Report of Sparse blind deconvolution

吴天宇 12334125

0.1题目

Sparse blind deconvolution. We are given a time series observation $y \in \mathbb{R}^T$, and seek a filter (convolution kernel) $w \in \mathbb{R}^k$, so that the convolution $x = w * y \in \mathbb{R}^{(T+k-1)}$ is sparse after truncating the first and last $k-1$ entries, i.e., $x_{k:T} = (x_k, x_{k+1}, \ldots, x_T)$ is sparse. Here $*$ denotes convolution,

$$x_i = \sum_{j=1}^{k} w_j y_{i-j}, \quad i = 1, \ldots, T+k-1$$

where we assume that $y_t = 0$ for $t \le 0$. Typically we have $k \ll T$.

As a convex surrogate for sparsity of x, we minimize its l_1-norm, $\|x\|_1$. To preclude the trivial solution $w = 0$, we normalize w by imposing the constraint $w_1 = 1$.

Interpretations. (These are not needed to solve the problem.) In signal processing dialect, we can say that w is a filter which, when applied to the signal y, results in x, a simpler, sparse signal. As a second interpretation, we can say that $y = w^{(-1)} * x$, where $w^{(-1)}$ is the convolution inverse of w, defined as

$$w^{(-1)} = F^{(-1)}(1/F(w)),$$

where F is discrete Fourier transform at length $N = T + k$ and $F^{(-1)}$ is its inverse transform. In this interpretation, we can say that we have decomposed the signal into the convolution of a sparse signal x and a signal with short (k-long) inverse, $w^{(-1)}$.

Carry out blind deconvolution on the signal given in `blind_deconv_data.*`. This file also defines the kernel length k. Plot optimal w and x, and also the given observation y. Also plot the inverse kernel $w^{(-1)}$, use the function `inverse_ker` that we provided in `blind_deconv_data.*`.

Hint. The function `conv(w, y)` is overloaded to work with CVX*.

0.2题目翻译

稀疏盲反卷积 我们得到一个时间序列观测值 $y \in \mathbb{R}^T$，并寻找一个滤波器（卷积核）$w \in \mathbb{R}^k$，使得卷积 $x = w * y \in \mathbb{R}^{(T+k-1)}$ 在截断首尾 $k-1$ 项后变得稀疏，即 $x_{k:T} = (x_k, x_{k+1}, \ldots, x_T)$ 是稀疏的。这里 $*$ 表示卷积，

$$x_i = \sum_{j=1}^{k} w_j y_{i-j}, \quad i = 1, \ldots, T+k-1$$

其中我们假设 $y_t = 0$ 对于 $t \le 0$。通常情况下我们有 $k \ll T$。

作为 x 的稀疏性的凸替代，我们最小化其 l_1-范数，$\|x\|_1$。为了避免平凡解 $w = 0$，我们通过施加约束 $w_1 = 1$ 来规范化 w。

解释（这些不是解决问题所必需的。）在信号处理术语中，我们可以说 w 是一个滤波器，当应用于信号 y 时，产生了 x，一个更简单、稀疏的信号。作为第二种解释，我们可以说 $y = w^{(-1)} * x$，其中 $w^{(-1)}$ 是 w 的卷积逆，定义为

$$w^{(-1)} = F^{(-1)}(1/F(w)),$$

其中 F 是长度为 $N = T + k$ 的离散傅里叶变换，$F^{(-1)}$ 是其逆变换。在这种解释中，我们可以说我们已经将信号分解成了稀疏信号 x 和具有短（长度为k）逆的信号 $w^{(-1)}$ 的卷积。

在 `blind_deconv_data.*` 中给出的信号上执行盲反卷积。此文件还定义了核长 k。绘制最优 w 和 x，以及给定的观测 y。还要绘制逆核 $w^{(-1)}$，使用我们在 `blind_deconv_data.*` 中提供的 `inverse_ker` 函数。

提示：函数 `conv(w, y)` 已被重载以适用于 CVX*。

1.1解答

给定一个观测时间序列 $y \in \mathbb{R}^T$，目标是找到一个滤波器 $w \in \mathbb{R}^k$，使得卷积 $x = w * y$ 在截断首尾 $k-1$ 个元素后变得稀疏，可以表述为如下优化问题：

$$\begin{array}{ll} \min & \|x\|_1 \\ \text{s.t.} & x = w * y, \\ & w_1 = 1, \\ & w \in \mathbb{R}^k, x \in \mathbb{R}^{(T+k-1)}. \end{array}$$

这里的 $\|x\|_1$ 表示 x 的 l_1-范数，即 x 各元素绝对值之和。目标是最小化 x 的 l_1-范数，而 x 由 w 和 y 通过卷积操作得到。此外，有一个约束条件 $w_1 = 1$ 以确保 w 不是零向量。因为目标函数和所有的约束都是凸的，故该优化问题为凸优化问题。

使用 python 调用 cvxpy 库求解该优化问题，恢复误差（RRMS）为0.0008676301267520908。

`blind_deconv_data.py` 模块定义了函数 `generate_waveform` 用于生成实验所需的波形数据,并定义 `inverse_ker` 函数用于计算卷积核 w 的逆，是解决稀疏盲反卷积优化问题的基础。

In [ ]:
```python
import numpy as np
import scipy.linalg as la
from numpy.random import RandomState
from scipy import signal


# 波形生成
def generate_waveform():
    # 参数设置
    T = 400        # 时间序列的长度
    k = 20         # 滤波器的长度
    N = T + k      # FFT计算中使用的长度
    p = 0.1        # 稀疏信号的稀疏程度
    sigma = 0.0001 # 噪声水平

    # 使用固定的随机数种子生成模型数据
    rn = RandomState(364)

    # 生成真实滤波器，并对其进行处理以满足特定的形式
    w_true = rn.rand(k)
    index = np.argmax(np.abs(w_true))
    w_true = np.roll(w_true, -index)
    w_true = w_true / w_true[0]

    # 生成稀疏信号和观测值
    x_true = rn.randn(T)
    x_true = rn.binomial(1, p, np.shape(x_true)) * x_true
    y_true = np.real(np.fft.ifft(np.fft.fft(x_true, N) / np.fft.fft(w_true, N), N))
    y = y_true[k:T+k] + sigma * rn.randn(T)

    return w_true, y, T, k, N

# 定义逆核函数
def inverse_ker(w, len):
    w_inv = np.real(np.fft.ifft(1/np.fft.fft(w, len), len))
    return w_inv
```

`optimization_solve.py` 模块使用cvxpy库函数实现针对稀疏盲反卷积问题的优化求解。此模块包含了定义目标函数、设置约束条件以及调用优化求解器。

In [ ]:
```python
import numpy as np
import cvxpy as cvx
from scipy import signal

# 优化算法
def optimization_solve(w_true, y, T, k):
    # 定义优化变量
    w = cvx.Variable(k)

    # 构造卷积矩阵
    Y = np.zeros((T, k))
    for i in range(T):
        Y[i, :min(i+1, k)] = y[max(0, i-k+1):i+1][::-1]

    # 定义卷积操作
    x = Y @ w

    # 目标函数：最小化 x 的 l1 范数
    objective = cvx.Minimize(cvx.norm(x, 1))

    # 约束条件：w 的第一个元素等于 1
    constraints = [w[0] == 1]

    # 定义并求解问题
    problem = cvx.Problem(objective, constraints)
    problem.solve(solver=cvx.ECOS)

    # 提取优化后的 w 和 x
    w_optimal = w.value
    x_optimal = Y @ w_optimal

    # 误差计算
    rmms = np.linalg.norm(w_optimal - w_true) / np.linalg.norm(w_true)      # RRMS (Relative Root Mean Square Error) 误差计算
    mse = np.mean((w_optimal - w_true) ** 2)        # 计算 MSE
    nmse = np.linalg.norm(w_optimal - w_true) ** 2 / np.linalg.norm(w_true) ** 2        # 计算 NMSE
    print('Optimize Success')

    return w_optimal, x_optimal, rmms, mse, nmse
```

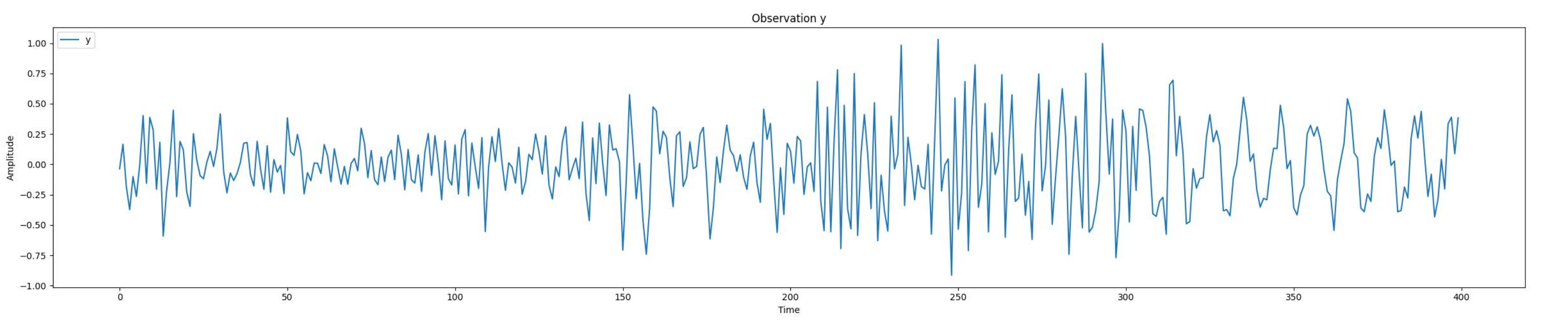
`results_output.py` 模块处理和展示输出结果，绘制图表并保存数据。

In [ ]:
```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os

def plot_and_save_with_csv(y1_data, title, xlabel, ylabel, output_folder_path, fig_size=(24, 5), plot_type='plot', y2_data=None, y2_label=None, legend1=None, legend2=None):
    # 确保输出文件夹存在
    if not os.path.exists(output_folder_path):
        os.makedirs(output_folder_path)

    # 绘制图像
    fig, ax1 = plt.subplots(figsize=fig_size)

    if plot_type == 'plot':
        ax1.plot(y1_data, color='tab:blue', label=legend1)
    elif plot_type == 'stem':
        ax1.stem(y1_data, linefmt='tab:blue', label=legend1)

    ax1.set_xlabel(xlabel)
    ax1.set_ylabel(ylabel)
    ax1.tick_params(axis='y')

    # 第二个 Y 轴的处理
    if y2_data is not None and y2_label is not None:
        ax2 = ax1.twinx()
        ax2.plot(y2_data, '--', color='tab:red', label=legend2)
        ax2.set_ylabel(y2_label)
        ax2.tick_params(axis='y')

    plt.title(title)

    # 显示图例
    if legend1 is not None:
        ax1.legend(loc='upper left')
    if y2_data is not None and legend2 is not None:
        ax2.legend(loc='upper right')

    plt.tight_layout()

    # 保存图像
    safe_title = title.replace(' ', '_').replace('.', '').replace('-', '_')
    image_file_name = f"{safe_title}.jpg"
    plt.savefig(os.path.join(output_folder_path, image_file_name))
    plt.show()

    # 合并数据并保存到 CSV
    csv_file_name = f"{safe_title}.csv"
    if y2_data is not None:
        # 确保数据是列表格式，即使只有一个元素
        y1_data = [y1_data] if np.isscalar(y1_data) else y1_data
        y2_data = [y2_data] if np.isscalar(y2_data) else y2_data
        combined_data = pd.DataFrame({legend1: y1_data, legend2: y2_data})
    else:
        y1_data = [y1_data] if np.isscalar(y1_data) else y1_data
        combined_data = pd.DataFrame({legend1: y1_data})
    combined_data.to_csv(os.path.join(output_folder_path, csv_file_name), index=False, header=True)
```

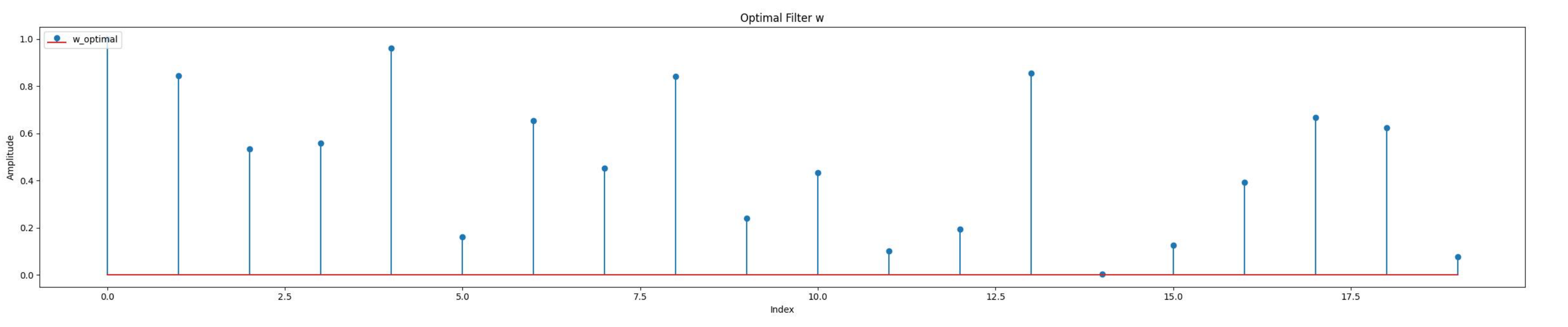
`main.py` 是整个程序的入口点，负责协调上述三个模块的功能，完成稀疏盲反卷积问题的优化求解，包含生成波形数据、执行优化算法以求解最佳滤波器和信号、计算卷积核的逆并输出结果。在 `main.py` 中先调用 `generate_waveform` 函数生成实验所需波形数据，随后通过 `optimization_solve` 函数进行数学优化以最小化 x 的 l_1-范数 $\|x\|_1$ 作为其稀疏性的量化指标，最后使用 `inverse_ker` 函数计算 w 的逆并将结果输出。

In [ ]:
```python
import os
from module.blind_deconv_data import generate_waveform, inverse_ker
from module.optimization_solve import optimization_solve
from module.results_output import plot_and_save_with_csv

# 生成波形
w_true, y, T, k, N = generate_waveform()

# 优化求解
w_optimal, x_optimal, rrms_values, mse_values, nmse_values = optimization_solve(w_true, y, T, k)

# 计算逆核
w_inverse = inverse_ker(w_optimal, N)

# 输出路径
current_dir = os.path.dirname(os.path.realpath(__file__))
results_output_folder_path = os.path.join(current_dir, 'outputs/results')
error_output_folder_path = os.path.join(current_dir, 'outputs/errors')

# 输出结果
plot_and_save_with_csv(y, 'Observation y', 'Time', 'Amplitude', results_output_folder_path, legend1='y')
plot_and_save_with_csv(w_optimal, 'Optimal Filter w', 'Index', 'Amplitude', results_output_folder_path, plot_type='stem', legend1='w_optimal')
plot_and_save_with_csv(x_optimal, 'Sparse Signal x', 'Time', 'Amplitude', results_output_folder_path, legend1='x_optimal')
plot_and_save_with_csv(w_inverse, 'Inverse Kernel w^-1', 'Index', 'Amplitude', results_output_folder_path, plot_type='stem', legend1='w_inverse')
plot_and_save_with_csv(rrms_values, 'RRMS Error', 'Iteration', 'Error', error_output_folder_path, legend1='RRMS')
plot_and_save_with_csv(mse_values, 'MSE Error', 'Iteration', 'Error', error_output_folder_path, legend1='MSE')
plot_and_save_with_csv(nmse_values, 'NMSE Error', 'Iteration', 'Error', error_output_folder_path, legend1='NMSE')
```

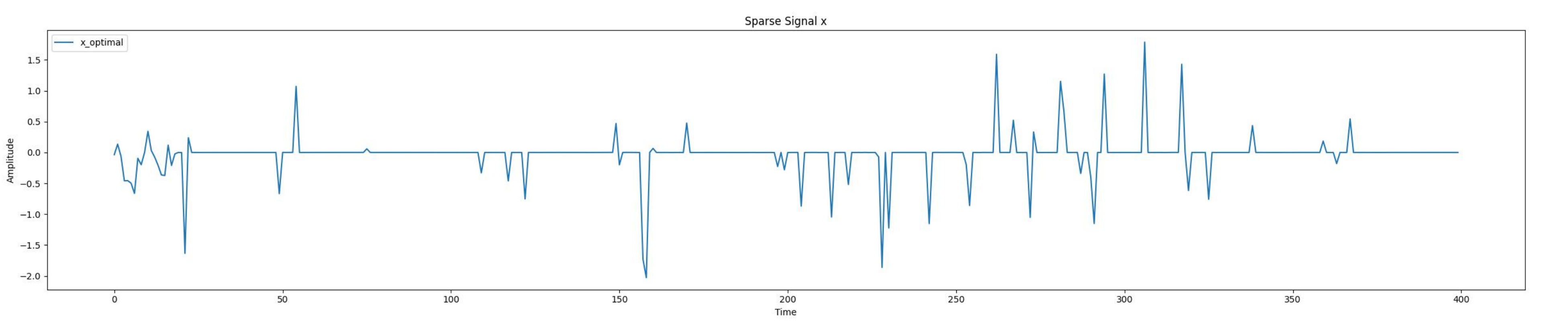
`Observation_y.jpg` 和 `Observation_y.csv` 为原始的观测信号 y。



`Optimal_Filter_w.jpg` 和 `Optimal_Filter_w.csv` 为通过优化问题求解得到的最佳滤波器 w 的幅度。



`Sparse_Signal_x.jpg` 和 `Sparse_Signal_x.csv` 为优化后的滤波器 w 与观测信号 y 的卷积得到的稀疏信号 x。



`Inverse Kernel w^-1.jpg` 和 `Inverse Kernel w^-1.csv` 为滤波器 w 的逆核 w^{-1} 。