4M17 Assignment 1

December 2019

1 Question 1

1.a)

For a vector $x \in \mathbf{R}^m$, the general p-norm can be defined as:

$$\|x\|_p = \sqrt[p]{\sum_{i=1}^{m} |x_i|^p}$$

Therefore using the definition, we can find expressions for the l_1, l_2 and l_∞ norms as follows:

$$\|x\|_1 = \sum_{i=1}^{m} |x_i|, \quad \|x\|_2 = \sqrt{\sum_{i=1}^{m} |x_i|^2}, \quad \|x\|_\infty = \max_i(|x_i|)$$

In the case of the l_2 norm, we can express the problem as an optimisation problem as follows:

$$\min_x \|Ax - b\|_2 \equiv \min_x((Ax - b)^T(Ax - b))$$

$$\frac{\partial}{\partial x}(x^T A^T Ax - 2x^T A^T b + b^T b) = 0$$

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

By squaring the solution, the minimum x value will not be affected, and we are able to differentiate the resulting quadratic and solve equal to zero for an analytic solution to x, which can be done by solving a linear system of equations as above.

1.b)

Given the l_1 norm approximation problem:

$$\min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|_1$$

from Q1.a), we can express this as:

$$\min_{\mathbf{x}} \sum_{i=1}^{m} |a_i^T x - b_i|$$

and using the fact:

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

we now look to minimise the sum of t_i. This can be expressed as the linear program:

$$\min_{\mathbf{x}} \tilde{\mathbf{c}}^T \tilde{\mathbf{x}}$$

$$\text{subject to } \tilde{\mathbf{A}}\tilde{\mathbf{x}} \geq \tilde{\mathbf{b}}$$

Where

$$\tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ \mathbf{t} \end{bmatrix},\ \tilde{\mathbf{c}} = \begin{bmatrix} \mathbf{0}_n \\ \mathbf{1}_m \end{bmatrix},\ \tilde{\mathbf{A}} = \begin{bmatrix} -\mathbf{A} & \mathbf{I}_m \\ \mathbf{A} & \mathbf{I}_m \end{bmatrix},\ \tilde{\mathbf{b}} = \begin{bmatrix} -\mathbf{b} \\ \mathbf{b} \end{bmatrix}$$

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

Beginning with the l_∞ norm approximation problem:

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_\infty$$

We can do similar to that done above, and fully express the l_∞ norm:

$$\min_{\mathbf{x}}\ \max_i |a_i^T b - b_i|$$

We note that

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

This gives us a linear program where we look to minimise t, and can be expressed fully as:

$$\min_{\mathbf{x}} \tilde{\mathbf{c}}^T \tilde{\mathbf{x}}$$

$$\text{subject to } \tilde{\mathbf{A}}\tilde{\mathbf{x}} \geq \tilde{\mathbf{b}}$$

Where

$$\tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ t \end{bmatrix},\ \tilde{\mathbf{c}} = \begin{bmatrix} \mathbf{0}_n \\ 1 \end{bmatrix},\ \tilde{\mathbf{A}} = \begin{bmatrix} -\mathbf{A} & \mathbf{1}_m \\ \mathbf{A} & \mathbf{1}_m \end{bmatrix},\ \tilde{\mathbf{b}} = \begin{bmatrix} -\mathbf{b} \\ \mathbf{b} \end{bmatrix}$$

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

1.c)

Using Matlab's inbuilt dual-simplex algorithm (`linprog`) the minimums for the l_1, l_2 and l_∞ norms were found, and the run times recorded for each of the 5 data sets:

Dataset	$\|Ax-b\|_1$	$\|Ax-b\|_2$	$\|Ax-b\|_\infty$	l_1 runtime /s	l_2 runtime /s	l_∞ runtime /s
A_1, b_1	10.26	2.39	0.61	0.020	0.00018	0.18
A_2, b_2	33.61	4.40	0.58	0.15	0.0060	0.055
A_3, b_3	143.26	9.39	0.61	3.27	0.010	1.39
A_4, b_4	277.18	12.91	0.59	37.14	0.045	13.92
A_5, b_5	571.63	18.55	0.60	485.41	0.77	163.12

From these results we can see that l_1 norm scaled linearly with th size of the dataset as we would expect, as it is the sum of the 'taxicab' distances of residuals. The l_2 norm appears to scale with the square root of the data size, and the l_∞ norm stays almost constant for each of the data sets, as it is a measure of the largest residual, so given the datasets are of similar order of magnitude, we can expect this result. The run times for l_1 and l_∞ increased greatly with the size of the data, with the l_2 run time hardly increasing at all even for the largest of the datasets. This is due to the fact we have an analytical solution for the l_2 norm minimisation problem, which involves solving a linear system of equations, and so we can expect this to be far faster than using an optimisation algorithm for the other norm problems.

1.d)

The histogram seen in Figure 1 shown the distribution of residuals for each of the norm approximation problems. Looking first at the l_1 norm problem, we can see that a large proportion of the residuals are 0, with very few non-zero. This is as we are looking to minimise the sum of all the residuals, so the optimisation has reduced this sum as much as possible. This means the method is very good for reducing the total residuals, and ignores outliers, but does not do well to reduce the residuals of outliers. The l_2 residuals plot resembles a Gaussian distribution, and this is expected as the l_2 norm problem also is a result of a maximum likelihood estimation with Gaussian likelihood, which produces Gaussian errors. Finally the residuals for the l_∞ problem shows a large bias towards minimising the outliers. This is shown by the fact that the largest residual is only 0.7 in magnitude, but that there are a large number of residuals at this value. From the above, it seems that a decrease in the p-value for the p-norm leads to the distribution shrinking inwards.

2 Question 2

2.a)

We look to find a solution to the problem:

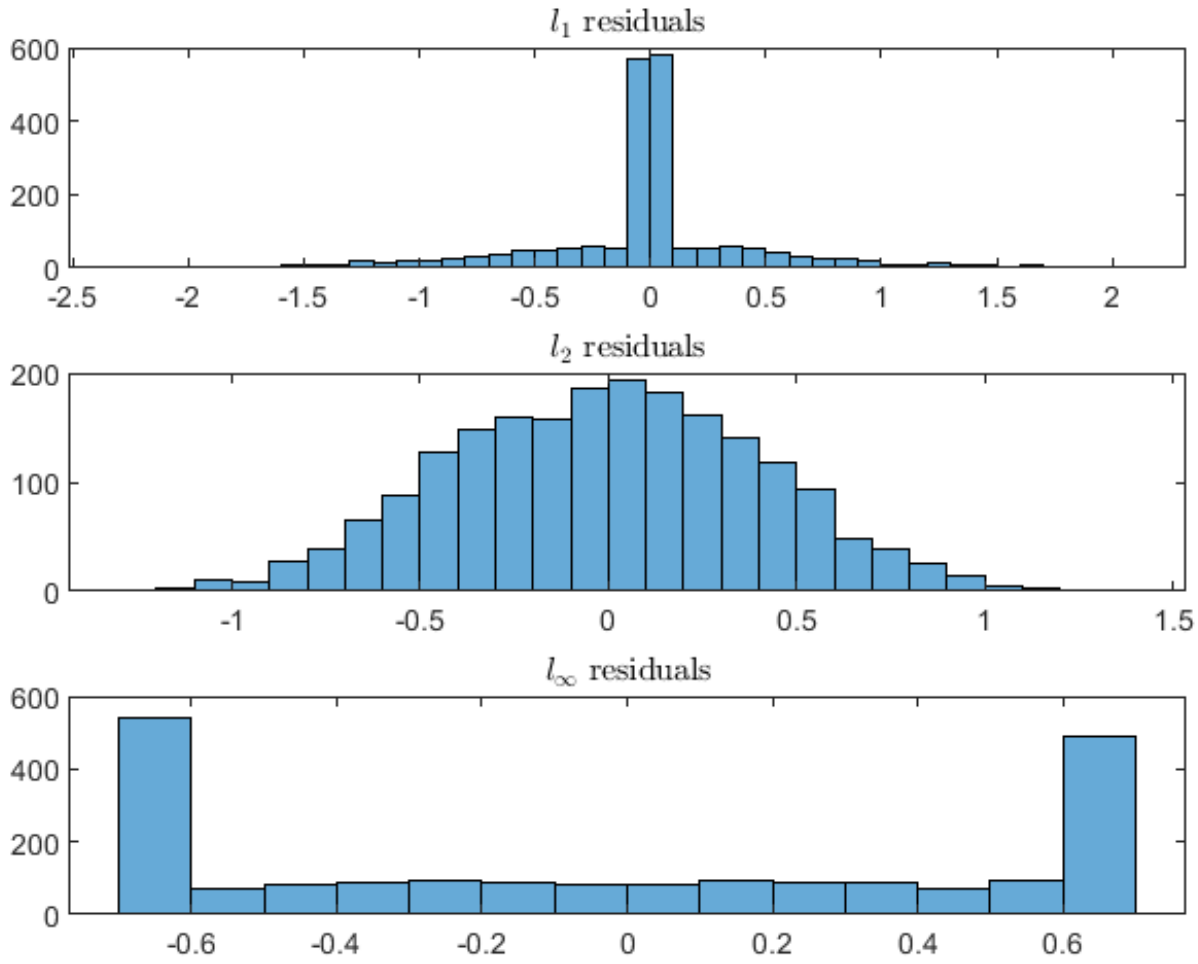
$$\min_x f_0(x)$$
$$\text{subject to } f_i(x) \leq 0, \quad i = 1, ..., m$$

This constrained problem can be converted to an unconstrained one by replacing the constraints with the logarithmic barrier function:

$$\phi(x) = \begin{cases} -\sum_{i=1}^m \log(-f_i(x)) & f_i(x) \leq 0 \\ \infty & f_i(x) > 0 \end{cases}$$

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

Figure 1: Histograms of residuals for the A_5, b_5 dataset for l_1, l_2 and l_∞ problems



The results in a central path formulation:

$$C(x) = tf_0(x) + \phi(x)$$

The explicit form of this corresponding to the LP formulation of the l_1 norm approximation is given as follows:

$$f_0(x) = \tilde{c}^T\tilde{x} \quad f_i(x) = \tilde{a}_i^T\tilde{x} - \tilde{b}_i$$

With $\tilde{a}_i$ being the ith row of $\tilde{A}$ and $\tilde{A}$, $\tilde{b}$, $\tilde{c}$ and $\tilde{x}$ being defined as in question 1.b). The gradient of the cost function $C(x)$, $\nabla C(x)$ is given as:

$$\nabla C(x) = t\tilde{c} + \nabla\phi(x) = t\tilde{c} + \sum_{i=1}^{m} \frac{\tilde{a}_i}{\tilde{b}_i - \tilde{a}_i^T\tilde{x}}$$

2.b)

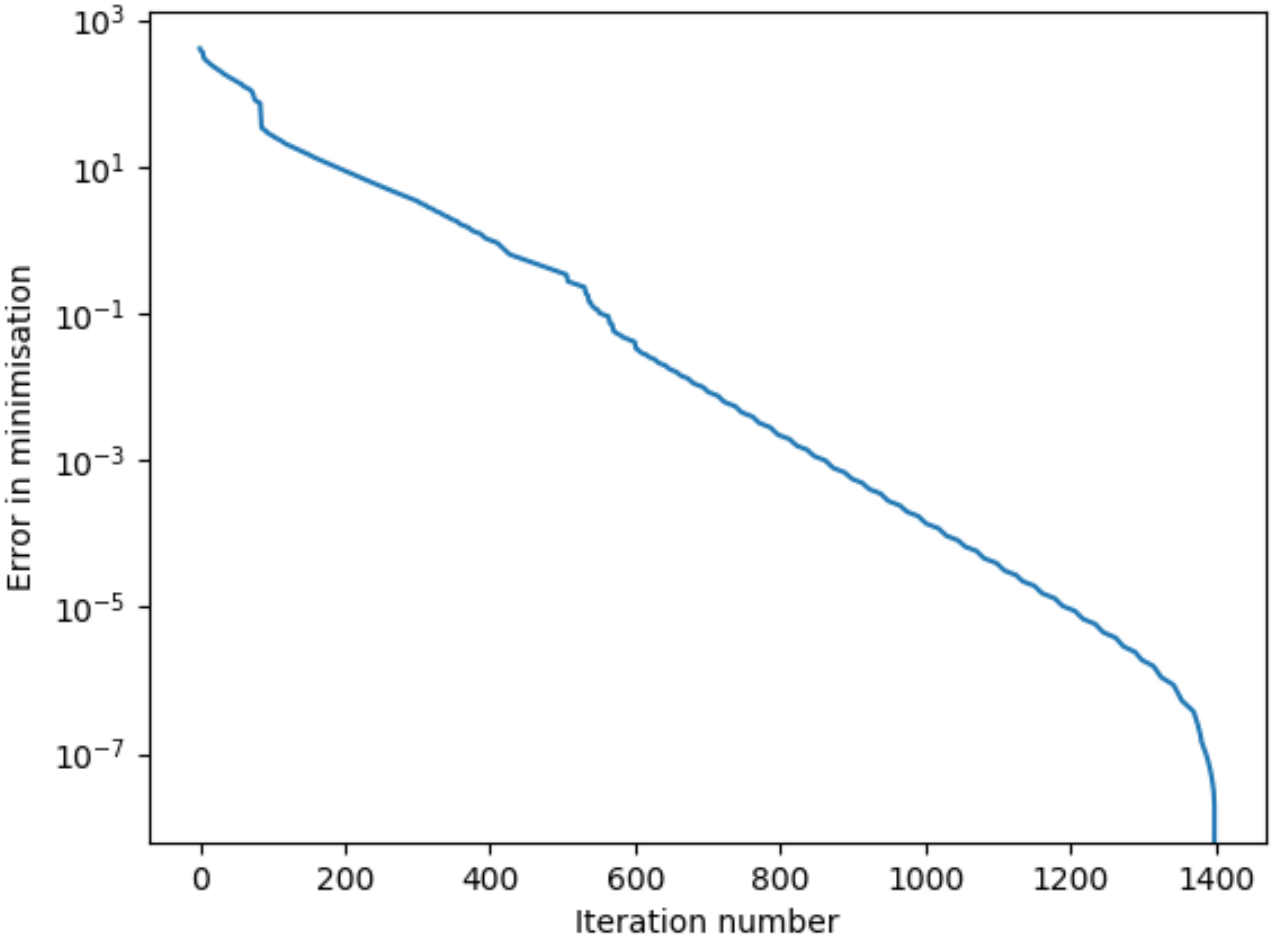
Using the code listed in Appendix B, the minimum value of the cost function was found to be 335.19281. The code uses an initial which is within the feasible set by solving $Ax = b$ using a pseudo-inverse, with

$\alpha = 0.25$, $\beta = 0.45$ and $t = 1$.

2.c)

Figure 2 shows the convergence of the minimisation error with each iteration of the gradient descent. Boyd and Vandenberghe [1] state that there is not a large variation in convergence time with different values for α and β, and that the convergence is found to be linear for a similar problem. This is what is observed in Figure 2, with a faster convergence when the error becomes less that 10e-6.

Figure 2: Convergence plot of the minimisation error with iteration number for a first order gradient method with backtracking linesearch on dataset A3, b3



3 Question 3

3.a)

We begin with the l_1 regularised least squares problem:

$$\min_x \ \|Ax - b\|_2^2 + \lambda \|x\|_1$$

This problem can be transformed into a quadratic problem with linear constraints using a similar technique to that seen in Question 1, where we introduce a new variable $\mathbf{u}$ to bound $\mathbf{x}$, replacing the absolute function.

$$\min_x \ \|Ax - b\|_2^2 + \lambda \sum_{i=1}^{n} u_i$$

$$\text{subject to } \ -u_i \leq x_i \leq u_i$$

Defining a logarithmic barrier function of both x and u to be

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

we can now represent the constrained problem above as a central path (unconstrained problem) by replacing the bounds with this logarithmic barrier function.

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

3.b)

The gradient of $\phi(x, u)$ must be defined over both $\mathbf{x}$ and $\mathbf{u}$, given we now have a problem with both variables:

$$\nabla_x \phi(x, u) = 2tA^T(Ax - b) + \begin{bmatrix} \frac{1}{u_1 - x_1} - \frac{1}{u_1 + x_1} \\ \vdots \\ \frac{1}{u_n - x_n} - \frac{1}{u_n + x_n} \end{bmatrix}$$

$$\nabla_u \phi(x, u) = t\lambda \mathbf{1} - \begin{bmatrix} \frac{1}{u_1 - x_1} + \frac{1}{u_1 + x_1} \\ \vdots \\ \frac{1}{u_n - x_n} + \frac{1}{u_n + x_n} \end{bmatrix}$$

And the hessian will then be a square matrix given by

$$\nabla^2 \phi(x, u) = \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix}$$

Where:

$$H_{11} = 2tA^T A + diag\left(\frac{2(u_1^2 + x_1^2)}{(u_1^2 - x_1^2)^2}, \ldots, \frac{2(u_n^2 + x_n^2)}{(u_n^2 - x_n^2)^2} \right)$$

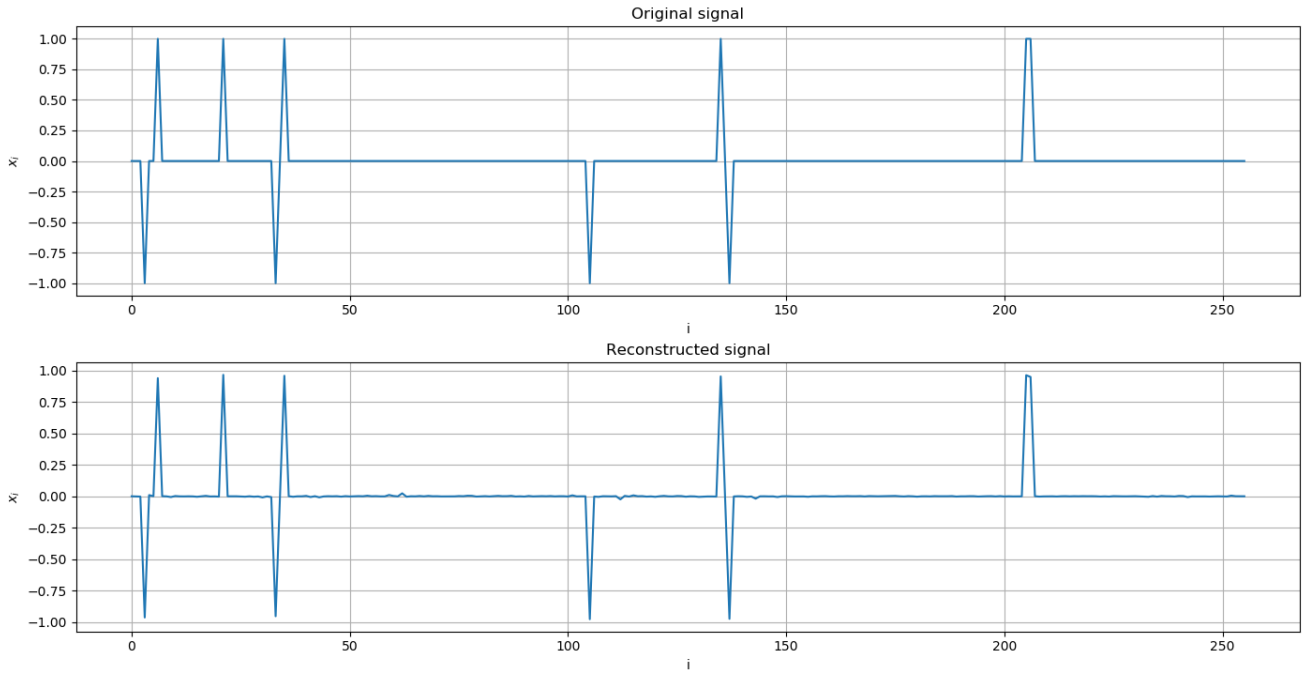
$$H_{12} = H_{21} = diag\left(\frac{-4x_1u_1}{(u_1^2 - x_1^2)^2}, \ldots, \frac{-4x_nu_n}{(u_n^2 - x_n^2)^2}\right)$$

$$H_{22} = diag\left(\frac{2(u_1^2 + x_1^2)}{(u_1^2 - x_1^2)^2}, \ldots, \frac{2(u_n^2 + x_n^2)}{(u_n^2 - x_n^2)^2}\right)$$

3.c)

Figure 3 shows both the original and reconstructed signals, and highlights the benefits of using the l_1 regularisation to act as a bound on the cardinality of the solution. There is some noise present in the reconstructed signal, but with an increase in the parameter t, this can be reduced even further. Figure 3 uses $t = 100000$, and converged using an exact Newton method in 18 iterations to a value of $\lambda^2/2 < 1e-10$, where $\lambda^2 = \nabla f(x)^T \nabla^2 f(x)^{-1} \nabla f(x)$. The method currently used requires a calculation of the inverse of the Hessian, which would become computationally prohibitive for very large problems, but this can be avoided using a Truncated Newton interior-point method (TNIPM), as detailed in [2]. This method is extremely useful for reconstructing signals from a smaller set of observations b, as in this case $b \in \mathbf{R}^{60}$ whereas $x \in \mathbf{R}^{256}$.

Figure 3: Original and reconstructed signal $\mathbf{x}$ found by minimising the l_1 regularised least squares problem with an exact Newton method

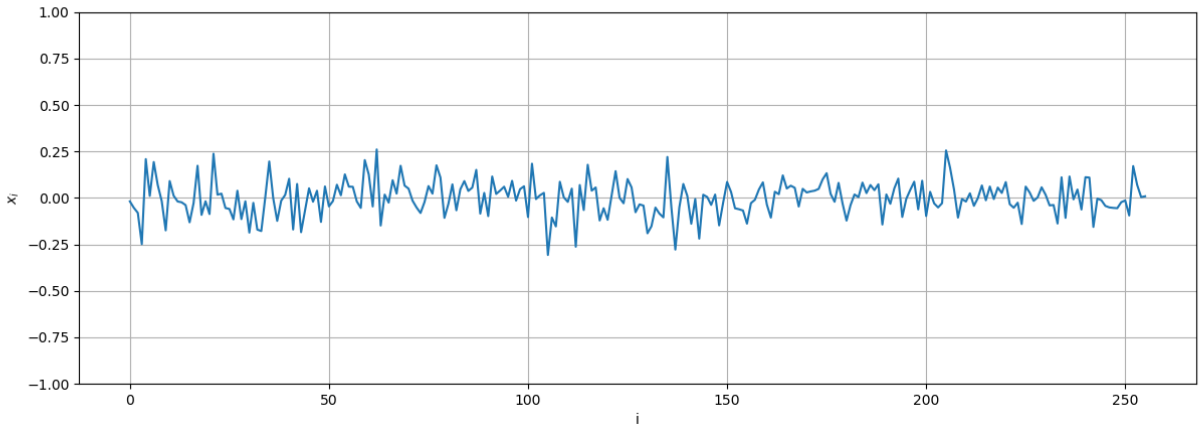


3.d)

The minimum energy reconstruction is the point in the set $\{x \in \mathbf{R}^{256} | A^T Ax = A^T b\}$. This is plotted in Figure 4, and is the result of the unregularized least squares problem. As there is no penalty for $\mathbf{x}$ being

far from the origin, the cardinality of the solution is high, and is a very poor representation of the original signal. As the matrix $A^T A$ was found to have a condition number (p=2) of 3.10e+18, the a pseudo-inverse could not be used to find the minimum energy reconstruction, and so instead the least squares problem was solved (using the `numpy.linalg.lstsq`) function.

Figure 4: Minimum energy reconstruction of the original signal, calculated using the pseudo-inverse as detailed in Q1.a)



A Question 1 code

```
clear
set(0,'defaulttextinterpreter','latex')
set(0,'DefaultTextFontname', 'CMU Serif')
set(0,'DefaultAxesFontName', 'CMU Serif')

load('A5.mat');
load('b5.mat');
A = A5;
b = b5;

dimens = size(A);

%l_1
A_ = [-A,eye(dimens(1));A,eye(dimens(1))];
b_ = [-b;b];
c_ = [zeros(dimens(2),1);ones(dimens(1),1)];

disp("l_1 problem")
tic
x_1 = linprog(c_, -A_, -b_);
time = toc;
disp(["time: " num2str(time)])
disp(["l1 norm:"  num2str(sum(abs(A*x_1(1:dimens(2))-b)))])

%l_2
disp("l_2 problem")
tic
x_2 = linsolve(A,b);
time = toc;
disp(["time: " num2str(time)])
disp(["l2 norm:"  num2str(sqrt(sum(power((A*x_2-b),2))))])

%l_inf
disp("l_inf problem")
A_ = [-A,ones(dimens(1),1);A,ones(dimens(1),1)];
b_ = [-b;b];
c_ = [zeros(dimens(2),1);1];

tic
x_inf = linprog(c_, -A_, -b_);
time = toc;
disp(["time: " num2str(time)])
disp(["linf norm:"  num2str(max(abs(A*x_inf(1:dimens(2))-b)))])

%histogram of results
figure
subplot(3,1,1)
```

```
histogram(A*x_1(1:dimens(2))-b)
title("$l_1$ residuals")
subplot(3,1,2)
histogram(A*x_2-b)
title("$l_2$ residuals")
subplot(3,1,3)
histogram(A*x_inf(1:dimens(2))-b)
title("$l_{\infty}$ residuals");
```

B Question 2 code

```
import numpy as np
import scipy.io as sio
from matplotlib import pyplot as plt

def loadData(number):
    A = sio.loadmat('A{}.mat'.format(number))['A{}'.format(number)]
    b = sio.loadmat('b{}.mat'.format(number))['b{}'.format(number)]
    return A,b

def l2_norm(x):
    return np.sqrt(np.sum(np.power(x,2)))

def phi(x):
    result = 0
    for i in range(2*m):
        if b_[i]-np.dot(A_[i],x) < 0:
            return np.inf
        result -= np.log(b_[i]-np.dot(A_[i],x))
    return result

def f(x):
    return t*np.dot(c_,x) + phi(x)

def grad_f(x):
    result = t*c_
    for i in range(2*m):
        result += A_[i]/(b_[i]-np.dot(A_[i],x))
    return result


def gradient_descent(x0):
    x = x0
    dx = -grad_f(x)
    allf = np.array([])
    iteration = 0
    while l2_norm(dx) > stopping_crit:
```

```
        dx = -grad_f(x)
        tau = 1
        while f(x+tau*dx) >= f(x) - alpha*tau*np.dot(dx, dx):
            tau = tau*beta
        x = x + tau*dx
        iteration += 1
        if iteration%10 == 0:
            print("Step: {}".format(iteration))
            print("Objective function value: {}".format(f(x)))
        allf = np.append(allf, f(x))

    print("Final objective function value: {}".format(allf[-1]))
    plt.plot(allf-allf[-1])
    plt.ylabel("Error in minimisation")
    plt.xlabel("Iteration number")
    plt.yscale("log")
    plt.show()

A, b = loadData(3)
(m,n) = A.shape
A_ = np.block([[A,-np.eye(m)],[-A,-np.eye(m)]])
b_ = np.concatenate((b,-b), axis=None)
c_ = np.concatenate((np.zeros(n), np.ones(m)), axis=None)
x0 = np.dot(np.dot(np.linalg.inv(np.dot(A_.T,A_)),A_.T),b_-5*np.ones(2*m))

t = 1
stopping_crit = 1e-3
alpha = 0.25
beta = 0.45

gradient_descent(x0)
```

C Question 3 code

```
import numpy as np
import scipy.io as sio
import scipy.optimize as opt
from matplotlib import pyplot as plt

def loadData():
    A = sio.loadmat('A.mat')['A']
    x0 = sio.loadmat('x0.mat')['x'].flatten()
    return A,x0

def problem(x):
    return l2_norm(np.dot(A,x)-b)**2 + l*l1_norm(x)
```

```
def l1_norm(x):
    return np.sum(np.absolute(x))

def l2_norm(x):
    return np.sqrt(np.sum(np.power(x,2)))

def inf_norm(x):
    return np.max(np.absolute(x))

def f(X):
    x = X[:n]
    u = X[n:]
    result = t*l2_norm(np.dot(A,x)-b)**2
    for i in range(n):
        result += t*l*u[i]
    return result + barrier(X)

def grad_f(X):
    x = X[:n]
    u = X[n:]
    g1 = 2*t*np.matmul(A.T, np.dot(A,x)-b)
    for i in range(n):
        g1[i] += (2*x[i])/(u[i]**2-x[i]**2)
    g2 = t*l*np.ones(n)
    for i in range(n):
        g2[i] -= (2*u[i])/(u[i]**2-x[i]**2)
    return np.concatenate((g1, g2))

def hessian(X):
    x = X[:n]
    u = X[n:]
    h_11 = np.zeros(n)
    h_22 = np.zeros(n)
    for i in range(n):
        h_11[i] += (-4*x[i]*u[i])/(u[i]**2-x[i]**2)**2
        h_22[i] += (2*(u[i]**2+x[i]**2))/(u[i]**2-x[i]**2)**2
    H_12 = np.diag(h_11)
    H_22 = np.diag(h_22)
    H_11 = 2*t*np.dot(A.T,A) + H_22
    return np.block([[H_11, H_12],[H_12, H_22]])

def barrier(X):
    x = X[:n]
    u = X[n:]
    result = 0
    for i in range(n):
        if u[i] - x[i] < 0 or u[i] + x[i] < 0:
            return np.inf
        result -= np.log(u[i]-x[i]) + np.log(u[i]+x[i])
    return result
```

```
def interior_point():
    iteration = 0
    x = np.concatenate((np.zeros(n),np.ones(n)))
    dx = -np.dot(np.linalg.inv(hessian(x)),grad_f(x))

    while -np.dot(grad_f(x).T, dx)/2 > stopping_crit:
        tau = 1
        tau = exact_linesearch(f,x,dx)
        x = x+tau*dx
        dx = -np.dot(np.linalg.inv(hessian(x)),grad_f(x))

        print("Iteration number {}".format(iteration))
        print("phi(x,u): "+str(f(x)))
        print(problem(x[:n]))
        print("-------------")

        iteration += 1
    return x

def exact_linesearch(f, x, dx):
    g = lambda tau: f(x+tau*dx)
    res = opt.minimize_scalar(g)
    return res.x


A,x0 = loadData()
b = np.dot(A,x0)
(m,n) = A.shape

xmin = np.linalg.lstsq(A,b)[0]
plt.plot(xmin)
plt.grid(True)
plt.xlabel('i')
plt.ylabel('$x_i$')
plt.show()


l_max = inf_norm(2*np.dot(A.T,b))
l = 0.01*l_max
t = 100000
stopping_crit = 1e-10
best_x = interior_point()

print("Final minimised value: " + str(problem(best_x[:n])))

fig, axs = plt.subplots(2, 1)
plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95, hspace=0.25, wspace=0.35)
axs[0].plot(x0)
axs[0].grid(True)
```

```
axs[0].set_xlabel('i')
axs[0].set_ylabel('$x_i$')
axs[0].set_title('Original signal')
axs[1].plot(best_x[:n])
axs[1].grid(True)
axs[1].set_xlabel('i')
axs[1].set_ylabel('$x_i$')
axs[1].set_title('Reconstructed signal')
plt.show()
```

References

[1] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[2] Seung-Jean Kim, Kwangmoo Koh, Michael Lustig, Stephen Boyd, and Dimitry Gorinevsky. An interior-point method for large-scale l_1-regularized least squares. *IEEE journal of selected topics in signal processing*, 1(4):606–617, 2007.