APPLICATION OF NORM OPTIMIZATION IN COMPRESSIVE SENSING

A graduate project submitted in partial fulfillment of the requirements

For the degree of Master of Science

in Electrical Engineering

By
Amir Sadri

May 2013
The graduate project of Amir Sadri is approved:

Prof. Benjamin Mallard  
Date

Dr. Ronald W. Mehler  
Date

Dr. Xiyi Hang, Chair  
Date

California State University, Northridge
Dedication

I dedicate this project to Cathrine Steenstrup.

Without her love and support I could not have achieved this accomplishment.
### Table of Contents

- Signature page........................................................................................................... ii
- Dedication.................................................................................................................... iii
- Abstract ......................................................................................................................... vi
- Introduction .................................................................................................................. 1
- Theory and background ............................................................................................... 6
  - Measurement limitations and characteristics................................................................. 6
  - Norm .............................................................................................................................. 7
- Constrained and Non-constrained Optimization............................................................. 8
- Lagrangian ....................................................................................................................... 9
- KKT conditions ........................................................................................................... 9
- Gradient Descent and Gradient Descent with Projection ............................................ 10
- MATLAB Scripts ......................................................................................................... 14
  - Structure of Gradient Descent with Projection implementation scripts ...................... 15
    - Implementation Based on Search Direction and Lagrange Multipliers .................... 18
    - Implementation Based on Signal Decrease ............................................................... 19
  - Structure of Caller Script .......................................................................................... 20
- Numerical Results ......................................................................................................... 22
- Discussions and Conclusions ....................................................................................... 30
- Future Enhancements ................................................................................................... 30
References .................................................................................................................................................. 32

Appendix A: MATLAB Script – Caller Routine .................................................................................. 35

Appendix B: MATLAB Script - Conjugate Gradient with Projection Based on Search Direction and Lagrange Multipliers .................................................................................................................. 40

Appendix C: MATLAB Script - Conjugate Gradient with Projection Based on Signal Decrease ................................................................................................................................. 47

Appendix D: MATLAB Script – Diagnostic Code, Caller Routine .................................................. 53

Appendix E: MATLAB Script - Diagnostic Code, Conjugate Gradient with Projection Based on Search Direction and Lagrange Multipliers .................................................................................. 57
Abstract

APPLICATION OF NORM OPTIMIZATION IN COMPRESSIVE SENSING

By

Amir Sadri

Master of Science in Electrical Engineering

There has been a lot of interest in the research community the recent years in Compressed Sensing for solving under-determined systems of equations or reconstruction of sparse signals from highly inadequate samples (in the original or in a transformed domain such as Fourier transform or Wavelet transform). To solve these problems, many techniques have been suggested and developed a lot of which have focused on optimization. Among different optimization techniques $\ell_1$ norm optimization has been source of much investigation [2, 3, 4, 5, 10]. But later it was suggested that $\ell_p$ norm optimization with $0<p<1$ can result in more accurate signal reconstruction [10].

In this project a review of Compressed Sensing is offered and some approaches to signal reconstruction are discussed. In particular methods involving $\ell_p$ norm optimization are evaluated. To this end Matlab scripts have been developed to solve the $\ell_p$ norm optimization problem ($0<p<1$). They will be discussed in details and their results will be compared with results of $\ell_1$ norm optimization.
**Introduction**

It is well known that a signal can be exactly recovered from samples taken at a rate of twice its highest frequency, commonly known as the Nyquist rate. However in many applications signals have very high frequency components which will result in high Nyquist rates and far too many samples for exact signal reconstruction. To overcome this problem different compression techniques such as Transform Coding have been developed [1, 16].

However one would rather sample the signal at a lower rate than first sample it at a high rate and then compress the samples. Related to this subject is reconstruction of a signal from fewer samples of its Fourier transform. In the early 1900s Carathéodory showed that a positive linear combination of any k sinusoids can be uniquely represented by any 2k samples in time plus the value at t=0 [1].

Later Beurling showed that for signals that consist of a finite number of impulses the Fourier transform can be recovered from any sufficiently large samples by finding the signal with smallest $\ell_1$ norm approximation [1].

In recent years, works by Candès, Romberg, Tao and Donoho have proved that finite-dimensional signals that have a sparse or compressible representation in some basis or frame can be exactly recovered from a set of linear, non-adaptive samples at a considerably lower rate than the Nyquist rate [2, 3, 4, 5]. This new technique is known as *Compressed Sensing* or *Compressed Sampling*, and is mainly concerned about the design of these measurements schemes and reconstruction techniques [1].
To reconstruct a signal, it is of interest to minimize the error between the reconstructed and the original signal. As norm is usually used as a measure of the size of the error signal, this is equivalent to finding a signal \( \hat{x} \) such that the norm \( \| x - \hat{x} \|_p \) is minimum. Therefore the problem of signal reconstruction becomes equivalent to solving an optimization problem. This results in a class of optimization based algorithms for sparse signal reconstruction [1].

Another class of algorithms is Greedy Algorithms such as Orthogonal Matching Pursuit (OMP) and Iterative Thresholding. These algorithms are based on iteratively approximating either the signal by attempting to improve its mismatch to the measurements or its support by meeting a convergence criterion. OMP iterations try to find the column of the measurement matrix most correlated with the difference of the estimated signal and the original measurements [1].

Iterative Thresholding algorithms iterate a gradient descent step followed by a thresholding until a convergence criterion. An example of thresholding is hard thresholding that sets all entries of the signal to zero except for those entries with largest magnitude [1].

In this project the focus is on optimization based algorithms.

In mathematical terms an \( n \)-dimensional signal \( x \in \mathbb{R}^n \) has been sampled using a sensing matrix \( \mathbf{A} \) of size \( m \times n \) to obtain an \( m \)-dimensional samples signal \( y \in \mathbb{R}^m \) such that \( y = \mathbf{A}x \).

In Compressed Sensing \( m<n \) (or \( m<<n \)) and the samples are independent, so the system of equations of \( y = \mathbf{A}x \) is under-determined and has many solutions.
Given the fact that $\ell_0$ norm is the number of non-zero elements of a signal, and considering that the signal is sparse it makes sense to try to recover the signal by finding a solution to $y = Ax$ that has the minimum $\ell_0$ norm error so that the sparsity of the original signal is preserved as closely as possible [1]. In this regard solving the under-determined system of equations will be equivalent to solving the following optimization problem:

Minimize $\|x\|_0$ subject to $y = Ax$

In the objective function of the problem above is $\ell_0$ norm which is non-convex and the problem is NP-hard to solve. Candès, Romberg, Tao and Donoho have shown that if the $\ell_0$ norm objective function is replaced by an $\ell_1$ norm objective function, the result is still with a high probability an exact recovery of the original signal [2, 3].

Minimize $\|x\|_1$ subject to $y = Ax$

The benefit of an $\ell_1$ norm objective function is that it is a convex function and the problem can be represented as Linear Programming, for which many efficient algorithms have been already developed [2, 3, 4, 10].

Based on this, Romberg and Candès developed a collection of MATLAB routines for solving the $\ell_1$ optimization programs using standard Interior-Point methods which were collectively called the $\ell_1$-Magic package [6]. Their work inspired the idea of using the $\ell_p$ norm optimization with $0 < p < 1$. Chartrand explored this idea and showed that using $0 < p < 1$ results in better reconstruction using substantially fewer measurements. Figure 1 shows a comparison of number of measurements for different values of $p$ [10].
He also concludes that $\ell_p$ norm optimization with $0 < p < 1$ results in a better reconstruction of the signal [10]:

Figure 1: probabilities of exact reconstruction for a signal of sparsity $k = 16$, for different numbers of measurements $M$ and four values of $p$ [10]

Figure 2: reconstruction of real and imaginary part of a signal with sparsity $k = 16$ and number of measurements $M = 48$ for $p = 1$ (top) and $p = 0.5$ (bottom) [10]
Because $\ell_p$ norm optimization with $0<p<1$ is a non-convex optimization problem, Chartrand has used the Gradient Descent with Projection algorithm to reconstruct a signal from fewer Fourier measurements [10].

Using his ideas this project was aimed to develop MATLAB scripts that solve the $\ell_p$ norm optimization with $0<p<1$ implementing the Gradient Descent with Projection. The scripts will be used to reconstruct a sparse signal from its incomplete measurements and its compare the results will be compared with $\ell_1$ norm optimization results from the $\ell_1$-Magic package.
Theory and background

Throughout this report this notation has been used when possible:

\( \mathbf{A} = \) matrix, \( \mathbf{x} = \) vector, and \( b = \) scalar.

Also, references to formulas may precede the formulas in order not to confusion them with the formulas themselves.

Measurement limitations and characteristics

Acquiring \( m \) measurements of a signal \( \mathbf{x} \in \mathbb{R}^n \) can be mathematically expressed as \( \mathbf{y} = \mathbf{Ax} \) where \( \mathbf{y} \in \mathbb{R}^m \) and \( \mathbf{A} \) is an \( m \times n \) measurement (also known as sensing) matrix. Two fundamental questions are how the sensing matrix should be designed and how the signal should be recovered.

One particular property of interest is Restricted Isometry Property (RIP). It is a measure of how well the measurement matrix preserves the distance between any two \( k \)-sparse vectors and is used to find the minimum number of measurements for successful reconstruction. It can be shown that in order for an \( m \times n \) sensing matrix to satisfy an RIP of \( 2k \), the number of measurements \( m \) must be at least \([1, 16]\):

\[
m \geq C k \ln(n/k), \text{ where } C = \ln(\sqrt{24} + 1) / 2 \approx 0.28
\]

Another property of interest is coherence. Coherence of a matrix is the largest absolute inner product between of any of its two columns and is useful in finding the upper bound on the sparsity of the signal. If denoted by \( \mu(\mathbf{A}) \) then \( \sqrt{\frac{n-m}{m(n-1)}} \leq \mu(\mathbf{A}) \leq 1 \). For a sparse matrix \( m \ll n \) so \( \mu(\mathbf{A}) \geq 1 / \sqrt{m} \). For any measurement vector \( \mathbf{y} = \mathbf{Ax} \) there is a unique
reconstruction $k$-sparse signal $x$ if $k < (1 + 1/\mu(A)) / 2$. So when $A$ is sparse the upper bond becomes [1]: $k < (1+1/\sqrt{m})/2$.

Another condition for successful reconstruction is incoherence between the measurement matrix and the basis of the original signal. If the sensing matrix is constructed randomly with entries chosen according to Gaussian distribution, both the RIP and incoherence can be achieved with high probability. Moreover the number of measurement can be kept to a practical value [1]. For example a simple way to construct the measurement matrix is by selecting independent and identically distributed (iid) random variables from a Gaussian probability density function with mean of zero and variance of $1/n$ [16].

**Norm**

For $p \in [1, \infty]$, $\ell_p$ norm is defined as:

$$
\|x\|_p = \begin{cases} 
(\sum_{i=1}^n |x_i|^p)^{1/p} & \text{when } p \in [1, \infty) \\
\max |x_i| & \text{when } p = \infty 
\end{cases} \quad (i = 1, 2, \ldots, n)
$$

This definition can be extended to $0 < p < 1$ but it will not satisfy the triangle inequality so it will be a *quasinorm*. For $p = 0$ the definition of $\|x\|_0$ is the number of non-zero elements of $x$ and is not even a quasinorm [1]. Nevertheless in this report both $0 < p < 1$ and $p=0$ have been referred to as norms.

In signal reconstruction we are interested in finding a signal $\hat{x}$ in a subspace $A$ of $\mathbb{R}^n$ such that $\|x - \hat{x}\|_p$ is minimum; we want to compute the closest point in $A$ to $x$. This can be visualized as growing an $\ell_p$ ball around $x$ until it intersects with $A$. Figures 3 and 4 depict these concepts [1].
Constrained and Non-constrained Optimization

An unconstrained minimization problem is of the form

Minimize $f(x)$ where $f: \mathbb{R}^n \rightarrow \mathbb{R}$

If $f(x)$ is differentiable and convex, a necessary and sufficient condition for a point $x^*$ to be an optimum solution is that the gradient at point $x^*$ is zero: $\nabla f(x^*) = 0$

In constrained optimization problem the solution needs to satisfy the minimization criterion as well as some equality and/or inequality constraints. The general form of a constraint optimization can be expressed as [11, 12]:

Figure 3: Unit balls in $\mathbb{R}^2$ of $\ell^p$ norm for $p = 1, 2, \infty, \text{and } 1/2$ [1]

Figure 4: Best approximation of a point on a one-dimensional subspace $A$ of $\mathbb{R}^2$ using growing $\ell^p$ norm ball for $p = 1, 2, \infty, \text{and } 1/2$ [1]
Minimize \( f(x) \) subject to
\[
\begin{align*}
    h_i(x) &= 0 \quad (i=1,\ldots,p) \\
    d_i(x) &\geq 0 \quad (i = 1,\ldots,m)
\end{align*}
\]

**Lagrangian**

If we add a weighted sum of the constraint functions to the objective function we get a

*Lagrangian* function [7]:

\[
L(x, \lambda, \nu) = f(x) + \sum_{i=1}^{m} \lambda_i d_i(x) + \sum_{i=1}^{p} \nu_i h_i(x)
\]

The coefficients \( \lambda_i \) and \( \nu_i \) are called the *Lagrange multipliers*, and the vectors \( \lambda \in \mathbb{R}^m \) and \( \nu \in \mathbb{R}^p \) are called the *dual variables*. The *Lagrangian dual function* is defined as the minimum of the Lagrangian function over \( x \) [7]:

\[
g(\lambda, \nu) = \inf_x L(x, \lambda, \nu)
\]

But if \( p^* \) is the optimal value of the objective function then \( g(\lambda, \nu) \leq p^* \). So for each \((\lambda, \nu)\) with \( \lambda \geq 0 \) the Lagrange dual function results in a lower bound for the objective function and the lowest bound gives the minimum value. As such we can consider solving the following optimization problem [7]:

Minimize \( g(\lambda, \nu) \) subject to \( \lambda \geq 0 \)

This problem is called the *Lagrange dual problem* and the corresponding values \( \lambda^* \) and \( \nu^* \) are called the *optimal Lagrangian multipliers* or the *dual optimal multipliers* [7].

**KKT conditions**

Let’s denote the primary and dual optimal points by \( x^* \) and \((\lambda^*, \nu^*)\) and the corresponding optimal values by \( p^* \) and \( d^* \). If \( p^* = d^* \), since \( x^* \) minimizes the
unconstrained Lagrangian function \( L(x, \lambda^*, \nu^*) \) over \( x \), then the gradient of \( L \) at \( x^* \) should be zero [7]:

\[
\nabla L(x^*, \lambda^*, \nu^*) = \nabla f(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla d_i(x^*) + \sum_{i=1}^{p} \nu_i^* \nabla h_i(x^*) = 0
\]

Therefore we have the following set of equations known as Karush-Kuhn-Tucker (KKT) conditions [7]:

\[
\begin{align*}
    h_i(x^*) &= 0 \quad (i=1,\ldots, p) \\
    d_i(x^*) &\geq 0 \quad (i = 1,\ldots, m) \\
    \lambda_i^* &\geq 0 \quad (i = 1,\ldots, m) \\
    \lambda_i^* d_i(x^*) &= 0 \quad (i = 1,\ldots, m) \\
    \nabla f(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla d_i(x^*) + \sum_{i=1}^{p} \nu_i^* \nabla h_i(x^*) &= 0
\end{align*}
\]

**Gradient Descent and Gradient Descent with Projection**

Descent methods are used in unconstrained minimization. They produce a minimization sequence \( x^{(k)} \) \( (k = 1, \ldots) \) in form of \( x^{(k+1)} = x^{(k)} + t^{(k)} \Delta x^{(k)} \). Vector \( \Delta x^{(k)} \) is also called the *step direction* or the *search direction* and the scalar \( t^{(k)} \geq 0 \) is called the *step size*. Descent methods try to minimize \( f(x) \) by securing that in each iteration the value of the objective function has decreased: \( f(x^{(k+1)}) < f(x^{(k)}) \). Since the gradient points in the direction in which a function experiences its maximum change, it will be a natural choice for the search direction: \( \Delta x^{(k)} = - \nabla f(x^{(k)}) \). This is called the direction of the *steepest descent* and algorithms that use this direction are known as Gradient Descent methods [7, 13, 15].

Calculating the step size is often called a *Line Search* as it tries to determine the step size \( t^{(k)} \) along the line \( x^{(k)} + t^{(k)} \Delta x^{(k)} \). An *exact line search* tries to find the value of \( t^{(k)} \) for which \( f(x^{(k+1)}) \) will be minimum. This is an unconstrained one-variable minimization problem which can be solved analytically by solving \( df(x^{(k)} + t^{(k)} \Delta x^{(k)}) /dt^{(k)} = 0 \). In
practice exact line search is performed when the computational cost to solve the above equation is not high [7, 13].

Most line searches used in practice are *inexact*. They try to find a step size that approximately minimizes the objective function or decreases it sufficiently. In *backtracking line search* the step size is started with one $t^{(k)} = 1$. If $f(x^{(k+1)}) > f(x^{(k)})$ then the step size is reduced by a factor of $\beta$ (usually $\beta = 2$), $x^{(k+1)}$ is set to $x^{(k)} + t^{(k)}/\beta \Delta x^{(k)}$ and $f(x^{(k+1)}) > f(x^{(k)})$ is tested again. This procedure is repeated until $f(x^{(k+1)}) < f(x^{(k)})$ is satisfied [7,13,15].

Gradient Descent with Projection (also known as Projected Gradient Descent or Gradient Projection) is a variant of the Gradient Descent method used in constrained optimization. It was first developed by Rosen and is based on projecting the search direction into the subspace tangent to the active constraints (tangent affine subspace). The step is taken in the direction of the gradient (steepest descent) and its projection onto the affine space guarantees that each iteration step remains feasible. Iterations continue until projection of gradient on the tangent subspace is zero (meaning that it is not possible to make progress in the gradient direction any more) *and* all Lagrange multipliers are positive [8].

Rosen initially introduced this method for finding the maximum of a convex objective function with linear constraints and with more constraints than the dimension of the signal [8]. But the algorithm can be applied to cases where the objective function is non-linear and/or the number of constraints is less than the dimension of the signal. Haftka and Gürdal explain the algorithm for case of linear constraints. Expressing the problem function as [9]:

11
Minimize \( f(x) \) subject to \( g_j(x) = \sum_{j=1}^{m} a_{ij} x_i - b_j \geq 0 \) \((j = 1, 2, \ldots, k)\)

Their algorithm can be summarized as [9]:

1) At iteration point \( x_i \), find the active constraints \((g_a(x_i) = 0)\) and construct a matrix \((N)\) whose columns are gradients of active constraints \((\nabla g_a(x_i))\).

(since constraints are linear, active constraints can be written as: \( g_a(x_i) = N^T x_i - b = 0 \))

2) Compute the projection matrix: \( P = I - N (N^T N)^{-1} N^T \)

3) Compute the search direction \( s = -P \nabla f \)

4) Compute Lagrange multipliers \( r = (N^T N)^{-1} N^T \nabla f \)

5) If \( s \neq 0 \) \( \rightarrow \) proceed to calculate the step size \( \alpha \)

6) Otherwise \( s = 0 \) and:

   a. If \( r \geq 0 \) (all Lagrange multipliers are non-negative) \( \rightarrow \) then \( x_i \) is the minimum

   b. Otherwise some Lagrange multipliers are negative:

      i. remove the constraint associated with the most negative multiplier

      ii. Construct the new matrix with columns equal to gradients of the remaining active constraints \((N)\), then compute the new projection matrix \((P)\), the new search direction \((s)\), and new Lagrange multipliers \((r)\)

      iii. Repeat to test the search direction for \( s = 0 \)

7) Calculate the step size \( \alpha \)

8) Update the new iteration point as \( x_{i+1} = x_i + \alpha s \) and repeat the iteration

As mentioned before, to calculate the step size one can perform an exact line search and find the solution to \( df/d\alpha = 0 \), or alternatively perform an inexact line search for a desired reduction in the objective function.
In the latter, if the reduction ratio is \( \gamma \approx \frac{f(x_i) - f(x_{i+1})}{f(x_i)} \) and by linear approximation of \( f \) around \( x_{i+1} = x_i + \alpha s \) we will have \( \alpha^* = -\gamma \frac{f(x_i)}{(s^T \nabla f)} \). It should now be verified that the objective function has decreased, otherwise backtracking should be performed. There is also an upper limit on the step size. As the step size increases, some of the inactive constraints (constraints that are not equal to zero; \( g_j(x) > 0 \)) become active and finally violated (\( g_j(x) < 0 \)). To avoid this we must have \( g_j(x_{i+1}) = a^T_j (x_i + \alpha s) - b_j \geq 0 \) which means \( \alpha \leq -\frac{(a^T_j x_i - b_j)}{a^T_j s} = -\frac{g_j(x_i)}{a^T_j s} \). This equation results in a different value of \( \alpha \) for each inactive constraint that we call \( \alpha_j \). The minimum of these values sets the upper limit on the step size: \( \alpha_{\text{upper}} = \min\{\alpha_j > 0\} \ [9] \).
**MATLAB Scripts**

Chartrand uses Gradient Descent with Projection to solve the $\ell_p$ norm optimization and recover a signal from its incomplete Fourier coefficients. He considers an $M \times N$ matrix $\Phi$ such that for a signal vector $x \in \mathbb{C}^N$ and a measurement signal $y \in \mathbb{C}^M$, $y = \Phi x$ is the vector of Fourier coefficients of $x$ at $M$ randomly chosen frequencies. He solves the following optimization problem as [10]:

Minimize $\|u\|_p$ subject to $\Phi u = \Phi x$

His implementation of Gradient Descent with Projection can be summarized as [10]:

1) Iterations are of the form $u_{n+1} = P\Phi u = \Phi x (u_n + t_n d_n)$ where $P\Phi u = \Phi x$ denotes projection onto the affine subspace

2) At iteration $u_n$, step direction $d_n$ is the direction of the steepest descent of $\ell_p$ norm (or more precisely of $\|u\|_p^p / p$) at $u_n$, so $d_n = -\frac{1}{u_n} |u_n|^{p-2} u_n$

3) Calculate the step size $t_n$ and take a step (with this step size) in $d_n$ direction

4) Project orthogonally onto the affine constraint space $\Phi u = \Phi x$. Projection is done by computing the DFT of $u$, setting the previously chosen $M$ frequency coefficients to those of $x$, and then computing the inverse DFT

5) Stop the iterations if the obtained minimizer $u^*$ satisfied $\|u^* - x\|_\infty \leq 10^{-6}$.

Two implementations of Gradient Descent with Projection have been developed. One version is based on Haftka and Güral’s [9] and the other is based Chartrand’s article [10].
a) Because the purpose is to reconstruct the signal from incomplete samples instead of incomplete Fourier transform samples, matrix $\Phi$ is simply the measurement matrix $A$ such that $Ax = y$

b) The step is in the gradient direction, which is the steepest descent direction. But instead of first calculating the step size and then projecting onto the affine space, the step direction is first projected on the affine space and then the step size is calculated. It can be shown that the order plays no role in the algorithm.

c) Projection to the affine subspace as “DFT and inverse DFT” is valid when signal is reconstructed from Fourier coefficients. This is because Fourier transform is mathematically the projection of the signal onto the space with (Fourier) sinusoidal signals basis. Therefore in $y = \Phi x$ the $\Phi$ operator acts as projection to the space with sinusoidal signals basis and projection to a subspace is equivalent to replacing some of these frequency (Fourier) coefficients.

Here we are dealing with reconstructing the signal from incomplete measurements so this method cannot be applied.

d) Step size is calculated every iteration by assuming a desired function decrease and performing backtracking if the objective function increases.

e) In calculating the gradient, $|u_n|$ is replaced by $\sqrt{|u_n|^2 + \varepsilon^2}$.

**Structure of Gradient Descent with Projection implementation scripts**

To verify Chartrand’s result applied to recovering a sparse signal from incomplete samples, the Gradient Descent with Projection algorithm should be implemented. In this
Two Matlab script have been developed to implement the algorithm; one version is based on Haftka and Gürdal’s [9] in which search direction and Lagrange multipliers are used to find the optimal point and the other is based on Chartrand’s approach in which sufficient signal decrease is used [10]. CG_Projection() is the Matlab function that implements the Gradient Descent with Projection algorithm. As such two versions of this function have been developed with the following commonalities in both implementations:

1) Approximations and rounding errors: Both methods need to compare values to zero in different places. However, because of rounding errors comparing to zero may not generate the correct result. So both methods sometimes resort to comparing to a very small number instead of zero.

2) Adjustment of the initial point: Gradient Descents with Projection is based on projecting the search direction onto that subspace tangent to the active constraints in order to keep the iteration points feasible. However, the initial point passed to the script is not necessarily on this subspace. Therefore after the initial iterate is adjusted by moving it to the subspace.

3) There is a main loop that runs at most for a maximum number of iterations (max_iter).

4) Tangent Subspace matrix: The columns of the Tangent Subspace matrix are the coefficients of active constraints. Here all constraints are active because \( Ax - b = 0 \), so this matrix is equivalent to \( A^T \).

5) Search Direction: In Gradient Descent with Projection search direction is the projection of the steepest descent direction onto the affine space. Because the direction of the steepest descent is equal to the negative of the gradient, search direction is the
projection of the negative of the gradient onto the tangent active subspace. \( s = - P \nabla f(x) \) [9]).

6) Projection matrix: projection matrix has been computed as [9]: \( P = I - N (N^T N)^{-1} N^T \)

7) Gradient: gradient is calculated using the following equation [14]:
\( \nabla f(x) = (\nabla_1 f(x), \nabla_2 f(x), \ldots, \nabla_m f(x) ) \) where \( x = (x_1, x_2, \ldots, x_m) \) and \( \nabla_i f(x) = |p| |x_i|^{p-2} x_i \)

8) Step Size: step size can be calculated using an exact line search or an inexact line search. It has been chosen to conduct an inexact line search with a desired reduction of 1% in the objective function. Because the line search is not exact, the upper limit of the step size has been calculated as well.

9) Backtracking: Because of the inexact line search, backtracking should be performed to test if the step size is too big. A factor of 2 has been chosen for reducing the step size if the objective function increases instead of decrease and step size reduction is continued until either the objective function decreases or a maximum number of backtracking is performed (max_bktrac), beyond which the step size is too small to have a significant effect.

10) Updating the iterations: The iteration points have been updated (mnemonically) as:
\[ x_{\text{new}} = x_{\text{old}} + \text{step}_\text{size} \times \text{step}_\text{direction} \]

11) Effect of the updated iteration: If the objective function does not decrease significantly for a number of successive iterations and the iteration point has not changed significantly either, the script is not changing the result of optimization significantly and the main loop stops to execute.
12) In calculating the gradient, to avoid division by zero resulted from $|x_i|^{p^2}$ with $p<1$, the grad_f() function that calculates the gradient replaces $|x_i|$ by $\sqrt{|x_i|^2 + \varepsilon^2}$.

Both implementations have the following differences with Chartrand’s implementation:

i) Chartrand updates the step size every 100 iterations using an *exact* line search [10]. The developed scripts update the step size every iteration and use an *inexact* line search.

ii) To avoid division by zero in calculating the gradient, Chartrand replaces $|x_n|$ by $\sqrt{|x_n|^2 + \varepsilon^2}$, where $\varepsilon$ is initially set to 0.01 and decremented by 5% every 100 iterations [10]. The developed scripts use a fixed value for $\varepsilon$.

iii) Chartrand uses the minimum $\ell_2$ norm fit to the data as the initial point (and expects that changing the initial point would change the computed local minimizer) [10]. The developed scripts use the solution of the $\ell_1$-Magic package as the initial point.

**Implementation Based on Search Direction and Lagrange Multipliers**

The code for this implementation can be found in Appendix B. The flow of the script can be summarized as:

a) After defining the global variables and constants and initializations, the initial iterate has been adjusted. The script then performs its main loop for at most a maximum number of iterations (*max_iter*).

b) The Tangent Subspace matrix is computed.

c) The Search Direction vector is computed and compared to zero.

d) If the search direction is not zero, step size is calculated.

d) If the search direction is zero, Lagrange multipliers are computed as [9]:

\[
\text{grad}_f() \text{ function that calculates the gradient replaces } |x_i| \text{ by } \sqrt{|x_i|^2 + \varepsilon^2}.
\]
\[ \lambda = (N^T N)^{-1} N^T \nabla f. \]

Lagrange multipliers and then tested for being positive.

d-1) If all Lagrange multipliers are positive then the optimum point is found.

d-2) If any of the Lagrange multipliers is not positive, the constraint corresponding with the most negative Lagrange multiplier has been removed and a new Tangent Subspace matrix along with a new projection matrix and a new search direction have been calculated. Then the new search direction has been compared to zero.

e) The step size has been computed using an inexact line search.

f) Iteration point has been updated: \[ x_{\text{new}} = x_{\text{old}} + \text{step}_\text{size} \times \text{step}_\text{direction} \]

g) New iteration point has been checked for backtracking.

h) The effect of the iteration on the objective function and the signal value will be checked to stop the main loop if the effect is not significant.

**Implementation Based on Signal Decrease**

The code for this implementation can be found in Appendix C.

a) After defining the global variables and constants and initializations, we adjust the initial iterate. The script then performs its main loop for at most a maximum number of iterations (\textit{max\_iter}).

b) The Tangent Subspace matrix is computed.

c) The Search Direction vector is computed.

d) The Step Size has been computed.
e) Iteration point has been updated: \( x_{\text{new}} = x_{\text{old}} + \text{step\_size} \times \text{step\_direction} \)

f) New iteration point has been checked for backtracking.

g) The effect of the iteration on the objective function and the signal value will be checked to stop the main loop if the effect is not significant.

**Structure of Caller Script**

In order to call CG_Projection() with different values of p, an auxiliary caller script is developed. The code for the script can be found in Appendix A.

Since the caller script solves the \( \ell_1 \) norm problem by calling the L1-Magic package routines [6], it should be placed in the main L1-Magic directory so that it has access to the files in the package.

Because results of \( \ell_p \) norm optimization with \( 0 < p < 1 \) should be compared to results of \( \ell_1 \) norm optimization using L1-Magic package [6], the method to generate the signal comes from the L1-Magic package so that the same signal is optimized in the two methods.

The following signal and measurement values have been selected: signal length (n) = 512, number of spikes (sparsity) of the signal (k) = 20, and number of observations to make (m) = 120.

It can be verified that the required minimum number of observations is satisfied:

\[
m \geq C \times k \ln(n/k), \quad \text{where} \quad C = \ln(\sqrt{24} + 1) / 2 \approx 0.28 \Rightarrow 120 > 0.28 \times 20 \times \ln(512/20) \Rightarrow 120 > 18.16
\]
As discussed before an easy way to achieve both the RIP and incoherence for the measurement matrix is to construct it randomly. So the script creates a sparse random discrete signal and a random measurement matrix.

The script first solves the $\ell_1$ norm problem using L1-Magic package in small and large scale modes and then calls the CG_Projection() routine to solve the $\ell_p$ norm problem for different values of $p$. As Chartrand has stated, it is expected that the value of the initial point affects the value of the computed minimum point [10]. Due to the non-convex nature of the $\ell_p$ norm objective function, as the dimension of the original signal increases the choice of the initial point becomes more important. So it is decided to us the result of the L1-Magic $\ell_1$ norm optimization as the initial point for the $\ell_p$ norm optimization solver script CG_Projection(). However the script allows this initial point to be placed away from this, with step sizes equal to the distance between the $\ell_1$ norm optimization solution and the original signal. To compare the results of $\ell_1$ and $\ell_p$ optimization, norm of the difference between the original signal and the reconstructed signal has been used as the measure of reconstruction.
**Numerical Results**

Table 1 shows the results of $\ell_p$ norm optimization using the script base on Lagrange multiplier and Search direction and Table 2 shows the result of using the script based signal decrease. It can be seen that in both case $\ell_p$ norm with 0<p<1 results in a closer reconstruction of the signal.

The results also show that due to the nature of Gradient Descent with Projection, the outcome of this method has a smaller deviation from $\mathbf{Ax} - \mathbf{b} = 0$. This can be easily attributed to the fact that in Gradient Descent with Projection we try to stay on the affine subspace, which in this case is the $\mathbf{Ax} - \mathbf{b} = 0$ hyperplanes subspace.
<table>
<thead>
<tr>
<th>Method</th>
<th>Norm$(x – x^*)$</th>
<th>Max. Deviation $\max( Ax^* - b)$</th>
<th>Norm of Deviation $|Ax^* - b|$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell_1$</td>
<td>0.000019030954064</td>
<td>0.000000000023106</td>
<td>0.000000000108625</td>
</tr>
<tr>
<td>$\ell_1$ Large Scale</td>
<td>0.000019032978731</td>
<td>0.0000000001616389</td>
<td>0.0000000001616389</td>
</tr>
<tr>
<td>$p = 0.1$</td>
<td>0.000016485248947</td>
<td>0.0000000000000001</td>
<td>0.0000000000000002</td>
</tr>
<tr>
<td>$p = 0.3$</td>
<td>0.000013740447193</td>
<td>0.0000000000000001</td>
<td>0.0000000000000002</td>
</tr>
<tr>
<td>$p = 0.5$</td>
<td>0.000013578628531</td>
<td>0.0000000000000001</td>
<td>0.0000000000000002</td>
</tr>
<tr>
<td>$p = 0.6$</td>
<td>0.000013663045843</td>
<td>0.0000000000000000</td>
<td>0.0000000000000002</td>
</tr>
<tr>
<td>$p = 0.8$</td>
<td>0.000010128304118</td>
<td>0.0000000000000000</td>
<td>0.0000000000000000</td>
</tr>
<tr>
<td>$p = 0.95$</td>
<td>0.000018383824895</td>
<td>0.0000000000000000</td>
<td>0.0000000000000000</td>
</tr>
<tr>
<td>$p = 1$</td>
<td>0.000019030954064</td>
<td>0.0000000000000000</td>
<td>0.0000000000000000</td>
</tr>
</tbody>
</table>

Table 1: $\ell_1$ vs. $\ell_1$ (0 < $p$ < 1) norm optimization results – Lagrange multipliers method

<table>
<thead>
<tr>
<th>Method</th>
<th>Norm$(x – x^*)$</th>
<th>Max. Deviation $\max( Ax^* - b)$</th>
<th>Norm of Deviation $|Ax^* - b|$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell_1$</td>
<td>0.000007549388703</td>
<td>0.00000000024468</td>
<td>0.000000000105644</td>
</tr>
<tr>
<td>$\ell_1$ Large Scale</td>
<td>0.000007552300972</td>
<td>0.000000001896926</td>
<td>0.000000001896926</td>
</tr>
<tr>
<td>$p = 0.1$</td>
<td>0.00002686445645</td>
<td>0.0000000000000000</td>
<td>0.0000000000000002</td>
</tr>
<tr>
<td>$p = 0.3$</td>
<td>0.00002123995889</td>
<td>0.0000000000000001</td>
<td>0.0000000000000003</td>
</tr>
<tr>
<td>$p = 0.5$</td>
<td>0.00001975850430</td>
<td>0.0000000000000001</td>
<td>0.0000000000000002</td>
</tr>
<tr>
<td>$p = 0.6$</td>
<td>0.00001693904412</td>
<td>0.0000000000000000</td>
<td>0.0000000000000000</td>
</tr>
<tr>
<td>$p = 0.8$</td>
<td>0.0000016493665</td>
<td>0.0000000000000000</td>
<td>0.0000000000000000</td>
</tr>
<tr>
<td>$p = 0.95$</td>
<td>0.00001261120623</td>
<td>0.0000000000000000</td>
<td>0.0000000000000000</td>
</tr>
<tr>
<td>$p = 1$</td>
<td>0.000018383824895</td>
<td>0.0000000000000000</td>
<td>0.0000000000000000</td>
</tr>
</tbody>
</table>

Table 2: $\ell_1$ vs. $\ell_1$ (0 < $p$ < 1) norm optimization results – Signal decrease method

During the development of the scripts it was important to visualize the progress of the script. Therefore diagnostic code was added to generate graphs when the original signal has a dimensional of two (so that $z = f(x,y)$ can be graphed). The code for the diagnostic version of the script can be found in Appendix F.

Figures 5 through 8 and figures 9 through 12 show screenshots of running the Lagrange multiplier based script for $p=0.3$ and $p=0.95$ respectively.
Figure 5: Overview of objective function, constraints plane and intersection line, Lagrange multiplier based script, $p = 0.3$

Figure 6: Gradient descent steps on x-y plane and intersection line, Lagrange multiplier based script, $p = 0.3$
Figure 7: Overview of gradient descent steps, Lagrange multiplier based script, $p = 0.3$

Figure 8: Last gradient descent steps, Lagrange multiplier based script, $p = 0.3$
Figure 9: Overview of objective function, constraints plane and intersection line, Lagrange multiplier based script, $p = 0.95$

Figure 10: Gradient descent steps on x-y plane and intersection line, Lagrange multiplier based script, $p = 0.95$
Figure 11: Overview of gradient descent steps, Lagrange multiplier based script, p = 0.95

Figure 12: Last gradient descent steps, Lagrange multiplier based script, p = 0.95
Figures 13 through 16 are screenshots of the signal decrease based script for $p=0.3$.

Figure 13: Overview of objective function, constraints plane and intersection line, signal decrease based script, $p = 0.3$

Figure 14: Gradient descent steps on x-y plane and intersection line, signal decrease based script, $p = 0.3$
Figure 15: Overview of gradient descent steps, signal decrease based script, $p = 0.3$

Figure 16: Last gradient descent steps, signal decrease based script, $p = 0.3$
Discussions and Conclusions

It has been shown that Gradient Descent with Projection can be applied to solve an $\ell_p$ norm optimization problem in the framework of sparse signal reconstruction and compressed sensing. It has been demonstrated that solving the $\ell_p$ norm optimization problem with $0 < p < 1$ can yield better results (closer reconstruction) than the $\ell_1$ norm optimization.

Due to the non-convex nature of the $\ell_p$ norm ($0 < p < 1$) optimization problem it is expected that the number of local minima will increase with the dimension of the signal and the choice of the initial iterates is important for convergence to the global minimum. Choosing the result of the convex $\ell_1$ norm optimization problem has been found to be a very good choice for the initial iterate. In experimenting how far the starting point can deviate from this, it was found that denoting the norm of the difference between the original signal and the solution to $\ell_1$ norm optimization problem by $\Delta$, then for most signals an initial point of at most $2\Delta$ away from the solution of $\ell_1$ norm optimization problem may still be acceptable.

Future Enhancements

Projection is performed by calculating the projection matrix and Lagrange multipliers using matrix inversion. A more efficient method suggested by Haftka and Gürdal would be to use QR factorization to avoid matrix inversions [9]. The QR factorization of the $N$ matrix is performed and the last $m - r$ rows of the Q factor are selected (m is the number of measurements and r is the number of active constraints). If we call this matrix $Q_r$, then the projection matrix can be computed as $P = Q_r^T Q_r$ [9].
A similar enhancement can be made in computing the Lagrange multipliers. Using the same QR factorization, Lagrange multipliers \( r \) can be computed by solving \( R \ r = Q_1 \nabla f \) for \( r \). Here \( Q_1 \) is a matrix consisting of the first \( r \) rows of the \( Q \) factor [9].

It should be noted that what Haftka and Gürdal call QR factorization is better termed as QR decomposition. It is to find matrices \( Q \) and \( R \) for a matrix \( A \) such that \( Q^*A = (R \ 0) \), where \( A \) is a \( nxr \) matrix, \( Q \) is an \( nxn \) orthogonal matrix, \( R \) is an \( rxr \) upper triangular matrix, and \( 0 \) is an \((n-r)xr\) matrix. However, in linear algebra QR factorization means finding matrices \( Q \) and \( R \) for a matrix \( A \) such that \( Q^*R = A \), where \( A \) is an \( nxr \) left-invertible matrix, \( Q \) is an \( nxn \) orthogonal matrix, and \( R \) is an \( rxr \) upper triangular matrix. Therefore using the Matlab qr() function for this purpose may not be correct.

Another possible enhancement is to modify the code such that it converges to the optimal point even if it starts from the least square (minimum energy) solution to \( A^*x=b \). It is worth to notice that per Chartrand, changing the initial iterate can change the final outcome [10]. So it may not always be possible to start from the least square solution and arrive at the global minimum. Figure 11 can demonstrate the situation where there are 2 minima. It may happen that the script will end at the local minimum and not the global one if it starts from a point on far left on the intersection line.
References


http://www.maths.bris.ac.uk/~maxmr/opt/nielsen_constrained.pdf

http://www.ee.ucla.edu/~vandenbe/103/reader.pdf


Appendix A: MATLAB Script – Caller Routine

```matlab
%% Initial setup
clc
close all;
% put key subdirectories in path if not already there
path(path, './Optimization');
path(path, './Data');
load RandomStates
rand('state', rand_state);
% log directory
dirname = 'CGProj_Log';
if (exist(dirname,'dir') == 0)
    mkdir (dirname);
end
filename = strcat('Results_', datestr(now,'yyyy-mm-dd_HH-MM-SS') , '.txt');
ResultFile = fopen( strcat(dirname,filename) , 'at');

%% Globals
N = 512; % signal length
K = 20; % number of spikes in the signal
M = 120; % number of observations to make

%% Original signal and measurements
format long;
% random +/- 1 signal
x = zeros(N,1);
q = randperm(N);
x(q(1:K)) = sign(randn(K,1));
% measurement matrix
A = randn(M,N);
```
A = orth(A');

% observations
b = A*x;

%%% initial guess = min energy
x0_1 = (A*A')\b;
x0 = A'*x0_1;

%%% solve the problem using the Primal-Dual Interior Point method (L1-Magic)
x_Interior = l1eq_pd(x0, A, [], b, 1e-3);

% large scale
Afun = @(z) A*z;
Atfun = @(z) A'*z;
x_InteriorLarge = l1eq_pd(x0, Afun, Atfun, b, 1e-3, 30, 1e-8, 200);

%%% Initial graphs
clc
disp(''); disp('----------- Lp Norm optimization -----------'); disp('');
results = [x x0];
header = sprintf('%20s', 'Original signal', 'Min Energy');

%graph the results
h1 = figure('Name','Signal Plot Window');
hold all; plotbrowser('on'); grid on;
stem(x,'LineWidth',2,'DisplayName','Original Signal');
stem(x0,'DisplayName','Min Energy');
stem(x_Interior,'DisplayName','L1 Magic');
stem(x_InteriorLarge,'DisplayName','L1 Magic Large');
norm_Interior = norm(x- x_Interior);
disp(sprintf( 'Norm of Interior = %20.15f' , norm_Interior ));
fprintf(ResultFile, 'Norm of Interior = %20.15f \n' , norm_Interior );
norm_InteriorLarge = norm(x - x_InteriorLarge);
disp(sprintf( 'Norm of Interior Large = %20.15f' , norm_InteriorLarge ));
fprintf(ResultFile, 'Norm of Interior Large = %20.15f
', norm_InteriorLarge );
dev_Interior = max(A*x_Interior - b);
norm_dev_Interior = norm(A*x_Interior - b);
disp(sprintf( 'Deviation of Interior: Max = %20.15f , Norm = %20.15f
' , dev_Interior, norm_dev_Interior));
fprintf(ResultFile, 'Deviation of Interior: Max = %20.15f , Norm = %20.15f
' , dev_Interior, norm_dev_Interior);
dev_InteriorLarge = max(A*x_InteriorLarge - b);
norm_dev_InteriorLarge = max(A*x_InteriorLarge - b);
disp(sprintf( 'Deviation of Interior Large: Max = %20.15f , Norm = %20.15f
' , dev_InteriorLarge, norm_dev_InteriorLarge));
fprintf(ResultFile, 'Deviation of Interior Large: Max = %20.15f , Norm = %20.15f
' , dev_InteriorLarge, norm_dev_InteriorLarge);

%% Solve the Lp norm problem using Conjugate Gradients with Projection method
% p holds the values to run the optimization on
p = [ 0.1 0.3 0.5 0.6 0.8 0.95 1 ];
% x_cgp holds optimization results using Conjugate Gradients with Projection method
x_cgp = zeros(N, length(p));
% maximum deviation of CGP method initial point from solution of Primary-Dual Interior method (L1-Magic)
max_deviation = 0;
delta_x = x - x_Interior;
x0_cgp = x_Interior - (max_deviation * delta_x);
disp(sprintf('Max deviation = %d', max_deviation));
fprintf(ResultFile, '
Max deviation = %d
', max_deviation);
for i = 1 : length(p)
    disp(''); disp(strcat('Norm_',num2str(p(i)))); disp('^^^^^^^^^^^^^^^^^^^^^^^^^');
fprintf(ResultFile, '
Norm_%1.2f
', p(i));
% Solve the norm_p problem

x_cgp(:,i) = CG_Projection(A, b, x0_cgp, p(i));

if (~isnan(x_cgp(:,i)))

    % logging the results
    figure(h1);
    results = [results, x_cgp(:,i)];
    norm_cgp = norm(x-x_cgp(:,i));
    dev_cgp = max(A*x_cgp(:,i) - b);
    norm_dev_cgp = norm(A*x_cgp(:,i) - b);
    disp('-----------------------
          |-----------------------
          |-----------------------
          -----------------------

    fprintf( ResultFile, '-----------------------
          |-----------------------
          |-----------------------
          -----------------------
    
    disp(sprintf( strcat( 'norm(x - p_',num2str(p(i)),') = %20.15f') , norm_cgp ));
    fprintf( ResultFile, 'norm(x - p_%1.2f) = %20.15f
', p(i), norm_cgp );
    disp(sprintf( 'Deviation of p_%1.2f: max = %20.15f, norm = %20.15f' , p(i), dev_cgp, norm_dev_cgp));
    fprintf( ResultFile, 'Deviation of p_%1.2f: max = %20.15f, norm = %20.15f
', p(i), dev_cgp, norm_dev_cgp);
    measure_of_improvement = norm_cgp - norm_Interior;
    if ( measure_of_improvement < 0 )
        disp('SUCCESS: Norm_p method improved over Interior method.');
        fprintf( ResultFile,'SUCCESS: Norm_p method improved over Interior method
');
    else
        disp('FAILURE:: Norm_p method did not improve over Interior method!');
        fprintf( ResultFile,'FAILURE: Norm_p method did not improve over Interior method!
');
    end

    stem(x_cgp(:,i),'DisplayName',strcat('Norm_',num2str(p(i))), 'Marker', '+' );
    header = strcat(header, sprintf('%20s',strcat('p_',num2str(p(i)))));

else
    disp('FAILURE: Conjugate Gradients with Projection method failed to solve the problem!!!');

end
fprintf(ResultFile,'FAILURE: Conjugate Gradients with Projection method failed to solve the problem!!!');
end

%% Log the signal values
fprintf (ResultFile, \n %s ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n', header);

for i = 1: length(results(:,1))
    fprintf(ResultFile, \%20.15f', results(i,:));
    fprintf(ResultFile, \n');
end

disp('Finished ...... !');
function xp = CG_Projection(A, b, x0, p )

%%% Globals
%clc;
%record iteration results in a log file
dirname = 'CGProj_Log';
filename = strcat('CGProjection_ ', datestr(now,'yyyy-mm-dd_HH-MM-SS') , '.txt');
LogFile = fopen( strcat(dirname,filename) , 'at');
format long;
%due to rounding errors, we compare to epsilon instead of zero
eps_0 = 1e-10;
eps_grad = 1e-4; %to calculate gradient
eps_s0 = 1e-14; %for step vector
eps_L0 = 1e-14; %for Lagrange multipliers
iter_num = 1; max_iter = 4e4;
%stop if objective function does not significantly change successively
tol = 1e-10;
decr_counter = 0; max_decr = 3;
max_bktrac = 50;
[m,n] = size(A);
xp = NaN; %initialize returned value to an invalid number

%%% Initial guess
%add a vector to x0 such that the initial guess lies on the Tangent Subspace
% that is: A*(x0 + delta_x) - b = 0
x1 = linsolve(A, b- A*x0);
x01 = x0 + x1;
%check if x01 is closer to the Tangent Subspace
if (max(A*x0 -b) < max(A*x01 -b))
x = x0;
else
    x = x01;
end

%% Iterations
% x = x0;
fx = f_x(x,p);
xp = NaN; %initialize returned value to an invalid number
fprintf(LogFile, '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
');
fprintf(LogFile, '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
');
done = 0;
while ( (iter_num <= max_iter) && (done == 0) )
    fprintf(LogFile, 'Iteration: %d 
', iter_num );

%% Tangent Subspace Matrix (N matrix) calculation
    gradF = grad_f(x,p, eps_grad);
    if isnan(gradF)
        disp('Error: unable to calculate the gradient of f!');
        fprintf(LogFile, 'Error: unable to calculate the gradient of f at a point! 
');
        fprintf(LogFile, '%20.16f', x);
        xp = NaN;
        break;
    end
    %general method to calulate the 'N' matrix
    g = A*x-b;
    M = length(g);
    gradG = A;
    N_Matrix = []; ga_idx = [];
    for i=1:M
% if ( 0 <= g(i) && g(i) <= eps_0 ) % if constraint is active (= 0)
% if ( abs(g(i)) <= eps_0 ) % if constraint is active (= 0)
% temp1 = horzcat(N_Matrix, gradG(i,:)') ;
% N_Matrix = temp1;
% temp2 = vertcat(ga_indx, i) ;
% ga_indx = temp2;
% end
% end

% g = A*x -b; % active constraints (all constraints are active because Ax -b =0 )

%% Search Direction
s = zeros(1,n); %initialize search direction to zero
% while ( max(s) == 0 ) %s is zero
while ( max(abs(s)) <= eps_s0 ) %s is (practically) zero

%% Projection Matrix calculation
if isempty(N_Matrix)
    Lambda = []; P = eye(n);
else
    P = eye(n) - N_Matrix* ((N_Matrix'*N_Matrix)
Lambda = ((N_Matrix'*N_Matrix)
end

%% Step Direction
s = -P * gradF ;
d = norm(s); s = s./d; %normalize s
    if ( max(abs(s)) <= eps_s0 ) %s is (almost) zero

%% active constraints (all constraints are active because Ax -b =0 )
% N_Matrix is Tangent subspace matrix with columns = gradient of constraints
N_Matrix = A';
M = length(g);
% ga_indx is vector of indices of active (= 0) constraints
ga_indx = (1:M); % all constraints are active (Ax -b =0)
if ( max(s) == 0 ) %s is zero

    [min_Lambda indx_min] = min(Lambda);
    if ( (min_Lambda >= -eps_L0) || (length(Lambda) == 1) )

%all Lagrange multipliers are non-negative or no more multiplier are left to remove

    disp('Optimum point calculated!');
    xp = x;
    fprintf(LogFile, 'Optimum point calculated! \n');
    fprintf(LogFile, '\n %f', x);
    done = 1;
else

%remove the active constraint corresponding to the most negative Lagrange multiplier

    if (indx_min == 1) %first column
        new_N_Matrix = N_Matrix(:,2:end);
        new_ga_indx = ga_indx(2:end);
    elseif (indx_min == length(Lambda)) %last column
        new_N_Matrix = N_Matrix(:,1:end-1);
        new_ga_indx = ga_indx(1:end-1);
    else
        new_N_Matrix = N_Matrix(:,[1:(indx_min-1), (indx_min+1):end] ) ;
        new_ga_indx = ga_indx([1:(indx_min-1) (indx_min+1):end] );
    end

    N_Matrix = new_N_Matrix;
    ga_indx = new_ga_indx;
end
end

%% Step size

%calculate the upper limit on step size (alpha )
gama = 0.01;  %desired decrease in objective function

step_upper = 1e100;  %practically infinite

for i=1:length(g)
    if (g(i) > eps_0)
        k = A(i,:)* s ;
        if ( k < -eps_0 )
            alpha_i = -g(i) / k ;
            step_upper = min (step_upper, alpha_i);
        end
    end
end

%one dimensional minimization for step size (alpha) using desired decrease in objective function
alpha_star = - (gama*fx) / (s'* gradF) ;
if (alpha_star < -eps_0)
    disp('Warning, alpha_star is negative!');
end
alpha = min(step_upper, alpha_star);

%% update the iteration
x_old = x;
fx_old = fx;
x = x_old + (alpha * s);
fx = f_x(x,p);

% perform backtracking if objective function has increased
while ( (fx - fx_old > tol) && (cntr <= max_bktrac) )
    gama = gama /2; %reduce the desired increase
    alpha_star = - (gama*fx) / (s'* gradF) ;
    alpha = min(step_upper, alpha_star ) ;
x = x_old + (alpha * s);
end
fx = f_x(x,p);
cntr = cntr + 1;
end
iter_num = iter_num + 1 ;
fprintf(LogFile,'fx_old: %12.8f , fx_new: %12.8f , delta_fx: %12.8f \n', fx_old, fx , fx - fx_old);
fprintf(LogFile, '----------------------------------- \n');

%%% check objective function change and iteration point change
if ( norm(x - x_old) < tol && (fx - fx_old < tol) && (fx < fx_old) )
decr_counter = decr_counter + 1;
if (decr_counter == max_decr)
disp('Iteration stopped to decrease significantly. Optimum point is found!');
fprintf(LogFile,'Iteration stopped to decrease significantly. Optimum point is found! \n');
fprintf(LogFile,'\n Number of Iterations = %d \n Optimum point: %20.16f \n', iter_num, x);
xp = x;
done = 1;
end
else
decr_counter = 0;
end
end
%% Exit the program
disp(' '); disp(sprintf('Number of iterations = %d',iter_num));
if (iter_num > max_iter)
disp('Error: Maximum iterations reached. Optimum point could not be found!');
fprintf(LogFile, 'Error: Maximum iterations reached. Optimum point could not be found! \n');
end
% return;
end

%%%
function fx = f_x(x,p)
    fx = sum( (abs(x).^p) );
end

%%
function [gradF] = grad_f(x,p, eps)
    N= length(x);
    gradF = zeros(N,1);
    for i = 1:N
        x(i) = sign(x(i))*sqrt(x(i)^2 + eps^2); %to avoid divid by zero if x(i)=0
        if (x(i) == 0)
            x(i) = sqrt(x(i)^2 + eps^2);
        end
        gradF(i) = p*x(i)* (abs(x(i))^(p-2)) ;
        if isnan(gradF(i))
            gradF = NaN;
            return
        end
    end
end
function xp = CG_Projection(A, b, x0, p )

%% Globals
clc;
%record iteration results in a log file
dirname = '\CGProj_Log_simple\';
filename = strcat('CGProjection_' , datestr(now,'yyyy-mm-dd_HH-MM-SS') , '.txt');
LogFile = fopen( strcat(dirname,filename) , 'at');
format long;
%due to rounding errors, we compare to epsilon instead of zero
eps_grad = 1e-5; %to calculate gradient
eps_0 = 1e-9; %for step vector
iter_num = 1; max_iter = 4e4;
delta_grad = 1e-5;
%stop if objective function does not significantly change successively
tol = 1e-10;
decr_counter = 0; max_decr = 3;
max_bktrac = 100;
[m,n] = size(A);
xp = NaN; %initialize returned value to an invalid number

%% Initial guess
%add a vector to x0 such that the initial guess lies on the Tangent Subspace
%that is: A*(x0 + delta_x) - b = 0
x1 = linsolve(A, b- A*x0);
x01 = x0 + x1;
%check if x01 is closer to the Tangent Subspace
if (max(A*x0 -b) < max(A*x01 -b))
x = x0;
else
    x = x01;
end

%% Iterations
%     x = x0;
fx = f_x(x,p);
xp = NaN;    %initialize returned value to an invalid number
fprintf(LogFile, '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
');
fprintf(LogFile, '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
');
done = 0;
while ( (iter_num <= max_iter) && (done == 0) )
    fprintf(LogFile, 'Iteration: %d 
', iter_num );
    fprintf(LogFile, '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
');
% % Tangent Subspace Matrix (N matrix) calculation
    gradF = grad_f(x,p, eps_grad);
    if isnan(gradF)
        disp('Error: unable to calculate the gradient of f!');
        fprintf(LogFile, 'Error: unable to calculate the gradient of f at a point! 
');
        fprintf(LogFile, 'Error: unable to calculate the gradient of f at a point! 
');
        fprintf(LogFile, '%20.16f', x);
        xp = NaN;
        break;
    end;
    GaussNewton_xp = N * gradF + g;
    % general method to calculate the 'N' matrix
    g = A*x-b;
    M = length(g);
    gradG = A;
    N_Matrix = [];
    ga_indx = [];
    for i=1:M
        if ( 0 <= g(i) && g(i) <= eps_0 )    %if constriant is active (= 0)
            if ( abs(g(i)) <= eps_0 )    %if constriant is active (= 0)
                %
            end
        end
    end
end

48
% if ( g(i) == 0 )  %if constraint is active (= 0)
% temp1 = horzcat(N_Matrix, gradG(i,:)) ;
% N_Matrix = temp1;
% temp2 = vertcat(ga_indx, i) ;
% ga_indx = temp2;
% end
% end

g = A*x -b;  % active constraints (all constraints are active because Ax -b =0 )
% N_Matrix is Tangent subspace matrix with columns = gradient of constraints
N_Matrix = A';  %

%%% Projection Matrix calculation
if isempty(N_Matrix)
    P = eye(n);
else
    P = eye(n) - N_Matrix* ((N_Matrix'*N_Matrix)
\N_Matrix');
end

%%% Step Direction
s = -P * gradF ;
d = norm(s); s = s/d;  %normalize s

%%% Step size
% calculate the upper limit on step size

gama = 0.01;  %desired decrease in objective function
step_upper = 1e100;  %practically infinite
for i=1:length(g)
    if (g(i) > eps_0)
        k = A(i,:)* s ;
        if ( k < -eps_0 )
\[
\alpha_i = -\frac{g(i)}{k};
\]
\[
\text{step_upper} = \min(\text{step_upper}, \alpha_i);
\]
end
end
end

% one dimensional minimization for step size (alpha) using desired decrease in objective function
\[
\alpha_{\text{star}} = -\left(\frac{\gamma \cdot f(x)}{(s^\top \cdot \text{gradF})}\right);
\]
if (\alpha_{\text{star}} < 0)
    disp('Warning, alpha_star is negative!');
end
\[
\alpha = \min(\text{step_upper}, \alpha_{\text{star}});
\]

%%% Update the iteration
\[
x_{\text{old}} = x;
\]
\[
f_{\text{x}}_{\text{old}} = f_{\text{x}};
\]
\[
x = x_{\text{old}} + (\alpha \cdot s);
\]
\[
f_{\text{x}} = f_{\text{x}}(x,p);
\]
\[
\text{cntr} = 1;
\]
% perform backtracking if objective function has increased
while (\ (f_{\text{x}} > f_{\text{x}}_{\text{old}} + \text{tol} ) && \ (\text{cntr} <= \text{max_bktrac}) )
    gama = gama / 2; % reduce the desired increase
    \[
    \alpha_{\text{star}} = -\left(\frac{\gamma \cdot f_{\text{x}}}{(s^\top \cdot \text{gradF})}\right);
    \]
    \[
    \alpha = \min(\text{step_upper}, \alpha_{\text{star}});
    \]
    \[
    x = x_{\text{old}} + (\alpha \cdot s);
    \]
    \[
    f_{\text{x}} = f_{\text{x}}(x,p);
    \]
    \[
    \text{cntr} = \text{cntr} + 1;
    \]
end
\[
\text{iter_num} = \text{iter_num} + 1;
\]
\[
\text{fprintf(LogFile,}'fx_{\text{old}} = \%20.16f , fx_{\text{new}} = \%20.16f , delta_{\text{fx}} = \%20.16f \ln' , fx_{\text{old}} , fx , fx-fx_{\text{old}})\);
\]
\[
\text{fprintf(LogFile,}'----------------------------------- \ln');
\]
%% Check objective function change
if ( norm(x - x_old) < tol && (fx - fx_old < tol) && (fx < fx_old) )
decr_counter = decr_counter + 1;
if (decr_counter == max_decr)
disp('Iteration stopped to decrease significantly. Optimum point is found!');
fprintf(LogFile,'Iteration stopped to decrease significantly. Optimum point is found! \n');
fprintf(LogFile,'\n Number of Iterations = %d \n Optimum point:\n %20.16f \n', iter_num, x);
xp = x;
done = 1;
else
decr_counter = 0;
end
end

%% Exit the program
disp(' '); disp(sprintf('Number of iterations = %d',iter_num));
if (iter_num > max_iter)
disp('Error: Maximum iterations reached. Optimum point could not be found!');
fprintf(LogFile, 'Error: Maximum iterations reached. Optimum point could not be found! \n');
end
end

%%
function fx = f_x(x,p)
    fx = sum( (abs(x).^p) );
end

%%
function [gradF] = grad_f(x,p, eps)
    N= length(x);
gradF = zeros(N, 1);
for i = 1:N
    x(i) = sign(x(i))*sqrt(x(i)^2 + eps^2); % to avoid divide by zero if x(i) == 0
    if (x(i) == 0)
        x(i) = sqrt(x(i)^2 + eps^2);
    end
    gradF(i) = x(i) * (abs(x(i))^(p-2)) ;
    if isnan(gradF(i))
        gradF = NaN;
        return
    end
end
end
%% Initial setup
clc
close all;
% put key subdirectories in path if not already there
path(path,'./Optimization');
path(path,'./Data');
load RandomStates
rand('state',rand_state);
% log directory
dirname = '.\CGProj_Log\';
if ( exist(dirname,'dir') == 0)
    mkdir (dirname);
end
filename = strcat('Results_', datestr(now,'yyyy-mm-dd_HH-MM-SS') , '.txt');
ResultFile = fopen( strcat(dirname,filename) , 'at');

%% Globals
N = 512; % signal length
K = 20;  % number of spikes in the signal
M = 120; % number of observations to make

%% Original signal and measurements
format long;
% random +/- 1 signal
x = zeros(N,1);
q = randperm(N);
x(q(1:K)) = sign(randn(K,1));
% measurement matrix
A = randn(M,N);
A = orth(A');
% observations
\[ b = A \cdot x; \]

%% initial guess = min energy
\[ x_{0_1} = (A^\cdot A') \cdot b; \]
\[ x_0 = A' \cdot x_{0_1}; \]

%% Uncomment for diagnostics
% Uncomment the following lines (and set your own values) to create a case
% for which you can visualize the algorithm progress
A = [2.1 1.5];
b = 3;
x_{0_1} = (A^\cdot A') \cdot b;
x_0 = A' \cdot x_{0_1};
% x_0 = [-5/7 3];
x = x_0;

%% Initial graphs
disp(''); disp(' ----------- Lp Norm optimization -----------'); disp(' ');
results = [x x_0];
header = sprintf('%20s' ,'Original signal', 'Min Energy');
%graph the results
h1 = figure('Name','Signal Plot Window');
hold all; plotbrowser('on'); grid on;
stem(x,'LineWidth',2,'DisplayName','Original Signal');
stem(x_0,'DisplayName','Min Energy');

%% Solve the Lp norm problem
% p is an array of Norm_p values to run optimization for
p = [0.3 0.5 0.6 0.8 0.95];
% x_cgp holds optimization results using Conjugate Gradients with Projection method
x_cgp = zeros(length(x), length(p));
for i = 1 : length(p)
    disp(''); disp(strcat('Norm_',num2str(p(i)))); disp('^^^^^^^^^^^^^^^^^^^^^^^^^');
    fprintf(ResultFile, 'n Norm_%1.2f \n', p(i));

    % Solve the norm_p problem
    x_cgp(:,i) = CG_Projection_diag(A, b, x0, p(i));

    if (~isnan(x_cgp(i)))
        % logging the results
        figure(h1);
        results = [results , x_cgp(:,i)];
        disp('-------------|---------------|------------------------|---------------|');
        fprintf(ResultFile, '-------------|---------------|------------------------|---------------|\n');
        disp(sprintf(strcat( 'norm(x-p_',num2str(p(i)),') = %20.15f' ), norm(x-x_cgp(:,i)) ));
        fprintf(ResultFile, 'norm(x-p_%1.2f) = %20.15f \n' ,p(i) ,norm(x-x_cgp(:,i)) );
        stem(x_cgp(:,i),'DisplayName',strcat('Norm_',num2str(p(i))), 'Marker', '+' );
        header = strcat(header, sprintf('20s',strcat('p_',num2str(p(i)))) );
    end
end

%% Log the signal values
disp(''); disp('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~'); disp('');
disp(results);
fprintf(ResultFile, strcat(header, '\n', '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~', '\n'));
for i = 1: length(results(:,1))
    fprintf(ResultFile, '%20.15f', results(i,:));
    fprintf(ResultFile, '\n');
end

% graphfile = strcat('Graph_', datestr(now,'yyyy-mm-dd_HH-MM-SS'));
saveas(gcf, strcat(dirname,graphfile,'.fig'),'fig');

saveas(gcf, strcat(dirname,graphfile,'.png'),'png');

disp('Finished ...... !');
Appendix E: MATLAB Script - Diagnostic Code, Conjugate Gradient with Projection Based on Search Direction and Lagrange Multipliers

function xp = CG_Projection_diag(A, b, x0, p)

%%% Globals
%clc;
%record iteration results in a log file
dirname = '\CGProj_Log';
filename = strcat('CGProjection_', datestr(now,'yyyy-mm-dd_HH-MM-SS'), '.txt');
LogFile = fopen( strcat(dirname,filename), 'at');
format long;
%due to rounding errors, we compare to epsilon instead of zero
eps_0 = 1e-10;
eps_grad = 1e-4; %to calculate gradient
eps_s0 = 1e-14; %for step vector
eps_L0 = 1e-14; %for Lagrange multipliers
iter_num = 1; max_iter = 4e4;
%stop if objective function does not significantly change successively
tol = 1e-10;
decr_counter = 0; max_decr = 3;
max_bktrac = 50;
[m,n] = size(A);
xp = NaN; %initialize returned value to an invalid number
f_values = []; x_values = [];

%%% Calculate the initial point
%add a vector to x0 such that the initial guess lies on the Tangent Subspace
% that is: A*(x0 + delta_x) - b = 0
x1 = linsolve(A, b - A*x0);
x01 = x0 + x1;
%check if x01 is closer to the Tangent Subspace
if (max(A*x0 -b) < max(A*x01 -b))
x = x0;
else
x = x01;
end

%% graph surfaces only for two dimensional case
if (n == 2)
    hc1 = figure('Name','Optimization graphs');
    hold all;
    axis on; grid on; plotbrowser('on'); title(strcat('Norm\_', num2str(p)) );
    xlabel('X-axis'); ylabel('Y-axis'); zlabel('Z-axis');
    %function surface
    [xx, yy] = meshgrid(-3:0.05:3);
    w = (abs(xx).^p) + (abs(yy).^p) ;
    mesh(xx,yy, w, 'EdgeColor', 'Red','DisplayName','Objective function');
    %constraint plane
    [X, Z] = meshgrid(-3:0.05:3 , min(min(w)):1: max(max(w)));
    Y= (-A(1,1)*X + b(1))./A(1,2);
    mesh(X,Y,Z, 'EdgeColor', 'Magenta','DisplayName','Constraint plane');
    %intersection line
    x1 = (-3:0.01:3);
    x2= (b(1)-A(1,1)*x1)./A(1,2);
    z = (abs(x1).^p) + (abs(x2).^p) ;
    plot3(x1,x2, z, 'Color', 'Green', 'LineWidth', 1.5,'DisplayName','Intersection');
    contour(xx, yy, w);
end

%% Iterations
fprintf(LogFile, '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
');
%f x = x0;
fx = f_x(x,p);
done = 0;
while ( (iter_num <= max_iter) && (done == 0) )
    fprintf(LogFile, 'Iteration: %d \n', iter_num);
    f_values = [f_values fx];
    x_values = [x_values x];

%% Tangent Subspace Matrix (N matrix) calculation
    gradF = grad_f(x,p, eps_grad);
    if isnan(gradF)
        disp('Error: unable to calculate the gradient of f!');
        fprintf(LogFile, 'Error: unable to calculate the gradient of f at a point! \n');
        fprintf(LogFile, '%20.16f', x);
        xp = NaN;
        break;
    end

    g = A*x - b;
    M = length(g);
    gradG = A;
    for i=1:M
        %if ( 0 <= g(i) && g(i) <= eps_0 ) %if constraint is active (= 0)
        if ( abs(g(i)) <= eps_0 ) %if constraint is active (= 0)
            temp1 = horzcat(N_Matrix, gradG(i,:)') ;
            N_Matrix = temp1;
            temp2 = vertcat(ga_indx, i) ;
            ga_indx = temp2;
        end
    end

    g = A*x - b; % active constraints (all constraints are active because A*x -b =0 )
    gradG = A; % gradient of constraints
    N_Matrix = A'; % columns of N= gradient of constraints
    M = length(g);
ga_indx = (1:M) ; % all constraints are active (Ax -b =0)

%%% Search Direction

s = zeros(1,n); %initialize search direction to zero
while ( max(abs(s)) <= eps_s0 ) %s is (practically) zero

%%% Projection Matrix calculation

if isempty(N_Matrix)
    Lambda = [] ; P = eye(n);
else
    P = eye(n) - N_Matrix* ((N_Matrix'*N_Matrix)
        \N_Matrix');
    Lambda = ( (N_Matrix'*N_Matrix)\N_Matrix') * gradF ;
end

%%% Step Direction

s = -P * gradF ;
d = norm(s); s = s/d; %normalize s
if ( max(abs(s)) <= eps_s0 ) %s is (almost) zero
    [min_Lambda indx_min] = min(Lambda);
    if ( (min_Lambda >= -eps_L0) || (length(Lambda) == 1) )
        %all Lagrange multipliers are non-negative or no more
        %multiplier left to remove
        disp('Optimum point calculated!');
        xp = x;
        fprintf(LogFile, 'Optimum point calculated! \n');
        fprintf(LogFile, ' \n %f', x);
        done = 1;
        if (n == 2)
            disp(x);
            graphfile = strcat('Gaph1_', datestr(now,'yyyy-mm-dd_HH-MM-SS'), '.txt');
            saveas(gcf, strcat(dirname,graphfile,'.fig'),'fig');
        end
    else
        % Optimum point is not reached, continue search
    end
end
% remove the active constraint corresponding to the most negative Lagrange multiplier

if (indx_min == 1)  % first column
    new_N_Matrix = N_Matrix(:,2:end);
    new_ga_indx = ga_indx(2:end);
elseif (indx_min == length(Lambda))  % last column
    new_N_Matrix = N_Matrix(:,1:end-1);
    new_ga_indx = ga_indx(1:end-1);
else
    new_N_Matrix = N_Matrix(:,[1:(indx_min-1), (indx_min+1):end] ) ;
    new_ga_indx = ga_indx([1:(indx_min-1) (indx_min+1):end]) ;
end
N_Matrix = new_N_Matrix;
ga_indx = new_ga_indx;
end
end
end

%% Step size

% gama = 0.05;  % desired decrease in objective function
%
calculate the upper limit on step size (alpha)

step_upper = 1e100;  % practically infinite
for i=1:length(g)
    if (g(i) > eps_0)
        k = A(i,:)* s ;
        if ( k < -eps_0 )
            alpha_i = -g(i) / k ;
            step_upper = min (step_upper, alpha_i);
        end
    end
end

% one dimensional minimization for step size (alpha) using desired decrease in objective function
\[ \alpha_{\text{star}} = - \frac{\gamma f(x)}{s^\top \nabla F} ; \]

if (\(\alpha_{\text{star}} < -\varepsilon_0\))

\[ \text{disp('Warning, } \alpha_{\text{star}} \text{ is negative!');} \]

end

\[ \alpha = \min(\text{step_upper}, \alpha_{\text{star}}) ; \]

%%% update the iteration

\[ x_{\text{old}} = x ; \]

\[ f_{x_{\text{old}}} = f(x) ; \]

\[ x = x_{\text{old}} + (\alpha \times s) ; \]

\[ f_x = f(x,p) ; \]

\[ \text{cntr} = 1 ; \]

\% perform backtracking if objective function has increased

while ((\(f_x - f_{x_{\text{old}}} > \text{tol}\)) 
& (\(\text{cntr} \leq \text{max_bktrac}\)) )

\[ gama = gama / 2 ; \% reduce the desired increase \]

\[ \alpha_{\text{star}} = - \frac{\gamma f(x)}{s^\top \nabla F} ; \]

\[ \alpha = \min(\text{step_upper}, \alpha_{\text{star}}) ; \]

\[ x = x_{\text{old}} + (\alpha \times s) ; \]

\[ f_x = f(x,p) ; \]

\[ \text{cntr} = \text{cntr} + 1 ; \]

end

\[ \text{iter_num} = \text{iter_num} + 1 ; \]

\[ \text{fprintf(LogFile,'fx_{\text{old}}: \%20.16f , fx_{\text{new}}: \%20.16f , delta_f: \%20.16f \n'} , \text{fx_{\text{old}}}, \text{fx} , \text{fx-fx_{\text{old}}}) ; \]

\[ \text{fprintf(LogFile,'----------------------------------- \n'}) ; \]

%\% graph the progress only for two dimensional case

%\% only for the first 20 iterations and then every 200 iterations, up to iteration 2000

if (\(n == 2\))

\[ \text{if ( ((mod(iter_num ,200) == 0 ) 
& (iter_num < 2000)) || (iter_num < 20)) )}

\[ \text{figure(hc1)} ; \]

\[ \text{if (iter_num < 20)} \]

\[ X0=[x_{\text{old}}(1), \ x(1)]; \]
Y0=[x_old(2) , x(2)];
Z0=[fx_old , fx];
plot3(X0,Y0,Z0, 'LineWidth', 1.5,'DisplayName','Function change');
plot3(X0,Y0,[0,0], 'LineWidth', 1.5,'DisplayName','Point change');
end
end
%% check objective function and iteration point change
if ( norm(x - x_old) < tol && (fx - fx_old < tol) && (fx < fx_old) )
decr_counter = decr_counter + 1;
if (decr_counter == max_decr)
disp('Iteration stopped to decrease significantly. Optimum point is found!');
fprintf(LogFile,'Iteration stopped to decrease significantly. Optimum point is found! 
');
fprintf(LogFile,'\n Number of Iterations = %d 
 Optimum point:
 %20.16f 
', iter_num, x);
    xp = x;
done = 1;
else
decr_counter = 0;
end
end
%% Exit the program
disp(' '); disp(sprintf('Number of iterations = %d',iter_num));
if (n == 2)
    graphfile = strcat('Graph1_', datestr(now,'yyyy-mm-dd_HH-MM-SS') );
saveas(gcf, strcat(dirname,graphfile,'.fig'),'fig');
end
if (iter_num > max_iter)
disp('Error: Maximum iterations reached. Optimum point could not be found!');
fprintf(LogFile, 'Error: Maximum iterations reached. Optimum point could not be found! 
');
end
function fx = f_x(x,p)
    fx = sum( (abs(x).^p) );
    %fx = sum( (x.^p) );
end

function [gradF] = grad_f(x,p, eps)
    N= length(x);
    %eps_0 = 1e-5;
    gradF = zeros(N,1);
    for i = 1:N
        x(i) = sign(x(i))*sqrt(x(i)^2 + eps^2); %to avoid divide by zero if x(i)=0
        if (x(i) == 0)
            x(i) = sqrt(x(i)^2 + eps^2);
        end
        gradF(i) = p*x(i)* (abs(x(i))^(p-2)) ;
        if isnan(gradF(i))
            gradF = NaN;
            return
        end
    end
end