MATH60005 Optimisation Coursework

CID: 02216531

14 November 2024

1 Unconstrained Optimisation

a) We are given the function $f(x_1, x_2) = 8x_1 + 12x_2 + x_1^2 - 2x_2^2$.

i) Let $\mathbf{x} = (x_1, x_2)$. We can rewrite f in standard form $f(\mathbf{x}) = \mathbf{x}^T \mathbf{A} x + 2\mathbf{b}^T \mathbf{x} + \mathbf{c}$ as

$$f(\mathbf{x}) = \mathbf{x}^T \begin{pmatrix} 1 & 0 \\ 0 & -2 \end{pmatrix} \mathbf{x} + 2 \begin{pmatrix} 4 \\ 6 \end{pmatrix}^T \mathbf{x} + 0.$$

The matrix $\mathbf{A} = \begin{pmatrix} 1 & 0 \\ 0 & -2 \end{pmatrix}$ has eigenvalues $1, -2$. Since have opposite signs, $\mathbf{A}$ is not positive definite, and by a lemma in lecture notes, f is not coercive.

Alternatively, by definition, f is coercive if $f \to \infty$ when $||x|| \to \infty$. However, we note that when $x_2 \to \infty$ while keeping x_1 constant, we have $f \to -\infty$ due to the $-2x_2^2$ term. Therefore, f is not coercive.

ii) We compute the first order partial derivatives of f as

$$\frac{\partial f}{\partial x_1} = 8 + 2x_1, \frac{\partial f}{\partial x_2} = 12 - 4x_2.$$

Setting them to zero gives $x_1 = -4, x_2 = 3$. Therefore, the stationary point of f is $(-4, 3)$.

To classify this point, we compute the Hessian of f. We have

$$\frac{\partial^2 f}{\partial x_1^2} = 2, \frac{\partial^2 f}{\partial x_2^2} = -4, \frac{\partial^2 f}{\partial x_1 \partial x_2} = 0, \frac{\partial^2 f}{\partial x_2 \partial x_1} = 0.$$

The Hessian is therefore given by $\begin{pmatrix} 2 & 0 \\ 0 & -4 \end{pmatrix}$ which has eigenvalues $2, -4$. Since they have opposite signs, the stationary point $(-4, 3)$ is a saddle point.

b) Yes, it is. We first notice that the matrix $\mathbf{A}$ can be rewritten as $\mathbf{A} = \operatorname{diag}(\mathbf{d}) - \mathbf{d}\mathbf{d}^T$.

Therefore, for every $\mathbf{x} \in \mathbb{R}^n$, we have $\mathbf{x}^T \mathbf{A} \mathbf{x} = \mathbf{x}^T \operatorname{diag}(\mathbf{d})\mathbf{x} - \mathbf{x}^T \mathbf{d}\mathbf{d}^T \mathbf{x} = \mathbf{x}^T \operatorname{diag}(\mathbf{d})\mathbf{x} - \left(\mathbf{x}^T \mathbf{d}\right)^2$.

Multiplying out these two terms, we get

$$\mathbf{x}^T \operatorname{diag}(\mathbf{d})\mathbf{x} = \sum_{i=1}^{n} d_i x_i^2, \left(\mathbf{x}^T \mathbf{d}\right)^2 = \left(\sum_{i=1}^{n} d_i x_i\right)^2.$$

Since $\mathbf{d} \in \Delta_n$, we have $d_i \geq 0$ and $\sum_{i=1}^{n} d_i = 1$. Therefore, $d_i x_i^2 \geq 0$ and we could apply the Cauchy-Schwarz inequality to get

$$\left(\sum_{i=1}^{n} d_i x_i\right)^2 \leq \left(\sum_{i=1}^{n} d_i\right)\left(\sum_{i=1}^{n} d_i x_i^2\right) = \sum_{i=1}^{n} d_i x_i^2.$$

We then have $\mathbf{x}^T \operatorname{diag}(\mathbf{d})\mathbf{x} \geq \left(\mathbf{x}^T\mathbf{d}\right)^2$ and therefore $\mathbf{x}^T\mathbf{A}\mathbf{x} \geq 0$. Thus $\mathbf{A}$ is positive semidefinite by definition.

2 Variational Signal Denoising

a) The smooth version problem is given by

$$\min_{\mathbf{x}} \frac{1}{2}||\mathbf{x}-\mathbf{f}||_2^2 + \frac{\omega}{2}||\mathbf{L}\mathbf{x}||_2^2.$$

We can write it in matrix form as

$$\min_{\mathbf{x}} \frac{1}{2}(\mathbf{x}-\mathbf{f})^T(\mathbf{x}-\mathbf{f}) + \frac{\omega}{2}(\mathbf{L}\mathbf{x})^T(\mathbf{L}\mathbf{x}).$$

To solve the problem, we take the gradient with respect to $\mathbf{x}$ and set it to zero, giving

$$\mathbf{x}-\mathbf{f}+\omega\mathbf{L}^T\mathbf{L}\mathbf{x} = 0,$$

which is equivalent to

$$(\mathbf{I}+\omega\mathbf{L}^T\mathbf{L})\mathbf{x} = \mathbf{f}.$$

We can then solve this equation and find the denoised signal $\mathbf{x}$ required.

To evaluate the problem numerically, we used Python. The relevant code is attached in Appendix I. The plot generated is attached below. It shows the denoised signals for different ω values together with the original noisy signal.

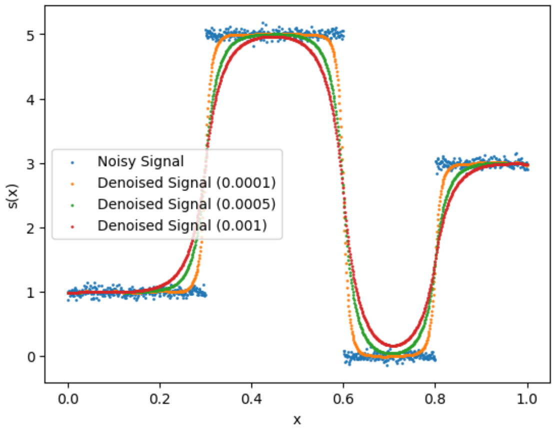

Figure 1: Smooth version problem for $\omega \in \{10^{-4}, 5\cdot 10^{-4}, 10^{-3}\}$.

b) We now consider the non-smooth problem.

i) We have

$$h_\gamma(t) = \begin{cases} \frac{\gamma t^2}{2} & \text{if } |t| \leq \frac{1}{\gamma}, \\ |t| - \frac{1}{2}\gamma & \text{otherwise.} \end{cases}$$

The derivative is evaluated piecewise as

$$h'_\gamma(t) = \begin{cases} \gamma t & \text{if } |t| \le \frac{1}{\gamma}, \\ 1 & \text{if } t > \frac{1}{\gamma}, \\ -1 & \text{if } t < -\frac{1}{\gamma}. \end{cases}$$

When $t > \frac{1}{\gamma}$, $t < -\frac{1}{\gamma}$ or $\frac{1}{\gamma} < t < -\frac{1}{\gamma}$, $h'_\gamma(t)$ is obviously continuous. As $t \to \left(\frac{1}{\gamma}\right)^+$, we have $h'_\gamma(t) = 1$; as $t \to \left(\frac{1}{\gamma}\right)^-$, we have $h'_\gamma(t) = \gamma t = 1$. They are equal, so $h'_\gamma(t)$ is continuous at $t = \frac{1}{\gamma}$. The $t = -\frac{1}{\gamma}$ case is similar. We conclude that $h'_\gamma(t)$ is indeed continuous, and we have

$$\nabla ||\mathbf{u}||_{1,\gamma,\bar{\omega}} = \sum \bar{\omega}_i h'_\gamma(u_i),$$

where $h'_\gamma(u_i)$ is calculated as above.
The non-smooth problem is given as

$$\left(\mathbf{x}^{k+1}, \mathbf{y}^{k+1}\right) = \arg\min_{\mathbf{x},\mathbf{y}} \frac{1}{2}\|\mathbf{x} - \mathbf{f}\|_2^2 + \frac{\lambda}{2}\|\mathbf{y} - \mathbf{L}\mathbf{x} - \mathbf{z}\|_2^2 + \omega\|\mathbf{y}\|_1,$$

and the required form is

$$\min_{\mathbf{u}} \frac{1}{2}\|\bar{\mathbf{A}}\mathbf{u} - \mathbf{f}\|_2^2 + \frac{\lambda}{2}\|\bar{\mathbf{L}}\mathbf{u} - \mathbf{z}\|_2^2 + \|\mathbf{u}\|_{1,\gamma,\bar{\omega}}.$$

By taking $\mathbf{u} = \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}$ and examining both problems, we have

$$\begin{cases} \bar{\mathbf{A}}\mathbf{u} = \mathbf{x}, \\ \bar{\mathbf{L}}\mathbf{u} = \mathbf{y} - \mathbf{L}\mathbf{x}, \\ \omega\|\mathbf{y}\|_1 = \|\mathbf{u}\|_{1,\gamma,\bar{\omega}}. \end{cases}$$

Again by comparison, we conclude that (where $n = 999$)

$$\begin{cases} \bar{\mathbf{A}} = \begin{pmatrix} \mathbf{I}_{n+1} & \mathbf{O}_{(n+1)\times n} \end{pmatrix}, \\ \bar{\mathbf{L}} = \begin{pmatrix} -\mathbf{L} & \mathbf{1}_n \end{pmatrix}, \\ \bar{\omega} = \begin{pmatrix} \mathbf{0}_{n+1} \\ \omega\mathbf{1}_n \end{pmatrix}, \end{cases}$$

where $\mathbf{I}_n$ is the $n \times n$ identity matrix, $\mathbf{O}_{n\times m}$ is the $n \times m$ zero matrix, $\mathbf{0}_n$ is the n-dimensional vector with zeroes and $\mathbf{1}_n$ is the n-dimensional vector with ones.
To solve the problem, we then write

$$\min_{\mathbf{u}} \frac{1}{2}\|\bar{\mathbf{A}}\mathbf{u} - \mathbf{f}\|_2^2 + \frac{\lambda}{2}\|\bar{\mathbf{L}}\mathbf{u} - \mathbf{z}\|_2^2 + \|\mathbf{u}\|_{1,\gamma,\bar{\omega}}$$

as

$$\min_{\mathbf{u}} \frac{1}{2}\left(\bar{\mathbf{A}}\mathbf{u} - \mathbf{f}\right)^T \left(\bar{\mathbf{A}}\mathbf{u} - \mathbf{f}\right) + \frac{\lambda}{2}\left(\bar{\mathbf{L}}\mathbf{u} - \mathbf{z}\right)^T \left(\bar{\mathbf{L}}\mathbf{u} - \mathbf{z}\right) + \sum \bar{\omega}_i h_\gamma(u_i).$$

Taking gradient with respect to $\mathbf{u}$ and setting the expression to zero gives

$$\bar{\mathbf{A}}^T \left(\bar{\mathbf{A}}\mathbf{u} - \mathbf{f}\right) + \lambda\bar{\mathbf{L}}^T \left(\bar{\mathbf{L}}\mathbf{u} - \mathbf{z}\right) + \sum \bar{\omega}_i h'_\gamma(u_i) = 0.$$

We now implement the algorithm in Python. The code is given in Appendix II. *Note that this assumes code in Appendix I is already run.* By running the code, we see that we performed 1164 iterations with execution time of around 1.25 seconds. We also attach the plot below.

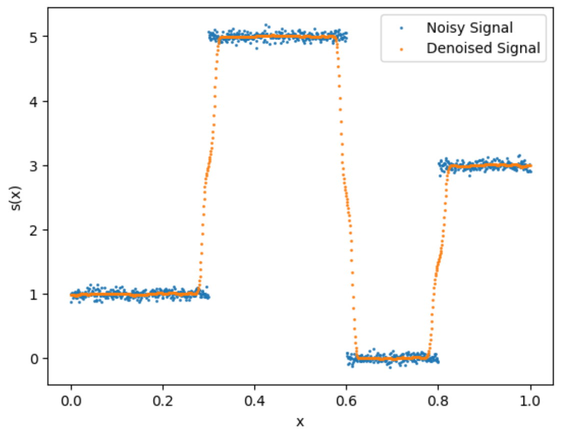

Figure 2: Non-smooth version problem.

ii) The smooth problem

$$\mathbf{x}^{k+1} = \arg\min_{\mathbf{x}} \frac{1}{2}\|\mathbf{x}-\mathbf{f}\|_2^2 + \frac{\lambda}{2}\|\mathbf{y}^k - \mathbf{L}\mathbf{x} - \mathbf{z}^k\|_2^2$$

is equivalent to

$$\mathbf{x}^{k+1} = \arg\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T\mathbf{x} - \mathbf{f}^T\mathbf{x} + \frac{\lambda}{2}\mathbf{x}^T\mathbf{L}^T\mathbf{L}\mathbf{x} - \lambda(\mathbf{y}^k)^T\mathbf{L}\mathbf{x} - \lambda(\mathbf{z}^k)^T\mathbf{L}\mathbf{x},$$

which, by taking gradient with respect to $\mathbf{x}$ and setting to zero, gives

$$\mathbf{x} - \mathbf{f} + \lambda\mathbf{L}^T\mathbf{L}\mathbf{x} - \lambda\mathbf{L}^T(\mathbf{y}^k - \mathbf{z}^k) = 0.$$

This can be rewritten as

$$(\mathbf{I} + \lambda\mathbf{L}^T\mathbf{L})\mathbf{x}^{k+1} = \mathbf{f} + \lambda\mathbf{L}^T(\mathbf{y}^k - \mathbf{z}^k).$$

We now solve the problem using Python. The code is given in Appendix III. *Note that this assumes code in Appendix I & II is already run. However, apparently, the results do not seem correct and there is probably some mistake in my code. I will still present them as if they were right, though.* By running the code, we see that we performed 29 iterations with execution time of around 1.33 seconds. We also attach the plot below.

iii) We have three methods as implemented in a), b)i) and b)ii).

For a), we used smooth approximation. This worked fine when ω is small. When ω is large, however, the resulting signal deviates significantly from the original signal.

For b)i), we used non-smooth technique. This worked better than a) but used too much iterations. When faced with a larger problem, this is probably not the optimal algorithm in terms of efficiency. We also notice that the signal was denoised poorly around the 'edges' or the step changes.

For b)ii), we combined smooth and non-smooth algorithms and further utilised recursion. Our current plot indicates that the signal is not adequately denoised. Had our code worked properly, we would expect that this produces better results than both a) and b)i).

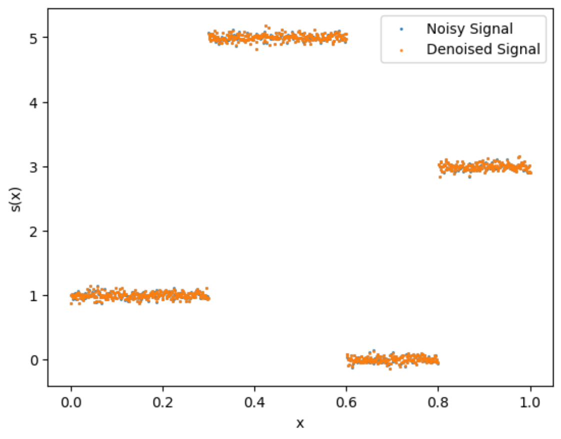

Figure 3: Combining smooth and non-smooth version problem.

3 Gradient Descent (Option A)

We are given the function $g : \mathbb{R}^3 \to \mathbb{R}$ such that

$$g(\mathbf{x}) = \frac{1}{4}\sum_{i=1}^{2} \cos(x_i - x_{i+1}) + \sum_{i=1}^{3} i x_i^2.$$

This can be written explicitly as

$$g(x_1, x_2, x_3) = \frac{1}{4}\left(\cos(x_1 - x_2) + \cos(x_2 - x_3)\right) + x_1^2 + 2x_2^2 + 3x_3^2.$$

i) We first calculate ∇g. We have

$$\frac{\partial g}{\partial x_1} = -\frac{1}{4}\sin(x_1 - x_2) + 2x_1,$$
$$\frac{\partial g}{\partial x_2} = \frac{1}{4}\left(\sin(x_1 - x_2) - \sin(x_2 - x_3)\right) + 4x_2,$$
$$\frac{\partial g}{\partial x_3} = \frac{1}{4}\sin(x_2 - x_3) + 6x_3.$$

So the gradient

$$\nabla g = \begin{pmatrix} -\frac{1}{4}\sin(x_1 - x_2) + 2x_1 \\ \frac{1}{4}\left(\sin(x_1 - x_2) - \sin(x_2 - x_3)\right) + 4x_2 \\ \frac{1}{4}\sin(x_2 - x_3) + 6x_3 \end{pmatrix}.$$

We can then compute the Hessian $\nabla^2 g$ as

$$\nabla^2 g = \begin{pmatrix} -\frac{1}{4}\cos(x_1 - x_2) + 2 & \frac{1}{4}\cos(x_1 - x_2) & 0 \\ \frac{1}{4}\cos(x_1 - x_2) & -\frac{1}{4}\left(\cos(x_1 - x_2) + \cos(x_2 - x_3)\right) + 4 & \frac{1}{4}\cos(x_2 - x_3) \\ 0 & \frac{1}{4}\cos(x_2 - x_3) & -\frac{1}{4}\cos(x_2 - x_3) + 6 \end{pmatrix}.$$

We can clearly see that g is bounded below since

$$g(\mathbf{x}) \geq \frac{1}{4}((-1) + (-1)) + 0 + 0 + 0 = -\frac{1}{2}.$$

From lecture, we know that for the gradient method to find $\min g$ to converge, we need to have a constant stepsize $\bar{t} \in \left(0, \frac{2}{L}\right)$ where L is the Lipschitz constant of g. We also know that L needs to satisfy $||\nabla^2 g|| \leq L$. From the hint given, we have that $||\nabla^2 g|| \leq ||\nabla^2 g||_F$. We can therefore choose L to be $||\nabla^2 g||_F$. Hence, we can choose a stepsize

$$\bar{t} = \frac{1}{L} = \frac{1}{||\nabla^2 g||_F}$$

since the Frobenius norm $||\nabla^2 g||_F$ is positive.

We can calculate the Frobenius norm as

$$||\nabla^2 g||_F = \sqrt{\sum_{i=1}^{3}\sum_{j=1}^{3} a_{ij}^2}$$

where

$$a_{11}^2 = \left(-\frac{1}{4}\cos(x_1 - x_2) + 2\right)^2 \leq \left(\frac{1}{4} + 2\right)^2 = \frac{81}{16},$$

$$a_{22}^2 = \left(-\frac{1}{4}\left(\cos(x_1 - x_2) + \cos(x_2 - x_3)\right) + 4\right)^2 \leq \left(\frac{1}{4}(1+1) + 4\right)^2 = \frac{81}{4},$$

$$a_{33}^2 = \left(-\frac{1}{4}\cos(x_2 - x_3) + 6\right)^2 \leq \left(\frac{1}{4} + 6\right)^2 = \frac{625}{16},$$

$$a_{12}^2 = a_{21}^2 = \left(\frac{1}{4}\cos(x_1 - x_2)\right)^2 \leq \left(\frac{1}{4}\right)^2 = \frac{1}{16},$$

$$a_{13}^2 = a_{31}^2 = 0^2 = 0,$$

$$a_{23}^2 = a_{32}^2 = \left(\frac{1}{4}\cos(x_2 - x_3)\right)^2 \leq \left(\frac{1}{4}\right)^2 = \frac{1}{16}.$$

So we have

$$||\nabla^2 g||_F = \sqrt{\frac{81}{16} + \frac{81}{4} + \frac{625}{16} + 2 \times \frac{1}{16} + 2 \times 0 + 2 \times \frac{1}{16}} = \frac{\sqrt{1034}}{4},$$

and we can therefore take the stepsize to be

$$\bar{t} = \frac{1}{||\nabla^2 g||_F} = \frac{4}{\sqrt{1034}} \left(= \frac{2\sqrt{1034}}{517}\right)$$

so that the gradient method is guaranteed to converge.

ii) To find the stationary point of g, we set $\nabla g = 0$ and we get

$$\begin{cases} -\frac{1}{4}\sin(x_1 - x_2) + 2x_1 = 0 \\ \frac{1}{4}\left(\sin(x_1 - x_2) - \sin(x_2 - x_3)\right) + 4x_2 = 0 \\ \frac{1}{4}\sin(x_2 - x_3) + 6x_3 = 0 \end{cases}$$

which has a solution $(x_1, x_2, x_3) = (0, 0, 0)$. This is the stationary point required.

To determine the nature of this stationary point, we compute the Hessian at this point as

$$\nabla^2 g(0,0,0) = \begin{pmatrix} -\frac{1}{4}\cos(0-0)+2 & \frac{1}{4}\cos(0-0) & 0 \\ \frac{1}{4}\cos(0-0) & -\frac{1}{4}\left(\cos(0-0)+\cos(0-0)\right)+4 & \frac{1}{4}\cos(0-0) \\ 0 & \frac{1}{4}\cos(0-0) & -\frac{1}{4}\cos(0-0)+6 \end{pmatrix}$$

$$= \begin{pmatrix} 7/4 & 1/4 & 0 \\ 1/4 & 3/2 & 1/4 \\ 0 & 1/4 & 23/4 \end{pmatrix}.$$

This matrix is strictly diagonally dominant since all the diagonal elements are strictly greater than the corresponding non-diagonal elements. Since it is also symmetric with positive diagonal elements, we know from lecture notes that it is positive definite. As a result, the point $(0, 0, 0)$ is a strict local minimum.

Appendix I

```
import pandas as pd
import numpy as np
from scipy.linalg import solve
import matplotlib.pyplot as plt
f = pd.read_csv("noisy_signal.csv", header=None)[0].values
n = len(f)
L = np.eye(n) - np.eye(n, k = 1)
L = L[:-1,:] * 999
solution = {}
for omega in [1e-4, 5e-4, 1e-3]:
    solution[omega] = solve(np.eye(n) + omega * (L.T @ L), f)
x = np.linspace(0, 1, n)
plt.figure()
plt.scatter(x, f, label = "Noisy Signal", s = 1)
for omega, sol in solution.items():
    plt.scatter(x, sol, label = f"Denoised Signal ({omega})", s = 1)
plt.xlabel("x")
plt.ylabel("s(x)")
plt.legend()
plt.show()
```

Appendix II

```
import scipy.sparse as sp
import time
n = 999
lamda = 5e-4
gamma = 1e3
omega = 75
alpha = 0.5
beta = 0.5
data = np.array([np.ones(n) * n, -np.ones(n) * n])
L = sp.diags(data, [0, 1], shape = (n, n + 1))
```

```
A_bar = sp.hstack((sp.eye(n + 1), sp.csr_matrix((n + 1, n))))
L_bar = sp.hstack((-L, sp.eye(n)))
omega_bar = np.concatenate([np.zeros(n + 1), omega * np.ones(n)])
x = f.copy()
y = np.zeros(n)
z = np.ones(n)
def h_gamma(t):
    return np.where(np.abs(t) <= 1 / gamma, 0.5 * gamma * t ** 2,
                    np.abs(t) - 1 / (2 * gamma))
def h_gamma_prime(t):
    return np.where(np.abs(t) <= 1 / gamma, gamma * t, np.sign(t))
def J(u, z):
    J1 = 0.5 * np.linalg.norm(A_bar @ u - f) ** 2
    J2 = 0.5 * lamda * np.linalg.norm(L_bar @ u - z) ** 2
    return J1 + J2 + np.dot(omega_bar, h_gamma(u))
def J_grad(u, z):
    J1_grad = A_bar.T @ (A_bar @ u - f)
    J2_grad = lamda * (L_bar.T @ (L_bar @ u - z))
    return J1_grad + J2_grad + omega_bar * h_gamma_prime(u)
outer_tolerance = 1e-5
inner_tolerance = 1e-4
iteration = 0
start_time = time.time()
while True:
    u = np.concatenate([x, y])
    for k in range(500):
        d = -J_grad(u, z)
        t = 1e-2
        while J(u, z) - J(u + t * d, z) < -alpha * t * np.dot(J_grad(u, z), d):
            t = t * beta
        u_new = u + d * t
        if np.linalg.norm(u_new - u) / np.linalg.norm(u_new) < inner_tolerance:
            iteration += k + 1
            break
        u = u_new
    x_new = u[:n + 1]
    y_new = u[n + 1:]
    z += L @ x_new - y_new
    if np.linalg.norm(x_new - x) / np.linalg.norm(x_new) < outer_tolerance:
        break
    x = x_new
    y = y_new
print(iteration, time.time() - start_time)
xx = np.linspace(0, 1, n + 1)
plt.figure()
plt.scatter(xx, f, label = 'Noisy Signal', s = 1)
plt.scatter(xx, x, label = 'Denoised Signal', s = 1)
plt.legend()
plt.xlabel('x')
plt.ylabel('s(x)')
plt.show()
```

Appendix III

```
lamda = 1e-2
omega = 2e-3
x = f.copy()
L = np.eye(n + 1, k = 0) - np.eye(n + 1, k = 1)
y = np.dot(L, x)
z = np.ones(n + 1)
def S(t, eta):
    return np.sign(t) * np.maximum(np.abs(t) - eta, 0)
def solve(y, z, f, lamda, L):
    A = np.eye(n + 1) + lamda * np.dot(L.T, L)
    b = f + lamda * np.dot(L.T, y - z)
    return np.linalg.solve(A, b)
iteration = 0
start_time = time.time()
while True:
    x_new = solve(y, z, f, lamda, L)
    y_new = S(np.dot(L, x_new) + z, omega / lamda)
    z += np.dot(L, x_new) - y_new
    if np.linalg.norm(x_new - x) / np.linalg.norm(x_new) < outer_tolerance:
        break
    x = x_new
    y = y_new
    iteration += 1
print(iteration, time.time() - start_time)
xx = np.linspace(0, 1, n + 1)
plt.figure()
plt.scatter(xx, f, label = 'Noisy Signal', s = 1)
plt.scatter(xx, x, label = 'Denoised Signal', s = 1)
plt.legend()
plt.xlabel('x')
plt.ylabel('s(x)')
plt.show()
```