4M17: Practical Optimization
Assignment 1

Student Candidate Number: 5490C

Dec 11, 2020

Abstract

In this report, the use of norm approximation in convex optimization problems are explored. Starting with the simplest form norm approximation using l_1, l_2 and l_∞ norms, their algorithmically-applicable forms were derived, such as linear programming and least squares method and were applied using MATLAB to 5 different overdetermined data pairs. The performance of the different norms were compared by comparing computation times and by histogramming the residuals, and it is found that there is a trade-off in using each norm for approximation. Further, the central path formulation using a logarithmic barrier function was explored. This unconstrained minimization problem was solved using gradient descent, and its performance was evaluated via a simple convergence analysis. Lastly, a sparse signal reconstruction was explored in a l_1-regularised Least squares problem. This optimization problem was solved using a Newton interior-point method and resulted in a very successful signal recovery. This was compared to a minimum-energy reconstruction result without l_1-heuristics, which showed very poor signal recovery.

1. Norm Approximation

a) The l_1, l_2 and l_∞ norm, and l_2 norm approximation using least squares

A norm of an object is a non-negative, real-valued number of 'how big' something is and allows ordering. A vector norm of a vector maps vector values to values in $[0,\infty)$. A function $\boldsymbol{f} : \mathbb{R}^m \to \mathbb{R}$ is a norm on $\mathbb{R}^m$ if the following 3 conditions are satisfied:

1. $\boldsymbol{f}$ is convex.
2. $\boldsymbol{f}$ is positively homogeneous, meaning that $\boldsymbol{f(\alpha x)= \alpha f(x)}$ for every $\boldsymbol{x}\in \mathbb{R}^m$ and $\alpha\in\mathbb{R}_+$
3. $\boldsymbol{f}$ is positive-definite: for every A$\boldsymbol{x}$-$\boldsymbol{b}\in \mathbb{R}^m$, $\boldsymbol{f(Ax\text{-}b)=0}$ implies $\boldsymbol{Ax\text{-}b\ =0.}$

A particular family of norms are called l_p norms.

The general l_p norm on $\mathbb{R}^m$ in the problem is defined as:

$$\|A\boldsymbol{x}-\boldsymbol{b}\|_p = \left(\sum_{i=1}^{m} |\boldsymbol{a}_i{}^T\boldsymbol{x}-b_i|^p\right)^{1/p}$$

where $\boldsymbol{a}_i$ is the vector formed by the entries of the $\boldsymbol{i}$th row of matrix A, that is $\boldsymbol{a}_i = (a_{i1}, a_{i2}, \dots, a_{im})$. Therefore the l_1, l_2 and l_∞ norms are defined as follows:

The l_1 norm,

$$\|A\boldsymbol{x}-\boldsymbol{b}\|_1 = |\boldsymbol{a}_1{}^T\boldsymbol{x}-b_1| + |\boldsymbol{a}_2{}^T\boldsymbol{x}-b_2| + \cdots + |\boldsymbol{a}_m{}^T\boldsymbol{x}-b_m| = \sum_{i=1}^{m} |\boldsymbol{a}_i{}^T\boldsymbol{x}-b_i|$$

The l_2 norm,

$$\|A\boldsymbol{x}-\boldsymbol{b}\|_2 = (|\boldsymbol{a}_1{}^T\boldsymbol{x}-b_1|^2 + |\boldsymbol{a}_2{}^T\boldsymbol{x}-b_2|^2 + \cdots + |\boldsymbol{a}_m{}^T\boldsymbol{x}-b_m|^2)^{\frac{1}{2}} = \left(\sum_{i=1}^{m} (\boldsymbol{a}_i{}^T\boldsymbol{x}-b_i)^2\right)^{\frac{1}{2}}$$

And the l_∞ norm,

$$\|A\boldsymbol{x}-\boldsymbol{b}\|_\infty = \max_i |\boldsymbol{a}_i{}^T\boldsymbol{x}-b_i|$$

When attempting to find the solution of the linear system of equations $A\boldsymbol{x} = \boldsymbol{b},$ where $A \in \mathbb{R}^{m\times n}$, $\boldsymbol{b} \in \mathbb{R}^m$ and $\boldsymbol{x} \in \mathbb{R}^n$, if there are more equations than unknowns, $(m > n)$, the system is overdetermined and typically, there is no exact solution. This can be overcome by finding an approximate solution that can be computed by minimizing the residual, hence the norm, $||A\boldsymbol{x} - \boldsymbol{b}||$. These norm-approximation problems are to minimize these norms with respect to $\boldsymbol{x}$ and find the approximate solution. Choosing different norms will result in different solutions. The l_1 norm approximation problem is:

$$\min_x \sum_{i=1}^{m} |\boldsymbol{a}_i{}^T\boldsymbol{x} - b_i|$$

The l_2 norm approximation problem:

$$\min_x \left(\sum_{i=1}^{m} (\boldsymbol{a}_i{}^T\boldsymbol{x} - b_i)^2\right)^{\frac{1}{2}}$$

And l_∞ norm approximation problem:

$$\min_x \max_i |\boldsymbol{a}_i{}^T\boldsymbol{x} - b_i|$$

The l_2 norm is the conceptually simplest form, otherwise known as Euclidean distance. The l_2 norm can be expressed as an optimization problem with a convex quadratic function with an analytic solution, which amounts to solving a linear system of equations as follows:
Minimising the l_2-norm is equivalent to minimizing the square of the l_2-norm. This is the sum of squares of the residuals which is a convex quadratic function.

$$f(\boldsymbol{x}) = ||A\boldsymbol{x} - \boldsymbol{b}||_2{}^2 = \sum_{i=1}^{m} (\boldsymbol{a}_i{}^T\boldsymbol{x} - b_i)^2$$

The criterion for optimal point for $f(\boldsymbol{x})$, as it is convex, is that at the minimum optimal point; $\nabla f(\boldsymbol{x}) = (0,0, \dots, 0)$ by the first order condition for optimality, where

$$\nabla f(\boldsymbol{x}) = (\partial_{x_1} f, \partial_{x_2} f, \dots, \partial_{x_m} f)$$

In matrix notation,

$$\nabla(||A\boldsymbol{x} - \boldsymbol{b}||_2{}^2) = \nabla((A\boldsymbol{x} - \boldsymbol{b})^T(A\boldsymbol{x} - \boldsymbol{b})) = \nabla(\boldsymbol{x}^T A^T A\boldsymbol{x} - 2\boldsymbol{x}^T A^T\boldsymbol{b} + \boldsymbol{b}^T\boldsymbol{b}) = 2A^T A\boldsymbol{x} - 2A^T\boldsymbol{b}$$

Setting this to zero, it can be seen that minima will occur at the optimal solution:

$$\boldsymbol{x}^* = (A^T A)^{-1} A^T\boldsymbol{b}$$

Therefore, this is a convex quadratic function which will have an analytic solution of $\boldsymbol{x}^*$ as above, and this amounts to approximately solving an overdetermined linear system of ‘m’ equations with ‘n’ unknowns using Least Squares.

b) LP formulation of l_∞ and l_1 norm approximation

The norm-approximation problems corresponding to the l_∞ norm on $\mathbb{R}^m$ is:

$$\min_x ||A\boldsymbol{x} - \boldsymbol{b}||_\infty = \min_x \max_i |\boldsymbol{a}_i{}^T\boldsymbol{x} - b_i|$$

For some z_i we observe that:

$$\max_i |z_i| = \max\{-z_i, z_i\} = \min\{t | z_i \le t, -z_i \le t\} = \min\{\mathrm{t} | -t \le z_i \le t\}$$

In the above equation, the second equality holds because taking the largest number among $-z_i, z_i$ is the same as minimizing a value, t, such that both $-z_i$ and z_i are smaller than $\boldsymbol{t}$.
Therefore, the l_∞ norm minimization problem can be transformed into:

$$\min_{x,t}\ t$$

With constraints: $|\boldsymbol{a}_i{}^T\boldsymbol{x} - b_i| \leq t$

Or equivalently,

$$-t \leq \boldsymbol{a}_i{}^T\boldsymbol{x} - b_i \leq t \qquad i = 1, \ldots, m$$

Here, t is a new scalar variable.

This can be solved via the linear program:

$$\min_{x} \tilde{\boldsymbol{c}}^T\tilde{\boldsymbol{x}}$$
$$\tilde{A}\tilde{\boldsymbol{x}} \leq \tilde{\boldsymbol{b}}$$

With

$$\tilde{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{x} \\ t \end{bmatrix}, \qquad \tilde{\boldsymbol{c}} = \begin{bmatrix} \boldsymbol{0} \\ 1 \end{bmatrix}, \qquad \tilde{A} = \begin{bmatrix} A & -\boldsymbol{1} \\ -A & -\boldsymbol{1} \end{bmatrix}, \qquad \tilde{\boldsymbol{b}} = \begin{bmatrix} \boldsymbol{b} \\ -\boldsymbol{b} \end{bmatrix}.$$

Where $\tilde{\boldsymbol{x}} \in \mathbb{R}^{n+1}$, $\tilde{\boldsymbol{c}} \in \mathbb{R}^{n+1}$, $\tilde{A} \in \mathbb{R}^{2m\times(n+1)}$, $\tilde{\boldsymbol{b}} \in \mathbb{R}^{2m}$.

The l_1 norm approximation problem can similarly be reduced to a linear program.

The norm-approximation problems corresponding to the l_1 norm on $\mathbb{R}^m$ is:

$$\min_{x} \|A\boldsymbol{x} - \boldsymbol{b}\|_1 = \min_{x} \sum_{i=1}^{m} |\boldsymbol{a}_i{}^T\boldsymbol{x} - b_i|$$

For some z_i, we observe that:

$$|z_i| = \max\{-z_i, z_i\} = \min\{t_i | z_i \leq t_i, -z_i \leq t_i\} = \min\{t_i | -t_i \leq z_i \leq t_i\}$$

Therefore, the l_1 norm minimization problem can be transformed into:

$$\min_{x,t_i} \sum_{i=1}^{m} t_i$$

with constraints:

$$-t_i \leq \boldsymbol{a}_i{}^T\boldsymbol{x} - b_i \leq t_i \qquad i = 1, \ldots, m$$

Here, $\boldsymbol{t}$ is a column vector of length m.

This can be solved via the linear program:

$$\min_{x} \tilde{\boldsymbol{c}}^T\tilde{\boldsymbol{x}}$$
$$\tilde{A}\tilde{\boldsymbol{x}} \leq \tilde{\boldsymbol{b}}$$

with

$$\tilde{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{t} \end{bmatrix}, \qquad \tilde{\boldsymbol{c}} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{1} \end{bmatrix}, \qquad \tilde{A} = \begin{bmatrix} A & -I \\ -A & -I \end{bmatrix}, \qquad \tilde{\boldsymbol{b}} = \begin{bmatrix} \boldsymbol{b} \\ -\boldsymbol{b} \end{bmatrix}.$$

Where $\tilde{\boldsymbol{x}} \in \mathbb{R}^{n+m}$, $\tilde{\boldsymbol{c}} \in \mathbb{R}^{n+m}$, $\tilde{A} \in \mathbb{R}^{2m\times(n+m)}$, $\tilde{\boldsymbol{b}} \in \mathbb{R}^{2m}$.

c) Solving LP and LS problems

For the l_1 and l_∞ norms, the ***linprog*** function in MATLAB was used to solve the problem using a linear program. After solving for $\tilde{\boldsymbol{x}}$ using the linear program, the desired optimal solution $\boldsymbol{x}$ was extracted by eliminating the added variable $\boldsymbol{t}$, then the values of the minimized norms were computed for each data pair.

For the l_2 norm, the solution $\boldsymbol{x}$ was computed directly using the closed-form expression derived using Least squares.

	$\min_x \|Ax - b\|_1$	$\min_x \|A\boldsymbol{x} - \boldsymbol{b}\|_2$	$\min_x \|A\boldsymbol{x} - \boldsymbol{b}\|_\infty$	l_1 runtime (s)	l_2 runtime (s)	l_∞ runtime (s)
(A1,b1)	9.0648	2.3268	0.5852	0.019	0.0007	0.020
(A2,b2)	42.4356	5.5259	0.7072	0.072	0.0012	0.043
(A3,b3)	141.4745	9.1825	0.5925	2.556	0.0075	1.339
(A4,b4)	308.7098	13.8558	0.6216	27.965	0.0394	12.177
(A5,b5)	575.5089	18.7329	0.6017	346.828	0.2341	135.106

Table 1. Table of minimized norms and runtimes for l_1, l_∞ and l_2

The runtimes were averaged with 3 trials each, for l_1, l_∞ norm approximations, and 10 trials for l_2 norm approximation to improve accuracy and eliminate anomalies in results. The minimized norm values are as expected. For the l_1-norm, the value increase as n increases, as the magnitudes of all residuals are being summed. The l_1-norm values seem to increase proportional to n. For the l_2 norm, the Euclidean distance is calculated, so the value increases at a rate that is half an order of magnitude smaller than the rate of increase of the l_1 norm. for example, taking the norm values of the first two datasets,

$$\frac{42.4356}{9.0648} \approx 2 * \frac{5.5259}{2.3268}$$

The l_∞ norm simply computes the maximum, so is similar for all datasets, and is the smallest among the norms. It can be seen that the computation times for the l_2 norm approximation increases proportionally with increasing data size. This is expected as it is simply performing a simple matrix multiplication. For the linear programming used in l_1, l_∞-norm computation, the ***linprog*** function was used, which uses the simplex algorithm. As it can be seen from the l_1, l_∞ runtimes in Table 1, the computation time increases exponentially with increasing data dimension, n. This is expected as the simplex algorithm is an iterative computational procedure: the minimum solution is found by rearranging columns and rows until the objective function optimum is reached. The computation for l_1 norm is more substantial than the l_∞-norm, as the dimensions are augmented in computation by vector $\boldsymbol{t}$ of length m, as opposed to the added scalar t. Performing least squares via computing a closed-form solution is the computationally most efficient, cheapest and fastest method, as can be seen from table 1. Computing the l_1, l_∞ norms via linear programming are costlier than computing the least squares solution, but are not much more costly than using least squares and has expanded the use of optimization techniques beyond using the simplest Least Squares method in the past few decades.

d) Histogram of residuals

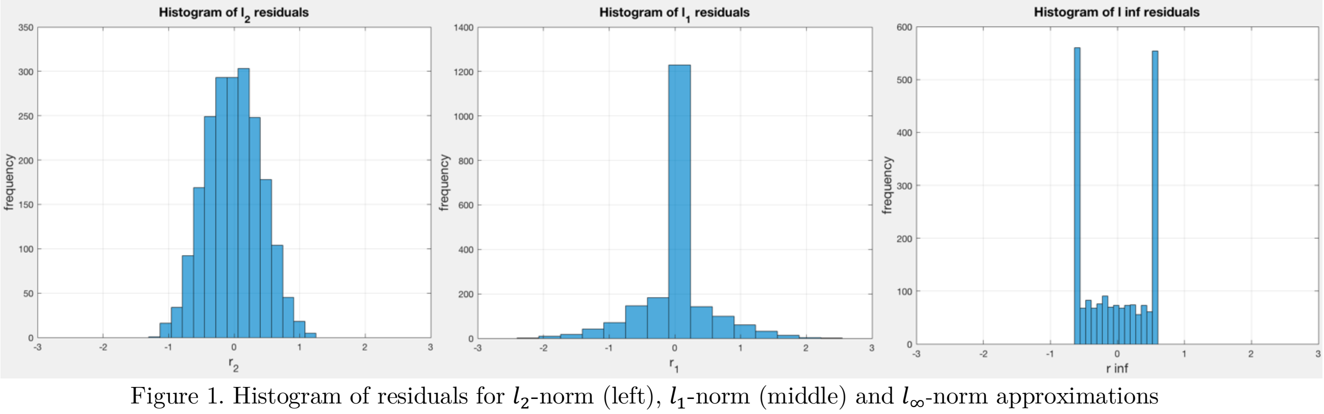


Figure 1. Histogram of residuals for l_2-norm (left), l_1-norm (middle) and l_∞-norm approximations

As it can be seen from the figures, the choice of norm clearly affects the statistical distribution of the residuals. In the l_2 norm LS minimization, the distribution of residuals approaches a gaussian distribution as the number of data points approaches infinity. For the l_1-norm, a high frequency of residuals around zero is observed, indicating an exact fit of high number of data points. This advantage of the exact fit is comprimised by the drawback of having other outlying residuals with larger magnitude than the residuals found in l_2 or l_∞-norms. In Figure 2, outlying residuals of magnitude of up to 2.5 are observed at the tails of the distribution. With the l_∞ norm, outliers of high magnitude are compressed, as can be from Figure 3, the largest residuals lie at around magnitude of 0.7. however, there is a large number of residuals at the tails, and there is a small number of residuals that have an exact fit. There is a trade-off between having a large number of outlying data, and having a few large-magnitude outliers in choosing to use the l_1 or the l_∞ norm. this will depend on every problem and the required result.

2. Unconstrained minimization

a) LP formulation of the problem

We now consider the convex optimization problem:

$$\min_x f_0(x)$$
$$subject\ to \quad f_i(x) \le 0, \qquad i = 1, \dots, m$$

Where f_0, f_i are covex and twice differentiable functions for $i = 1, \dots, m$.

The central path is the solution of the unconstrained minimization with the objective function:

$$\min_x t f_0(x) + \phi(x)$$

For any $t \le 0$, where $\phi(x)$ is the logarithmic barrier function of the feasibility set $S = \{x : f_i(x) \le 0\}$.

The explicit form of this objective function corresponding to the LP formulation of the l_1 norm approximation is derived as follows. Comparing the forms of the general convex optimization problem and that of LP formulation, in the case of LP formulation of the l_1 norm, $f_0(x) = \tilde{\boldsymbol{c}}^{\boldsymbol{T}}\tilde{\boldsymbol{x}}$ and $f_i(x) = \tilde{A}\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{b}}$. The central path of this convex optimization problem with convex inequality constraint can be solved by applying equation (4). The logarithmic barrier function as defined by the question is:

$$\phi(x) = -\sum \log(-f_i(x))$$

In the case of the LP formulation of the l_1 norm,

$$\phi(\tilde{\boldsymbol{x}}) = -\sum_{i=1}^{m} \log(-f_i(\tilde{\boldsymbol{x}})) = -\sum_{i=1}^{m} \log(\tilde{b}_i - \tilde{\boldsymbol{a}}_i{}^T\tilde{\boldsymbol{x}})$$

Therefore, the explicit form of this objective function corresponding to the LP formulation of the l_1 norm approximation is

$$\min_x t\tilde{\boldsymbol{c}}^{\boldsymbol{T}}\tilde{\boldsymbol{x}} - \sum_{i=1}^{m} \log(\tilde{b}_i - \tilde{\boldsymbol{a}}_i{}^T\tilde{\boldsymbol{x}})$$

where $\tilde{\boldsymbol{x}}$, $\tilde{\boldsymbol{c}}$, $\tilde{\boldsymbol{A}}$, and $\tilde{\boldsymbol{b}}$ are as specified in section 1 b).

When preforming a descent method to search for the optimum, one needs to search within a sets that is bounded: all the sublevel sets must be closed. This can be satisfied by using this logarithmic barrier function, where the domain of the function will be exactly the polyhedron where all the inequality constraints will be satisfied. $\phi(\tilde{\boldsymbol{x}})$ will only be finite when we are inside the feasibility set. In the problem, we are essentially augmenting the cost with a penalty that punishes when it tries to escape the polyhedron; hence why this kind of problem is otherwise known as the interior point method. As the number of iteration increases, there is less and less weight put on the barrier function until only $\tilde{\boldsymbol{c}}^{\boldsymbol{T}}\tilde{\boldsymbol{x}}$ is minimized.

b) Gradient descent

Now, a gradient descent algorithm with backtracking line search is applied to solve this problem with $t = 1$ to the pair of data (A3,b3). Hence, we perform an unconstrained minimization with gradient descent algorithm to solve for optimal $\tilde{\boldsymbol{x}}$, for

$$\min_x \psi(\boldsymbol{x})$$

where

$$\psi(\boldsymbol{x}) = \tilde{\boldsymbol{c}}^{\boldsymbol{T}}\tilde{\boldsymbol{x}} - \sum_{i=1}^{m} \log(\tilde{b}_i - \tilde{\boldsymbol{a}}_i{}^T\tilde{\boldsymbol{x}})$$

The gradient can be computed as

$$\nabla\psi(\boldsymbol{x}) = \tilde{\boldsymbol{c}} - \sum_{i=1}^{m} \frac{\tilde{\boldsymbol{a}}_i}{\tilde{b}_i - \tilde{\boldsymbol{a}}_i^{T} \tilde{\boldsymbol{x}}}$$

In the gradient descent algorithm, we start from a point $\tilde{\boldsymbol{x}}$**,** choose a search direction, $\Delta\tilde{\boldsymbol{x}}$, and we search for a point along the line $\tilde{\boldsymbol{x}}+t_{step}\Delta\tilde{\boldsymbol{x}}$ with a lower cost at each iteration by choosing a step size t_{step}, and stop until the stopping criterion is satisfied. $\Delta\tilde{\boldsymbol{x}}$ is defined as the negative gradient, and the step size is found using backtracking line search, using an appropriate choice of α and β. $\alpha = 0.2$ and $\beta = 0.4$ were used in my algorithm. The algorithm stops when the distance to the optimum is smaller than a threshold value, ϵ.

The gradient descent algorithm used is given in Appendix A.4

c) Convergence analysis

Convergence analysis of the gradient descent algorithm used above is performed by plotting a semilogarithmic plot of the minimization error against the number of iterations. The minimization error is the difference between $\psi(\boldsymbol{x}_k)$ at an iteration, k, and $\psi(\boldsymbol{x}^*)$, where $\boldsymbol{x}^*$ is the optimum.

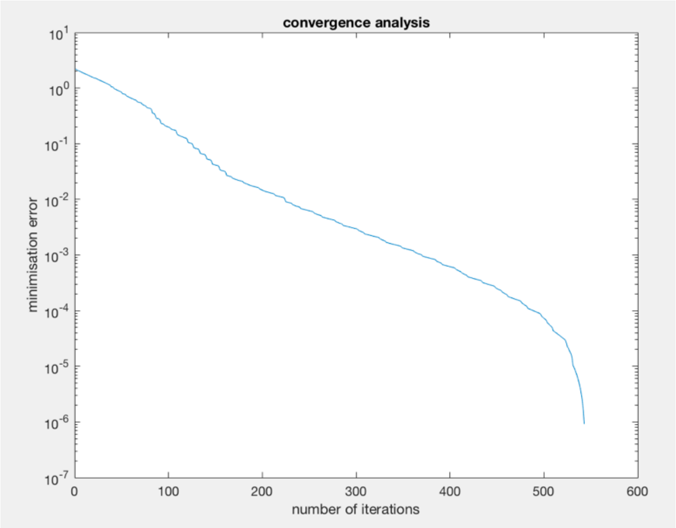


Figure 2. Plot of number of iterations against the minimization error

Our objective function $\psi(\boldsymbol{x})$ is strongly convex, so we expect an exponential decay with increasing number of iterations. This can be seen in Figure 4; as the number of iterations increases, we are approach the optimum at the vertex. This can be seen in Figure 4, in which we observe an exponential decay of the iterates (a linear convergence) until around 400 iterations and $\epsilon = 10^{-4}$. The convergence is good in theory, but the convergence is slow, as can be seen from the figure, it converges only after around 550 iterations for $\epsilon = 10^{-6}$. The rather slow convergence is because of the simple descent algorithm used that is cheap and easy to compute (as one only needs to compute the gradient), but is usually slow to converge. Moreover, as strictly convex functions, the convergence rate depends on the Hessian of the objective function, perhaps, our objective function is not convex enough to give a very fast convergence.

3. l_1-Regularized Least Squares

a) Central path formulation

Sparse modeling has become a very prominent finding in the field of machine learning. When there is an underdetermined solution in a linear problem, we try to find the solution with maximal possible components of the solution that are exactly zero. Because this sparsity is incredibly cheap to store, there is an interest in obtaining and optimizing these sparse solutions in large scale problems. This amounts to optimizing the 'cardinality'. However, as cardinality is a discrete property, and discrete constraints are to be avoided at any rate, hence, an l_1-regularization term is added to the Least squares problem to replace this cardinality constraint. We will try to solve the following l_1-regularized least squares problem to reconstruct a sparse signal given a noisy signal. the noisy observations are of the form, $\boldsymbol{b} = A\boldsymbol{x} + \boldsymbol{v},$ where $\boldsymbol{v}$ is the random noise with mean of zero. The cost function for the problem is generally:

$$\min_{x} \|A\boldsymbol{x} - \boldsymbol{b}\|_2{}^2 + \lambda\|\boldsymbol{x}\|_1$$

Where $\lambda > 0$ is a regularization parameter.

This can be transformed into a convex quadratic problem with linear inequality constraints. Following the convention that $A \in \mathbb{R}^{m\times n}$, $\boldsymbol{b} \in \mathbb{R}^m$ and $\boldsymbol{x} \in \mathbb{R}^n$, the problem is equivalent to:

$$\min_{x} \|A\boldsymbol{x} - \boldsymbol{b}\|_2{}^2 + \lambda\sum_{i=1}^{n}|x_i|$$

We observe that

$$\max\{-x_i, x_i\} = \min\{u_i | x_i \le u_i, -x_i \le u_i\}$$

Which is equivalent to

$$\min_{x,u}\left(\|A\boldsymbol{x} - \boldsymbol{b}\|_2{}^2 + \lambda\sum_{i=1}^{n}u_i\right),$$

subject to $|x_i| \le u_i$, or equivalently,

$$-u_i \le x_i \le u_i, \text{ for } i = 1, \dots n.$$

In order to define the logarithmic barrier function, $\Phi(\boldsymbol{x}, \boldsymbol{u})$ more easily, we put the two inequality constraints in the form:

$$\begin{aligned} x_i - u_i \le 0 \qquad & \text{for } i = 1, \dots n \\ -x_i - u_i \le 0 \qquad & \text{for } i = 1, \dots n \end{aligned}$$

where the added variable $\boldsymbol{u} \in \mathbb{R}^n$.

To formulate the central path of this problem, a suitable logarithmic barrier function. This function characterizes the two inequality constraints, so calling the two inequality constraints $f1_i(x, u) = x_i - u_i$ and $f2_i(x, u) = -x_i - u_i$,

$$\begin{aligned} \Phi(\boldsymbol{x}, \boldsymbol{u}) &= -\left(\sum \log\left(-f1_i(\boldsymbol{x}, \boldsymbol{u})\right) + \sum \log\left(-f2_i(\boldsymbol{x}, \boldsymbol{u})\right)\right) \\ &= -\left(\sum_{i=1}^{n}\log(-x_i + u_i) + \sum_{i=1}^{n}\log(x_i + u_i)\right) \\ &= -\sum_{i=1}^{n}\log(u_i + x_i) - \sum_{i=1}^{n}\log(u_i - x_i) \end{aligned}$$

Now, by the central path formulation from Question 2, our problem transforms to solving for a unique minimizer $(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t))$ by solving

$$\begin{aligned} &\min_{x,u} t\left(\|A\boldsymbol{x} - \boldsymbol{b}\|_2{}^2 + \lambda\sum_{i=1}^{n}u_i\right) + \Phi(\boldsymbol{x}, \boldsymbol{u}) \\ &= \min_{x,u} t\|A\boldsymbol{x} - \boldsymbol{b}\|_2{}^2 + \lambda t\sum_{i=1}^{n}u_i + \Phi(\boldsymbol{x}, \boldsymbol{u}) \end{aligned}$$

where parameter t varies from 0 to ∞.

We can call this convex function ϕ_t:

$$\phi_t(\boldsymbol{x}, \boldsymbol{u}) = t\|A\boldsymbol{x} - \boldsymbol{b}\|_2{}^2 + \lambda t\sum_{i=1}^{n}u_i + \Phi(\boldsymbol{x}, \boldsymbol{u})$$

b) Derivation of gradient and Hessian

Denote the gradient as $\boldsymbol{g}$,

$$\boldsymbol{g} = \begin{bmatrix} \nabla_{\boldsymbol{x}} \phi_t(\boldsymbol{x}, \boldsymbol{u}) \\ \nabla_{\boldsymbol{u}} \phi_t(\boldsymbol{x}, \boldsymbol{u}) \end{bmatrix} \equiv \begin{bmatrix} \boldsymbol{g_1} \\ \boldsymbol{g_2} \end{bmatrix}$$

$$\boldsymbol{g_1} = \nabla_{\boldsymbol{x}} \phi_t(\boldsymbol{x}, \boldsymbol{u}) = t(2A^T A\boldsymbol{x} - 2A^T \boldsymbol{b}) + \nabla_{\boldsymbol{x}} \Phi(\boldsymbol{x}, \boldsymbol{u})$$

As

$$\frac{\delta(\Phi(\boldsymbol{x}, \boldsymbol{u}))}{\delta x_i} = \frac{-1}{u_i + x_i} + \frac{1}{u_i - x_i}$$

Hence,

$$\nabla_{\boldsymbol{x}} \Phi(\boldsymbol{x}, \boldsymbol{u}) = \begin{bmatrix} \frac{-1}{u_1 + x_1} + \frac{1}{u_1 - x_1} \\ \vdots \\ \frac{-1}{u_n + x_n} + \frac{1}{u_n - x_n} \end{bmatrix} = \begin{bmatrix} \frac{2x_1}{{u_1}^2 - {x_1}^2} \\ \vdots \\ \frac{2x_n}{{u_n}^2 - {x_n}^2} \end{bmatrix}$$

Therefore,

$$\boldsymbol{g_1} = 2tA^T(A\boldsymbol{x} - \boldsymbol{b}) + \begin{bmatrix} \frac{2x_1}{{u_1}^2 - {x_1}^2} \\ \vdots \\ \frac{2x_n}{{u_n}^2 - {x_n}^2} \end{bmatrix}$$

And,

$$\boldsymbol{g_2} = \nabla_{\boldsymbol{u}} \phi_t(\boldsymbol{x}, \boldsymbol{u}) = t\lambda \boldsymbol{1} + \nabla_{\boldsymbol{u}} \Phi(\boldsymbol{x}, \boldsymbol{u})$$

where

$$\nabla_{\boldsymbol{u}} \Phi(\boldsymbol{x}, \boldsymbol{u}) = \begin{bmatrix} \frac{-1}{u_1 + x_1} + \frac{-1}{u_1 - x_1} \\ \vdots \\ \frac{-1}{u_n + x_n} + \frac{-1}{u_n - x_n} \end{bmatrix} = \begin{bmatrix} \frac{-2u_1}{{u_1}^2 - {x_1}^2} \\ \vdots \\ \frac{-2u_n}{{u_n}^2 - {x_n}^2} \end{bmatrix}$$

Therefore,

$$\boldsymbol{g_2} = t\lambda \boldsymbol{1} - \begin{bmatrix} \frac{2u_1}{{u_1}^2 - {x_1}^2} \\ \vdots \\ \frac{2u_n}{{u_n}^2 - {x_n}^2} \end{bmatrix}$$

We have computed the gradient.

The Hessian is defined as

$$H = \nabla^2 \phi_t(\boldsymbol{x}, \boldsymbol{u}) = \begin{bmatrix} \frac{\delta^2 \phi_t}{\delta \boldsymbol{x^2}} & \frac{\delta^2 \phi_t}{\delta \boldsymbol{x} \delta \boldsymbol{u}} \\ \frac{\delta^2 \phi_t}{\delta \boldsymbol{u} \delta \boldsymbol{x}} & \frac{\delta^2 \phi_t}{\delta \boldsymbol{u^2}} \end{bmatrix}$$

where

$$\frac{\delta^2 \phi_t}{\delta \boldsymbol{x^2}} = \frac{\delta(\boldsymbol{g_1})}{\delta x} = 2tA^T A + \ diag\left(\frac{2({u_1}^2 + {x_1}^2)}{({u_1}^2 - {x_1}^2)^2}, \dots, \frac{2({u_n}^2 + {x_n}^2)}{({u_n}^2 - {x_n}^2)^2}, \right)$$

and

$$\frac{\delta^2 \phi_t}{\delta \boldsymbol{x} \delta \boldsymbol{u}} = \frac{\delta^2 \phi_t}{\delta \boldsymbol{u} \delta \boldsymbol{x}} = diag\left(\frac{-4u_1 x_1}{({u_1}^2 - {x_1}^2)^2}, \dots, \frac{-4u_n x_n}{({u_n}^2 - {x_n}^2)^2}, \right)$$

and

$$\frac{\delta^2 \phi_t}{\delta \boldsymbol{u^2}} = diag\left(\frac{2({u_1}^2 + {x_1}^2)}{({u_1}^2 - {x_1}^2)^2}, \dots, \frac{2({u_n}^2 + {x_n}^2)}{({u_n}^2 - {x_n}^2)^2}, \right)$$

Where $diag(A_1, \dots, A_p)$ means a diagonal matrix with diagonal blocks $A_1, \dots, A_p$.

c) Sparse Signal recovery using Newton interior-point method

A sparse signal recovery problem is considered. Signal $\boldsymbol{x}_0$ consists of 10 spikes of amplitude ± 1, which is the signal we aim to reconstruct: $\boldsymbol{card}\ (\boldsymbol{x}) = 10$. $\boldsymbol{x}_0 \in \mathbb{R}^{256}$ and measurement matrix $A \in \mathbb{R}^{60\times 256}$ are given. The vector of observations, $\boldsymbol{b}$ is computed simply by $\boldsymbol{b} = A\boldsymbol{x}_0$. The regularization parameter is computed as $\lambda = 0.01\|2A^T\boldsymbol{b}\|_\infty$. This penalty constraint turs out to be quite small, $\lambda = 0.0061$. Choosing an initial $\boldsymbol{x}$ and $\boldsymbol{u}$ which satisfies $u_i \le x_i \le u_i$ for $i = 1, \ldots n$, we aim to recover the exact signal $\boldsymbol{x}_0$ from the noisy measurement, by solving the convex optimization problem specified in part a) using a newton interior-point method. In this algorithm, we use the newton method for the centering step: the search direction is computed as: $H\begin{bmatrix}\Delta\boldsymbol{x}\\ \Delta\boldsymbol{u}\end{bmatrix} = -\boldsymbol{g}$, and using Cholesky factorization, we compute the decrement also. The solution is updated at each step as $\boldsymbol{x}+\ t_{step}\Delta\boldsymbol{x}$ and a backtracking line search is used again to compute t_{step} at each iterate. The decrement calculated is used as a stopping criterion.

Once we have centered and updated the solution using Newton's method, we increase $\boldsymbol{t}$ variable in the objective function $\boldsymbol{\phi}_t$, by a scaling parameter μ at each step until a stopping criterion, $\frac{m}{t} < \varepsilon$, that ensures that the minimization error is smaller than ε.

For the Newton method, the cost for every step of computation of the algorithm is $\approx On^3$, whereas it was $\approx On$ for the gradient descent method; it is more costly as computing the Hessian is costly, but is much faster to converge as it gives a quadratic convergence.

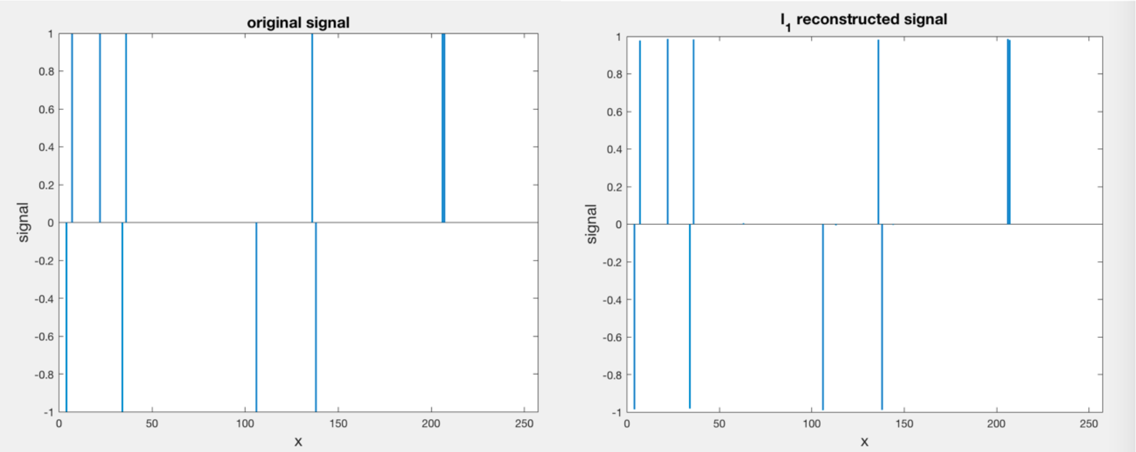


Figure 3. Comparison plot of the original sparse signal $\boldsymbol{x}_0$ (left) compared to the reconstructed signal using l_1-regularization (right)

It can be seen from Figure 5 that using l_1 regularization, the original signal can be reconstructed almost perfectly using convex optimization by newton interior-point method. It can be seen that there is negligible noise of the recovered signal around the origin, and the signal spikes of ± 1 are recovered at the exact spots, although some spikes have magnitude slightly < 1. In theory, this result be improved by polishing; by fixing the sparsity pattern of the signal obtained, $\boldsymbol{x}$, and repeating re-solving the convex optimization problem with this new sparsity pattern to obtain a better solution.

d) Minimum energy reconstruction

Generally, another form of regularizing the least squares solution is to regularize using the l_2-norm, which is:

$$\min_x \|A\boldsymbol{x} - \boldsymbol{b}\|_2{}^2 + \lambda\|\boldsymbol{x}\|_2$$

This can be solved by using least squares:

$$\boldsymbol{x}^* = (A^T A + \lambda I)^{-1}A^T\boldsymbol{b}$$

In minimum energy reconstruction, we are searching for the limit point that has the minimum l_2-norm among all points that satisfy $A^T(A\boldsymbol{x}-\boldsymbol{b}) = \boldsymbol{0}$, that is closest to the origin. Therefore, as $\lambda \rightarrow 0$, the optimal point $\boldsymbol{x}^*$converges to the solution without the added l_2-norm-term:

$$\boldsymbol{x}^* = (A^T A)^{-1} A^T \boldsymbol{b}$$

which can be solved using Least Squares.

This is equivalent to just solving for the least squares solution without any penalty added to the cost.

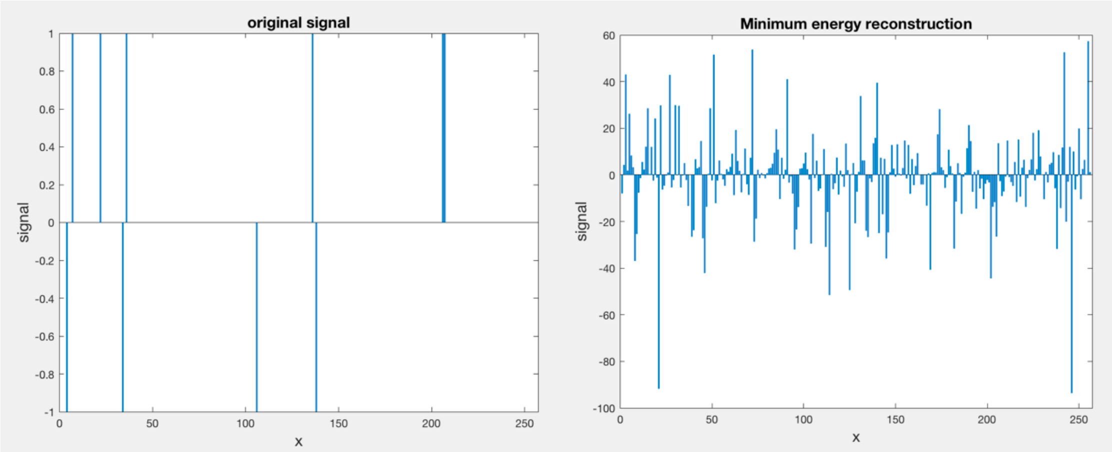


Figure 4. Comparison plot of the original sparse signal $\boldsymbol{x}_0$ (left) compared to the reconstructed signal using least squares (right)

From Figure 6, it can be observed that the least squares solution completely fails at reconstructing the original signal. As opposed to the l_1-regularized reconstruction problem, where we were given A priori information about sparsity in the form of the l_1-regularization, here we do not have the penalty function, which leads to failure of reconstruction. This result seems to be in contradiction to Shannon's theorem. Conventionally, it is found that one must sample at twice the maximum frequency to sample a signal without loss of information. This unusual result that is in contradiction to Shannon's theory, as one can reconstruct the exact signal from a very little measurements. This result has led to a new field in information engineering called 'compressive sampling', which challenges such fundamental theorems.

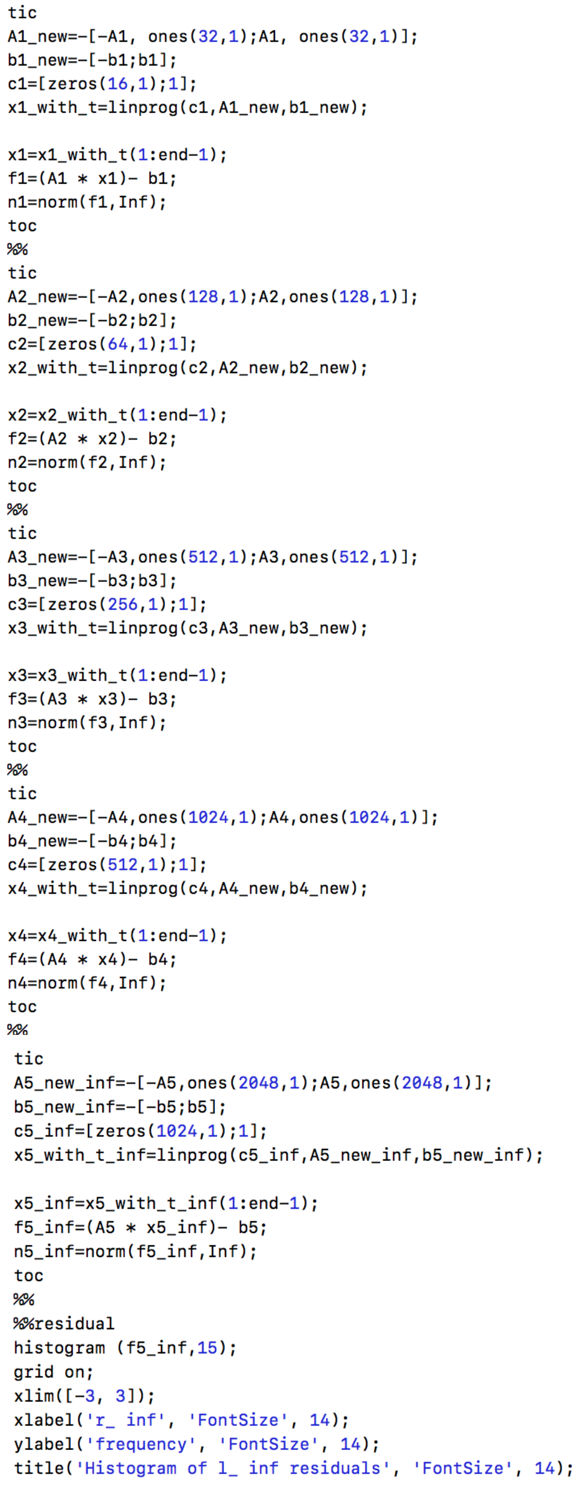
4. Conclusion

In this report, we have explored the usage of norms in convex optimization problems. First, the simple l_1, l_2 and l_∞ norm- approximation was explored and computed to find solutions for overdetermined systems, and we found that the usage of l_1 and l_∞ norms had a trade-off between having many exact fits but large magnitude outliers, and small number of exact fits but having may outliers. It was seen that although the l_2-norm Least squares solution was easiest to compute, the linear programming of l_1 and l_∞ norms were efficient enough to give scope for large usage in many problems in engineering. Further, in the unconstrained convex optimization problem formed by central path formulation, we explored the concept of augmenting the cost function with a penalty log barrier function. from the convergence analysis, it was found that the gradient descent method gives a linear convergence in finding the optimal solution. Lastly, a sparse signal reconstruction problem was explored that is a very prominent field of research especially in Machine learning, and it was found that using the l_1-norm regularization was very effective in recovering the sparse signal. However, the recovery without any penalty failed to reconstruct the signal, which is an astounding finding as it seems to go against conventional information principles.

5. Appedix A: Matlab code

A.1

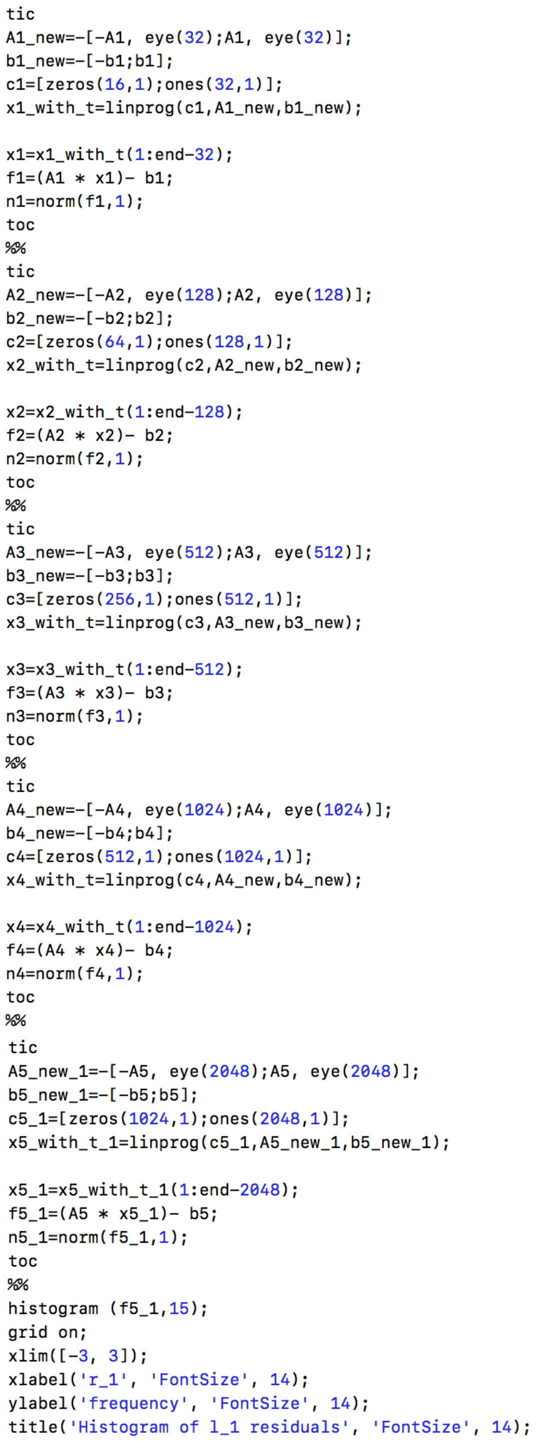
Question 1 : l_∞ norm approximation



```
tic
A1_new=-[-A1, ones(32,1);A1, ones(32,1)];
b1_new=-[-b1;b1];
c1=[zeros(16,1);1];
x1_with_t=linprog(c1,A1_new,b1_new);

x1=x1_with_t(1:end-1);
f1=(A1 * x1)- b1;
n1=norm(f1,Inf);
toc
%%
tic
A2_new=-[-A2,ones(128,1);A2,ones(128,1)];
b2_new=-[-b2;b2];
c2=[zeros(64,1);1];
x2_with_t=linprog(c2,A2_new,b2_new);

x2=x2_with_t(1:end-1);
f2=(A2 * x2)- b2;
n2=norm(f2,Inf);
toc
%%
tic
A3_new=-[-A3,ones(512,1);A3,ones(512,1)];
b3_new=-[-b3;b3];
c3=[zeros(256,1);1];
x3_with_t=linprog(c3,A3_new,b3_new);

x3=x3_with_t(1:end-1);
f3=(A3 * x3)- b3;
n3=norm(f3,Inf);
toc
%%
tic
A4_new=-[-A4,ones(1024,1);A4,ones(1024,1)];
b4_new=-[-b4;b4];
c4=[zeros(512,1);1];
x4_with_t=linprog(c4,A4_new,b4_new);

x4=x4_with_t(1:end-1);
f4=(A4 * x4)- b4;
n4=norm(f4,Inf);
toc
%%
tic
A5_new_inf=-[-A5,ones(2048,1);A5,ones(2048,1)];
b5_new_inf=-[-b5;b5];
c5_inf=[zeros(1024,1);1];
x5_with_t_inf=linprog(c5_inf,A5_new_inf,b5_new_inf);

x5_inf=x5_with_t_inf(1:end-1);
f5_inf=(A5 * x5_inf)- b5;
n5_inf=norm(f5_inf,Inf);
toc
%%
%%residual
histogram (f5_inf,15);
grid on;
xlim([-3, 3]);
xlabel('r_ inf', 'FontSize', 14);
ylabel('frequency', 'FontSize', 14);
title('Histogram of l_ inf residuals', 'FontSize', 14);
```

Appendix A.1. l_∞-norm approximation Matlab code

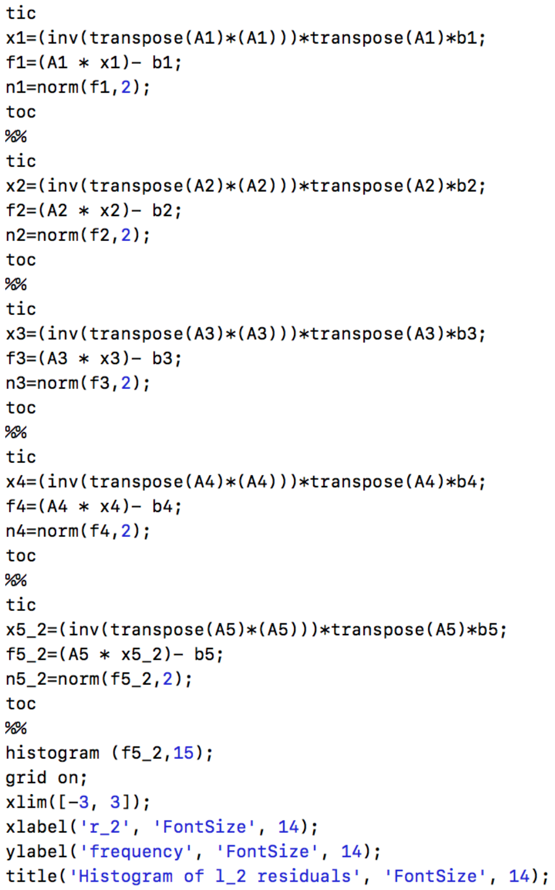
A.2
Question 1: l_1 norm approximation



```
tic
A1_new=-[-A1, eye(32);A1, eye(32)];
b1_new=-[-b1;b1];
c1=[zeros(16,1);ones(32,1)];
x1_with_t=linprog(c1,A1_new,b1_new);

x1=x1_with_t(1:end-32);
f1=(A1 * x1)- b1;
n1=norm(f1,1);
toc
%%
tic
A2_new=-[-A2, eye(128);A2, eye(128)];
b2_new=-[-b2;b2];
c2=[zeros(64,1);ones(128,1)];
x2_with_t=linprog(c2,A2_new,b2_new);

x2=x2_with_t(1:end-128);
f2=(A2 * x2)- b2;
n2=norm(f2,1);
toc
%%
tic
A3_new=-[-A3, eye(512);A3, eye(512)];
b3_new=-[-b3;b3];
c3=[zeros(256,1);ones(512,1)];
x3_with_t=linprog(c3,A3_new,b3_new);

x3=x3_with_t(1:end-512);
f3=(A3 * x3)- b3;
n3=norm(f3,1);
toc
%%
tic
A4_new=-[-A4, eye(1024);A4, eye(1024)];
b4_new=-[-b4;b4];
c4=[zeros(512,1);ones(1024,1)];
x4_with_t=linprog(c4,A4_new,b4_new);

x4=x4_with_t(1:end-1024);
f4=(A4 * x4)- b4;
n4=norm(f4,1);
toc
%%
tic
A5_new_1=-[-A5, eye(2048);A5, eye(2048)];
b5_new_1=-[-b5;b5];
c5_1=[zeros(1024,1);ones(2048,1)];
x5_with_t_1=linprog(c5_1,A5_new_1,b5_new_1);

x5_1=x5_with_t_1(1:end-2048);
f5_1=(A5 * x5_1)- b5;
n5_1=norm(f5_1,1);
toc
%%
histogram (f5_1,15);
grid on;
xlim([-3, 3]);
xlabel('r_1', 'FontSize', 14);
ylabel('frequency', 'FontSize', 14);
title('Histogram of l_1 residuals', 'FontSize', 14);
```

Appendix A.2. l_1-norm approximation Matlab code

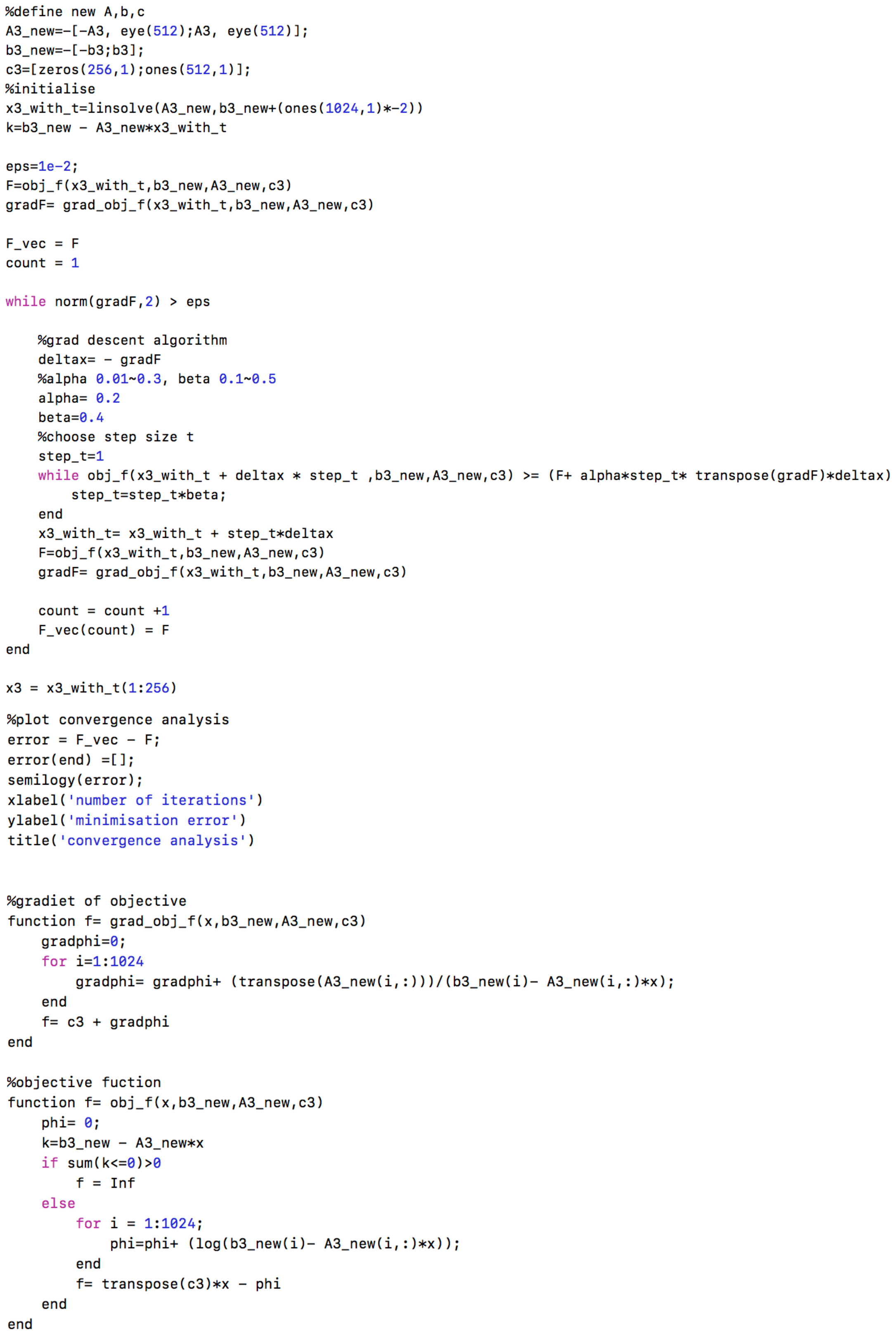
A.3
Question 1: l_2 norm approximation



```
tic
x1=(inv(transpose(A1)*(A1)))*transpose(A1)*b1;
f1=(A1 * x1)- b1;
n1=norm(f1,2);
toc
%%
tic
x2=(inv(transpose(A2)*(A2)))*transpose(A2)*b2;
f2=(A2 * x2)- b2;
n2=norm(f2,2);
toc
%%
tic
x3=(inv(transpose(A3)*(A3)))*transpose(A3)*b3;
f3=(A3 * x3)- b3;
n3=norm(f3,2);
toc
%%
tic
x4=(inv(transpose(A4)*(A4)))*transpose(A4)*b4;
f4=(A4 * x4)- b4;
n4=norm(f4,2);
toc
%%
tic
x5_2=(inv(transpose(A5)*(A5)))*transpose(A5)*b5;
f5_2=(A5 * x5_2)- b5;
n5_2=norm(f5_2,2);
toc
%%
histogram (f5_2,15);
grid on;
xlim([-3, 3]);
xlabel('r_2', 'FontSize', 14);
ylabel('frequency', 'FontSize', 14);
title('Histogram of l_2 residuals', 'FontSize', 14);
```

Appendix A.3. l_2-norm approximation Matlab code

A.4
Question 2

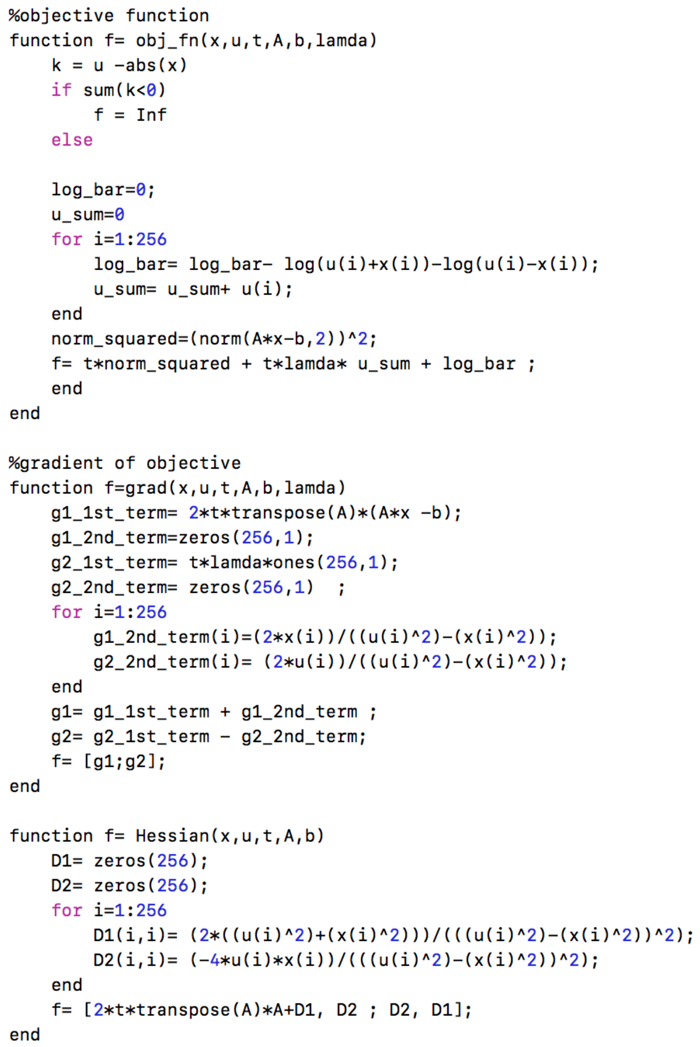


```
%define new A,b,c
A3_new=-[-A3, eye(512);A3, eye(512)];
b3_new=-[-b3;b3];
c3=[zeros(256,1);ones(512,1)];
%initialise
x3_with_t=linsolve(A3_new,b3_new+(ones(1024,1)*-2))
k=b3_new - A3_new*x3_with_t

eps=1e-2;
F=obj_f(x3_with_t,b3_new,A3_new,c3)
gradF= grad_obj_f(x3_with_t,b3_new,A3_new,c3)

F_vec = F
count = 1

while norm(gradF,2) > eps

    %grad descent algorithm
    deltax= - gradF
    %alpha 0.01~0.3, beta 0.1~0.5
    alpha= 0.2
    beta=0.4
    %choose step size t
    step_t=1
    while obj_f(x3_with_t + deltax * step_t ,b3_new,A3_new,c3) >= (F+ alpha*step_t* transpose(gradF)*deltax)
        step_t=step_t*beta;
    end
    x3_with_t= x3_with_t + step_t*deltax
    F=obj_f(x3_with_t,b3_new,A3_new,c3)
    gradF= grad_obj_f(x3_with_t,b3_new,A3_new,c3)

    count = count +1
    F_vec(count) = F
end

x3 = x3_with_t(1:256)

%plot convergence analysis
error = F_vec - F;
error(end) =[];
semilogy(error);
xlabel('number of iterations')
ylabel('minimisation error')
title('convergence analysis')


%gradiet of objective
function f= grad_obj_f(x,b3_new,A3_new,c3)
    gradphi=0;
    for i=1:1024
        gradphi= gradphi+ (transpose(A3_new(i,:)))/(b3_new(i)- A3_new(i,:)*x);
    end
    f= c3 + gradphi
end

%objective fuction
function f= obj_f(x,b3_new,A3_new,c3)
    phi= 0;
    k=b3_new - A3_new*x
    if sum(k<=0)>0
        f = Inf
    else
        for i = 1:1024;
            phi=phi+ (log(b3_new(i)- A3_new(i,:)*x));
        end
        f= transpose(c3)*x - phi
    end
end
```

Appendix A.4. Question 2 Matlab code

A.5
Question 3

```
%define observation and lamda
x0=x
b= A*x
lamda= 0.01* norm (2*transpose(A)*b, inf)
%need to initialize u &t
u=(ones(256,1))*2;
v=[x;u]

t=1
mu = 15

while 256/t >= 0.00001
    eps=1e-4
    %define obj fn,Hessian, grad, L
    F= obj_fn(x,u,t,A,b,lamda)
    H=Hessian(x,u,t,A,b)
    g=grad(x,u,t,A,b,lamda)
    L=chol(Hessian(x,u,t,A,b))
    L = transpose(L)
    dec=(norm(inv(L)*g,2))^2

    while dec/2>eps

        %newton-interior
        deltav=(-inv(transpose(L)))*inv(L)*g
        deltax=deltav(1:256)
        deltau=deltav(257:512)

        %alpha 0.01~0.3, beta 0.1~0.5
        alpha= 0.3
        beta=0.3
        %choose step size t
        step_t=1
        while obj_fn(x+deltax * step_t ,u+deltau*step_t,t,A,b,lamda) >= (F+ alpha*step_t* transpose(g)*deltav)
            step_t=step_t*beta;
        end
        x= x+ step_t*deltax
        u=u+step_t*deltau
        F= obj_fn(x,u,t,A,b,lamda)
        g=grad(x,u,t,A,b,lamda)
        H=Hessian(x,u,t,A,b)
        L=chol(Hessian(x,u,t,A,b))
        L = transpose(L)
        dec=(norm(inv(L)*g,2))^2
    end
t = t* mu

end

x
%%
%original
figure (1)
bar(x0)
xlabel('x', 'FontSize', 14);
ylabel('signal', 'FontSize', 14);
title('original signal', 'FontSize', 14);
%obtained
figure (2)
bar(x)
xlabel('x', 'FontSize', 14);
ylabel('signal', 'FontSize', 14);
title('l_1 reconstructed signal', 'FontSize', 14);

x_new = inv(transpose(A)*A)*transpose(A)*b
figure (3)
bar(x_new)
xlabel('x', 'FontSize', 14);
ylabel('signal', 'FontSize', 14);
title('Minimum energy reconstruction', 'FontSize', 14);
%%
```



```
%objective function
function f= obj_fn(x,u,t,A,b,lamda)
    k = u -abs(x)
    if sum(k<0)
        f = Inf
    else

    log_bar=0;
    u_sum=0
    for i=1:256
        log_bar= log_bar- log(u(i)+x(i))-log(u(i)-x(i));
        u_sum= u_sum+ u(i);
    end
    norm_squared=(norm(A*x-b,2))^2;
    f= t*norm_squared + t*lamda* u_sum + log_bar ;
    end
end

%gradient of objective
function f=grad(x,u,t,A,b,lamda)
    g1_1st_term= 2*t*transpose(A)*(A*x -b);
    g1_2nd_term=zeros(256,1);
    g2_1st_term= t*lamda*ones(256,1);
    g2_2nd_term= zeros(256,1)  ;
    for i=1:256
        g1_2nd_term(i)=(2*x(i))/((u(i)^2)-(x(i)^2));
        g2_2nd_term(i)= (2*u(i))/((u(i)^2)-(x(i)^2));
    end
    g1= g1_1st_term + g1_2nd_term ;
    g2= g2_1st_term - g2_2nd_term;
    f= [g1;g2];
end

function f= Hessian(x,u,t,A,b)
    D1= zeros(256);
    D2= zeros(256);
    for i=1:256
        D1(i,i)= (2*((u(i)^2)+(x(i)^2)))/(((u(i)^2)-(x(i)^2))^2);
        D2(i,i)= (-4*u(i)*x(i))/(((u(i)^2)-(x(i)^2))^2);
    end
    f= [2*t*transpose(A)*A+D1, D2 ; D2, D1];
end
```

Appendix A.5. Question 3 Matlab code