linear-regression

December 6, 2017

```
In [1]: import numpy as np
        import pandas as pd
        from matplotlib import pyplot as plt
        from scipy import sparse as sp
        from scipy.sparse.linalg import lsqr
        import time
        %pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

1 Question 1: Constructing X

```
In [2]: def construct_X(M, alphas, Np=None):
            # calculating Np if not given
            if Np == None:
                Np_estimate = int(np.floor(np.sqrt(2)*M))
                Np = Np_estimate if Np_estimate % 2 == 1 else Np_estimate + 1
                print('Use Np={:d}'.format(Np))

            # defining the dimensions
            D = M*M
            N = Np * len(alphas)
            # creating the normal vectors
            n = np.array([[np.cos(alpha*np.pi/180), -np.sin(alpha*np.pi/180)] for alpha in alph
            # coordinates of detector rotation center
            # M - 1, because indexing starting from 0
            s0 = np.array([(M-1)/2, (M-1)/2])

            beta_flat_index = np.arange(D) # just an array with all indices of beta

            # C contains the vector from center of rotation to beta element
            C = np.empty((2, D)) # create C
            C[0,:] = -s0[0] + np.mod(beta_flat_index, M) # x-value: x(beta) = modulo
            C[1,:] = -s0[1] + np.floor_divide(beta_flat_index, M) #y-value: y(beta) = floor_di
```

```
#fig, ax = plt.subplots(1, 1)
#ax.quiver([s0[0] for _ in range(D)], [s0[1] for _ in range(D)], C[0], C[1], angle
#ax.set_xlim(-1, M)
#ax.set_ylim(-1, M)
#ax.set_title('Construction of C')
#plt.show()

#fig, ax = plt.subplots(1, 1)
#for i in range(len(alphas)):
#    ax.quiver(s0[0], s0[1], n[i][0], n[i][1], angles='xy', scale_units='xy', scal
#ax.set_xlim(s0[0]-1.2, s0[0]+1.2)
#ax.set_ylim(s0[1]-1.2, s0[1]+1.2)
#ax.set_title('Alphas')
#plt.show()

# np.tensordot gives the projected length of C vectors on
# Since they are measured from the rotation center, 0 corresponds to the rotation
p = (Np-1)/2 - np.tensordot(n, C, axes=((1), (0)))
# TODO: what to do with values smaller than 0?

# calculate weights and indices
# detector_index_1 is the integral part of p, i.e. the first (most left) sensor th
# beta is contributing to detector_index_1 with weight_1 = 1 - weight_2, where wei
# therefore weight_2 is the fractional part of p
# the neighbouring element of detector_index_2 is the one right of it, so just + 1
weight_2, detector_index_1 = np.modf(p)
weight_1 = 1 - weight_2
detector_index_2 = detector_index_1 + 1

# now it can happen, that some are out of bounds. Here we just replace these value
# TODO: performance?
mask_detector_index_1 = np.logical_or(detector_index_1 < 0, detector_index_1 >= Np
weight_1[mask_detector_index_1] = 0
detector_index_1[mask_detector_index_1] = 0 # just to avoid later errors
mask_detector_index_2 = np.logical_or(detector_index_2 < 0, detector_index_2 >= Np
weight_2[mask_detector_index_2] = 0
detector_index_2[mask_detector_index_2] = 0 # just to avoid later index errors


# merge arrays
weights = np.array([])
weights = np.append(weights, [weight_1[angle_index] for angle_index in range(len(a
weights = np.append(weights, [weight_2[angle_index] for angle_index in range(len(a
# this is what is called i_indices
detector_indices = np.array([])
detector_indices = np.append(detector_indices, [Np*angle_index + detector_index_1[
detector_indices = np.append(detector_indices, [Np*angle_index + detector_index_2[
```

```
    # create j indices
    beta_indices = np.array([])
    # we have to flip the beta_flat_index array, because otherwise the picture is upsi
    beta_indices = np.append(beta_indices, [beta_flat_index[::-1] for _ in range(len(al
    beta_indices = np.append(beta_indices, [beta_flat_index[::-1] for _ in range(len(al

    # i hope duplicate entries will sum
    X = sp.coo_matrix((weights, (detector_indices, beta_indices)), shape=(N, D), dtype
    return X
```

1.1 Checking example

```
In [3]: # There is a mistake in the exercise sheet. The provided example is Np=15 (result of e
        example_x = construct_X(10, [-33, 1, 42])#, Np=9)
        # compare with provided
        provided_example_x = np.load('hs_tomography/X_example.npy')
        print('Me == Example?: ', np.array_equal(example_x.toarray(), provided_example_x))
```

```
Use Np=15
Me == Example?:  True
```

```
In [4]: def get_beta(M, Np, alphas, y, error=1e-5):
            t0 = time.time()
            x = construct_X(M, alphas, Np)
            t1 = time.time()
            print('Constructed X in {:f}s'.format(t1 - t0))
            print('Sparsity:', x.nnz/(x.get_shape()[0]*x.get_shape()[1]))
            t0 = time.time()
            beta = lsqr(x, y, atol=error, btol=error)[0]
            t1 = time.time()
            print('Solved for beta in {:f}s'.format(t1 - t0))
            return beta
```

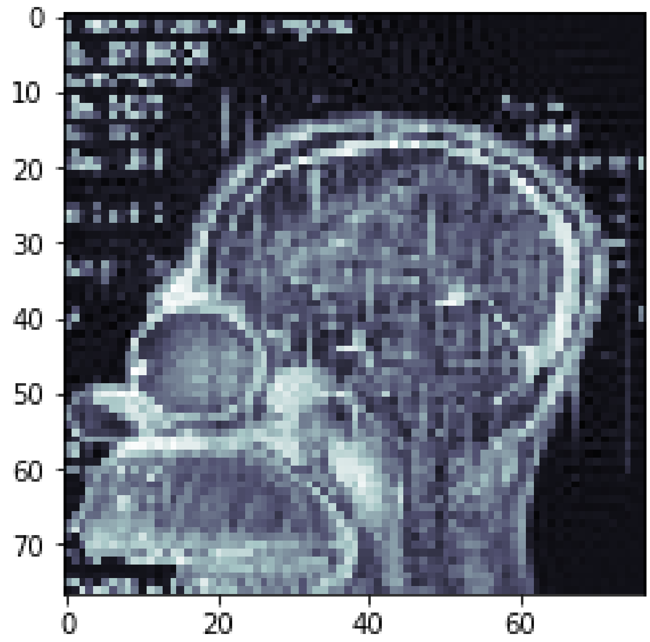
2 Question 2: Recovering images

2.1 Low resolution

```
In [5]: alphas_77 = np.load('hs_tomography/alphas_77.npy')
        y_77 = np.load('hs_tomography/y_77.npy')
        beta_77 = get_beta(77, 109, alphas_77, y_77)
```

```
Constructed X in 0.098470s
Sparsity: 0.01834862385321101
Solved for beta in 1.120535s
```

```
In [6]: fig, ax = plt.subplots(1, 1)
        ax.imshow(beta_77.reshape(77, 77), cmap='bone')
        plt.show()
```

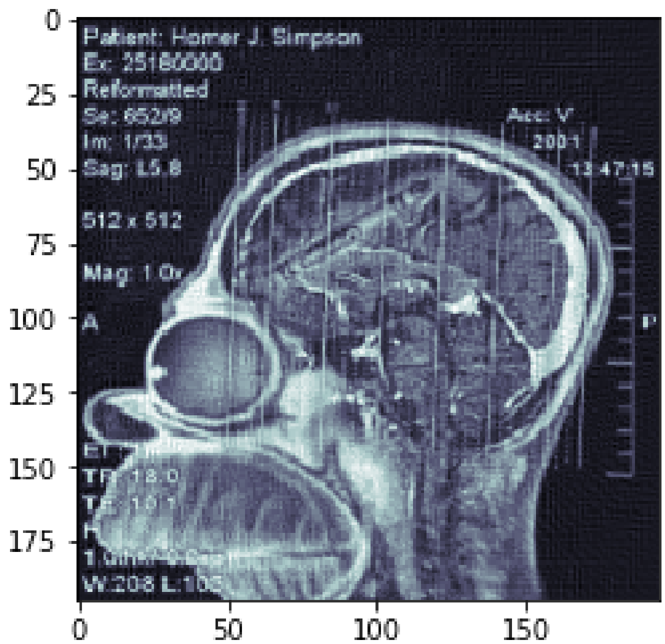


2.2 High resolution

```
In [7]: alphas_195 = np.load('hs_tomography/alphas_195.npy')
        y_195 = np.load('hs_tomography/y_195.npy')
        beta_195 = get_beta(195, 275, alphas_195, y_195)
```

```
Constructed X in 1.127091s
Sparsity: 0.007272727272727273
Solved for beta in 19.894776s
```

```
In [8]: fig, ax = plt.subplots(1, 1)
        ax.imshow(beta_195.reshape(195, 195), cmap='bone')
        plt.show()
```



3 Question 3: Reduce dosis

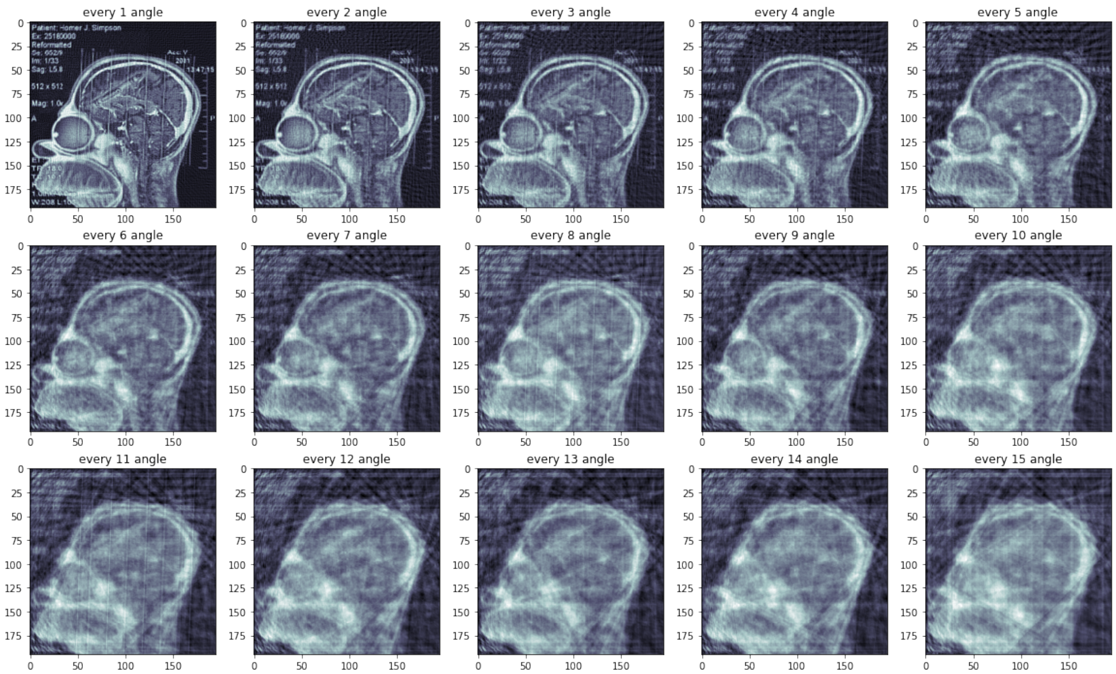
```
In [9]: sorted_alphas = np.argsort(alphas_195)

        fig, ax = plt.subplots(3, 5, figsize=(20, 12))

        for j in range(1, 16):
            y_reduced = np.array([])
            y_reduced = np.append(y_reduced, [y_195[i*275:(i+1)*275] for i in range(0, len(alph
            beta = get_beta(195, 275, alphas_195[sorted_alphas][::j], y_reduced, error=1e-4)
            ax[(j - 1)//5, (j-1)%5].imshow(beta.reshape(195, 195), cmap='bone')
            ax[(j - 1)//5, (j-1)%5].set_title('every {:d} angle'.format(j), fontsize='12')
        plt.show()
```

```
Constructed X in 0.803263s
Sparsity: 0.007272727272727273
Solved for beta in 4.698782s
Constructed X in 0.471093s
Sparsity: 0.007272727272727273
Solved for beta in 1.979965s
Constructed X in 0.308867s
Sparsity: 0.007272727272727273
Solved for beta in 1.248384s
Constructed X in 0.217159s
Sparsity: 0.007272727272727273
```

```
Solved for beta in 0.965225s
Constructed X in 0.182880s
Sparsity: 0.007272727272727273
Solved for beta in 0.710432s
Constructed X in 0.148927s
Sparsity: 0.007272727272727273
Solved for beta in 0.592606s
Constructed X in 0.136932s
Sparsity: 0.007272727272727273
Solved for beta in 0.515982s
Constructed X in 0.124040s
Sparsity: 0.007272727272727273
Solved for beta in 0.457629s
Constructed X in 0.091393s
Sparsity: 0.007272727272727273
Solved for beta in 0.393850s
Constructed X in 0.083721s
Sparsity: 0.007272727272727273
Solved for beta in 0.337122s
Constructed X in 0.076381s
Sparsity: 0.007272727272727273
Solved for beta in 0.319285s
Constructed X in 0.069027s
Sparsity: 0.007272727272727273
Solved for beta in 0.319436s
Constructed X in 0.060316s
Sparsity: 0.007272727272727273
Solved for beta in 0.269991s
Constructed X in 0.048987s
Sparsity: 0.007272727272727273
Solved for beta in 0.266880s
Constructed X in 0.052239s
Sparsity: 0.007272727272727273
Solved for beta in 0.317231s
```



So it should be sufficient to take only 30% (every third angle).

```
In [ ]:
```