IEICE TRANS. COMMUN., VOL.E107–B, NO.1 JANUARY 2024

INVITED PAPER

Introduction to Compressed Sensing with Python

Masaaki NAGAHARA†a), ***Member***

SUMMARY Compressed sensing is a rapidly growing research field in signal and image processing, machine learning, statistics, and systems control. In this survey paper, we provide a review of the theoretical foundations of compressed sensing and present state-of-the-art algorithms for solving the corresponding optimization problems. Additionally, we discuss several practical applications of compressed sensing, such as group testing, sparse system identification, and sparse feedback gain design, and demonstrate their effectiveness through Python programs. This survey paper aims to contribute to the advancement of compressed sensing research and its practical applications in various scientific disciplines.

key words: *compressed sensing, sparse representation, convex optimization, group testing, system identification, feedback control, Python*

1. Introduction

Compressed sensing is a mathematical framework for reconstructing sparse signals from a limited number of measurements [1], [2]. Namely, compressed sensing attempts to find the *sparsest* vector among vectors satisfying underdetermined linear equations. This formulation appears in many scientific and technological problems in many research fields such as machine learning, signal/image processing, and statistics. One example is audio signal reconstruction. Audio signals have frequency components only in the low frequencies, and hence they can be treated as sparse signals in the frequency domain. In particular, the human voice can be assumed to be in the frequency range of the so-called telephone bandwidth of 30 Hz–3400 Hz, by which the human voice can be coded at the sampling rate 8000 Hz [3]. Also, a pulse signal, which is active (i.e., non-zero) only in a short duration of time, can be considered as a sparse signal in the time domain. This property is effectively used for the seismic reflection survey in geophysics [4]–[6]. Other practical examples of compressed sensing can be found in many fields such as image processing [7], medical imaging [8], and signal processing [7].

This is formulated as a mathematical optimization problem to find a minimum ℓ^0-norm vector among the feasible solutions. Since this problem is NP-hard [9], many heuristic methods have been proposed. An effective method to solve the compressed sensing problem is to approximate the ℓ^0 norm by the ℓ^1 norm, which makes the problem convex. Then we can easily solve the ℓ^1 optimization with linear or convex constraints by using numerical optimization software such as CVX [10] on MATLAB and CVXPY [11] on Python.

More recently, the compressed sensing approach has been extended to *optimal control*. In particular, the papers of [12], [13] have introduced a new type of optimal control called the *maximum hands-off control*, which minimizes the L^0 norm, the Lebesgue measure (i.e., the length) of the support, of a continuous-time control signal, under a control objective with control constraints. Namely, maximum hands-off control has the minimum length of time duration on which the control is active (i.e., non-zero) among all feasible controls. We note that maximum hands-off control has a long period over which the control is exactly zero, which is used in practical control systems, sometimes called *gliding* or *coasting*. An example of hands-off control is a stop-start system in automobiles where the engine is automatically shut down when the car is stationary [14], [15]. Another example can be found in a hybrid vehicle, where the internal combustion engine is stopped when the vehicle is at a stop or a low speed while the electric motor is alternatively used [16]–[18]. These systems can effectively reduce fuel consumption and CO or CO_2 emissions. Railway vehicles [19], [20] and free-flying robots [21] take advantage of hands-off control as well. By these properties, hands-off control is sometimes called *green control* [22].

Mathematical properties of maximum hands-off control have also been explored recently [23]. Maximum hands-off control is mathematically described as an L^0 optimal control, which is very hard to solve. Borrowing the idea of the ℓ^1-norm heuristic in compressed sensing, L^1 optimal control has been proposed in [13] to solve the L^0 optimal control. The L^1 optimal control, also known as the *minimum fuel control*, has been extensively studied since the 1960s (see, e.g., [24]), and the problem is easily solved. In [13], the equivalence between L^0 and L^1 optimal controls is established under some mild assumptions. Fundamental properties of maximum hands-off control, such as the value function [25] and necessary conditions [26], have been investigated. The maximum hands-off control has also been extended to time-optimal control [27], control in the presence of denial-of-service attacks [28], distributed control [29], [30], continuous control [31], infinite-dimensional systems [32], optimal multiplexing [33], stochastic control [34], [35], minimum attention control [36], and time-space sparse control [37], [38]. Efficient numerical algorithms for

Manuscript received April 30, 2023.
Manuscript revised June 5, 2023.
Manuscript publicized August 15, 2023.

†The author is with the Graduate School of Advanced Science and Engineering, Hiroshima University, Higashi-Hiroshima-shi, 739-8521 Japan.
a) E-mail: nagahara@ieee.org

DOI: 10.1587/transcom.2023EBI0002

maximum hands-off control have been proposed in recent papers of [39]–[41]. Discrete-time hands-off control is also essential in digital control systems, on which a recent line of research has focused [42]–[45]. Finally, practical applications of maximum hands-off control have been reported for electrically tunable lens [46], spacecraft maneuvering [47], thermally activated building systems [48], and quadrotor groups [49].

The organization of this paper is as follows: Section 2 gives mathematical formulations of compressed sensing. Section 3 shows the core idea of compressed sensing to solve the optimization problem as an ℓ^1 optimization problem. Section 4 discusses the problem of group testing as a compressed sensing problem. Section 5 introduces an application of compressed sensing to dynamical system identification. Section 6 shows sparse feedback gain design using the concept of compressed sensing. Finally, Sect. 7 makes the conclusion.

Notation

Let x be a vector. The ℓ^p norm $\|x\|_p$ with $p \in (0, \infty)$ is defined by

$$\|x\|_p \triangleq \left(\sum_i |x_i|^p \right)^{1/p}, \tag{1}$$

where x_i is the i-th element of x. In particular, the ℓ^1 norm and the ℓ^2 norm are important in this article:

$$\|x\|_1 \triangleq \sum_i |x_i|, \quad \|x\|_2 \triangleq \sqrt{\sum_i |x_i|^2} = \sqrt{\langle x, x \rangle}, \tag{2}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product. For $p = 0$, the ℓ^0 norm is defined by

$$\|x\|_0 \triangleq \#\bigl(\mathrm{supp}(x)\bigr), \tag{3}$$

where $\mathrm{supp}(x)$ is the support set of x, that is,

$$\mathrm{supp}(x) \triangleq \bigl\{ i : x_i \neq 0 \bigr\}, \tag{4}$$

and $\#\bigl(\mathrm{supp}(x)\bigr)$ is the number of elements in the finite set $\mathrm{supp}(x)$.

Let A be a matrix. The transpose of A is denoted by $A^\top$, and the rank by $\mathrm{rank}(A)$. The i-th largest singular value of A is denoted by $\sigma_i(A)$, and the maximum singular value by $\sigma_{\max}(A)$. The mutual coherence of A is denoted by $\mu(A)$.

For a closed subset C of a normed space $\mathcal{S}$ with norm $\|\cdot\|$, the projection of $x \in \mathcal{S}$ onto C is denoted by $\Pi_C(x)$, that is,

$$\Pi_C(x) \in \operatorname*{arg\,min}_{z \in C} \|z - x\|. \tag{5}$$

For a real number x, $\mathrm{sign}(x)$ is the sign of x, namely,

$$\mathrm{sign}(x) \triangleq \begin{cases} 1, & \text{if } x > 0, \\ -1, & \text{if } x < 0, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

Python codes

The Python codes in this survey paper can be downloaded from the following web page:

`github.com/nagahara-masaaki/IEICECS2023`

2. Mathematical Formulation of Compressed Sensing

In this section, we briefly review the basics of compressed sensing.

2.1 The ℓ^0 Norm

First, we consider the ℓ^0 norm of a vector in a finite-dimensional vector space, which plays an important role in compressed sensing. The ℓ^0 norm $\|x\|_0$ of $x = [x_1, \dots, x_n]^\top \in \mathbb{R}^n$ is defined in (3), and it counts *the number of nonzero elements* in x. We should note that $\|x\|_0$ is actually not a proper norm since it does not necessarily satisfy the positive homogeneity property. For example, a vector $x \in \mathbb{R}^n$ has the same ℓ^0 norm as $2x$, and hence

$$\|2x\|_0 = \|x\|_0 \neq 2\|x\|_0, \tag{7}$$

whenever $x \neq 0$. We say $x \in \mathbb{R}^n$ is *sparse* if $\|x\|_0 \ll n$. Also, if a vector $x \in \mathbb{R}^n$ satisfies $\|x\|_0 \leq s$ with $s \in \mathbb{N}$, then it is called *s-sparse*, and we denote the set of all s-sparse vectors in $\mathbb{R}^n$ by $\tilde{\Sigma}_s$. Namely,

$$\tilde{\Sigma}_s \triangleq \bigl\{ x \in \mathbb{R}^n : \|x\|_0 \leq s \bigr\}. \tag{8}$$

2.2 Signal Reconstruction Problem

In compressed sensing, we consider the system of linear equations

$$Ax = b, \tag{9}$$

where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ are given. We call A the measurement matrix, and b the observation vector. In particular, we assume $m < n$ and $\mathrm{rank}(A) = m$ (i.e., A has full row rank). That is, we consider a signal reconstruction problem where the number m of the observations in b is less than the length n of the unknown vector x. In this case, there are infinitely many solutions for (9), and we need to uniquely identify x with some additional information. In compressed sensing, we assume that x is known to be sparse (i.e., $\|x\|_0 \ll n$ holds), and solve the following optimization problem:

$$\underset{x}{\text{minimize}} \; \|x\|_0 \quad \text{subject to} \quad Ax = b. \tag{10}$$

We call this the ℓ^0 *optimization*.

We first discuss the uniqueness of the solution of (10). For this, we introduce the *mutual coherence* of a matrix. For a matrix $A = [a_1, a_2, \dots, a_n] \in \mathbb{R}^{m \times n}$ with column vectors $a_i \in \mathbb{R}^m, i = 1, 2, \dots, n$, the mutual coherence $\mu(A)$ is defined by

$$\mu(A) \triangleq \max_{\substack{i,j=1,\dots,n\\ i\neq j}} \frac{|\langle a_i, a_j\rangle|}{\|a_i\|_2\|a_j\|_2}, \tag{11}$$

where $\langle a_i, a_j\rangle$ is the inner product of a_i and a_j, that is, $\langle a_i, a_j\rangle = a_i^\top a_j$ and $\|a_i\|_2 = \sqrt{\langle a_i, a_i\rangle}$. Then, we have the following uniqueness theorem [50], [51]:

Theorem 1: If the linear Eq. (9) has a solution x satisfying

$$\|x\|_0 < \frac{1}{2}\left(1 + \frac{1}{\mu(A)}\right), \tag{12}$$

then x is the unique solution of (10). □

It is easily checked whether a *given* solution x of (9) satisfies (12) or not. However, to seek a solution x that satisfies (12) is very hard, and actually it is NP hard [9]. Hence, for large-scale problems, we need to employ a heuristic method to efficiently solve the ℓ^0 optimization (10).

3. The ℓ^1-Norm Heuristic

A widely used method is to adopt the ℓ^1 norm

$$\|x\|_1 \triangleq \sum_{i=1}^{n} |x_i|, \tag{13}$$

as a surrogate of the ℓ^0 norm. Namely, we solve the following optimization problem instead:

$$\underset{x}{\text{minimize}}\ \|x\|_1 \ \text{ subject to }\ Ax = b. \tag{14}$$

This is a convex optimization problem, and efficiently solved by, e.g., CVX [10] on MATLAB and CVXPY [11] on Python.

We then discuss the relation between the ℓ^0 optimization (10) and the ℓ^1 optimization (14). For this, we define the *restricted isometry property* (RIP) [1, Section 1.4.2] of matrix A:

Definition 1: A matrix A is said to satisfy the restricted isometry property of order $s \in \mathbb{N}$ if there exists $\delta_s \in (0,1)$ such that

$$(1-\delta_s)\|x\|_2^2 \le \|Ax\|_2^2 \le (1+\delta_s)\|x\|_2^2, \tag{15}$$

holds for any $x \in \tilde{\Sigma}_s$, the set of s-sparse vectors defined in (8).

Then, we have the following theorem [1, Section 1.5.1]:

Theorem 2: Suppose that matrix A satisfies the RIP of order $2s$ with $\delta_{2s} < \sqrt{2}-1$. Suppose also that there exists $x^* \in \tilde{\Sigma}_s$ that satisfies $Ax^* = b$. Then, the solution $\hat{x}$ of the ℓ^1 optimization (14) is equivalent to x^*.

We should note that to check if the condition in Theorem 2 is satisfied or not is known to be NP hard [52]. Hence, in practical applications, we usually do not check the condition using the RIP before we just solve the ℓ^1 optimization (14).

In the noisy case, where the observation vector b suffers from noise, the reconstruction problem is formulated as regularization:

$$\underset{x}{\text{minimize}}\ \|Ax-b\|_2^2 + \lambda\|x\|_0, \tag{16}$$

where $\lambda > 0$ is the regularization parameter that should be chosen appropriately. The optimization problem (16) is called the ℓ^0 *regularization*. This problem is also hard to solve, and is relaxed to the following convex optimization problem using the ℓ^1 heuristic approach:

$$\underset{x}{\text{minimize}}\ \|Ax-b\|_2^2 + \lambda\|x\|_1. \tag{17}$$

This is called the ℓ^1 *regularization*, or *LASSO* (least absolute shrinkage and selection operator).

3.1 Numerical Optimization with CVXPY

The optimization problems with the ℓ^1 norm are convex optimization and we can solve them easily using efficient programming packages. In particular, `CVXPY` [11] in Python is suitable. `CVXPY` is a Python-embedded modeling language for convex optimization problems. It provides a simple and intuitive syntax for formulating and solving convex optimization problems including linear programming, quadratic programming, second-order cone programming, semi-definite programming, and many others. We here use `CVXPY` to solve the ℓ^1 optimization problem (14) with specific measurement matrix A and observation vector b.

To use `CVXPY` we need to install the package. This is done for example by

```
!pip install cvxpy
```

Then, we define the measurement matrix A in (14) as a random matrix. The Python code is given as follows:

```python
# import packages
import numpy as np
import cvxpy as cp
# random seed
np.random.seed(1)
# Problem size
n = 1000
m = 100
# random measurement matrix
A = np.random.randn(m, n)
```

We first import two packages of `numpy` and `cvxpy`. The package `numpy` is for numerical computation with vectors and matrices. The third command `np.random.seed(1)` sets the seed of the random number generator. Then, we set $n = 1000$ and $m = 100$, and obtain a random matrix $A \in \mathbb{R}^{100\times1000}$ whose elements are drawn from the normal distribution with mean 0 and variance one.

We then set the original vector x_{orig} to be estimated. We assume $x_{\text{orig}} \in \mathbb{R}^{1000}$ is a 10-sparse vector (i.e., $\|x_{\text{orig}}\|_0 = 10$) whose nonzero elements are 1. This is done by the following Python code:

```
# sparse vector (n-dimensional, s-
    sparse)
s = 10
x_orig = np.zeros(n)
S = np.random.randint(n,size=s)
x_orig[S] = 1
```

Using A and x_{orig} above, we compute the observation vector $b \in \mathbb{R}^{100}$ as:

```
# observation vector
b = A @ x_orig
```

Now, we solve the optimization problem (14) by CVXPY. The Python code is given as follows:

```
# optimization by cvxpy
x = cp.Variable(n)
objective = cp.Minimize(cp.norm(x, 1)
    )
constraints = [A @ x == b]
prob = cp.Problem(objective,
    constraints)
result = prob.solve()
```

The first command defines the n-dimensional optimization variable `x` by `cp.Variable(n)`. The second command defines the cost function $\|x\|_1$ by `cp.Minimize(cp.norm(x,1))` where $cp.norm(x, 1)$ means the ℓ^1 norm of x. The third command defines the equality constraint $Ax = b$, where `A @ x` is the multiplication Ax. The fourth command defines the optimization problem with the cost function and the constraint. Finally, the fifth command solves the optimization problem.

To see the result, we show the original vector x_{orig} and the reconstructed vector x in Fig. 1. The figure is obtained by the following Python code:

```
# reconstructed vector
fig = plt.figure()
ax1 = fig.add_subplot(1, 2, 1)
ax1.stem(x_orig)
ax2 = fig.add_subplot(1, 2, 2)
ax2.stem(x.value)
```

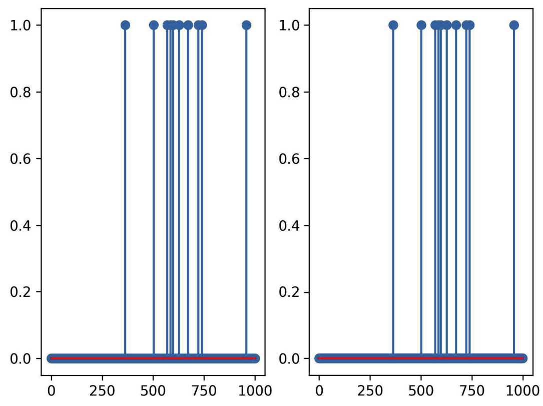

Fig. 1 The original vector (left) and the reconstructed vector (right).

We can see the reconstruction is almost exact. In fact, the reconstruction error $\|x_{\text{orig}} - x\|_2$ is computed as 3.305×10^{-10} by the Python code:

```
error = np.linalg.norm(x_orig-x.value)
```

3.2 Fast Algorithms

The Python package CVXPY is very useful for numerical optimization. However, for very large-scale problems or real-time applications like feedback control, more fast algorithms that can be easily and compactly implemented are preferable. Here we show fast algorithms to numerically solve convex optimization problems (14) and (17). First, we define an important function called the *soft-thresholding function*. Let $v = [v_1, \dots, v_n]^\top \in \mathbb{R}^n$. Then the soft-thresholding function $S_\gamma : \mathbb{R}^n \to \mathbb{R}^n$ with positive parameter γ is defined by

$$[S_\gamma(v)]_i \triangleq \begin{cases} v_i - \gamma, & \text{if } v_i \geq \gamma, \\ 0, & \text{if } -\gamma < v_i < \gamma, \\ v_i + \gamma, & \text{if } v_i \leq -\gamma, \end{cases} \tag{18}$$

for $i = 1, 2, \dots, n$, where $[S_\gamma(v)]_i$ is the i-th entry of $S_\gamma(v) \in \mathbb{R}^n$. Figure 2 shows the graph of this function.

Then, the ℓ^1 optimization in (14) can be efficiently solved by the Douglas-Rachford splitting algorithm [53] given as follows:

$$\begin{aligned} x[k+1] &= S_\gamma(z[k]), \\ z[k+1] &= z[k] + \Pi_C(2x[k+1] - z[k]) - x[k+1], \\ &\quad k = 0, 1, 2, \dots, \end{aligned} \tag{19}$$

with initial vector $z[0] \in \mathbb{R}^n$, where $\gamma > 0$ is the step size and Π_C is the projection onto the constraint set

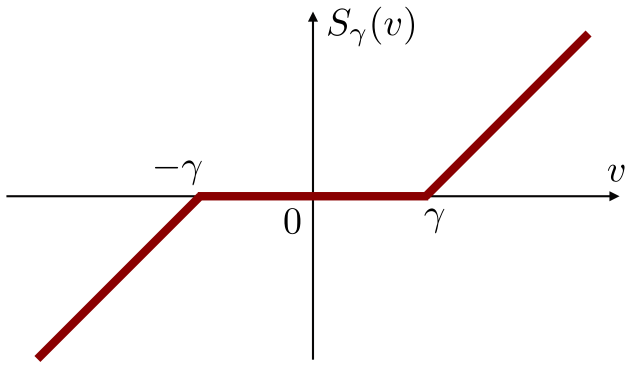

Fig. 2 Soft-thresholding function.

$$C \triangleq \{x \in \mathbb{R}^n : Ax = b\}, \tag{20}$$

for (9), which is given by

$$\Pi_C(v) = v + A^\top (AA^\top)^{-1}(b - Av). \tag{21}$$

Note that $AA^\top$ is invertible if A has full row rank. The convergence result is given as follows [53]:

Theorem 3: For any $\gamma > 0$ and any $z[0] \in \mathbb{R}^n$, the sequence $\{x[k]\}_{k=0}^{\infty}$ generated by the Douglas-Rachford splitting algorithm (19) converges to a solution of the optimization problem (14). □

For the LASSO problem in (17), we have the following algorithm:

$$\begin{aligned} x[k+1] &= S_{\gamma\lambda}\big(x[k] - \gamma A^\top (Ax[k] - b)\big),\\ k &= 0, 1, 2, \ldots, \end{aligned} \tag{22}$$

with initial vector $x[0] \in \mathbb{R}^n$, where $\gamma > 0$ is the step size and $S_{\gamma\lambda}$ is the soft-thresholding function defined in (18). This algorithm is called the *proximal gradient algorithm*, or the *iterative shrinkage thresholding algorithm* (ISTA). For the convergence, the following theorem holds [54]:

Theorem 4: Suppose that the step size γ satisfies

$$0 < \gamma \leq \frac{1}{\sigma_{\max}(A)^2}, \tag{23}$$

where $\sigma_{\max}(A)$ is the maximum singular value of A. Then, for any $x[0] \in \mathbb{R}^n$, the sequence $\{x[k]\}_{k=0}^{\infty}$ generated by the proximal gradient algorithm (22) converges to a solution of the optimization problem (17). □

4. Group Testing

Group testing has been used for a variety of infectious diseases such as HIV, hepatitis B and C, and COVID-19 [55]. This is a technique to find a few infected individuals from a large number, say n, of people with much fewer tests than n. The key idea is to apply the technique of compressed sensing as described below.

Group testing was first proposed by R. Dorfman in 1948 [56]. To explain the method, let us suppose that only one of eight patients is infected with a disease that can be detected by examining the blood. We are given eight blood samples from the eight patients. It easily checks who is infected by testing all the blood samples by eight tests. Then, we want to identify the infected individual by fewer tests than eight, since blood testing is expensive and time-consuming.

For this purpose, we adopt the following strategy (see also Fig. 3):

- **TEST 1:** We first divide the blood samples of the eight individuals into two groups of four individuals, and take a little bit of blood from each of them, and mix them for each group. Since there is only one infected individual, the blood from either group will test positive.

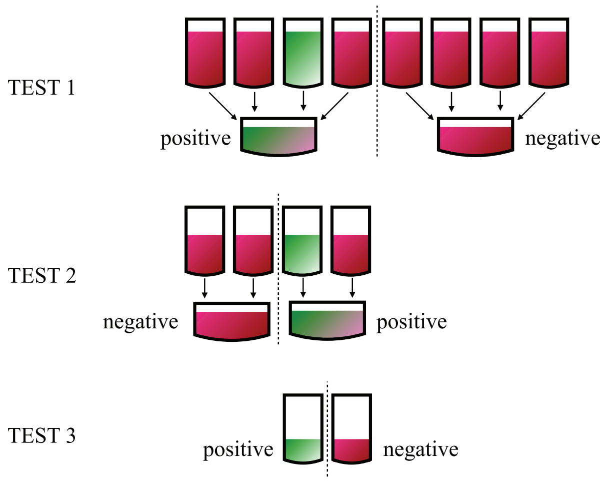

Fig. 3 Group testing from eight blood samples.

- **TEST 2:** Divide the group that tested positive into two groups of two patients, and do the same thing. At this point, the number of suspicious persons has been narrowed down to two.
- **TEST 3:** Finally, by examining the blood of the two individuals separately, the infected person can be uniquely identified.

By this method, we can identify the infected individual in six tests, whereas eight tests would be required for an individual blood test. In general, if there is only one infected individual among 2^T individuals, we can identify the infected one in less than $2T$ tests. For example, for 1024 patients, only 20 tests are needed to identify the infected individual. Therefore, group testing can dramatically reduce the number of tests compared to testing all individuals' blood separately. A question here is how to obtain a sophisticated method like this in a general situation where a few people in 100,000 for example are infected, instead of examining the blood of 100,000 individuals one by one. This is the problem of group testing.

Now we describe the problem of group testing in detail. Let n be the number of people to be tested. Define a variable, x_i $(i \in \{1, 2, \ldots, n\})$ representing whether the i-th individual is infected or not. Namely,

$$x_i \triangleq \begin{cases} 1, & \text{if the } i\text{-th individual is infected,} \\ 0, & \text{otherwise.} \end{cases} \tag{24}$$

Then define an n-dimensional binary vector, called called the *infection vector*, that takes values of 0 or 1 as

$$x \triangleq [x_1, x_2, \ldots, x_n]^\top \in \mathbb{R}^n. \tag{25}$$

Then our problem is to predict this n-dimensional unknown binary vector. Of course, if we examine each one of them individually, we can determine the vector x with n tests, but here we want to identify x with a much smaller number of tests.

Now let us formulate the process of group testing. For this, we first define the *testing vector* a_j, $j = 1, 2, \dots, m$, where m represents the number of tests. To see the role of this vector, we consider the eight blood samples shown in Fig. 3. In this case, the infection vector is given by

$$x = [0, 0, 1, 0, 0, 0, 0, 0]^\top \in \mathbb{R}^n \tag{26}$$

Namely, the third element is active and hence the third individual is infected. Then the first testing vector is given by

$$a_1 = [1, 1, 1, 1, 0, 0, 0, 0]^\top. \tag{27}$$

This means that the first test is applied to the first four samples of the eight samples. Then taking the inner product of a_1 and x, we have the test result, or the *observation*:

$$b_1 \triangleq \langle a_1, x \rangle = 1. \tag{28}$$

This means that there is one infected individual among the first four individuals. Then if the second testing vector is given by

$$a_2 = [0, 0, 0, 0, 1, 1, 1, 1]^\top, \tag{29}$$

The observation $b_2 \triangleq \langle a_2, x \langle = 0$ implies that the last four individuals are all negative.

In general, the observation b_j or the inner product of a_j and x shows the number of infected individuals in the group defined by the testing vector a_j. For simplicity, we assume this number can be obtained by blood testing. Then, we formulate the above process using a matrix and vectors. First, we define the *measurement matrix* A by

$$A = \begin{bmatrix} a_1 & a_2 & \dots & a_m \end{bmatrix}^\top \in \mathbb{R}^{m \times n}, \tag{30}$$

and the *observation vector* y by

$$b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} = \begin{bmatrix} \langle a_1, x \rangle \\ \langle a_2, x \rangle \\ \vdots \\ \langle a_m, x \rangle \end{bmatrix} = \begin{bmatrix} a_1^\top x \\ a_2^\top x \\ \vdots \\ a_m^\top x \end{bmatrix} = \begin{bmatrix} a_1^\top \\ a_2^\top \\ \vdots \\ a_m^\top \end{bmatrix} x = Ax \tag{31}$$

Now, the problem of group testing is to reconstruct the infection vector $x \in \mathbb{R}^n$ from the observation vector $b \in \mathbb{R}^m$ that satisfies the linear Eq. (31). Since the purpose of group testing is to dramatically reduce the number m of tests, this should be much smaller than n, the number of individuals. Therefore, the linear Eq. (31) has infinitely many solutions in general (if solutions exist). This means that we cannot uniquely determine the original vector x only from the observation vector b with Eq. (31). However, if we assume that the number of infected people is much smaller than n, or vector x is *sparse* (i.e., x has very few nonzero elements), then we can formulate the problem of group testing as the following optimization problem:

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \ \|x\|_0 \ \text{ subject to } \ Ax = b, \tag{32}$$

which is the problem of compressed sensing as discussed in the previous sections. Many methods to efficiently solve this problem have been proposed since the Dorfman's paper [56]. For recent methods, you can refer to, for example, [57], [58].

Python Example 1: Here we execute a Python program to see if the problem of group testing is solved by using techniques of compressed sensing.

We solve the problem in (32) by the ℓ^1-norm heuristic mentioned in Section 3. That is, we solve the ℓ^1 optimization (14) instead of (32).

First, we import two packages:

```
import numpy as np
import matplotlib.pyplot as plt
```

The package `matplotlib` is for plotting results. We set the parameters as follows: the number of individuals $n = 1000$, the number of infected individuals $s = 5$, and the number of tests $m = 40$. The infection vector is randomly generated such that the vector contains $s = 5$ ones and $n - s_9 5$ zeros. This is done by the following code:

```
# vector size (number of individuals)
n = 1000
# number of positives
s = 5
# random seed
np.random.seed(1)
# original vector (n-dimensional, s-
  sparse)
x_orig = np.zeros(n)
S = np.random.randint(n,size=s)
x_orig[S] = 1
# number of tests
m = 40
```

Then, we need to design the measurement matrix A. For this, we simply adopt a random 0-1 valued matrix. This means that the groups are formed randomly. Using this A, we also define the observation vector b. The Python code is given as follows:

```
# measurement matrix
A = np.random.randint(2,size=(m, n))
# observation vector
b = A @ x_orig
```

We solve the ℓ^1 optimization in (14) by Douglas-Rachford splitting algorithm in (19). For this, we fist define the soft-thresholding function (18):

```
# Soft-thresholding function
  def St(lmbd, v):
  Sv = np.sign(v) * np.maximum(np.abs(v
    ) - lmbd, 0)
  return Sv
```

We choose the step size as $\gamma = 1$, and we iterate the algorithm 5000 times. The algorithm is implemented as follows:

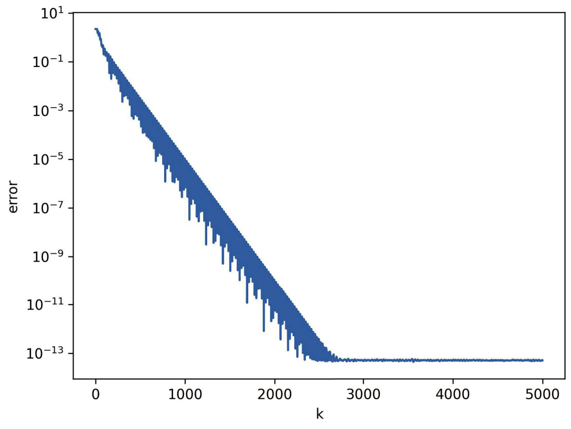

Fig. 4 The error $\|x[k] - x_{\text{orig}}\|$.

```
# Parameter setting
gamma = 1 # step size
max_itr = 5000 # number of iterations
x = np.zeros(n) # initial guess

Ap = np.linalg.pinv(A) # pseudo
    inverse of A
M = np.eye(n) - Ap @ A
v = Ap @ b

error = np.zeros(max_itr) # residual
z = x # variable z
# Iteration
for k in range(max_itr):
  error[k] = np.linalg.norm(x_orig -
     x)
  x = St(gamma,z)
  z = z + M @ (2*x - z) + v - x
```

In each iteration, we compute the error $\|x[k] - x_{\text{orig}}\|$. Figure 4 shows the error.

We can see the error rapidly decreases before it achieves a sufficiently small value. To check the reconstructed result, we show Fig. 5. Note that we round the values in the reconstructed vector to their nearest integers. Then, the reconstruction is perfect.

Finally, we show the Python code to draw these figures:

```
## Error analysis
fig = plt.figure()
plt.semilogy(error)
plt.xlabel("k")
plt.ylabel("error")
## Reconstructed vector
fig = plt.figure()
ax1 = fig.add_subplot(1, 2, 1)
ax1.stem(x_orig,use_line_collection=
    True)
ax2 = fig.add_subplot(1, 2, 2)
ax2.stem(x,use_line_collection=True)
```

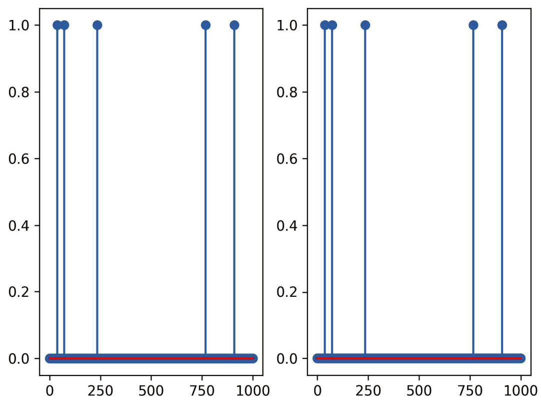

Fig. 5 The original vector (left) and the reconstructed vector (right).

5. System Identification

System identification is the process of building mathematical models of dynamical systems based on input-output data. In system identification, we use the input-output data to estimate the characteristics of a system, such as its impulse response, transfer function, and state-space equation.

The accuracy of system identification depends highly on how a priori information is utilized. The *kernel regularization* is one of the most effective methods to take such information into account, as discussed in [59]. For example, in [60], [61], it was shown that the kernel method can significantly improve the accuracy of system identification of a stable linear system by adopting the exponential convergence of impulse response. More recently, the method has been extended to a priori information on the DC gain [62], the frequency domain decay characteristics [63], [64], and the relative degree [65].

On the other hand, the notion of *sparsity* also plays an important role in system identification [66], [67]. In this section, we introduce system identification of sparse impulse response that has only a few nonzero coefficients.

For this, we consider the FIR (finite impulse response) model described by

$$y_k = \sum_{i=0}^{m-1} g_i u_{k-i} + \epsilon_k, \quad k = 0, 1, 2, \dots, \tag{33}$$

where $\{u_k\}$ and $\{y_k\}$ are respectively the input and output sequences, $\{\epsilon_k\}$ is noise, and $\{g_i\}$ is the impulse response. Figure 6 shows the block diagram of this system, where G is the system to be identified. The problem is to identify the impulse response $\{g_i : i = 0, 1, \dots, m-1\}$ from the input/output data $\{(u_k, y_k) : k = 0, 1, \dots, N\}$. For this, we define the following vectors:

$$y \triangleq \begin{bmatrix} y_0 \\ \vdots \\ y_N \end{bmatrix}, \ \epsilon \triangleq \begin{bmatrix} \epsilon_0 \\ \vdots \\ \epsilon_N \end{bmatrix}, \ g \triangleq \begin{bmatrix} g_0 \\ \vdots \\ g_{m-1} \end{bmatrix}, \tag{34}$$

and the following matrix:

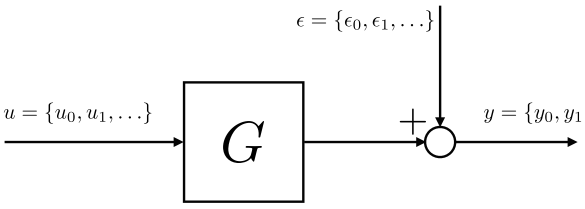
$\epsilon = \{\epsilon_0, \epsilon_1, \dots\}$
$u = \{u_0, u_1, \dots\}$
G
$+$
$y = \{y_0, y_1, \dots\}$

Fig. 6 Block diagram.

$$U \triangleq \begin{bmatrix} u_0 & 0 & \dots & 0 \\ u_1 & u_0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ u_{N-1} & u_{N-2} & \dots & u_{N-m} \end{bmatrix}. \tag{35}$$

Then, we consider the squared ℓ^2 error $E(g)$ as

$$E(g) \triangleq \frac{1}{2}\|y - Ug\|_2^2. \tag{36}$$

The minimizer of $E(g)$ is the least-square solution, which is used when we have no prior information on g. However, it may cause overfitting [68] in particular when the parameter size m is large. To avoid this, we adopt *regularization* with the following regularization term:

$$\Omega(g) \triangleq \frac{\alpha}{2}g^\top Qg + (1-\alpha)\|g\|_1, \tag{37}$$

where $Q = [Q_{ij}]$ is a positive definite matrix, and the second term with the ℓ^1 norm of g is for the sparsity of the impulse response. That is, we minimize the following cost function:

$$\begin{aligned} J(g) &\triangleq E(g) + \lambda\Omega(g) \\ &= \frac{1}{2}\|y - Ug\|_2^2 + \frac{\lambda\alpha}{2}g^\top Qg + \lambda(1-\alpha)\|g\|_1. \end{aligned} \tag{38}$$

This minimization problem can be equivalently transformed into the ℓ^1 regularization (or LASSO) as in (17). To do this, we first define a matrix $\sqrt{\lambda\alpha Q}$ such that

$$\sqrt{\lambda\alpha Q}^\top \sqrt{\lambda\alpha Q} = \lambda\alpha Q. \tag{39}$$

Since Q is positive definite and $\lambda > 0$, $\alpha \geq 0$, there exists $\sqrt{\lambda\alpha Q}$ that satisfies (39). Then, define the following matrix and vector:

$$\widetilde{U} = \begin{bmatrix} U \\ \sqrt{\lambda\alpha Q} \end{bmatrix}, \quad \widetilde{y} = \begin{bmatrix} y \\ 0 \end{bmatrix}. \tag{40}$$

Then we have

$$\|\widetilde{y} - \widetilde{U}g\|_2^2 = (y - Ug)^\top(y - Ug) + \lambda\alpha g^\top Qg. \tag{41}$$

Therefore, the cost function in (38) can be rewritten as

$$\frac{1}{2}\|\widetilde{y} - \widetilde{U}g\|_2^2 + \widetilde{\lambda}\|g\|_1, \tag{42}$$

where $\widetilde{\lambda} = \lambda(1-\alpha)$. Hence, we adopt the proximal gradient algorithm (22) to minimize $J(g)$ for sparse system identification.

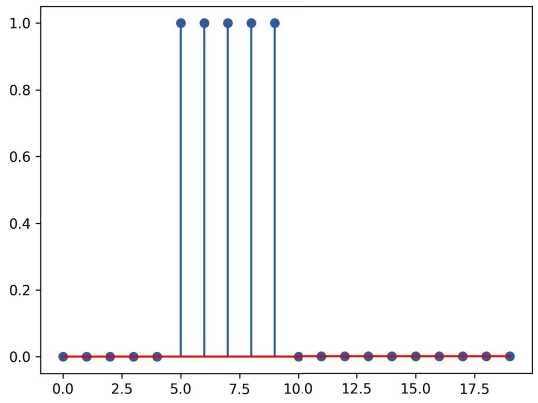

Fig. 7 True impulse response.

Python Example 2: Here we consider a Python simulation of sparse system identification.

First, we import some libraries:

```
import numpy as np
from scipy.linalg import toeplitz
from numpy.linalg import inv
import matplotlib.pyplot as plt
```

In the second line, we import `toeplitz` to compute the matrix U in (35), and the third one is for matrix inversion.

First, we make the input sequence $\{u_k\}$ of length $N = 100$ as

```
# input u
np.random.seed(0)
N = 100
u = np.random.rand(N)
u = np.where(u >= 0.5, 1, -1)
```

We use random variables to obtain a ± 1-valued random vector.

Then, the impulse response g^* to be estimated is set as

```
# true impulse response g*
m = 20
gstar = np.zeros([m,1])
gstar[5:10] = 1
```

Figure 7 shows this impulse response.

With the input and the impulse response, we then compute the output $\{y_k\}$. Namely, we compute the output by (33), where we add Gaussian noise with mean 0 and variance $\sigma^2 = 0.1$.

```
# output y
sigma2 = 0.1 # noise variance
y = np.convolve(gstar.ravel(), u.
    ravel(), mode='full')[:N] + np.
    sqrt(sigma2)*np.random.randn(N)
```

Note that the noise variance has a significant impact on the accuracy of the model's estimation as shown in Fig. 8 (see also the explanations of Fig. 8 below).

We then compute the matrix U as follows:

```
# Toeplitz matrix U
U = toeplitz(np.concatenate(([u[0]],
    np.zeros(m - 1))), u).T
```

We choose the positive definite matrix Q in (37) as *kernel regularization* with the kernel matrix $K = [K_{ij}]$. In particular, we adopt the *tuned-correlated* kernel, also known as *first-order stable-spline kernel* [60], [61], defined by

$$K_{ij} \triangleq \begin{cases} \beta a^{\max(i,j)}, & \text{if} i = j, \\ 0, & \text{otherwise}. \end{cases} \tag{43}$$

Using this, the matrix Q is defined as $Q = K^{-1}$. The Python code is given as follows:

```
# TC Kernel
T1 = np.ones((m, 1)) * np.arange(1, m
    +1)
TCbeta = 100
TCalpha = 0.9
K = TCbeta * TCalpha**(np.maximum(T1,
     T1.T))
K = np.diag(np.diag(K))
```

With this kernel, we fist solve the problem of impulse response estimation by minimizing

$$J_{\text{kernel}}(g) \triangleq \frac{1}{2}\|y - Ug\|_2^2 + \frac{\lambda}{2} g^\top K^{-1} g. \tag{44}$$

Namely, we minimize $J(g)$ in (38) with $\alpha = 1$. The solution is easily obtained by

$$g_2^\star \triangleq K(U^\top UK + \lambda I)^{-1} U^\top y. \tag{45}$$

The associated Python code is given as follows:

```
lmbd = 1
g2 = K @ (inv(U.T @ U @ K + lmbd * np
    .eye(m)) @ U.T @ y)
```

Now, let's solve the sparse regularization. We take the parameters $\alpha = 0.8$ and $\lambda = 50$ in (38). First, we compute the matrix $\widetilde{U}$ and $\widetilde{y}$ in (40). The Python code is given as follows:

```
Q = inv(K)
alpha = 0.8
lmbd = 50
if alpha > 0:
  tU = np.vstack((U, np.linalg.
      cholesky(lmbd * alpha * Q)))
  ty = np.vstack((y.reshape(-1,1), np
      .zeros((m, 1))))
  tlambda = lmbd * (1 - alpha)
elif alpha == 0:
  tU = U
  ty = y
  tlambda = lmbd
else:
  print('alpha should be in [0,1]')
  exit()
```

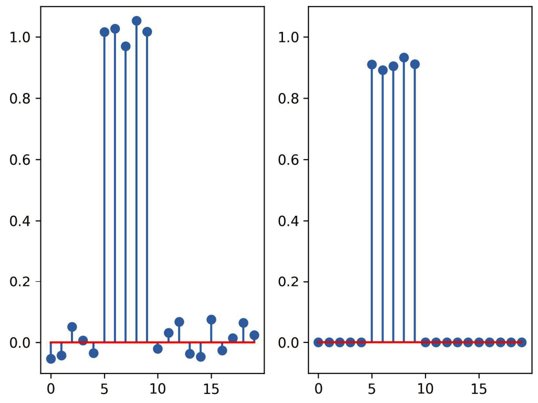

Fig. 8 Reconstructed impulse response by kernel regularization (left) and sparse regularization (right).

Note that we assume $\alpha \in [0, 1]$. Then, we implement the proximal gradient algorithm (22) with step size $\gamma = 0.01$. The Python code is given as follows:

```
# Soft-thresholding function
def St(lmbd, v):
  Sv = np.sign(v) * np.maximum(np.abs
      (v) - lmbd, 0)
  return Sv
# Proximal gradient algorithm
gamma = 0.01
g = np.zeros((m, 1))
for k in range(1000):
  g = St(gamma*tlambda,g + gamma * tU
      .T @ (ty - tU @ g))
```

Figure 8 shows the results of the kernel regularization (45) and the sparse regularization. From this figure, we can see that the zero values in the impulse response are exactly reconstructed by the sparse regularization. This property is useful when we detect the delay time in the response. Since the sparse regularization can reconstruct the nonactive response exactly, it can also detect the delay time precisely.

6. Sparse Feedback Gain

In this section, we show an application of compressed sensing to the design of a sparse feedback gain.

Let us consider the following linear time-invariant system:

$$\dot{x}(t) = Ax(t) + Bu(t), \quad t \geq 0, \tag{46}$$

where $x(t) \in \mathbb{R}^n$, $u(t) \in \mathbb{R}^m$, $A \in \mathbb{R}^{n\times n}$, and $B \in \mathbb{R}^{n\times m}$. We assume (A, B) is stabilizable [69]. Then there exists a state feedback gain $K \in \mathbb{R}^{m\times n}$ such that the state feedback control

$$u(t) = Kx(t), \tag{47}$$

asymptotically stabilizes the system (46). Inserting (47) into (46), we have

$$\dot{x}(t) = (A + BK)x(t), \quad t \geq 0. \tag{48}$$

Then, if K asymptotically stabilizes the feedback system, then the eigenvalues of the matrix $A + BK$ have negative real parts [69, Proposition 5.5.6]. This is equivalent to the existence of $Q \succ 0$ such that the following matrix inequality holds [70, Corollary 3.5.1]:

$$(A + BK)^\top Q + Q(A + BK) \prec 0. \tag{49}$$

In this inequality, K and Q are both unknown variables, and hence it is nonlinear. To make the inequality linear, we introduce new variables $P \triangleq Q^{-1}$ and $Y \triangleq KP$. Then, from matrix inequality (49), we have

$$P \succ 0, \quad AP + PA^\top + BY + Y^\top B^\top \prec 0. \tag{50}$$

These are called *linear matrix inequalities* (LMIs). LMIs play an important role in linear control systems design [70].

Now, the problem of sparse feedback gain design is formulated as follows:

Problem 1 (Sparse feedback gain): Find matrix Y that has the minimum ℓ^0 norm among matrices satisfying the LMIs (50).

If Y is sparse, or contains many zeros, then choosing the output as $y = P^{-1}x$, one can implement a sparse output feedback gain $u = Yy$.

Let us consider how to obtain such a sparse solution. First, we slightly change the LMIs in (50) as follows:

$$P \succeq \epsilon I, \quad AP + PA^\top + BY + Y^\top B^\top \preceq -\epsilon I, \tag{51}$$

with a small number $\epsilon > 0$. Then the set

$$\Lambda \triangleq \{Y : \exists P \succeq \epsilon I, AP + PA^\top + BY + Y^\top B^\top \preceq -\epsilon I\}, \tag{52}$$

becomes a closed subset of $\mathbb{R}^{m\times n}$. We note that if $Y \in \Lambda$ then this Y satisfies (50).

Now, Problem 1 is described as a matrix optimization problem of

$$\underset{Y}{\text{minimize}}\ \|Y\|_0 \ \text{ subject to } \ Y \in \Lambda, \tag{53}$$

where $\|Y\|_0$ is the ℓ^0 norm of Y. As in compressed sensing, this is a combinatorial optimization and hard to solve. Therefore, we approximate the ℓ^0 norm $\|Y\|_0$ by the ℓ^1 norm $\|Y\|_1$, the sum of absolute values of the entries of Y. With the ℓ^1 norm, the ℓ^0 optimization in (53) is reduced to the following *convex* optimization problem:

$$\underset{Y}{\text{minimize}}\ \|Y\|_1 \ \text{ subject to } \ Y \in \Lambda. \tag{54}$$

The ℓ^1 norm heuristic approach for sparse feedback gains has been proposed in [71], [72].

To numerically solve the convex optimization problem (54), we can use CVXPY discussed in Sect. 3 as shown in the example below.

Python Example 3: Here we design a sparse feedback gain for the linear plant (46) with

$$A = \begin{bmatrix} 0 & 0 & 1.132 & 0 & -1 \\ 0 & -0.0538 & -0.1712 & 0 & 0.0705 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0.0485 & 0 & -0.8556 & -1.013 \\ 0 & -0.2909 & 0 & 1.0532 & -0.6859 \end{bmatrix},$$
$$B = \begin{bmatrix} 0 & 0 & 0 \\ -0.12 & 1 & 0 \\ 0 & 0 & 0 \\ 4.419 & 0 & -1.665 \\ 1.575 & 0 & -0.0732 \end{bmatrix}. \tag{55}$$

This model, named AC2, is taken from the benchmark problem set in COMPL$_e$ib library [73].

First, we solve the ℓ^1 norm optimization in (54). For this, we use CVXPY package [11]. So, we first install this package.

```
!pip install cvxpy
```

Then we import necessary packages.

```
import cvxpy as cp
import numpy as np
```

We define the system matrices A and B in (55) as

```
# System matrices
n = 5
m = 3
A = np.matrix(
  [[0,0,1.1320,0,-1],
   [0,-0.0538,-0.1712,0,0.0705],
   [0,0,0,1,0],
   [0,0.0485,0,-0.8556,-1.0130],
   [0,-0.2909,0,1.0532,-0.6859]])
B = np.matrix(
  [[0,0,0],
   [-0.12,1,0],
   [0,0,0],
   [4.419,0,-1.6650],
   [1.575,0,-0.0732]])
```

Then we set the cost function and the LMI constraints (51) with $\epsilon = 0.01$. The Python code is given by

```
epsil = 0.01
eI = epsil * np.eye(n)
P = cp.Variable((n,n), symmetric=True
   )
```

```
Y = cp.Variable((m,n))
objective = cp.Minimize(cp.norm1(Y))
constraints = [P - eI >> 0]
constraints += [A @ P + P @ A.T + B @
    Y + Y.T @ B.T + eI << 0]
```

Finally, we solve the optimization problem (54) by Python as follows:

```
prob = cp.Problem(objective,
   constraints)
prob.solve()
Y_ = Y.value
Y_[np.abs(Y_) < 1e-6] = 0
```

In this code, `cp.Problem` defines the optimization problem to be solved, and `prob.solve()` solves it. To extract the numeric value in the optimization variable `Y`, we write `Y.value`. Then, we shrink the values whose absolute values are less than 10^{-6} to zero, by the last command. After running the codes, we obtain the following solution:

$$Y = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -0.00552915 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{56}$$

This is a sparse matrix, and we successfully obtain a sparse feedback gain.

7. Conclusion

In this survey paper, we have provided a comprehensive review of the theoretical foundations and practical applications of compressed sensing, with a focus on group testing, system identification, and feedback control. The concept of sparsity, which underlies compressed sensing, holds great potential for advancing research in a wide range of scientific fields. We hope that this survey paper has provided valuable insights into the possibilities and challenges of compressed sensing, and will inspire further exploration and development of this important research area.

Acknowledgments

This research was partly supported by JSPS KAKENHI Grant Nos. 23H01436, 22H00512, 22H01653, and 22KK0155.

References

[1] Y.C. Eldar and G. Kutyniok, Compressed Sensing: Theory and Applications, Cambridge University Press, Cambridge, 2012.

[2] M. Vidyasagar, An Introduction to Compressed Sensing, SIAM, Philadelphia, 2019.

[3] R.V. Cox, S.F. De Campos Neto, C. Lamblin, and M.H. Sherif, “ITU-T coders for wideband, superwideband, and fullband speech communication,” IEEE Commun. Mag., vol.47, no.10, pp.106–109, 2009.

[4] J.F. Claerbout and F. Muir, “Robust modeling with erratic data,” Geophysics, vol.38, no.5, pp.826–844, 1973.

[5] F. Santosa and W.W. Symes, “Linear inversion of band-limited reflection seismograms,” SIAM J. Sci. Stat. Comp., vol.7, no.4, pp.1307–1330, 1986.

[6] H.L. Taylor, S.C. Banks, and J.F. McCoy, “Deconvolution with the ℓ_1 norm,” Geophysics, vol.44, no.1, pp.39–52, 1979.

[7] J.L. Starck, F. Murtagh, and J.M. Fadili, Sparse Image and Signal Processing, Cambridge University Press, 2010.

[8] M. Lustig, D.L. Donoho, J.M. Santos, and J.M. Pauly, “Compressed sensing MRI,” IEEE Signal Process. Mag., vol.25, no.2, pp.72–82, March 2008.

[9] B.K. Natarajan, “Sparse approximate solutions to linear systems,” SIAM J. Comput., vol.24, no.2, pp.227–234, 1995.

[10] M. Grant and S. Boyd, “Graph implementations for nonsmooth convex programs,” Recent Advances in Learning and Control, V. Blondel, S. Boyd, and H. Kimura, eds., Lecture Notes in Control and Information Sciences, vol.371, pp.95–110, Springer, London, 2008.

[11] S. Diamond and S. Boyd, “CVXPY: A Python-embedded modeling language for convex optimization,” Journal of Machine Learning Research, vol.17, no.1, pp.2909–2913, 2016.

[12] M. Nagahara, D.E. Quevedo, and D. Nešić, “Maximum hands-off control and L^1 optimality,” 52nd IEEE Conference on Decision and Control (CDC), pp.3825–3830, Dec. 2013.

[13] M. Nagahara, D.E. Quevedo, and D. Nešić, “Maximum hands-off control: A paradigm of control effort minimization,” IEEE Trans. Autom. Control, vol.61, no.3, pp.735–747, 2016.

[14] B. Dunham, “Automatic on/off switching gives 10-percent gas saving,” Popular Science, vol.205, no.4, p.170, Oct. 1974.

[15] R. Kirchhoff, M. Thele, M. Finkbohner, P. Rigley, and W. Settgast, “Start-stop system distributed in-car intelligence,” ATZextra Worldw., vol.15, no.11, pp.52–55, Jan. 2010.

[16] C. Chan, “The state of the art of electric, hybrid, and fuel cell vehicles,” Proc. IEEE, vol.95, no.4, pp.704–718, April 2007.

[17] P. Shakouri, A. Ordys, P. Darnell, and P. Kavanagh, “Fuel efficiency by coasting in the vehicle,” International Journal of Vehicular Technology, vol.2013, p.14, 2013.

[18] M. Nalbach, A. Korner, and S. Kahnt, “Active engine-off coasting using 48 V: Economic reduction of CO_2 emissions,” 17th International Congress ELIV, pp.41–51, Oct. 2015.

[19] E. Khmelnitsky, “On an optimal control problem of train operation,” IEEE Trans. Autom. Control, vol.45, no.7, pp.1257–1266, 2000.

[20] C. Chang and S. Sim, “Optimising train movements through coast control using genetic algorithms,” IEE Proceedings-Electric Power Applications, vol.144, no.1, pp.65–73, 1997.

[21] G. Vossen and H. Maurer, “On L^1-minimization in optimal control and applications to robotics,” Optimal Control Applications and Methods, vol.27, no.6, pp.301–321, 2006.

[22] M. Nagahara, D.E. Quevedo, and D. Nešić, “Hands-off control as green control,” SICE Control Division Multi Symposium 2014, March 2014.

[23] M. Nagahara, “Sparse control for continuous-time systems,” International Journal of Robust and Nonlinear Control, vol.33, no.1, pp.6–22, Jan. 2022.

[24] M. Athans and P.L. Falb, Optimal Control, Dover Publications, New York, 2007. an unabridged republication of the work published by McGraw-Hill in 1966.

[25] T. Ikeda and M. Nagahara, “Value function in maximum hands-off control for linear systems,” Automatica, vol.64, pp.190–195, 2016.

[26] D. Chatterjee, M. Nagahara, D.E. Quevedo, and K.M. Rao, “Characterization of maximum hands-off control,” Systems & Control Letters, vol.94, pp.31–36, 2016.

[27] T. Ikeda and M. Nagahara, “Time-optimal hands-off control for linear time-invariant systems,” Automatica, vol.99, pp.54–58, 2019.

[28] S. Sukumar and D. Chatterjee, “A jammer’s perspective of reachability and LQ optimal control,” Automatica, vol.70, pp.295–302, 2016.

[29] T. Ikeda, M. Nagahara, and K. Kashima, “Maximum hands-off distributed control for consensus of multi-agent systems with sampled-

data state observation," IEEE Trans. Control Netw. Syst., vol.6, no.2, pp.852–862, June 2019.

[30] T. Ikeda, D. Zelazo, and K. Kashima, "Maximum hands-off distributed bearing-based formation control," 2019 IEEE 58th Conference on Decision and Control (CDC), pp.4459–4464, 2019.

[31] M. Nagahara, D. Chatterjee, N. Challapalli, and M. Vidyasagar, "CLOT norm minimization for continuous hands-off control," Automatica, vol.113, p.108679, 2020.

[32] T. Ikeda and K. Kashima, "On sparse optimal control for general linear systems," IEEE Trans. Autom. Control, vol.64, no.5, pp.2077–2083, 2019.

[33] Y. Kumar, S. Srikant, and D. Chatterjee, "Optimal multiplexing of sparse controllers for linear systems," Automatica, vol.106, pp.134–142, 2019.

[34] I. Exarchos, E.A. Theodorou, and P. Tsiotras, "Stochastic L^1-optimal control via forward and backward sampling," Systems & Control Letters, vol.118, pp.101–108, 2018.

[35] R.P. Aguilera, R. Delgado, D. Dolz, and J.C. Agüero, "Quadratic MPC with ℓ_0-input constraint," IFAC Proceedings Volumes, vol.47, no.3, pp.10888–10893, 2014. 19th IFAC World Congress.

[36] M. Nagahara and D. Nešić, "An approach to minimum attention control by sparse derivative," 2020 59th IEEE Conference on Decision and Control (CDC), pp.5005–5010, 2020.

[37] T. Ikeda and M. Nagahara, "Maximum hands-off control with time-space sparsity," IEEE Control Syst. Lett., vol.5, no.4, pp.1213–1218, Oct. 2021.

[38] T. Ikeda and M. Nagahara, "Resource-aware time-optimal control with multiple sparsity measures," Automatica, vol.135, p.109957, 2022.

[39] B. Polyak and A. Tremba, "Sparse solutions of optimal control via Newton method for under-determined systems," J. Glob. Optim., vol.76, pp.613–623, 2020.

[40] K. Hamada, I. Maruta, K. Fujimoto, and K. Hamamoto, "Locally deforming continuation method based on a shooting method for a class of optimal control problems," SICE Journal of Control, Measurement, and System Integration, vol.14, no.2, pp.80–89, 2021.

[41] Y. Kumar, S. Sukumar, D. Chatterjee, and M. Nagahara, "Sparse optimal control problems with intermediate constraints: Necessary conditions," Optimal Control, Applications and Methods, vol.43, no.2, pp.369–385, 2022.

[42] C.V. Rao, "Sparsity of linear discrete-time optimal control problems with l_1 objectives," IEEE Trans. Automa. Control, vol.63, no.2, pp.513–517, 2018.

[43] P.K. Mishra, D. Chatterjee, and D.E. Quevedo, "Resource efficient stochastic predictive control under packet dropouts," IET Control Theory & Applications, vol.11, no.11, pp.1666–1673, 2017.

[44] P.K. Mishra, D. Chatterjee, and D.E. Quevedo, "Sparse and constrained stochastic predictive control for networked systems," Automatica, vol.87, pp.40–51, 2018.

[45] A. Sachan, S. Kamal, S. Olaru, D. Singh, and X. Xiong, "Discrete-time sector based hands-off control for nonlinear system," International Journal of Robust and Nonlinear Control, vol.30, no.6, pp.2443–2460, 2020.

[46] D. Iwai, H. Izawa, K. Kashima, T. Ueda, and K. Sato, "Speeded-up focus control of electrically tunable lens by sparse optimization," Sci. Rep., vol.9, p.12365, 2019.

[47] M. Leomanni, G. Bianchini, A. Garulli, A. Giannitrapani, and R. Quartullo, "Sum-of-norms model predictive control for spacecraft maneuvering," IEEE Control Syst. Lett., vol.3, no.3, pp.649–654, 2019.

[48] Y. Shiraishi, M. Nagahara, and D. Saelens, "Optimal control of TABS by sparse MPC," Building Simulation 2021 Conference, 2021.

[49] K. Motonaka, T. Watanabe, Y. Kwon, M. Nagahara, and S. Miyoshi, "Control of a quadrotor group based on maximum hands-off distributed control," International Journal of Mechatronics and Automation, vol.8, no.4, pp.200–207, 2021.

[50] D. Donoho and M. Elad, "Optimally sparse representation in general (nonorthogonal) dictionaries via l_1 minimization," Proc. Natl. Acad. Sci., vol.100, no.5, pp.2197–2202, 2003.

[51] M. Elad, Sparse and Redundant Representations, Springer, New York, 2010.

[52] A.S. Bandeira, E. Dobriban, D.G. Mixon, and W.F. Sawin, "Certifying the restricted isometry property is hard," IEEE Trans. Inf. Theory, vol.59, no.6, pp.3448–3450, 2013.

[53] P.L. Combettes and J.C. Pesquet, "Proximal splitting methods in signal processing," Fixed-Point Algorithms for Inverse Problems in Science and Engineering, pp.185–212, Springer New York, New York, NY, 2011.

[54] A. Beck and M. Teboulle, "Gradient-based algorithms with applications to signal-recovery problems," Convex Optimization, Cambridge University Press, Cambridge, 2010.

[55] H.Y. Kim, M.G. Hudgens, J.M. Dreyfuss, D.J. Westreich, and C.D. Pilcher, "Comparison of group testing algorithms for case identification in the presence of test error," Biometrics, vol.63, no.4, pp.1152–1163, 2007.

[56] R. Dorfman, "The detection of defective members of large populations," Ann. Math. Statist., vol.14, no.4, pp.436–440, Dec. 1943.

[57] G.K. Atia and V. Saligrama, "Boolean compressed sensing and noisy group testing," IEEE Trans. Inf. Theory, vol.58, no.3, pp.1880–1901, March 2012.

[58] M. Aldridge, L. Baldassini, and O. Johnson, "Group testing algorithms: Bounds and simulations," IEEE Trans. Inf. Theory, vol.60, no.6, pp.3671–3687, June 2014.

[59] G. Pillonetto, F. Dinuzzo, T. Chen, G. De Nicolao, and L. Ljung, "Kernel methods in system identification, machine learning and function estimation: A survey," Automatica, vol.50, no.3, pp.657–682, 2014.

[60] G. Pillonetto and G. De Nicolao, "A new kernel-based approach for linear system identification," Automatica, vol.46, no.1, pp.81–93, 2010.

[61] T. Chen, H. Ohlsson, and L. Ljung, "On the estimation of transfer functions, regularizations and Gaussian processes–Revisited," Automatica, vol.48, no.8, pp.1525–1535, 2012.

[62] Y. Fujimoto and T. Sugie, "Kernel-based impulse response estimation with *a priori* knowledge on the DC gain," IEEE Control Syst. Lett., vol.2, no.4, pp.713–718, 2018.

[63] Y. Fujimoto, "Kernel regularization in frequency domain: Encoding high-frequency decay property," IEEE Control Syst. Lett., vol.5, no.1, pp.367–372, 2021.

[64] Y. Fujimoto, "Kernel regularization for low-frequency decay systems," 60th IEEE Conference Decision and Control (CDC 2021), pp.308–3023, Dec. 2021.

[65] Y. Fujimoto, I. Maruta, and T. Sugie, "Extension of first-order stable spline kernel," 20th IFAC World Congress (IFAC 2017), pp.15481–15486, July 2017.

[66] Y. Chen, Y. Gu, and A.O. Hero, "Sparse LMS for system identification," Proc. IEEE ICASSP2009, pp.3125–3128, 2009.

[67] S. Fattahi and S. Sojoudi, "Data-driven sparse system identification," Proc. 56th An. Allerton Conf. Commun., Contr., Computing, pp.462–469, 2018.

[68] C.M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.

[69] E.D. Sontag, Mathematical Control Theory, 2nd ed., Springer, New York, 1998.

[70] R.E. Skelton, T. Iwasaki, and K. Grigoriadis, A Unified Algebraic Approach to Linear Control Design, Taylor & Francis, London, 1998.

[71] F. Lin, M. Fardad, and M.R. Jovanović, "Design of optimal sparse feedback gains via the alternating direction method of multipliers," IEEE Trans. Autom. Control, vol.58, no.9, pp.2426–2431, 2013.

[72] B. Polyak, M. Khlebnikov, and P. Shcherbakov, "An LMI approach to structured sparse feedback design in linear control systems," 2013 European Control Conference (ECC), pp.833–838, 2013.

[73] F. Leibfritz, "Compl$_e$ib: constraint matrix optimization problem library," Technical Report, 2005.

Masaaki Nagahara received a bachelor's degree in engineering from Kobe University in 1998, and a master's degree and a Doctoral degree in informatics from Kyoto University in 2000 and 2003, respectively, under the supervision of Prof. Yutaka Yamamoto. He is currently a Professor at the Graduate School of Advanced Science and Engineering, Hiroshima University. He has been a Visiting Professor at Indian Institute of Technology Bombay since 2017. His research interests include control theory, machine learning, and sparse modeling. He received two remarkable international awards: George S. Axelby Outstanding Paper Award in 2018 and Transition to Practice Award in 2012 from the IEEE Control Systems Society. Also, he received many awards from Japanese research societies such as SICE Young Authors Award in 1999, SICE Best Paper Award in 2012, SICE Best Book Authors Awards in 2016 and 2021, SICE Control Division Research Award (Kimura Award) in 2020, and the Best Tutorial Paper Award from the IEICE Communications Society in 2014. He is a senior member of the IEEE. He has been serving as General Co-Chair of IEEE CCTA2027, Awards Chair of IFAC NMPC2024, IPC Vice-Chair of IFAC WC 2023, Delegate of ISCIE (2022–), Director of Journal of SICE (2022–), Editor of SICE JCMSI (2022–), and Associate Editor of Asian Journal of Control (2019–) and Advanced Robotics (2021–).