MTH 510

Inverse Problems and Data Assimilation

Homework #1

Submitted by Andrew Huffman

October 14, 2019

Portland State
UNIVERSITY

- **Task 1:** Provide the plot of the condition number of the matrix **A** = $\mathbf{B}^k$ for k = 1 : 20.

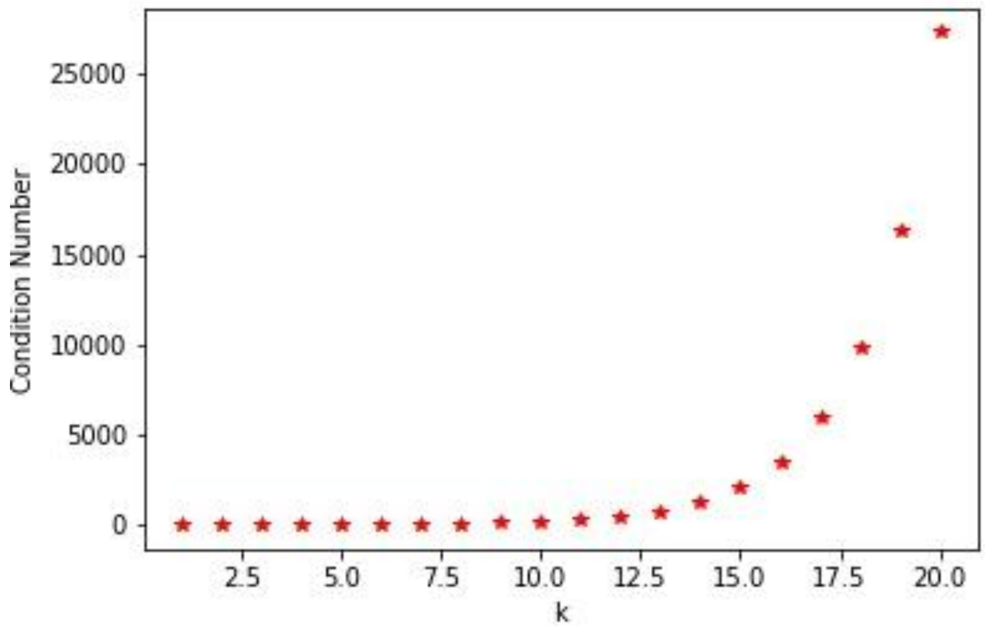

Figure 1: Plot of condition number of B^k for k = 1:20

- **Task 2:** Generate the noisy receive data image $\widehat{D}$ as in (4) and the reconstructed image $\hat{X}$ given by the direct inversion approach (5). Provide the plots of the absolute error and the relative error in the reconstructed image $\hat{X}$ for k = 1:20.

$$\widehat{D} = AX + \varepsilon \qquad (4)$$

$$\hat{X} = A^{-1}\widehat{D} \qquad (5)$$

$$Absolute\ Error = \left\|X - \hat{X}\right\|$$

$$Relative\ Error = \frac{\left\|X - \hat{X}\right\|}{\|X\|}$$

Note: I am using the Frobenius norm

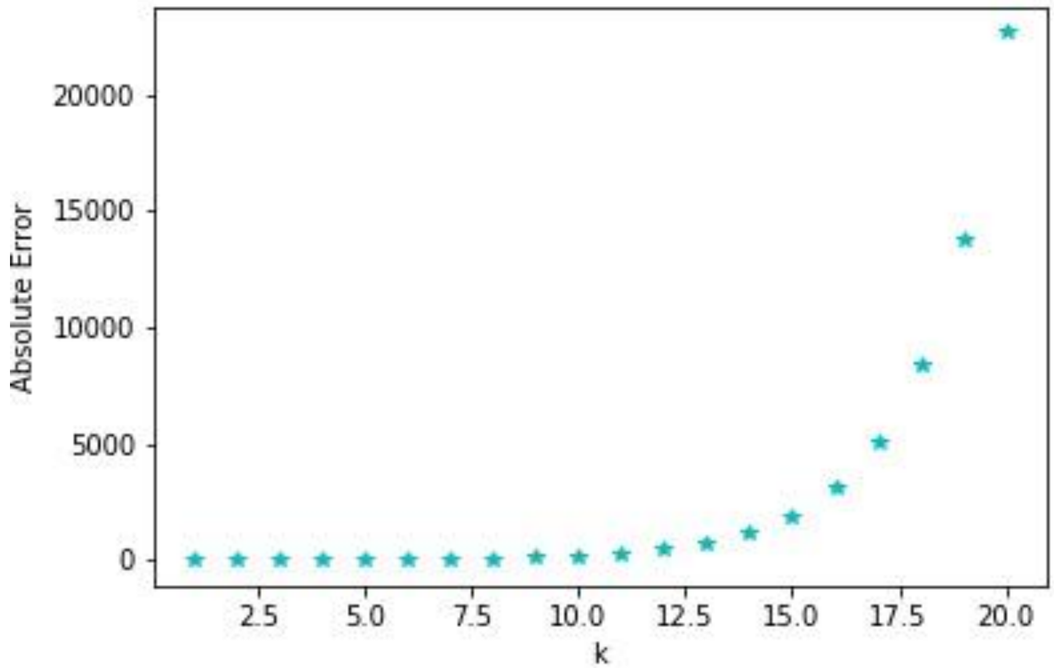

Figure 2: Plot of absolute error vs k, σ = 0.01

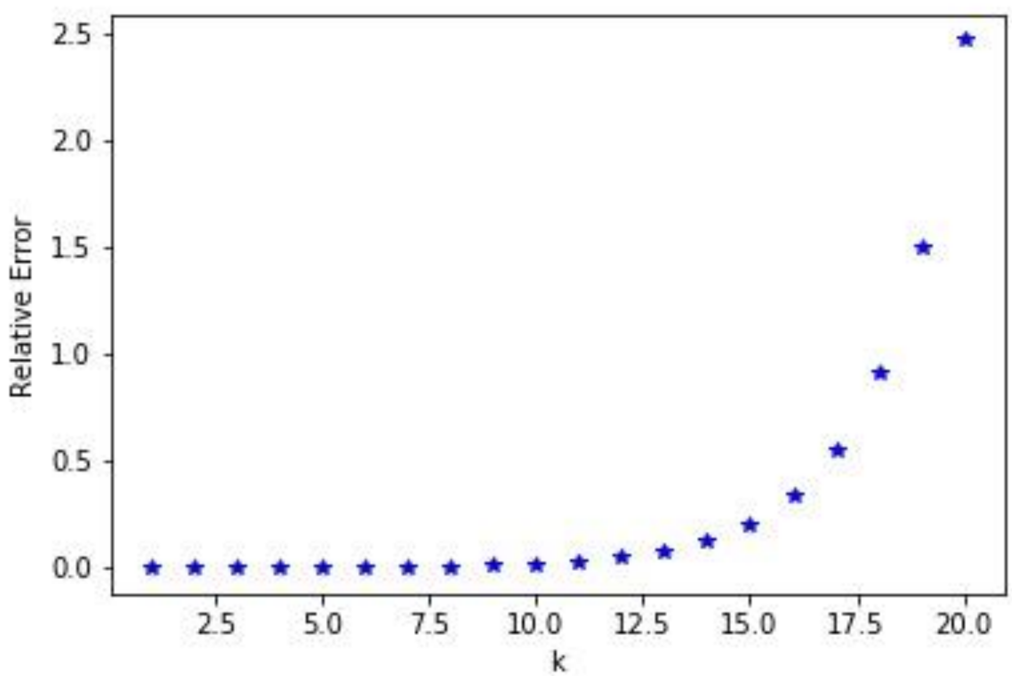

Figure 3: Plot of relative error vs k, σ = 0.01

- **Task 3:** Provide the reconstructed image for k = 1, k =5, and k =20.

Figure 4: Reconstructed image for k = 1, σ = 0.01

Figure 5: Reconstructed image for k = 5, σ = 0.01

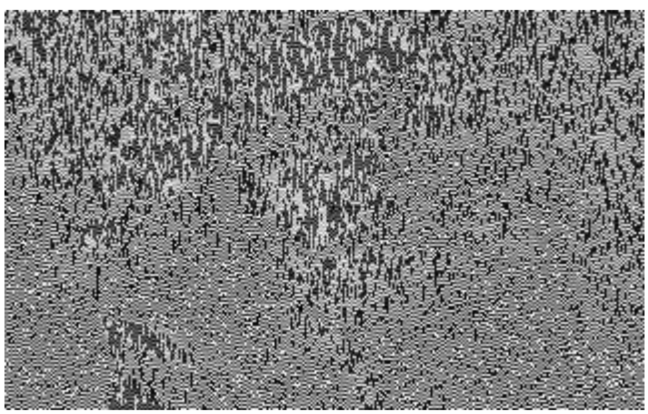

Figure 6: Reconstructed image for k = 20, σ = 0.01

- **Task 4:** Repeat tasks 2 and 3 for a noise matrix with standard deviation 0.1.

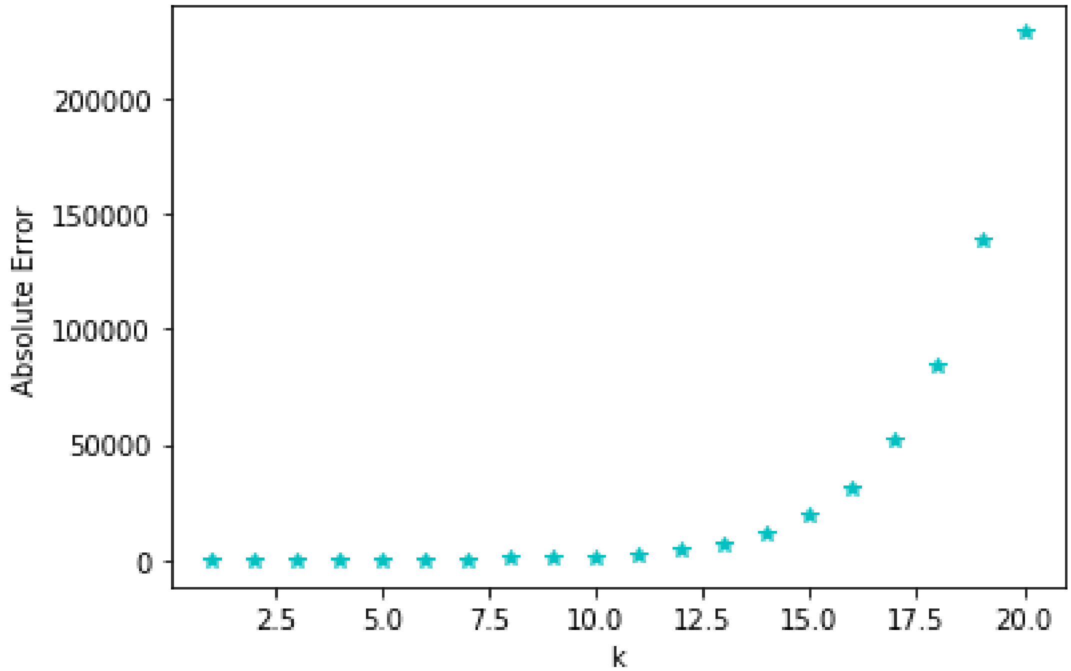

Figure 7: Plot of absolute error vs. k

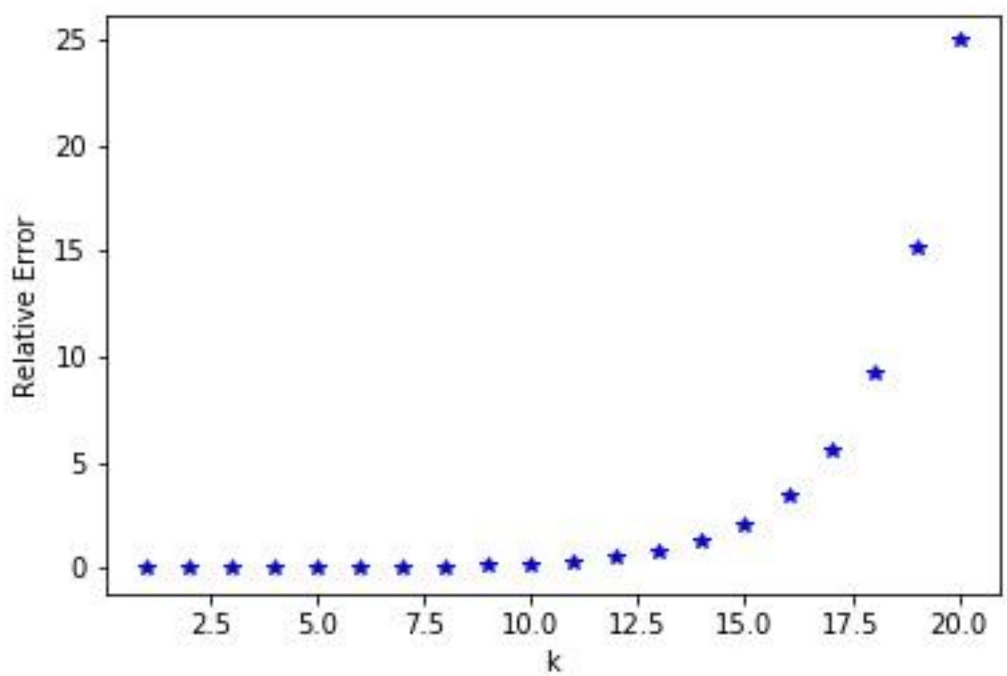

Figure 8: Plot of relative error vs. k

Figure 9: Reconstructed image for k =1, σ = 0.1

Figure 10: Reconstructed image for k =5, σ = 0.1

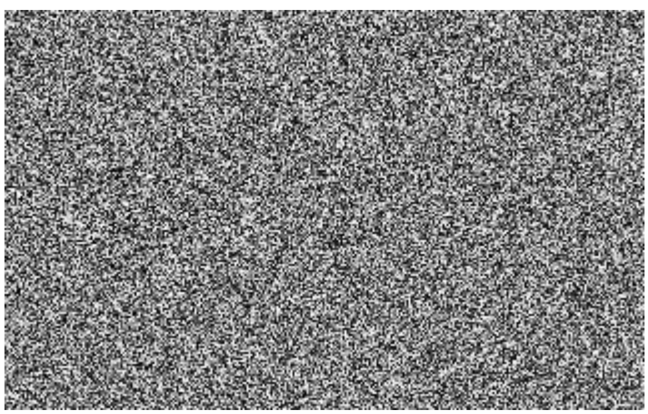

Figure 11: Reconstructed image for k =20, σ = 0.1

Appendix: *Script used to generate solutions*

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 7 11:52:05 2019

@author: Andrew
"""

from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

## Load image and convert to matrix of grayscale form ##
clown = Image.open("clown1.jpg")
clown = clown.convert("L")
image = clown
clown_matrix = np.asarray(clown.getdata(),
dtype=np.float64).reshape((clown.size[1],clown.size[0]))
X = clown_matrix

# Generate noise matrix
eps = np.zeros((200,320))
mu = 0
sigma = 0.1
for i in range(0,image.size[1]):
    for j in range(0,image.size[0]):
        eps[i][j] = np.random.normal(mu,sigma)

# Generate blurring operator and reconstruct blurred image
B = np.zeros((image.size[1],image.size[1]))
L = 0.1
con_num = np.zeros((20,1))
k_vector = np.zeros((20,1))
absolute_error = np.zeros((20,1))
relative_error = np.zeros((20,1))
for i in range (0,200):
    if i<199:
        B[i][i] = (1-(2*L))
        B[i][i+1] = L
        B[i+1][i] = L
    else:
        B[i][i] = (1-(2*L))

for k in range(1,21):
```

```
    k_vector[k-1][0] = k
    A = np.linalg.matrix_power(B,k)
    A_inv = np.linalg.inv(A)
    con_num[k-1][0] = np.linalg.cond(A)
    D_noisy = np.matmul(A,X)+eps
    X_noisy = np.matmul(A_inv,D_noisy)
    error_abs = np.linalg.norm((X-X_noisy),"fro")
    absolute_error[k-1][0] = error_abs
    error_rel = np.linalg.norm((X-X_noisy),"fro")/np.linalg.norm(X,"fro")
    relative_error[k-1][0] = error_rel
    if k == 1:
        clown_noisy = Image.fromarray(X_noisy.astype("uint8"),"L")
        clown_noisy.save("Reconstructed_Image_k1.jpg")
    if k == 5:
        clown_noisy = Image.fromarray(X_noisy.astype("uint8"),"L")
        clown_noisy.save("Reconstructed_Image_k5.jpg")
    if k == 10:
        clown_noisy = Image.fromarray(X_noisy.astype("uint8"),"L")
        clown_noisy.save("Reconstructed_Image_k10.jpg")
    if k == 15:
        clown_noisy = Image.fromarray(X_noisy.astype("uint8"),"L")
        clown_noisy.save("Reconstructed_Image_k15.jpg")
    if k == 20:
        clown_noisy = Image.fromarray(X_noisy.astype("uint8"),"L")
        clown_noisy.save("Reconstructed_Image_k20.jpg")


# Plot condition number and error vs k
f1 = plt.figure()
plt.plot(k_vector,con_num,"r*")
plt.xlabel("k")
plt.ylabel("Condition Number")
f1.savefig("Condition_Number_vs_k.jpg")


f2 = plt.figure()
plt.plot(k_vector,absolute_error,"c*")
plt.xlabel("k")
plt.ylabel("Absolute Error")
f2.savefig("Absolute_Error_vs_k.jpg",edgecolor="k")

f3 = plt.figure()
plt.plot(k_vector,relative_error,"b*")
plt.xlabel("k")
plt.ylabel("Relative Error")
```

```
f3.savefig("Relative_Error_vs_k.jpg",edgecolor="k")
```