MTH 510

Inverse Problems and Data Assimilation

Homework #3 (Resubmission)

Submitted by Andrew Huffman

November 20, 2019

Portland State University

- **TSVD Reconstruction**

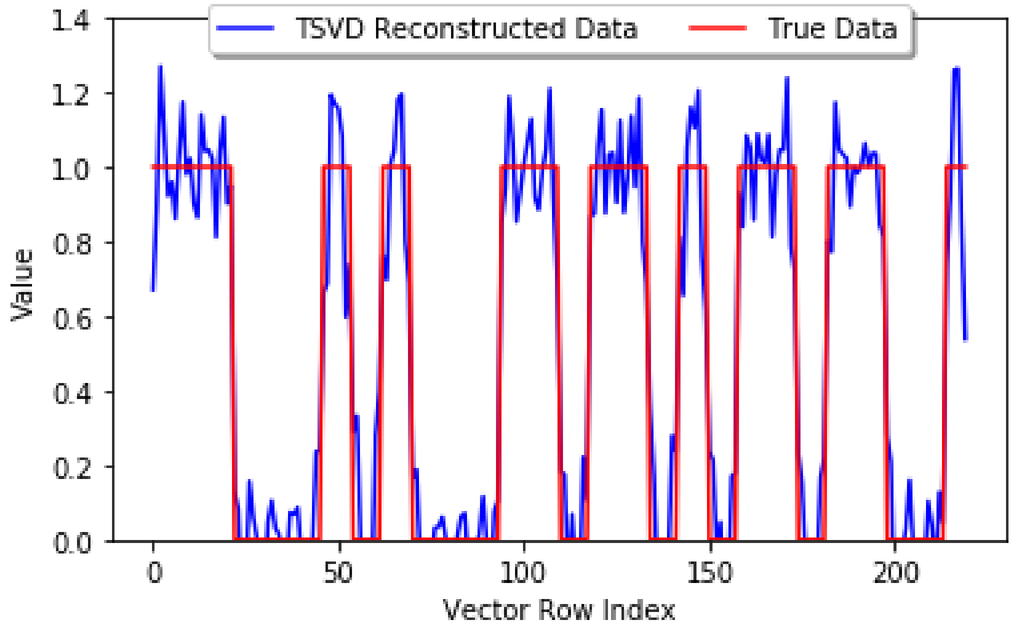

Figure 1: TSVD reconstruction (truncation index = 134)

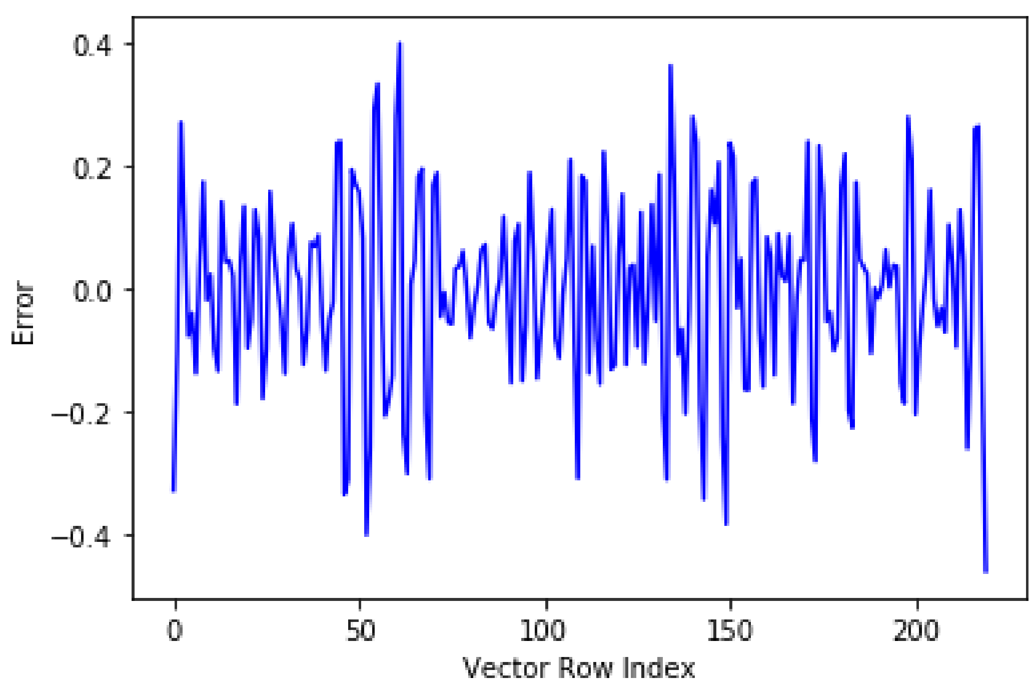

Figure 2: Elementwise reconstruction error ($\hat{x} - x_t$)

- **Tikhonov Reconstruction for L=I**

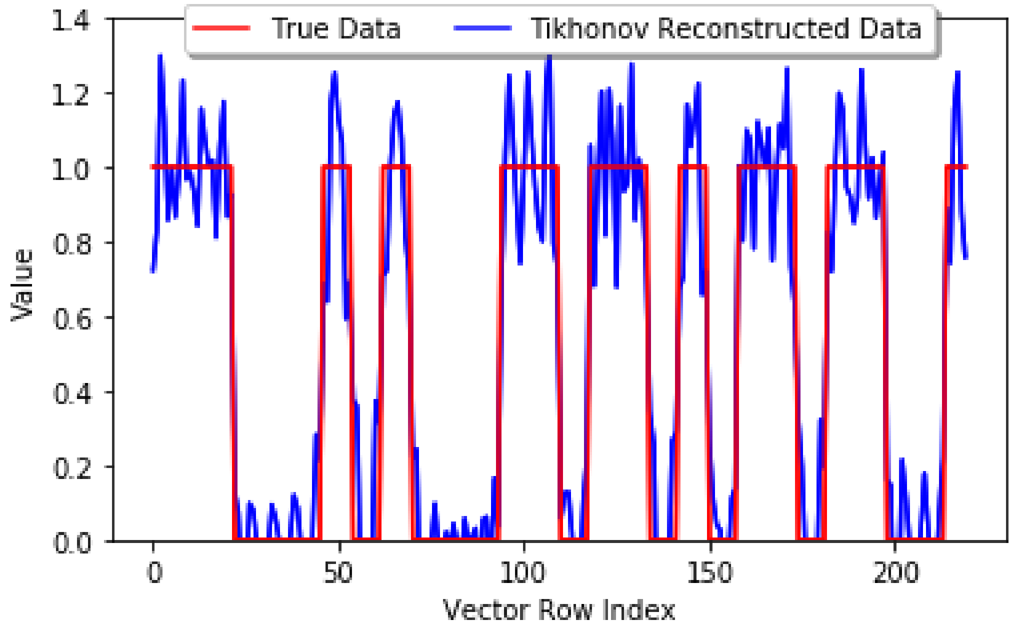

Figure 3: Tikhonov reconstructed data (λ= 0.0006)

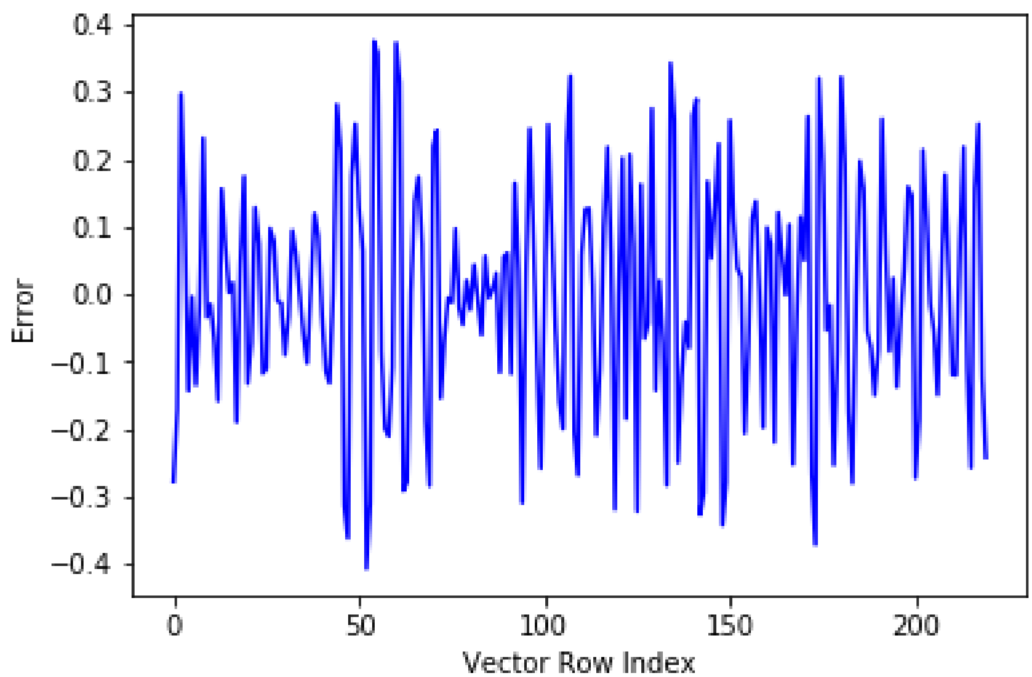

Figure 4: Elementwise reconstruction error ($\hat{x} - x_t$)

- **Tikhonov Reconstruction for L=L_1**

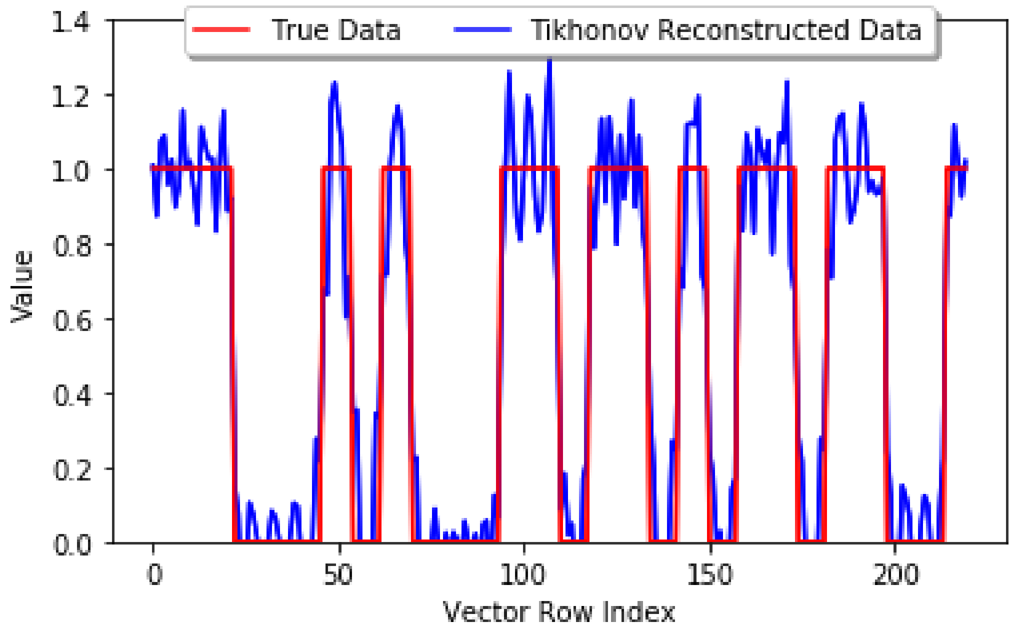

Figure 5: Tikhonov reconstructed data (λ= 0.0006)

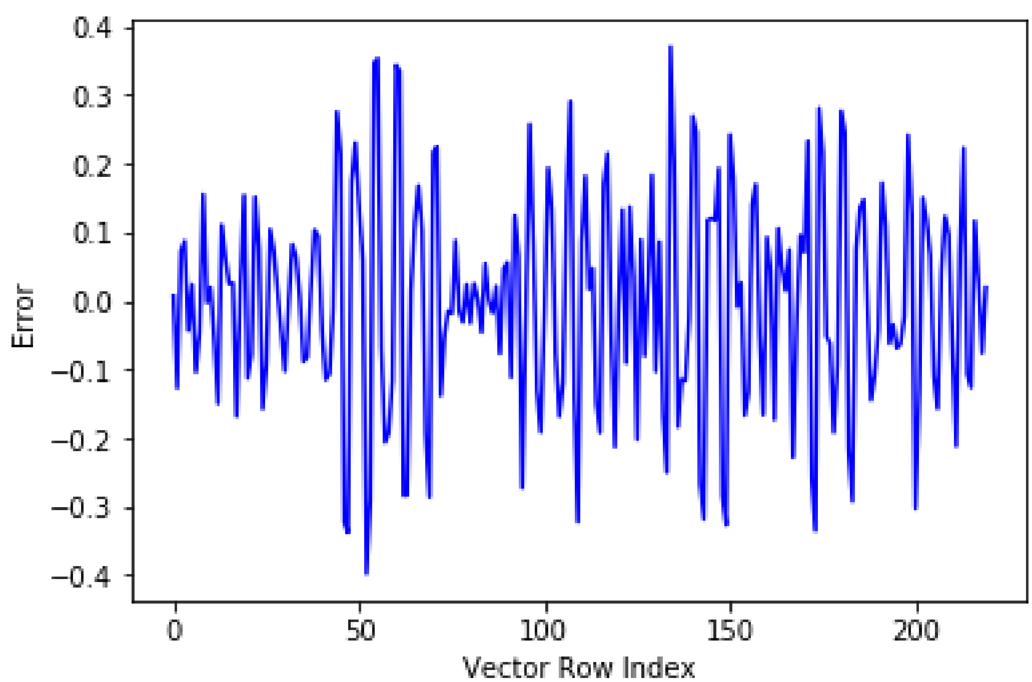

Figure 6: Elementwise reconstruction error ($\hat{x} - x_t$)

- **Tikhonov Reconstruction for L=L_2**

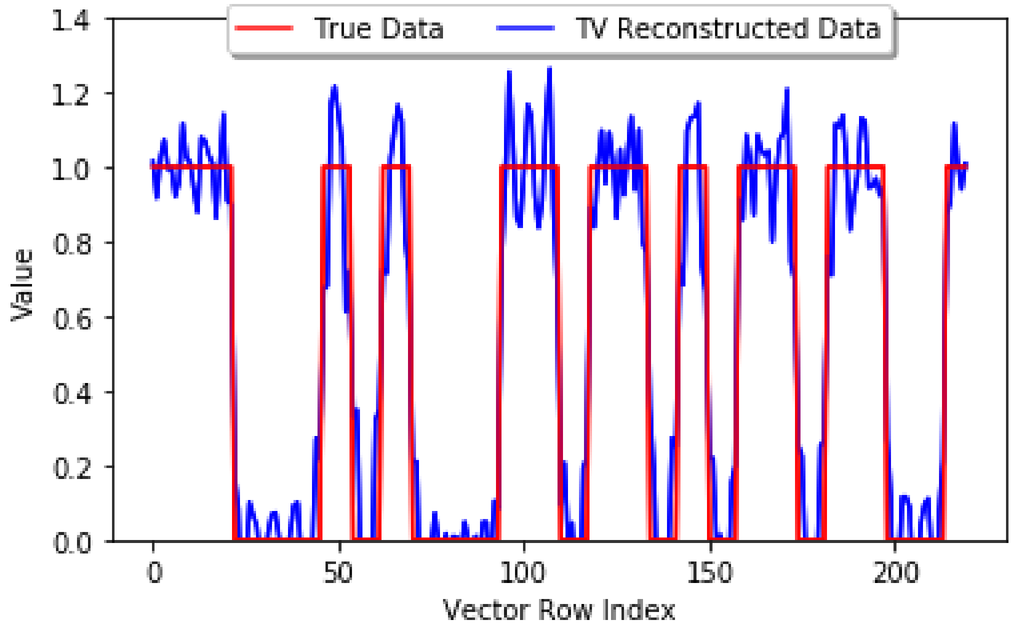

Figure 7: Tikhonov reconstructed data (λ= 0.0006)

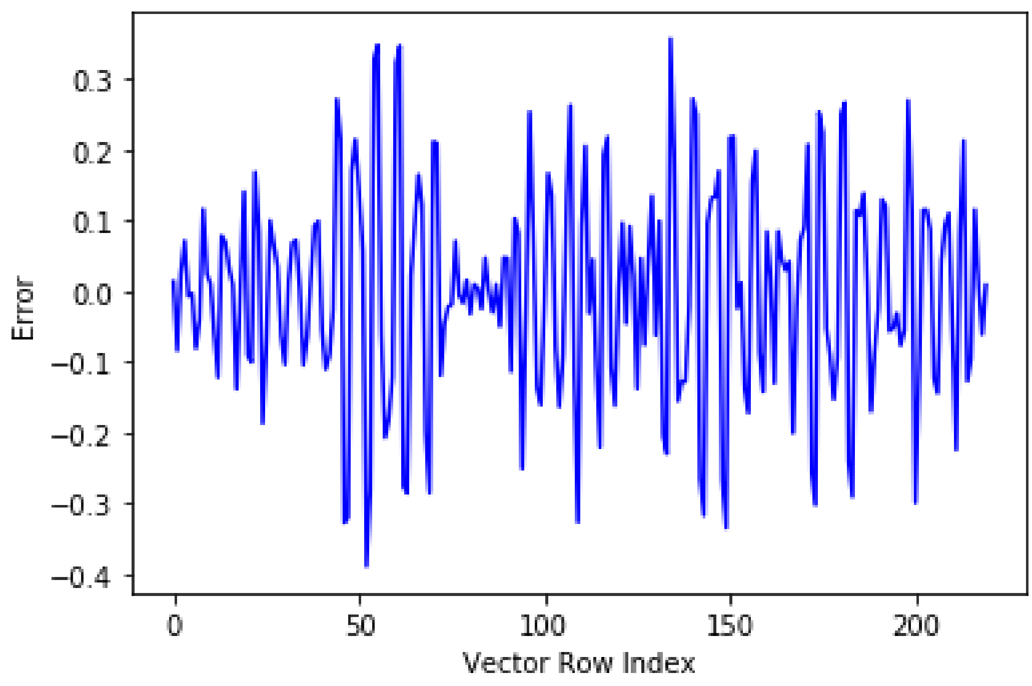

Figure 8: Elementwise reconstruction error ($\hat{x} - x_t$)

- **Total Variation Reconstruction**

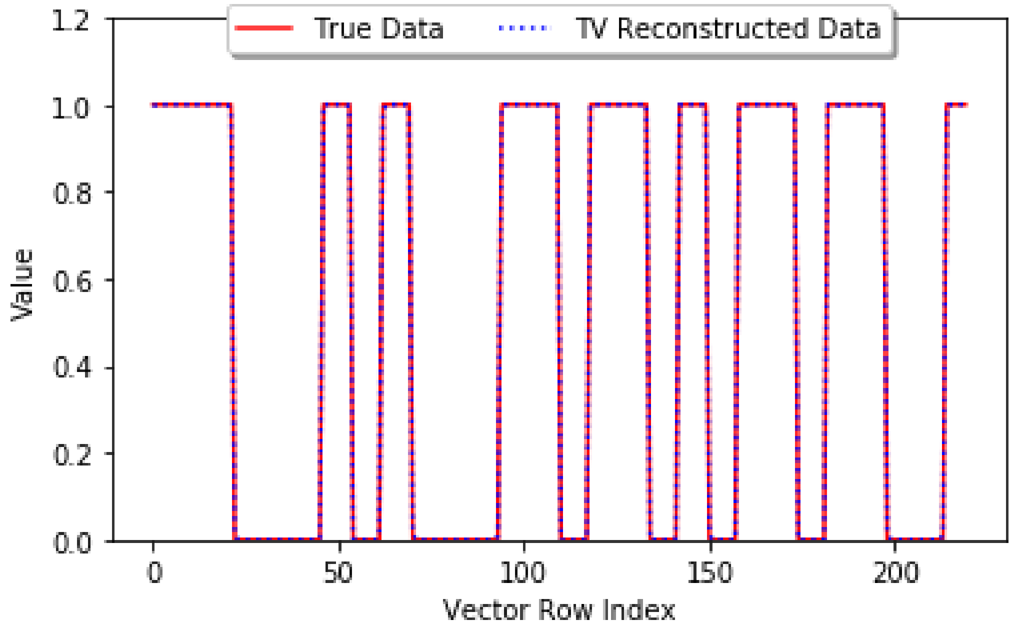

Figure 9: Total Variation reconstructed data (α = 0.0001, β = 0.1)

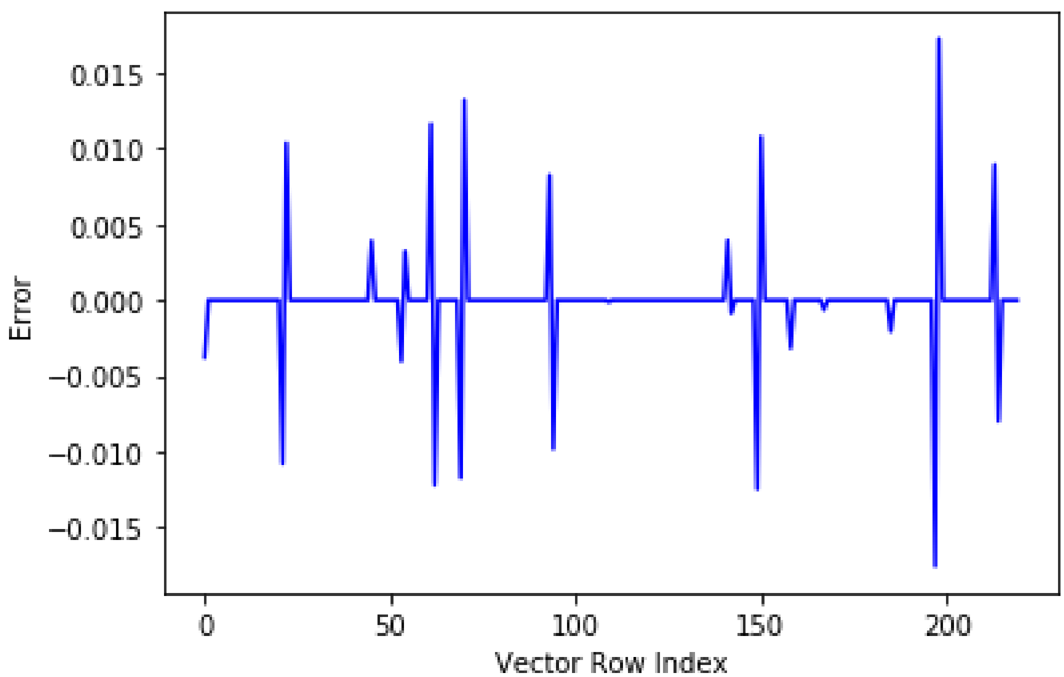

Figure 10: Elementwise reconstruction error ($\hat{x} - x_t$)

Appendix: Script used to produce the above results

```
# -*- coding: utf-8 -*-
"""
Created on Wed Nov  6 14:10:32 2019

@author: Andrew
"""

import pandas as pd
import numpy as np
import scipy
from scipy.sparse import diags
from scipy import optimize
import matplotlib.pyplot as plt
from PIL import Image
import math
import time

start_time = time.time()
D_hat_df = pd.read_csv("BlurData.txt",sep="\s+",header=None)
D_hat = pd.DataFrame.to_numpy(D_hat_df)
x_t_df = pd.read_csv("TrueData.txt",sep="\s+",header=None)
x_t = (pd.DataFrame.to_numpy(x_t_df))


B = np.zeros((220,220))
L = 0.45
power = 10
for i in range(0,B.shape[1]):
    if i<B.shape[1]-1:
        B[i][i] = (1-(2*L))
        B[i][i+1] = L
        B[i+1][i] = L
    else:
        B[i][i] = (1-(2*L))
A = np.linalg.matrix_power(B,power)

Position = np.zeros((220,1))
for i in range(0,220):
    Position[i][0] = i


### TSVD Regularization ###
method = "TSVD"
```

```
if method == "TSVD":
    error_TSVD = np.zeros((220,1))
    p = 134
    U,S,V = np.linalg.svd(A,full_matrices=True)
    S = S.reshape(A.shape[0],1)
    X_hat = np.zeros((D_hat.shape[0],D_hat.shape[1]))
    x_hat = np.zeros((D_hat.shape[0],D_hat.shape[1]))
    for i in range(0,p):
        sigma_i = S[i][0]
        u_i = U[:,i].reshape(U.shape[0],1)
        v_i = V[i,:].reshape(V.shape[0],1)
        x_hat = x_hat+(((np.transpose(u_i)@D_hat)/sigma_i)*v_i)
    x_hat_tsvd = x_hat
    x_error_tsvd = x_hat_tsvd-x_t

    TSVD_Reconstruction = plt.figure()
    fig = plt.subplot()
    fig.plot(Position,x_hat_tsvd,"-b",label="TSVD Reconstructed Data")
    fig.plot(Position,x_t,"-r",label="True Data")
    fig.set_xlabel ("Vector Row Index")
    fig.set_ylabel ("Value")
    fig.set_ylim(bottom=0,top=1.4)
    box = fig.get_position()
    fig.set_position([box.x0, box.y0 + box.height * 0.1, box.width,
box.height * 0.9])
    fig.legend(loc='upper center', bbox_to_anchor=(0.5, 1.05), ncol=3,
fancybox=True, shadow=True)

    TSVD_Error = plt.figure()
    fig = plt.subplot()
    fig.plot(Position,x_error_tsvd,"-b")
    fig.set_xlabel("Vector Row Index")
    fig.set_ylabel("Error")


### Tikhonov Regularization ###
method = "Tikhonov"
if method == "Tikhonov":
    L_type = 0
    L_0 = np.identity(220)
    L_1 = diags([-1,1],[0,1],shape=(219,220)).toarray()
    L_2 = diags([1,-2,1],[0,1,2],shape=(218,220)).toarray()

    if L_type == 0:
        L = L_0
```

```
    elif L_type == 1:
        L = L_1
    elif L_type == 2:
        L = L_2
    for j in range(0,1):
        l = 0.0006+j
        B = (np.transpose(A)@A)+((l**2)*(np.transpose(L)@L))
        x_hat_tk = (np.linalg.inv(B)@np.transpose(A)@D_hat)
        error_TK = x_hat_tk-x_t
        TK_Reconstruction = plt.figure()
        fig = plt.subplot()
        fig.plot(Position,x_t,"-r",label="True Data")
        fig.plot(Position,x_hat_tk,"-b",label="Tikhonov Reconstructed Data")
        fig.set_xlabel ("Vector Row Index")
        fig.set_ylabel ("Value")
        fig.set_ylim(bottom=0,top=1.4)
        box = fig.get_position()
        fig.set_position([box.x0, box.y0 + box.height * 0.1, box.width,
box.height * 0.9])
        fig.legend(loc='upper center', bbox_to_anchor=(0.5, 1.05), ncol=3,
fancybox=True, shadow=True)
        plt.plot(Position,x_hat_tk,"-b",label="Reconstructed Data")
        plt.plot(Position,x_t,"-r",label="True Data")
        TK_error = plt.figure()
        fig = plt.subplot()
        fig.plot(Position,error_TK,"-b")
        fig.set_xlabel("Vector Row Index")
        fig.set_ylabel("Error")



### Total Variation Regularization ####
alpha = 0.0001
beta = 0.1
x0 = np.zeros((220,1)).tolist()

### Define function, J, to minimize
def J_alpha_beta (x,A,D_hat,alpha,beta):
    global output
    x_hat = np.array(x).reshape(220,1)
    Ax = A.dot(x_hat)
    Ax_b = Ax-D_hat
    norm = np.linalg.norm(Ax_b,2)
    residual = norm**2
    T = 0
```

```
    for i in range(0,(A.shape[0]-1)):
        T = T+math.sqrt((beta**2)+(((abs(x_hat[i+1][0]-x_hat[i][0]))**2)))
    output = (residual-((alpha**2)*T))
    return (output**2)

### Define gradient of J
def grad_J(x, A,D_hat,alpha,beta):
    global grad
    x_hat = np.array(x).reshape(220,1)
    for i in range(0,A.shape[0]):
        if i == 0:
            dt_dx = (x_hat[0][0]-
x_hat[1][0])/(math.sqrt(beta**2+(x_hat[1][0]-x_hat[2][0])**2))
        elif (i>0) & (i<(A.shape[0]-1)):
            dt_dx = dt_dx+((x_hat[i][0]-x_hat[i-
1][0])/math.sqrt((beta**2)+(x_hat[i][0]-x_hat[i-1][0])**2))+((x_hat[i][0]-
x_hat[i+1][0])/math.sqrt((beta**2)+(x_hat[i+1][0]-x_hat[i][0])**2))
        elif i == A.shape[0]-1:
            dt_dx = dt_dx+((x_hat[i][0]-x_hat[i-
1][0])/(math.sqrt((beta**2)+(x_hat[i][0]-x_hat[i-1][0])**2)))
    Ax = A.dot(x_hat)
    Ax_b = Ax-D_hat
    grad = ((2*np.transpose(A).dot(Ax_b))+((alpha**2)*dt_dx))
    grad = np.ndarray.flatten(grad)
    return grad

### Define optimization parameters
lower_bnds = np.zeros((220,1))
upper_bnds = np.ones((220,1))
bnds = np.zeros((220,2))
for i in range(0,220):
    bnds[i][0] = lower_bnds[i][0]
    bnds[i][1] = upper_bnds[i][0]
bnds_tuple = tuple(bnds)

optimization =
scipy.optimize.minimize(J_alpha_beta,x0,args=(A,D_hat,alpha,beta),method="TNC
",bounds=bnds_tuple,jac=grad_J)

x_hat_TV = optimization.x.reshape(220,1)
Error_TV = x_hat_TV-x_t
TV_Reconstruction = plt.figure()
fig = plt.subplot()
fig.plot(Position,x_t,"-r",label="True Data")
fig.plot(Position,x_hat_TV,":b",label="TV Reconstructed Data")
```

```
fig.set_xlabel ("Vector Row Index")
fig.set_ylabel ("Value")
plt.xlabel = "Vector Row Index"
plt.ylabel = "Value"
fig.set_ylim(bottom=0,top=1.2)
box = fig.get_position()
fig.set_position([box.x0, box.y0 + box.height * 0.1, box.width, box.height *
0.9])
fig.legend(loc='upper center', bbox_to_anchor=(0.5, 1.05), ncol=3,
fancybox=True, shadow=True)

TV_error = plt.figure()
fig = plt.subplot()
fig.plot(Position,Error_TV,"-b")
fig.set_xlabel("Vector Row Index")
fig.set_ylabel("Error")

run_time_total = time.time()-start_time
```