**CUED - Engineering Tripos Part IIB 2021-2022** **Module Coursework**

| Module | 4M17 | Title of report | Convex Optimisation |
|---|---|---|---|
| Date submitted: 30/11/2021 | | Assessment for this module is ☒ 100% / ☐ 25% coursework<br>of which this assignment forms 50 % | |

| UNDERGRADUATE STUDENTS ONLY | | POST GRADUATE STUDENTS ONLY | | | |
|---|---|---|---|---|---|
| Candidate number: | 5585G | Name: | | College: | |

| Feedback to the student<br>☐ **See also comments in the text** | | Very good | **Good** | Needs improvmt |
|---|---|---|---|---|
| CONTENT | **Completeness, quantity of content:**<br>Has the report covered all aspects of the lab? Has the analysis been carried out thoroughly? | | | |
| | **Correctness, quality of content**<br>Is the data correct? Is the analysis of the data correct? Are the conclusions correct? | | | |
| | **Depth of understanding, quality of discussion**<br>Does the report show a good technical understanding? Have all the relevant conclusions been drawn? | | | |
| | Comments: | | | |
| PRESENTATION | **Attention to detail, typesetting and typographical errors**<br>Is the report free of typographical errors? Are the figures/tables/references presented professionally? | | | |
| | Comments: | | | |

***Indicative grades are not provided for the FINAL piece of coursework in a module***

| **Assessment (circle one or two grades)** | | A* | A | B | C | D |
|---|---|---|---|---|---|---|
| Indicative grade guideline | | >75% | 65-75% | 55-65% | 40-55% | <40% |
| *Penalty for lateness:* | | *20% of maximum achievable marks per week or part week that the work is late.* | | | | |

Marker: Date:

# 1 Norm Approximation

We first consider unconstrained problems of the form

$$\min \quad ||Ax - b||$$

where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ are given in the problem and $x \in \mathbb{R}^n$ is the variable with respect to which the minimisation is considered.

## 1.1 Part (a)

Here, $|| \cdot ||_p$ represents the $l_p$ norm on $\mathbb{R}^m$. Of primary interest in this report are the $l_1$, $l_2$ and $l_\infty$ norms, defined for this problem as

$$\begin{aligned} l_1 \quad & ||Ax - b||_1 = \sum_{i=1}^{m} |\{Ax - b\}_i| \\ l_2 \quad & ||Ax - b||_2 = \sqrt{\sum_{i=1}^{m} (\{Ax - b\}_i)^2} = \sqrt{(Ax - b)^T (Ax - b)} \\ l_\infty \quad & ||Ax - b||_\infty = \max_i |\{Ax - b\}_i| \end{aligned} \tag{1}$$

Where $|\cdot|$ is the absolute value and $\{\cdot\}$ is used to represent indexing a vector. Note that $Ax - b$ is a vector in $\mathbb{R}^m$. The corresponding norm approximation problems are simply the minimisation of these norms with respect to the variable $x$.

For the $l_2$ norm we can expand the norm as

$$\begin{aligned} ||Ax - b||_2^2 &= (Ax - b)^T (Ax - b) \\ &= x^T A^T A x - 2b^T A x + b^T b \end{aligned}$$

Hence, since the norm is strictly non-negative by definition, the minimisation problem can have as its objective the squared norm instead of the norm itself. Further, since $b^T b$ does not depend on $x$, we can drop it entirely. Then, provided $A$ is not singular, we have

$$\min ||Ax - b||_2 = \min\{x^T A^T A x - 2b^T A x\}$$

which is an unconstrained optimisation problem with a convex quadratic cost (since $A^T A$ is positive definite for non-singular $A$). An analytical solution can be found by differentiation of the cost

$$\frac{d}{dx}\{x^T A^T A x - 2b^T A x\} = 2A^T A x - 2A^T b$$

Setting this to zero we get

$$(A^T A)x^* = A^T b$$

Which is easily recognisable as the least-squares formulation and amounts to solving a system of $n$ linear equations. Since the cost function is convex, this optimum is a both a minimum and a global optimum.

## 1.2 Part (b)

One of the key breakthroughs in convex optimisation was the formulation of the $l_1$ and $l_\infty$ norm approximation problems as Linear Programs (LPs). This allowed for a broader range of choices of norm beyond the $l_2$ which had previously not been possible due to the non-linear $l_1$ and $l_\infty$ objective functions.

For the $l_1$ norm, the key insight is that $|a_i^T x - b_i|$ is the smallest number $z_i$ that satisfies

$$b_i - a_i^T x \leqslant z_i, \quad \forall i \quad \text{and} \quad a_i^T x - b_i \leqslant z_i, \quad \forall i$$

This defines a new minimisation problem over a set of $m$ variables

$$\begin{aligned} \min \quad & \sum_{i=1}^{m} z_i \\ \text{s.t.} \quad & a_i^T x - b_i \leqslant z_i \\ & b_i - a_i^T x \leqslant z_i \end{aligned} \tag{2}$$

Which is a linear program with a $2m$ inequality constraints and can equivalently be written as

$$\begin{aligned} \min \quad & \tilde{c}^T \tilde{x} \\ \text{s.t.} \quad & \begin{bmatrix} A & -I \\ -A & -I \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} \preceq \begin{bmatrix} b \\ -b \end{bmatrix} \end{aligned} \tag{3}$$

Where the inequality constraints have been represented as $\tilde{A}\tilde{x} \preceq \tilde{b}$ with $\tilde{A} \in \mathbb{R}^{(2m)\times(n+m)}$, $\tilde{x} \in \mathbb{R}^{(n+m)}$ and $\tilde{b} \in \mathbb{R}^{2m}$. The cost function vector is[1] $\tilde{c} = \begin{bmatrix} \mathbb{0}_n & \mathbb{1}_m \end{bmatrix}^T \in \mathbb{R}^{(n+m)}$, $\tilde{x} = \begin{bmatrix} x & z \end{bmatrix}^T \in \mathbb{R}^{(n+m)}$ with $z \in \mathbb{R}^m$.

For the $l_\infty$ norm we reformulate the max expression using a scalar $t$ as

$$\max\{a_1^T x - b_1, \ a_2^T x - b_2, \ \ldots, \ a_m^T x - b_m\} = \min\{t \ : \ |a_i^T x - b_i| \leqslant t, \ \forall i\}$$

Using the same trick with the absolute value as for the $l_1$ norm we can write this more canonically

$$\begin{aligned} \min \quad & t \\ \text{s.t.} \quad & a_i^T x - b_i \leqslant t \\ & b_i - a_i^T x \leqslant t \end{aligned} \tag{4}$$

Or

$$\begin{aligned} \min \quad & \tilde{c}^T \tilde{x} \\ \text{s.t.} \quad & \begin{bmatrix} A & -1 \\ -A & -1 \end{bmatrix} \begin{bmatrix} x \\ t \end{bmatrix} \preceq \begin{bmatrix} b \\ -b \end{bmatrix} \end{aligned} \tag{5}$$

Again this is a linear program with $2m$ inequality constraints, but this time $\tilde{A} \in \mathbb{R}^{(2m)\times(n+1)}$, $\tilde{x} \in \mathbb{R}^{(n+1)}$ and $\tilde{b} \in \mathbb{R}^{2m}$. The new cost function vector is $\tilde{c} = \begin{bmatrix} \mathbb{0}_n & 1 \end{bmatrix}^T \in \mathbb{R}^{(n+1)}$ and $\tilde{x} = \begin{bmatrix} x & t \end{bmatrix}^T$ where $t \in \mathbb{R}$.

## 1.3 Part (c)

Here we apply these norm approximation methods to test data. The data consisted of 5 pairs of $A$ and $b$ with increasing $n$ and $m = 2n$.

[1] An alternative formulation is $\tilde{c} = \mathbb{1}_m$, with cost function $\tilde{c}^T z$, but the other method was easier to implement in code as we can lump together the $x$ and $z$ vectors

| n | $\min \|\|Ax-b\|\|_1$ | $\min \|\|Ax-b\|\|_2$ | $\min \|\|Ax-b\|\|_\infty$ | $l_1$ Runtime (s) | $l_2$ Runtime (s) | $l_\infty$ Runtime (s) |
|---|---|---|---|---|---|---|
| 16 | 7.8475 | 4.4091 | 0.5643 | 0.0092 | 0.0050 | 0.0030 |
| 64 | 40.9069 | 29.6898 | 0.7036 | 0.0722 | 0.0010 | 0.0131 |
| 256 | 144.8620 | 87.5455 | 0.6066 | 4.6794 | 0.0092 | 0.5077 |
| 512 | 308.5018 | 192.0742 | 0.6211 | 40.3364 | 0.0080 | 4.2612 |
| 1024 | 575.9003 | 350.2085 | 0.6014 | 372.6677 | 0.0299 | 38.4004 |

Table 1: Minimised norms and runtimes for $l_1$, $l_2$, $l_\infty$ norms

Table (1) is rich with quantitative comparisons between the different norms. Firstly, as expected the $l_1$ is the most expensive problem to solve because we are optimising over $3n$ variables. The $l_2$ norm was quick to compute for all $n$ as the method is not iterative; there is $\mathcal{O}(n^3)$ cost in calculating the Cholesky decomposition of $A^TA$, but we only require this calculation once. The cost of calculating the $l_\infty$ norm did grow with $n$, but involved only $n+1$ variables thus it was faster than the $l_1$ norm. Figure (1) shows how the cost of the $l_1$ and $l_\infty$ problems grow at the same rate. The $l_2$ problem was not included here as $n$ was too low for random fluctuations at compute time to not affect the results.

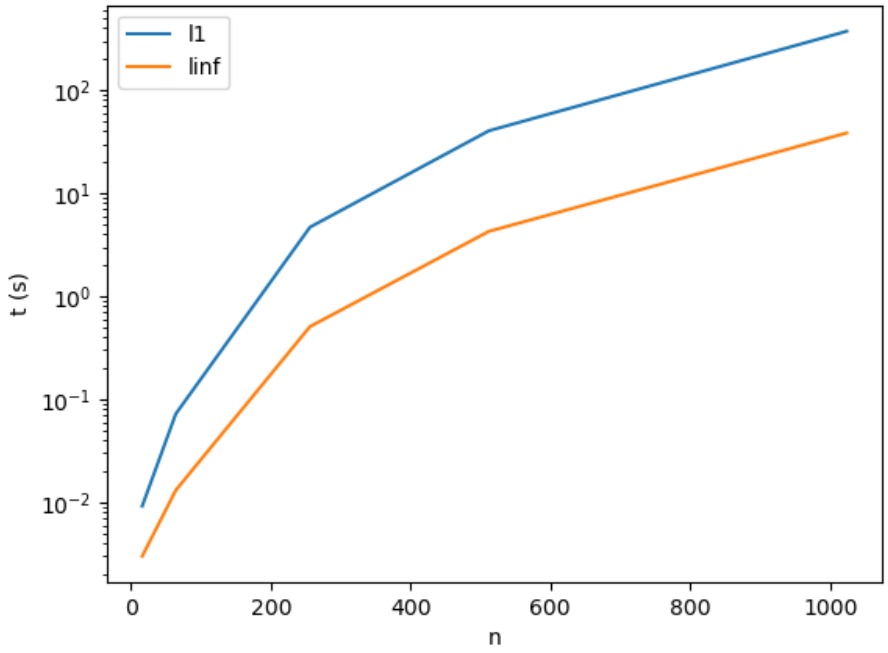
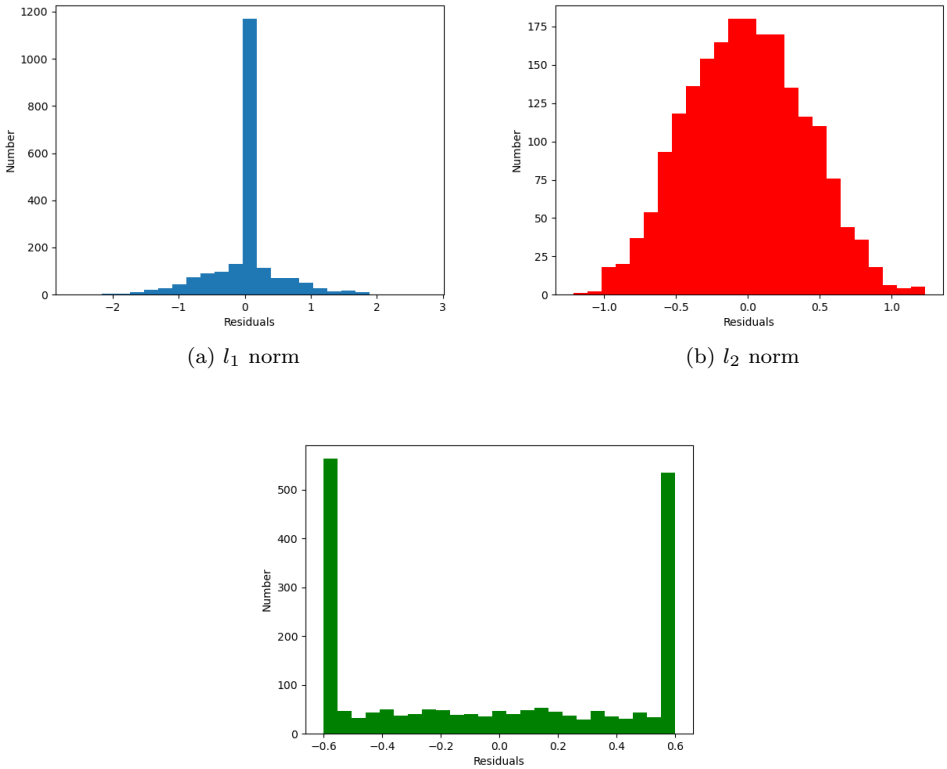



Figure 1: Semilog plots of complexity

The actual minimised norm values are not helpful when compared across norms as they are using different criterion, however we can draw comparisons within the same norm. We see that for the $l_1$ and $l_2$ norms the minimised value grows with $n$ as there are more data points added into the sum. For the $l_\infty$ norm, the minimised values were of the same order of magnitude as we're only interested in the single maximum value.

### 1.4 Part (d)




(a) $l_1$ norm

(b) $l_2$ norm

(c) $l_\infty$ norm

Figure 2: Histograms of residuals

Figure (2) demonstrates what each norm approximation problem is doing. Each LP can be seen as a trade-off between the size of the maximum outlier and the number of residuals that are close to zero. The $l_1$ norm problem tries to minimise the distance for as many datapoints as possible by considering all points and ignoring the most extreme outliers. Hence the residuals are concentrated heavily around zero, and there are a small number of points which the LP has ignored. The $l_2$ is similar, except it uses a squared distance function which weights extreme outliers more heavily - this means it takes a more balanced approach to the trade-off. The $l_\infty$ norm is only concerned with minimising the distance for the most extreme outlier, hence most residuals tend to bunch around this value as the LP is not concerned with how many residuals are close to zero.

## 2 Barrier Functions

In this section we consider constrained optimisation problems of the form

$$\begin{aligned} \min \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \leqslant 0, \quad i \in \{1, \ \ldots, \ p\} \end{aligned} \tag{6}$$

for convex and twice differentiable functions $f_0$ and $f_i$. We define the central path for $t \geqslant 0$ as the solution to

$$\min \quad tf_0(x) + \phi(x) \tag{7}$$

Where $\phi$ is a barrier function for the feasibility set $S = \{x \ : \ f_i(x) \leqslant 0\}$. One choice for $\phi$ is the logarithmic barrier function over the $p$ constraints

$$\phi(x) = -\sum_{i=1}^{p} \log(-f_i(x))$$

## 2.1 Part (a)

This central path formulation can be used to solve linear programs, for example the $l_1$ norm approximation problem from the first section. For this problem we had $\tilde{A} \in \mathbb{R}^{(2m)\times(n+m)}$, $\tilde{x} \in \mathbb{R}^{(n+m)}$, $\tilde{b} \in \mathbb{R}^{(2m)}$, $\tilde{c} \in \mathbb{R}^{(n+m)}$ and $\tilde{x} \in \mathbb{R}^{(n+m)}$. In this case we have $f_0(x) = \tilde{c}^T\tilde{x}$ and define the log barrier function

$$\phi(x) = \begin{cases} -\sum_{i=1}^{2m} \log(\tilde{b}_i - \tilde{a}_i^T\tilde{x}) & \text{no violated constraints} \\ +\infty & \text{at least one constraint violated} \end{cases} \tag{8}$$

where $\tilde{c} = \begin{bmatrix} \mathbb{0}_n & \mathbb{1}_m \end{bmatrix}^T$ and $\tilde{x} = \begin{bmatrix} x & z \end{bmatrix}^T$.

The full, unconstrained central path formulation is

$$\min \quad t\tilde{c}^T\tilde{x} - \sum_{i=1}^{2m} \log(\tilde{b}_i - \tilde{a}_i^T\tilde{x}) \tag{9}$$

This particular choice of barrier function allows for an analytical gradient expression, and hence gradient descent. The gradient of this cost function with respect to $\tilde{x}$ is

$$t\tilde{c} + \sum_{i=1}^{2m} \frac{\tilde{a}_i}{\tilde{b}_i - \tilde{a}_i^T\tilde{x}} \tag{10}$$

## 2.2 Part (b)

With this gradient in hand we can apply a first order gradient descent method using backtracking line search. In this question we only consider the case $t = 1$, but later on consider full barrier methods which adjust the value of t. For gradient descent, the parameter $\epsilon$ measures convergence using the norm of the gradient and should be chosen to be appropriately small (since the norm of the gradient decreases as we approach an optimum). We must then decide how to calculate this step size, $h$. Backtracking line search is used here, though numerous alternatives exist including the exact line search. Backtracking line search is parameterised by $(\alpha, \beta)$ - $\beta$ controls the coarseness of the search and $\alpha$ is a tolerance parameter for the quality of our solution. This can be extended using the Armijo rule which prevents overly large steps, but this is not included here. The following algorithms outline these two key methods.

**Algorithm 1** Gradient descent

1: Initialise $x \in \mathbf{dom}\phi$
2: **while** $||\nabla f(x)|| > \epsilon$ **do**
3: Step direction: $\Delta x \leftarrow -\nabla f(x)$
4: Find step length, $h$, by backtracking line search
5: Update: $x \leftarrow x + h\Delta x$
6: **end while**
7: Return $x$

**Algorithm 2** Backtracking line search

1: Initialise $h \leftarrow 1$
2: **while** $f(x + h\Delta x) > f(x) + \alpha h \nabla f(x)^T \Delta x$ **do**
3: $\quad h \leftarrow \beta h$
4: **end while**
5: Return $h$

Importantly, to use interior point methods, the problem must be initialised in the feasible region. Hence, we need a method to find a basic feasible solution. This can be achieved by solving a separate linear program

$$\begin{aligned} \min \quad & s \in \mathbb{R} \\ \text{s.t.} \quad & s \geqslant f_i(x), \quad \forall i \\ & s \leqslant 0 \end{aligned} \tag{11}$$

Note that we can drop the constraint $s \leqslant 0$ since if a basic feasible solution exists for this problem then $\min s$ must be negative. Using the fact that $f_i(x) = \tilde{A}\tilde{x} - \tilde{b}$ we can rewrite this as

$$\begin{aligned} \min \quad & s \\ \text{s.t.} \quad & \begin{bmatrix} \tilde{A} & -\mathbb{1}_{2m \times 1} \end{bmatrix} \begin{bmatrix} \tilde{x} \\ s \end{bmatrix} \preceq \tilde{b} \end{aligned} \tag{12}$$

This problem is simple to solve using the LP solvers used in the first question of this report.

The first order method was applied to the data pair $(A_3, b_3)$ where $n = 256$. With $t = 1$ and $\epsilon = 0.1$ the algorithm converged after 709 iterations to the value $tf_0(x^*) + \phi(x^*) = 335.0739$. It is worth noting that the algorithm converged to a different optimum to that in question 1 ($\min ||Ax - b||_1 = 144.8620$). This is because the central path defines a locus of optima for $t > 0$ in the output space, of which we have chosen the point corresponding to $t = 1$. To get closer to the true optimum full barrier methods should be considered which consider an increasing set of $t$ values.

### 2.3 Part (c)

A semi-log plot of errors (against the LP value from the first question) is shown in figure (3). The convergence is approximately linear in the central section. This tells us that each iteration makes an improvement in error which is equally spaced from the last on a log scale. I

## 3 Sparse Signal Modelling

$l_1$ regularisation has garnered increasing interest in the field of sparse signal modelling. The $l_1$ regularised least squares problem is

$$\min \quad ||Ax - b||_2^2 + \lambda ||x||_1 \tag{13}$$

where the $||x||_1$ term penalises a lack of sparseness in the solution $x^*$ and $\lambda > 0$. $x \in \mathbb{R}^{256}$ is the original signal and $A \in \mathbb{R}^{60 \times 256}$ is the measurement matrix such that $b = Ax$. In this case, the signal is known a-priori to be sparse - there are only 10 non-zero values.

In contrast to the $l_2$ regularised problem

$$\min \quad ||Ax - b||_2^2 + \lambda ||x||_2 \tag{14}$$

the $l_1$ objective function is non-linear in $x$, and we cannot obtain a closed form solution to the minimisation problem. For this reason we seek a form of the problem that can be implemented using barrier methods.

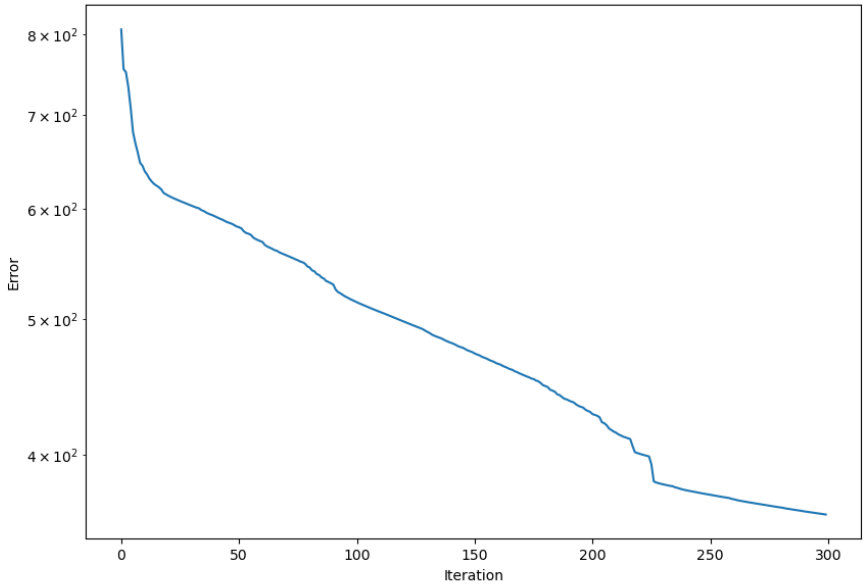



Figure 3: Semi-log plot for convergence of gradient descent

### 3.1 Part (a)

An expanded form for this objective is (dropping any terms not involving $x$)

$$\begin{aligned} f(x) &= (Ax-b)^T(Ax-b) + \lambda \sum_{i=1}^{256} |x_i| \\ &= x^T A^T Ax - 2b^T Ax + \lambda \sum_{i=1}^{256} |x_i| \end{aligned} \tag{15}$$

We can linearise the absolute function, $|\cdot|$, by converting this unconstrained problem into a constrained problem of the form

$$\begin{aligned} \min \quad & x^T A^T Ax - 2b^T Ax + \lambda \sum_{i=1}^{256} u_i \\ \text{s.t.} \quad & -u_i \leqslant x_i \leqslant u_i \quad i \in \{1, \dots, 256\} \end{aligned} \tag{16}$$

Therefore we have a convex quadratic objective and $2n = 512$ constraints. We define the barrier function

$$\Phi(x,u) = -\sum_{i=1}^{256} \log(u_i + x_i) - \sum_{i=1}^{256} \log(u_i - x_i)$$

from which we can formulate a central path for this constrained problem for $t \in (0, \infty)$

$$\phi_t(x,u) = t||Ax-b||_2^2 + t\lambda \sum_{i=1}^{256} u_i + \Phi(x,u) \tag{17}$$

## 3.2 Part (b)

To apply the Newton barrier method to this central path we require its gradient and Hessian. The gradient can be constructed as

$$g = \nabla\phi_t = \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} \in \mathbb{R}^{2n}$$

Where

$$V_1 = \frac{\partial\phi_t}{\partial x} = 2tA^T(Ax-b) + \frac{\partial\Phi}{\partial x} \quad \text{and} \quad V_2 = \frac{\partial\phi_t}{\partial u} = t\lambda\mathbb{1}_n + \frac{\partial\Phi}{\partial u} \tag{18}$$

with barrier derivatives

$$\frac{\partial\Phi}{\partial x} = \mathbf{ones}_n \frac{2x_i}{u_i^2 - x_i^2} \quad \text{and} \quad \frac{\partial\Phi}{\partial u} = \mathbf{ones}_n \frac{-2u_i}{u_i^2 - x_i^2} \tag{19}$$

Similarly

$$H = \nabla^2\phi_t = \begin{bmatrix} M_1 & M_2 \\ M_2 & M_3 \end{bmatrix} \in \mathbb{R}^{2n\times 2n}$$

$$M_1 = \frac{\partial^2\phi_t}{\partial x^2} = 2tA^TA + \frac{\partial^2\Phi}{\partial x^2} \quad \text{and} \quad M_2 = \frac{\partial^2\phi_t}{\partial x\partial u} = \frac{\partial^2\Phi}{\partial x\partial u} \quad \text{and} \quad M_3 = \frac{\partial^2\phi_t}{\partial u^2} = \frac{\partial^2\Phi}{\partial u^2} \tag{20}$$

with barrier second derivatives

$$\frac{\partial^2\Phi}{\partial x^2} = \frac{\partial^2\Phi}{\partial u^2} = \mathbf{diag}_{n\times n}\frac{1}{(u_i - x_i)^2} + \frac{1}{(u_i + x_i)^2} \quad \text{and} \quad \frac{\partial^2\Phi}{\partial x\partial u} = \mathbf{diag}_{n\times n}\frac{1}{(u_i + x_i)^2} - \frac{1}{(u_i - x_i)^2} \tag{21}$$

Where $\mathbf{ones}_n f(x_i, u_i)$ represents a length $n$ vector whose $i^{th}$ value is $f(x_i, u_i)$ and $\mathbf{diag}_{n\times n} f(x_i, u_i)$ represents a diagonal, $n \times n$ matrix whose $i^{th}$ value is $f(x_i, u_i)$.

These rather lengthy expressions allow us to calculate a step size and direction for minimisation with the Newton method for fixed $t$. These steps are calculated at each iteration by solving the system of equations

$$H_k x_{k+1} = v_k \quad \text{where} \quad v_k = H_k x_k - g_k \tag{22}$$

using the Cholesky decomposition of $H$. As mentioned in question 2, full barrier methods must consider an increasing sequence of $t$ values and so we define the full Newton barrier method in algorithm (3).

**Algorithm 3** Newton Barrier Method

```
Initialise x ∈ dom φ_t
t ← 1
while 2n/t > δ do
    while ||∇f(x)|| > ϵ do
        Calculate H and g
        Calculate v ← Hx − g
        Solve Hx = g via Cholesky decomposition
    end while
    t ← μt
end while
Return x
```

In this case, $n = 256$ and several basic feasible solution were immediately obvious - for example setting all $x_i = 0$ and all $u_i = 1$. The parameter $\epsilon$ measures convergence in the gradient descent as before, whilst $\delta$ measures convergence as we travel along the central path. $\mu$ is the proportion by which we increase $t$ before each *centering step*. Smaller $\mu$ values correspond to a finer search along the central path. The *centering step* is the inner loop for each value of $t$ in which we navigate back to the central path after increasing $t$.

## 3.3 Part (c)

This algorithm was applied to some supplied test data. The regularisation parameter was set to $\lambda = 0.01\lambda_{\max}$ where $\lambda_{\max} = ||2A^T b||_\infty$, as suggested in the course notes.

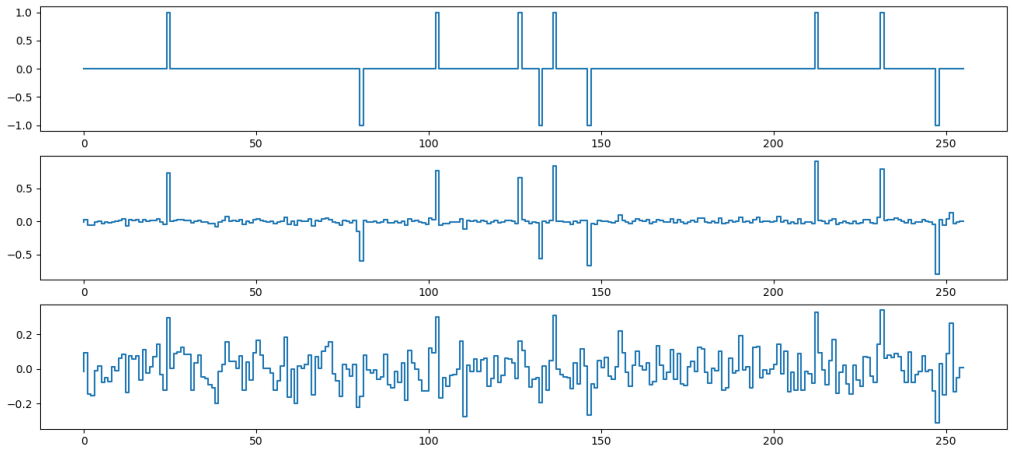



Figure 4: Top: original signal. Middle: $l_1$ regularised reconstruction. Bottom: minimum energy reconstruction

The top plot shows the original signal with 10 spikes of amplitude $\pm 1$. Below this is the $l_1$ reconstructed signal. The regularisation has clearly encouraged sparsity as there are 10 very clear spikes which line up with the original signal, with the rest of the signal values being very close to zero. This is consistent with our observations for $l_1$ norm approximation - the spikes are treated as outliers, whilst the $l_1$ regularisation tries to bring every other residual as close to zero as possible.

## 3.4 Part (d)

The bottom plot shows the minimum energy reconstruction of the signal, which is the point in $\{x \in \mathbb{R}^{256} : A^T Ax = A^T b\}$ that is closest to the origin in $l_2$ norm. If we consider the corresponding constrained minimisation problem this is

$$\begin{aligned} \min \quad & ||x||_2^2 \\ \text{s.t.} \quad & A^T Ax = A^T b \end{aligned} \tag{23}$$

Or recasting as an unconstrained problem using Lagrange multipliers

$$\min \quad x^T x + \lambda A^T (b - Ax) \tag{24}$$

This problem has the closed form solution $x^* = (A^T A)^{-1} A^T b$ which can easily be implemented with standard least squares solvers.

Comparing reconstruction errors for the two reconstructions

$$\epsilon = ||Ax - b||_2^2$$

for the $l_1$ regularised problem we get $\epsilon = 311.28$ and for the minimum energy we get $\epsilon = 315.12$. This would suggest that these two reconstructions are very similar in quality quantatively, despite the minimum energy

qualitatively being much poorer than the $l_1$ regularised solution. For this reason, when signal sparsity is given as prior information, $l_1$ regularisation should be preferred.

# 4 Appendix

## 4.1 Code for Q1

```
import numpy as np
from cvxopt import matrix, solvers
import matplotlib.pyplot as plt
from scipy.linalg import cho_factor, cho_solve


def solve_linf(a, b):
        """Construct appropriate matrices then solve the l infinity norm approximation LP
            """
        m = a.shape[0]
        n = a.shape[1]

        astack = matrix(np.vstack((a, -a)))
        tm_ones = matrix(-1*np.ones(2*m))
        Atilde = matrix([[astack], [tm_ones]])
        btilde = matrix(np.vstack((b, -1*b)))
        ctilde = np.zeros(n+1)
        ctilde[-1] = 1
        ctilde = matrix(ctilde)

        sol = solvers.lp(ctilde, Atilde, btilde)

        return sol['x'], sol['s']

def solve_lone(a, b):
        """Construct appropriate matrices then solve the l1 norm approximation LP"""
        m = a.shape[0]
        n = a.shape[1]

        astack = matrix(np.vstack((a, -a)))
        tm_ones = matrix(np.vstack((-1*np.eye(m), -1*np.eye(m))))
        Atilde = matrix([[astack], [tm_ones]])
        btilde = matrix(np.vstack((b, -1*b)))

        czs = np.zeros(n)
        cones = np.ones(m)
        cstack = np.hstack((czs, cones))
        ctilde = matrix(cstack)

        sol = solvers.lp(ctilde, Atilde, btilde)

        return sol['x'], sol['s']
```

```
def solve_ltwo(a, b):
        """Solve l2 problem using least squares"""
        S = np.matmul(a.T, a)
        q = np.matmul(a.T, b)
        cho_fac = cho_factor(S)
        x = cho_solve(cho_fac, q)
        residuals = b - np.matmul(a, x)
        return x, residuals
```

Figure 5: Code for Q1

### 4.2 Code for Q2

```
import numpy as np
from cvxopt import matrix, solvers
from scipy.linalg import norm

def central_path(x, c, a, b, t):
        """Central path in q2"""
        x = matrix(x)
        term1 = t*np.dot(c.T, x)
        term2 = np.sum(np.log(b - (a*x)), axis=0)
        if np.sum(np.isnan(term2))>0:
                return matrix(np.inf*np.ones(1))
        else:
                return term1 - term2

def central_path_grad(x, c, a, b, t):
        """Gradient of central path in q2"""
        term1 = np.array(t*c)
        term2 = np.sum(a/np.array(b - a*x), axis=0)
        total = term1.flatten() + term2
        return matrix(total)

def build_l1_tildes(a, b):
        """Build the Atilde and Btilde matrices for the l1 norm approximation LPs"""
        m = a.shape[0]
        n = a.shape[1]
        at = np.block([[a, -1*np.eye(m)],[-a, -1*np.eye(m)]])
        bt = np.block([[b], [-1*b]])
        return matrix(at), matrix(bt)

def find_bfs(a, b, m, n):
        """Solve the initialisation LP for q2"""
        aones = matrix(-1*np.ones(2*m))
        ai = matrix(np.hstack((matrix(a), aones)))
        bi = matrix(b)
        c = np.zeros(n+m+1)
        c[-1] = 1
        ci = matrix(c)

        sol = solvers.lp(ci, ai, bi)
        bfs = sol['x'][:-1]

        return bfs
```

```
def backtrack(x, data, func, grad, alpha, beta, cvec, u):
        """Backtracking linesearch algorithm"""
        x = matrix(x)
        t = 1
        current_func = func(x, cvec, data[0], data[1], u)
        gradient = grad(x, cvec, data[0], data[1], u)
        step = -1*alpha*(np.linalg.norm(gradient)**2)
        new_func = func((x-t*gradient), cvec, data[0], data[1], u)
        while new_func[0] > (current_func[0]+t*step):
                t *= beta
                new_func = func((x-t*gradient), cvec, data[0], data[1], u)
        return t

def first_order_method(data, func, grad, cvec, epsilon, m, n, u=1.):
        """Full gradient descent method for q2"""
        x = find_bfs(data[0], data[1], m, n)
        iteration=0
        errors = []
        while np.linalg.norm(grad(x, cvec, data[0], data[1], u)) > epsilon:
                step_dir = -1*grad(x, cvec, data[0], data[1], u)
                step_len = backtrack(x, data, func, grad, 0.2, 0.4, cvec, u)
                x += (step_len * step_dir)
                iteration +=1
                errors.append(func(x, cvec, data[0], data[1], u)[0][0])
        return x, func(x, cvec, data[0], data[1], u)[0], errors
```

Figure 6: Code for Q2

### 4.3 Code for Q3

```
import numpy as np
from scipy.linalg import cho_factor, cho_solve

def grad_phi(A, b, l, t, x, u):
        """
        Gradient of central path for q3
        """
        n = x.shape[0]

        gradx1 = 2.*t*np.matmul(A.T, np.matmul(A, x)-b.flatten())
        gradx2 = np.divide(2.*x, np.square(u)-np.square(x))
        gradx = np.add(gradx1, gradx2)

        gradu1 = t*l * np.ones(n)
        gradu2 = np.divide(-2.*u, np.square(u)-np.square(x))
        gradu = np.add(gradu1, gradu2)

        return np.hstack((gradx, gradu))

def hessian_phi(A, b, t, x, u):
        """
        Hessian of central path for q3
        """
        gradxx1 = 2.*t*np.matmul(A.T, A)
        gradxx2 = np.diag(np.divide(1., np.square(u+x)) + np.divide(1., np.square(u-x)))
        gradxx = np.add(gradxx1, gradxx2)

        graduu = np.diag(np.divide(1., np.square(u+x)) + np.divide(1., np.square(u-x)))

        gradux = np.diag(np.divide(1., np.square(u+x)) - np.divide(1., np.square(u-x)))

        return np.block([[gradxx, gradux], [gradux, graduu]])

def central_phi(A, b, l, t, x, u):
        """
        Central path for q3
        """
        term1 = np.linalg.norm(np.matmul(A, x) - b)
        term2 = t*l*np.sum(u)
        term3 = -1.*np.sum(np.log(u+x))-1.*np.sum(np.log(u-x))

        return term1 + term2 + term3
```

```
def newton_step(A, b, l, t, coords):
        """
        Take a newton step
        """
        x = coords[:256]
        u = coords[256:]

        grad = grad_phi(A, b, l, t, x, u)
        hessian = hessian_phi(A, b, t, x, u)

        # avoid inverting the 512x512 hessian -- xk+1 = xk - h where h =H^-1g
        c = cho_factor(hessian)
        step = cho_solve(c, grad)
        coord_new = coords - step
        # vec = np.matmul(hessian, coords) - grad
        # c = cho_factor(hessian)
        # xnew = cho_solve(c, vec)
        return coord_new


def barrier_method(A, b, l, t0, coords, n, epsilon, delta, mu, grad):
        """Full barrier method"""
        t = t0
        while 2*n/t > delta:
                while np.linalg.norm(grad(A, b, l, t, coords[:256], coords[256:])) >
                    epsilon:
                        coords_old = coords
                        coords = newton_step(A, b, l, t, coords_old)
                t *= mu
        return coords, t

def minimum_energy(A, b, delta):
        """
        Minimum Energy Reconstruction
        """
        return np.linalg.lstsq(A, b)[0]
```

Figure 7: Code for Q3