Machine Learning

Exercise 5: Application of the linear regression in tomography

December 7, 2014

Group: Sergej Kraft
Elias Roeger
Ekaterina Tikhoncheva

Contents

1 Construction of A

To construct matrix A we adopted the idea, that sensor is places in the middle of the image x and that coordinate center also lies in the middle of the image x.
For each image pixel (x, y), where $x = -\frac{(N-1)}{2} \dots \frac{(N-1)}{2}, y = -\frac{(N-1)}{2} \dots \frac{(N-1)}{2}$ and N is the size of the image, we calculated it's projection on the sensor line. The equation of the sensor line is $y = \tan(180° - \alpha)x$ and the coordinates of the projection (px, py) of the pixel (x, y) are described by:

$$\begin{cases} px = \frac{x+\tan(180° - \alpha)y}{\tan(180° - \alpha)^2+1} \\ py = \tan(180° - \alpha)px \end{cases}$$

According to the values (px, py) one decides, which sensor pixel receives the corresponding ray passing through the pixel (x, y). The sensor pixel, which is the closest to the projection point, is selected. The value of this sensor pixel will be increased by the intensity of the ray. The intensity of the ray is a function

of the distance between image pixel and sensor pixel. We used the function $f(dist) = N - dist$.
If an ray crosses the sensor in between of two centres of the sensor pixels, than it's intensity is divided between those sensor pixel and absorbed value is calculated by the formula $\frac{intensity*(dist_{s_1}+dist_{s_2}-dist_{s_i})}{dist_{s_1}+dist_{s_2}}$, where $intensity$ is the intensity of the ray, $dist_{s_i}$ is the distance between projection (px, py) and center of the sensor pixel $s_i, i = 1, 2$.
The listing of the function $makeA(shape, alphas)$ with $numpy$ arrays can be found below:

```
#-------------------------------------------------------------------------------
def makeA_numpyArray(shape, alphas):
    assert shape[0]==shape[1], 'Expect square matrix'

    N = shape[0]     # NxN shape of the image
    M = len(alphas) # number of alphas
    K = int(N*np.sqrt(2))

    if K%2==0:
        K = K + 1   # sensor length is always a odd number

    sensorcenter = np.zeros((2,K), dtype = np.float32)
    A = np.zeros((M*K, N*N), dtype = np.float32)

    for a in range(0,M):

        alpha = alphas[a]
        ralpha = np.pi*(180 - alpha)/180. # alpha in radians

        for s in range(0,K):
            if alpha==-90 or alpha== 90:
                sensorcenter[0][s]= 0
                sensorcenter[1][s]= np.sign(alpha)*(K -s -1 - (K-1)/2)
            else :
                sensorcenter[0][s]= np.cos(ralpha)*(K - s - 1 - (K-1)/2 )
                sensorcenter[1][s]= np.sin(ralpha)*(K - s - 1 - (K-1)/2 )
        # end for i

        # for each pixel calculate contribution to absorption along a rai
        for i in range(0,N*N):

            # coordinates of the image pixel
            # (coordinate center is shifted to the picture center)
            x = i%N - (N-1)/float(2)
            y = i/N - (N-1)/float(2)

            # px,py - projection of the pixel on the sensor
```

```
            if alpha==-90 or alpha == 90:
                py = y
                px = 0
            else:
                px = (y*np.tan(ralpha)+x)/(np.tan(ralpha)*np.tan(ralpha)+1)
                py = np.tan(ralpha)*px
            # end if

            distToProj = np.abs(x*np.tan(ralpha)-y)/ \
                         np.sqrt(np.tan(ralpha)*np.tan(ralpha) + 1)

            pixelcontribution = N-distToProj

            #distance between projection of (x,y) and centers of the sensorpixel
            dist =  np.zeros(K, dtype = np.float32)
            for s in range(0,K):
                dist[s] = (sensorcenter[0][s]-px)*(sensorcenter[0][s]-px)\
                        + (sensorcenter[1][s]-py)*(sensorcenter[1][s]-py)
                dist[s] = np.sqrt(dist[s])
            #end for s

            # find receiver sensorpixel
            ind = np.argsort(dist)

            if np.abs(dist[ind[0]]-0.5)> 0.1:
                A[a*K+ind[0]][i] += pixelcontribution
            else :
                # if ray meets sensor in between of two sensor pixels
                # intensity of the ray is devided between those pixels
                A[a*K+ind[0]][i] += pixelcontribution*(dist[ind[0]]/ \
                                            (dist[ind[0]]+dist[ind[1]]))
                A[a*K+ind[1]][i] += pixelcontribution*(dist[ind[1]]/ \
                                            (dist[ind[0]]+dist[ind[1]]))
            #end if
        #end for i
    #end for alpha

    return A
# end def makeA
#-----------------------------------------------------------------------------
```

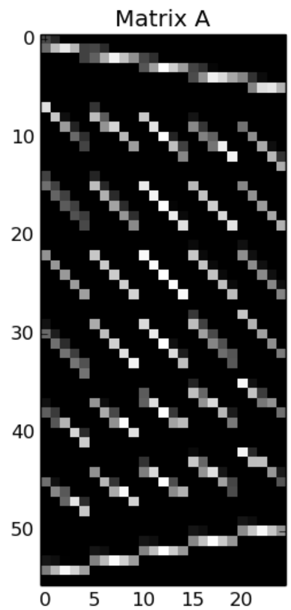

Figure 1: Matrix A for the image 5×5 and angles $[-77,\ -33,\ -12,\ 3,\ 21,\ 42,\ 50,\ 86]$

2 Reconstruction of the image

The results of the image reconstruction for two sets of arrays (y, α) are shown on the images 2 and 3.

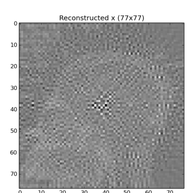

Figure 2: Reconstruction of the image x (77×77)

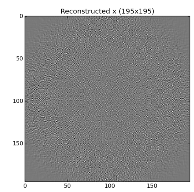

Figure 3: Reconstruction of the image x (195×195)

From the first image we can recognize the patient H.S., but because of the image quality we cann't state the cause of the headache by the patient. Probably the reason for this is the calculation of the Ray intensity. We used simple line function (see section 1).

3 Minimization of the radiation doze

By the image size 77×77 we tried to reduce in half projection number. In this case it was not possible to recognize anything (see image 4).

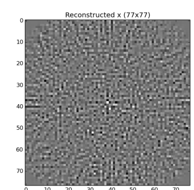

Figure 4: Reconstruction of the image x (77×77)

4 Listing *ex05.py*

../ex05.py

```
"""
Exercise 5 : Linear regression

"""

import time
import numpy as np
import matplotlib.pyplot as plot
from scipy.sparse import dok_matrix
from scipy.sparse.linalg import lsqr
```

```python
#-----------------------------------------------------------------------------
def makeA_numpyArray(shape, alphas):
    assert shape[0]==shape[1], 'Expect square matrix'

    N = shape[0]    # NxN shape of the image
    M = len(alphas) # number of alphas
    K = int(N*np.sqrt(2))

    if K%2==0:
        K = K + 1   # sensor length is always a odd number

    sensorcenter = np.zeros((2,K), dtype = np.float32)
    A = np.zeros((M*K, N*N), dtype = np.float32)

    for a in range(0,M):

        alpha = alphas[a]
        ralpha = np.pi*(180 - alpha)/180. # alpha in radians

        for s in range(0,K):
            if alpha==-90 or alpha== 90:
                sensorcenter[0][s]= 0
                sensorcenter[1][s]= np.sign(alpha)*(K -s -1 - (K-1)/2)
            else :
                sensorcenter[0][s]= np.cos(ralpha)*(K - s - 1 - (K-1)/2 )
                sensorcenter[1][s]= np.sin(ralpha)*(K - s - 1 - (K-1)/2 )
        # end for i

        # for each pixel calculate contribution to absorption along a rai
        for i in range(0,N*N):

            # coordinates of the image pixel
            # (coordinate center is shifted to the picture center)
            x = i%N - (N-1)/float(2)
            y = i/N - (N-1)/float(2)

            # px,py - projection of the pixel on the sensor
            if alpha==-90 or alpha == 90:
                py = y
                px = 0
            else:
                px = (y*np.tan(ralpha)+x)/(np.tan(ralpha)*np.tan(ralpha)+1)
                py = np.tan(ralpha)*px
            # end if

            distToProj = np.abs(x*np.tan(ralpha)-y)/ \
                         np.sqrt(np.tan(ralpha)*np.tan(ralpha) + 1)

            pixelcontribution = N-distToProj
```

```python
            #distance between projection of (x,y) and centers of the sensorpixel
            dist =  np.zeros(K, dtype = np.float32)
            dist = np.sqrt(np.square(sensorcenter[0][:]-px)
                 + np.square(sensorcenter[1][:]-py))

            # find receiver sensorpixel
            indMin1 = np.argmin(dist)
            dist1 = np.delete(dist, [indMin1])
            indMin2 = np.argmin(dist1)

            A[a*K+indMin1][i] += pixelcontribution*dist1[indMin2]/ \
                                            (dist[indMin1]+dist1[indMin2])
            A[a*K+indMin2][i] += pixelcontribution*dist[indMin1]/ \
                                                (dist[indMin1]+dist1[indMin2])

#            if np.abs(dist[indMin1] - 0.5)<= 0.1:
#                # if ray meets sensor in between of two sensor pixels
#                # intensity of the ray is devided between those pixels
#
#                A[a*K+indMin1][i] += pixelcontribution*dist1[indMin2]/ \
#                                                (dist[indMin1]+dist1[indMin2])
#                A[a*K+indMin2][i] += pixelcontribution*dist[indMin1]/ \
#                                                (dist[indMin1]+dist1[indMin2])
#            else :
#                A[a*K+indMin1][i] += pixelcontribution
            #end if
        #end for i
    #end for alpha

    return A
# end def makeA
#------------------------------------------------------------------------------

def makeA(shape, alphas):
    assert shape[0]==shape[1], 'Expect square matrix'

    N = shape[0]    # NxN shape of the image
    M = len(alphas) # number of alphas
    K = int(N*np.sqrt(2))

    if K%2==0:
        K = K + 1   # sensor length is always a odd number

    sensorcenter = np.zeros((2,K), dtype = np.float32)
    A = dok_matrix((M*K, N*N), dtype = np.float32)

    for a in range(0,M):

        alpha = alphas[a]
        ralpha = np.pi*(180 - alpha)/180. # alpha in radians
```

```
    for s in range(0,K):
        if alpha==-90 or alpha== 90:
            sensorcenter[0][s]= 0
            sensorcenter[1][s]= np.sign(alpha)*(K -s -1 - (K-1)/2)
        else :
            sensorcenter[0][s]= np.cos(ralpha)*(K - s - 1 - (K-1)/2 )
            sensorcenter[1][s]= np.sin(ralpha)*(K - s - 1 - (K-1)/2 )
    # end for i

    # for each pixel calculate contribution to absorption along a rai
    for i in range(0,N*N):
        # coordinates of the image pixel
        # (coordinate center is shifted to the picture center)
        x = i%N - (N-1)/float(2)
        y = i/N - (N-1)/float(2)

        # px,py - projection of the pixel on the sensor
        if alpha==-90 or alpha == 90:
            py = y
            px = 0
        else:
            px = (y*np.tan(ralpha)+x)/(np.tan(ralpha)*np.tan(ralpha)+1)
            py = np.tan(ralpha)*px
        # end if

        distToProj = np.abs(x*np.tan(ralpha)-y)/ \
                     np.sqrt(np.tan(ralpha)*np.tan(ralpha) + 1)

        pixelcontribution = N-distToProj

        #distance between projection of (x,y) and centers of the sensorpixel
        dist =  np.zeros(K, dtype = np.float32)
        dist = np.sqrt(np.square(sensorcenter[0][:]-px)
             + np.square(sensorcenter[1][:]-py))

        # find receiver sensorpixel
        indMin1 = np.argmin(dist)
        dist1 = np.delete(dist, [indMin1])
        indMin2 = np.argmin(dist1)

        A[a*K+indMin1, i] += pixelcontribution*dist1[indMin2]/ \
                                            (dist[indMin1]+dist1[indMin2])
        A[a*K+indMin2, i] += pixelcontribution*dist[indMin1]/ \
                                                (dist[indMin1]+dist1[indMin2])

#            if np.abs(dist[indMin1] - 0.5)<= 0.1:
#                # if ray meets sensor in between of two sensor pixels
#                # intensity of the ray is devided between those pixels
#
```

```
#                A[a*K+indMin1, i] += pixelcontribution*dist1[indMin2]/ \
#                                                (dist[indMin1]+dist1[indMin2])
#                A[a*K+indMin2, i] += pixelcontribution*dist[indMin1]/ \
#                                                (dist[indMin1]+dist1[indMin2])
#            else :
#                A[a*K+indMin1][i] += pixelcontribution
            #end if
        #end for i
    #end for alpha

    return A.tocsc()
# end def makeA
#------------------------------------------------------------------------------
#                              Main Function
def main():
    plot.close('all')

    print
    print "Construction of A"
    print

    Atest = makeA_numpyArray([5,5], [-77, -33, -12, 3, 21, 42, 50, 86])

    f = plot.figure()
    plot.gray()
    plot.imshow(Atest, interpolation = 'nearest')
    plot.title("Matrix A")
    plot.show()
    f.savefig("matrixA.png")


    print
    print "Reconstruction of the Image"
    print

    print "Experiment 1: x = 77x77"

    N = 77
    y_77 = np.load('y_77_77.npy')
    alphas_77 = np.load('y_77_alphas.npy')

    # construct matrix A
    tstart = time.time()
    A_77 = makeA([N,N], alphas_77)
    tstop = time.time()
    print "makeA 77 took {} sec". format(tstop-tstart)

    np.save('A_77.npy', A_77)

    # reconstruct x
```

```
tstart = time.time()
resultLSQR = lsqr(A_77, y_77)
tstop = time.time()
print "reconstruct x 77 took {} sec".format(tstop-tstart)

x_77 = resultLSQR[0]
x_77 = np.reshape(x_77, (N,N))
np.save('x_77.npy', x_77)

f = plot.figure()
plot.gray()
plot.imshow(x_77, interpolation = 'nearest')
plot.title("Reconstructed x (77x77)")
plot.show()
f.savefig("x_77.png")

print "Experiment 2: x = 195x195"

N = 195
y_195 = np.load('y_195_195.npy')
alphas_195 = np.load('y_195_195_alphas.npy')

# construct matrix A
tstart = time.time()
A_195 = makeA([N,N], alphas_195)
tstop = time.time()
print "makeA 195 took {} sec".format(tstop-tstart)

np.save('A_195.npy', A_195)

# reconstruct x
tstart = time.time()
resultLSQR = lsqr(A_195, y_195)
tstop = time.time()

print "reconstruct x 195 took {} sec".format(tstop-tstart)

x_195 = resultLSQR[0]
x_195 = np.reshape(x_195, (N,N))
np.save('x_195.npy', x_195)

f = plot.figure()
plot.gray()
plot.imshow(x_195, interpolation = 'nearest')
plot.title("Reconstructed x (195x195)")
plot.show()
f.savefig("x_195.png")

print
print "Minimization of the radiation dose"
```

```
    print

    print "Experiment 1: x = 77x77"

    N = 77
    y_77 = np.load('y_77_77.npy')

    alphas_77 = np.load('y_77_alphas.npy')
    alphas_77 = alphas_77[0:alphas_77.shape[0]:2]

    y_77 = y_77[0:y_77.shape[0]:2]

    # construct matrix A
    tstart = time.time()
    A_77 = makeA([N,N], alphas_77)
    tstop = time.time()

    print "makeA 77 took {} sec". format(tstop-tstart)

    np.save('A_77r.npy', A_77)

    # reconstruct x
    tstart = time.time()
    resultLSQR = lsqr(A_77, y_77)
    tstop = time.time()
    print "reconstruct x 77 took {} sec". format(tstop-tstart)

    x_77 = resultLSQR[0]
    x_77 = np.reshape(x_77, (N,N))
    np.save('x_77r.npy', x_77)

    f = plot.figure()
    plot.gray()
    plot.imshow(x_77, interpolation = 'nearest')
    plot.title("Reconstructed x (77x77)")
    plot.show()
    f.savefig("x_77r.png")
#
 #end main


if __name__ == "__main__":
    main()
```