HW3

June 7, 2022

1 EE798 HW3

1.1 Kutay Ugurlu

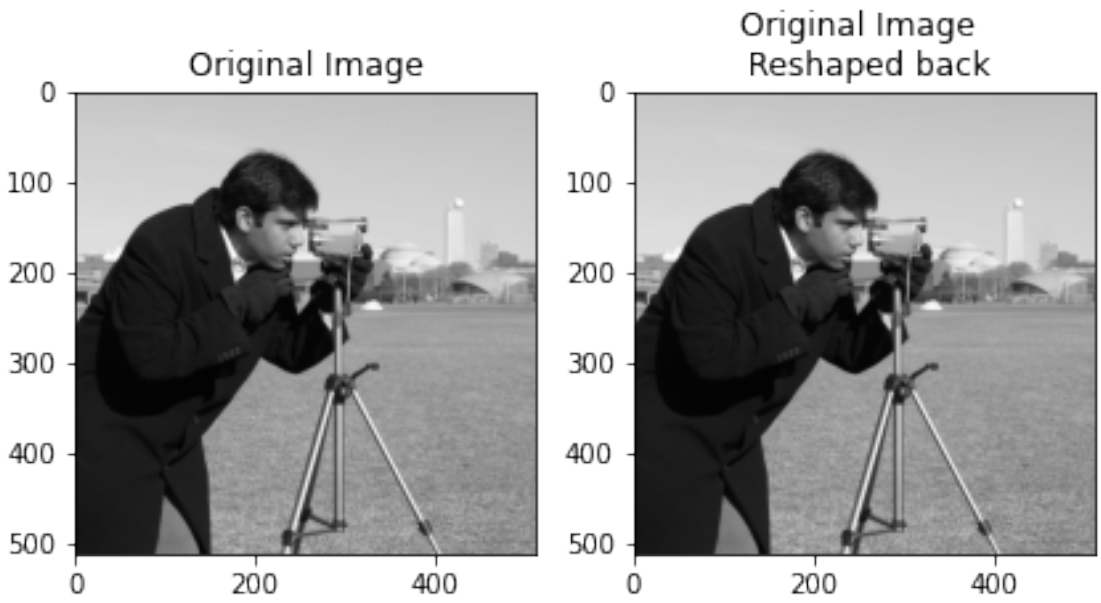
1.1.1 To achieve the notebook and the MATLAB scripts, click here.

[ ]:
```
from scipy.linalg import toeplitz, dft, inv, eigvals, eig, kron, eigh
from scipy.signal import convolve2d, convolve
from sklearn.preprocessing import normalize
import numpy as np
from numpy.fft import fft, ifft, fft2, ifft2
import matplotlib.pyplot as plt
from skimage.data import camera
```

2 Q1 Matrix Representation of 2D Convolution

2.1 Q1 a

[ ]:
```
I = camera()
I_flatten = I.flatten(order="F")
I_back = np.reshape(I,(I.shape))
plt.subplot(1,2,1)
plt.imshow((I),cmap="gray")
plt.title("Original Image")
plt.subplot(1,2,2)
plt.imshow((I_back),cmap="gray")
plt.title("Original Image \n Reshaped back")
plt.tight_layout()
```



2.1.1 The vectors stacked in matrix H are flattened version of the convolved vectors of basis pixels. Since flattened basis vectors are lexicographically ordered, we observed shifted versions of block of the output vectors in the matrix, hence in a Toeplitz form.

2.1.2 This matrix may be too huge to store for a huge image size.

3 Q1 b

3.0.1 Given that H is of size $(P1, P2)$ and image of size $(N1, N2)$, FFT points of

$$(N1 + P1 - 1, N2 + P2 - 1) = (L_1, L_2)$$

should be used to have linear convolution.

3.0.2 If FFT points (N_1, N_2) are higher than the image sizes, the images are padded zeros until they have FFT points size, as in the case of linear convolution calculation. On the other hand, if we were to use fewer points FFT, then the images are rolled onto themselves in the following manner:

$$\hat{x}(m, n) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} x(m + N1i, n + N2j)$$

[ ]:
```
def cconv2(x:np.ndarray, y:np.ndarray,N1:int,N2:int):
    sizes = (N1,N2)
    return ifft2(np.multiply(fft2(x,sizes),fft2(y,sizes)))
```

3.0.3 Tests

[ ]:
```
for _ in range(200):
    N1 = np.random.randint(1,5)
    N2 = np.random.randint(1,10)
    N3 = np.random.randint(1,20)
    x = np.random.randint(0,100,(N1,N1))
    y = np.random.randint(0,100,(N2,N3))
    cconv_size_1 = N1+N2-1
    cconv_size_2 = N1+N3-1
    res = convolve2d(x,y,mode="full")
    res2 = np.round(cconv2(y,x,cconv_size_1,cconv_size_2))
    assert np.sum(np.isclose(res,res2)) == res.size
print("All tests are success!")
```

```
All tests are success!
```

4 Q1 c

[ ]:
```
def cconvmtx2(h:np.ndarray, image_shape, N1, N2):
    L1,L2 = image_shape
    H = np.empty((N1*N2,0))
    image_size = L1*L2
    for i in range(image_size):
        row = np.mod(i,L1)
        col = i//L1
        basis_vec = np.zeros((L1,L2))
        basis_vec[row,col] = 1
        basis_vec_output = cconv2(h,basis_vec,N1,N2)
        transformed_basis_vec = basis_vec_output.flatten(order="F")
        H = np.column_stack((H, transformed_basis_vec))
    return H
```

4.1 Verification

[ ]:
```
for trial in range(100):

    P1 =  np.random.randint(1,3)
    P2 =  np.random.randint(1,3)
    L1 = np.random.randint(P1,10)
    L2 = np.random.randint(P2,10)
    N1 = np.random.randint(L1,20)
    N2 = np.random.randint(L2,20)


    h = np.random.randint(1,5,(P1,P2))
    test_image = np.random.randint(1,100,(L1,L2))
```

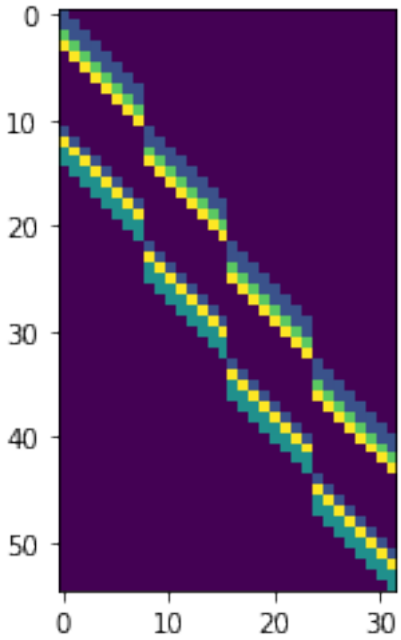
```
    res = cconv2(h,test_image, N1, N2)
    H = cconvmtx2(h, test_image.shape, N1, N2)
    res2 = np.reshape(H.dot(test_image.flatten(order="F")),(N1,N2),order="F")
    assert np.sum(np.isclose(res,res2)) == res.size

print("All tests are success!")
```

```
All tests are success!
```

A linear convolution example ↓

[ ]:
```
P1 =  np.random.randint(1,6)
P2 =  np.random.randint(1,6)
L1 = np.random.randint(P1,10)
L2 = np.random.randint(P2,10)
N1 = L1 + P1 - 1
N2 = L2 + P2 - 1


h = np.random.randint(1,5,(P1,P2))
test_image = np.random.randint(1,100,(L1,L2))
res = cconv2(h,test_image, N1, N2)
H = cconvmtx2(h, test_image.shape, N1, N2)
plt.imshow(np.real(H))
```

[ ]:
```
<matplotlib.image.AxesImage at 0x270aacb49d0>
```



5 Q1 d

[ ]:
```
def dftmtx2(N1:int, N2:int):
    F = np.empty((N1*N2,0))
    for i in range(N1*N2):
        row = np.mod(i,N1)
        col = i//N1
        basis_vec = np.zeros((N1,N2))
        basis_vec[row,col] = 1
        basis_vec_output = fft2(basis_vec,(N1,N2))
        transformed_basis_vec = basis_vec_output.flatten(order="F")
        F = np.column_stack((F, transformed_basis_vec))
    return F
```

5.1 Verification

5.1.1 $(2DFFT(I))_{flat} == FI_{flat}$

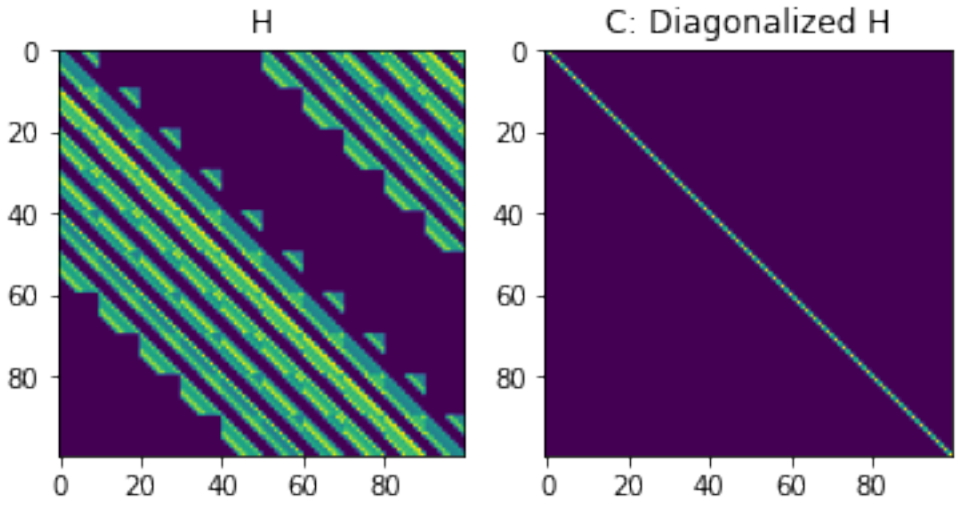
[ ]:
```
for trial in range(50):

    N1 = np.random.randint(1,20)
    N2 = np.random.randint(1,20)


    test_image = np.random.randint(1,100,(N1,N2))
    res = fft2(test_image, (N1, N2))
    F = dftmtx2(N1, N2)
    res2 = np.reshape(F.dot(test_image.flatten(order="F")),(N1,N2),order="F")
    F2 =  dftmtx2(N1, N1)
    assert np.sum(np.isclose(res,res2)) == res.size and np.sum(np.
 ↪isclose(inv(F2),F2.conjugate().T/N1**2)) == F2.size

print("All tests are success!")
```

```
All tests are success!
```

5.1.2 Since FFT matrix entries are symmetrical with respect to variables $n1$, $k1$ and $n2$, $k2$ the matrix is symmetric, *i.e.* $ F = F^T$. Inverse FFT matrix F^{-1} have just conjugated entries scaled with $\frac{1}{N_1 N_2}$ of F. Hence $F^{-1} = \frac{F^H}{N_1 N_2}$

[ ]:
```
for _ in range(100):

    N = np.random.randint(2,20,(1,1)).item()
    P = np.random.randint(1,N,(1,1)).item()
    h = np.random.randint(40,100,(P,P))
    Image = np.random.randint(1,30,(N,N))
    H = cconvmtx2(h,Image.shape,N,N)
```

```
    h = np.pad(h,((0,N-P),(0,N-P)))
    F = dftmtx2(N,N)
    C = F.dot(H).dot(F.conjugate().T)/N**2
    diagC = np.diag(C)
    diagC_check = fft2(h,(N,N)).flatten(order="F")
    assert np.sum(np.isclose(diagC,diagC_check)) == diagC_check.size # Check
 ↪whether diagonalized matrix stores the flattened FFT2 coefficients
    x = Image.flatten(order="F")
    y = H.dot(x)
    Y = F.dot(y)
    H = F.dot(h.flatten(order="F"))
    X = F.dot(x)
    assert np.sum(np.isclose(Y,H*X)) == Y.size # Check whether the given
 ↪relation is true

H = cconvmtx2(h,Image.shape,N,N)
F = dftmtx2(N,N)
C = F.dot(H).dot(F.conjugate().T)/N**2
plt.subplot(1,2,1)
plt.imshow(np.real(H))
plt.title("H")
plt.subplot(1,2,2)
plt.imshow(np.abs(normalize(np.abs(C),norm="max")))
plt.title("C: Diagonalized H")
plt.show()
```



5.2 $\tilde{\mathbf{C}}$ is a diagonal matrix storing the flattened version of 2D FFT coefficients of the filter.

5.3

$$diag(\tilde{\mathbf{C}}) = 2DFFT(H)_{flat}$$

5.4 In addition, $\tilde{\mathbf{C}} = \frac{FHF^H}{N_1 N_2}$ corresponds to the eigen decomposition of matrix H, hence diagonal matrix $\tilde{\mathbf{C}}$ corresponds to the matrix storing the eigen values. Therefore, eigenvalues of H are the 2-dimensional FFT coefficients of the filter H.

5.5 Let b_i's be the basis vectors of $(N1, N2)$ grid and (:) operator be the lexicographic ordering operator, F_N is the N dimensional FFT operator.

2D FFT matrix F_2 can be considered as follows:

$$F_2 = [F_2 b_1(:)\ F_2 b_2(:)\ \ ...\ \ F_2 b_N(:)]$$

Circular convolution matrix H has columns as follows:

$$[{F_2}^{-1} diag(F_2 h(:)) F_2 b_1(:)\ {F_2}^{-1} diag(F_2 h(:)) F_2 b_2(:)\ \ ... {F_2}^{-1} diag(F_2 h(:)) F_2 b_N(:)]$$

Then $\tilde{\mathbf{C}} = \frac{F_2 H {F_2}^H}{N_1 N_2}$ can be rewritten as:

$$[\underbrace{F_2 {F_2}^{-1}}_{I} diag(\underbrace{F_2 h(:)}_{H}) F_2 b_1(:) {F_2}^{-1}\ \ \underbrace{F_2 {F_2}^{-1}}_{I} diag(\underbrace{F_2 h(:)}_{H}) F_2 b_2(:) {F_2}^{-1} ...]$$

The terms can be collected as follows:

$$diag(H)\ \underbrace{\underbrace{[F_2 b_1(:) F_2 b_2(:) F_2 b_3(:) ...]}_{F_2}\ {F_2}^{-1}}_{I}$$

Hence

$$\tilde{\mathbf{C}} = diag(H)$$

IF the problem was not shift invariant, we could not utilize convolution matrix to define the relationship between the input and output of the system. Hence, utilization of DFT via circular convolution to conduct linear convolution would become impossible.

6 Q2 a

$$\delta(t - s_1 \cos\theta - s_2 \sin\theta) = \begin{cases} 1 & t = s_1 \cos\theta + s_2 \sin\theta \\ 0 & otherwise \end{cases}$$

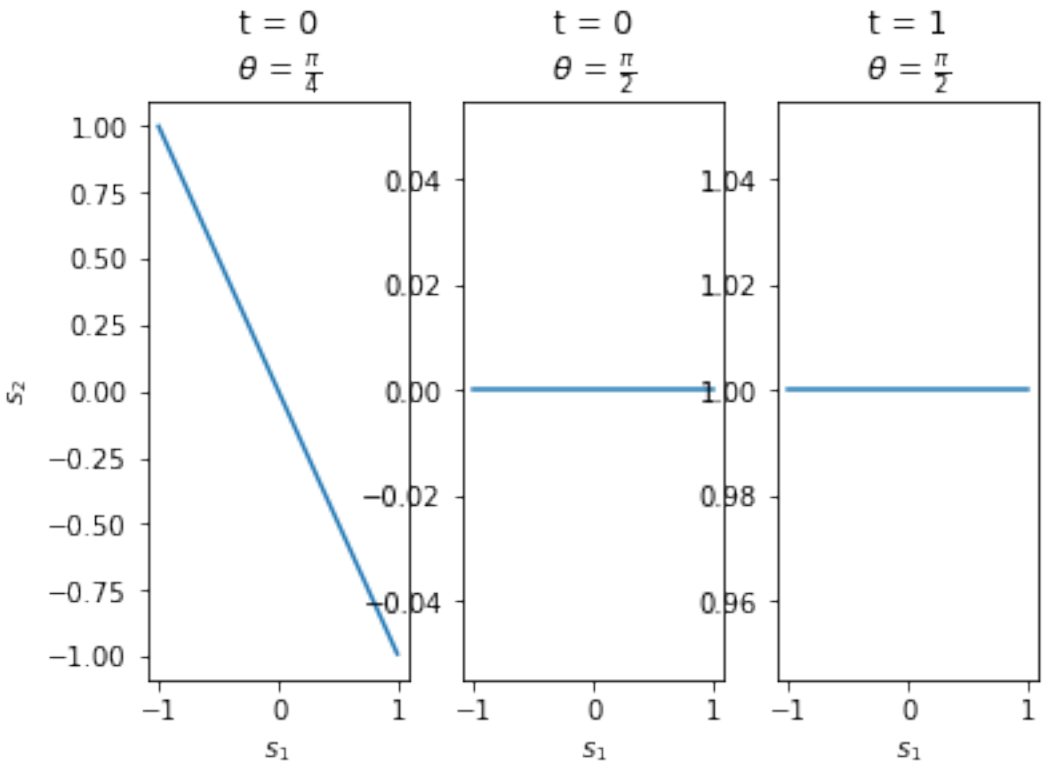
6.1 In s_1,s_2 space equation $t = s_1 \cos\theta + s_2 \sin\theta$ corresponds to the line in the form

$$s_2 = m s_1 + b$$

where $m = -\cot\theta$ and $b = \frac{t}{\sin\theta}$

[ ]:
```
s1 = np.arange(-1,1,0.01)
i = 1
thetas = [np.pi/4, np.pi/2]
theta_names = [r'$\frac{\pi}{4}$',r'$\frac{\pi}{2}$']
others = [0,1,1]
for t in [0,0,1]:
    theta = thetas[others[i-1]]
    print(theta)
    s2 = -1/np.tan(theta) * s1 + t/np.sin(theta)
    plt.subplot(1,3,i)
    plt.plot(s1,np.round(s2,4))
    plt.title("t = " + str(t) + "\n" + r'$\theta$ = ' +␣
 ↪theta_names[others[i-1]])
    plt.xlabel(r'$s_1$')
    if i == 1:
        plt.ylabel(r'$s_2$')
    i += 1
```

```
0.7853981633974483
1.5707963267948966
1.5707963267948966
```



6.2 As can be seen in the above figure, θ changes the angle which the line makes with the axes and t changes the shift in the s_2 direction.

6.3 $y(\theta, t)$ is the integration of the object along a line specified by the projection variables. The reason behind the line integral is that the impulse function given here zeroes out the whole function except for the line specified by the arguments of itself.

7 Q2 b

7.1 Projection operation is defined as follows:

$$y(\theta, t) = \iint_{-\infty}^{\infty} \delta(t - s_1 \cos\theta - s_2 \sin\theta) x(s_1, s_2) ds_1 ds_2$$

7.2 Using the given note that The adjoint of a general integral equation over an infinite domain is given by the same integral equation integrated with respect to the output variables. adjoint of the projection operation can be defined as follows:

$$\begin{aligned} x(s_1, s_2) &= \int_{\theta=0}^{\pi} \int_{t=-\infty}^{\infty} \delta(t - s_1 \cos\theta - s_2 \sin\theta) y(\theta, t) d\theta dt \\ &= \int_{\theta=0}^{\pi} y(\theta, s_1 \cos\theta + s_2 \sin\theta) d\theta dt \end{aligned}$$

Physically this equation describes a procedure where projection matrix is integrated over a line described by s_1, s_2 and θ and the value of $x(s_1, s_2)$ is computed. This projection is known as back-projection.

[ ]:
```
from skimage.transform import radon, resize
from skimage.data import camera, shepp_logan_phantom
```

8 Q2 c

8.1

$$y(\theta, t) = \iint_{-\infty}^{\infty} \delta(t - s_1 \cos\theta - s_2 \sin\theta) x(s_1, s_2) ds_1 ds_2$$

8.1.1 The matrix T that conducts discrete random transform operation is composed of elements t_{ij}'s representing the line integral of i^{th} ray along the j^{th} basis vector, *i.e.*, j^{th} pixel. In the given discretization scheme, the integral boundaries regarding $s1$ and s_2 directions can be found as follows:

$$\begin{aligned} s_1 &\in [\tfrac{N}{2} - i \bmod N, \tfrac{N}{2} - i \bmod N - 1] \\ s_2 &\in [\tfrac{-N}{2} + \lfloor j/N \rfloor, \tfrac{-N}{2} + \lfloor j/N \rfloor + 1] \end{aligned}$$

8.2 In more analytical notation,

$$b_j(s_1, s_2) = \Pi(s_1 - (\frac{N}{2} - 0.5 - i \bmod N), s_2 - (\frac{-N}{2} + 0.5 + \lfloor j/N \rfloor))$$

where Π represent the rectangular function.

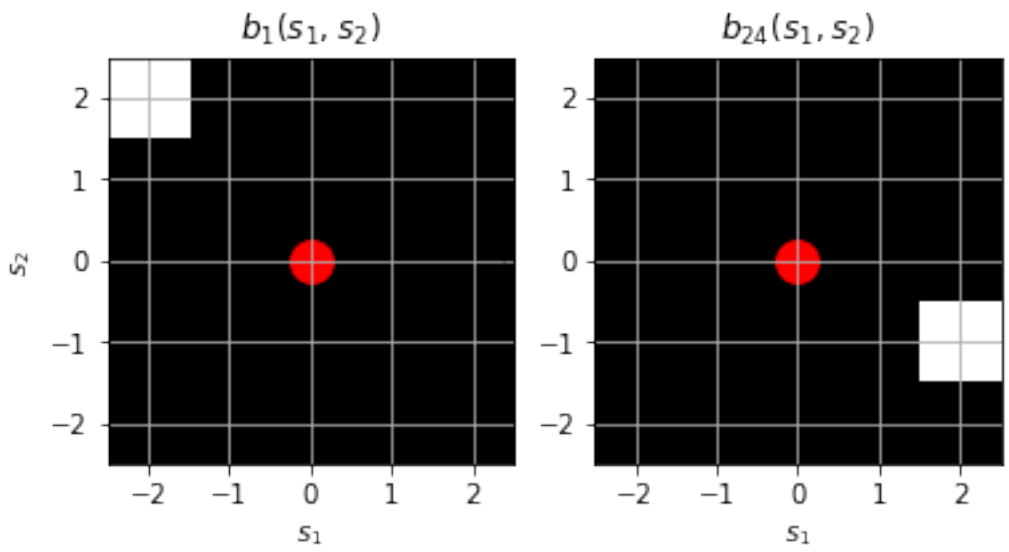
8.3 Hence, entries of T can be written as

$$T_{ij} = y(\theta, t) = \int_{s_1=\frac{N}{2}-i \bmod N}^{\frac{N}{2}-i \bmod N-1} \int_{s_2=\frac{-N}{2}+\lfloor j/N \rfloor}^{\frac{-N}{2}+\lfloor j/N \rfloor+1} \delta(t - s_1 \cos\theta - s_2 \sin\theta) \underbrace{x(s_1, s_2)}_{=1} ds_1 ds_2$$

8.4 Soem example basis functions can be found below.

[ ]:
```
A = np.zeros((5,5))
A[0,0] = 1
plt.subplot(1,2,1)
plt.imshow(A,extent=[-A.shape[1]/2., A.shape[1]/2., -A.shape[0]/2., A.shape[0]/
  ↪2. ],cmap="gray")
plt.grid()
plt.scatter(0,0,c="red",s=250)
plt.title(r'$b_1(s_1,s_2)$')
plt.xlabel(r'$s_1$')
plt.ylabel(r'$s_2$')
plt.subplot(1,2,2)
A[0,0] = 0
A[3,4] = 1
plt.title(r'$b_{24}(s_1,s_2)$')
plt.xlabel(r'$s_1$')
plt.xlabel(r'$s_2$')
plt.imshow(A,extent=[-A.shape[1]/2., A.shape[1]/2., -A.shape[0]/2., A.shape[0]/
  ↪2. ],cmap="gray")
plt.grid()
plt.scatter(0,0,c="red",s=250)
plt.xlabel(r'$s_1$')
plt.ylabel(r'$s_2$')
```

[ ]:
```
Text(0, 0.5, '$s_2$')
```



9 Q2 d

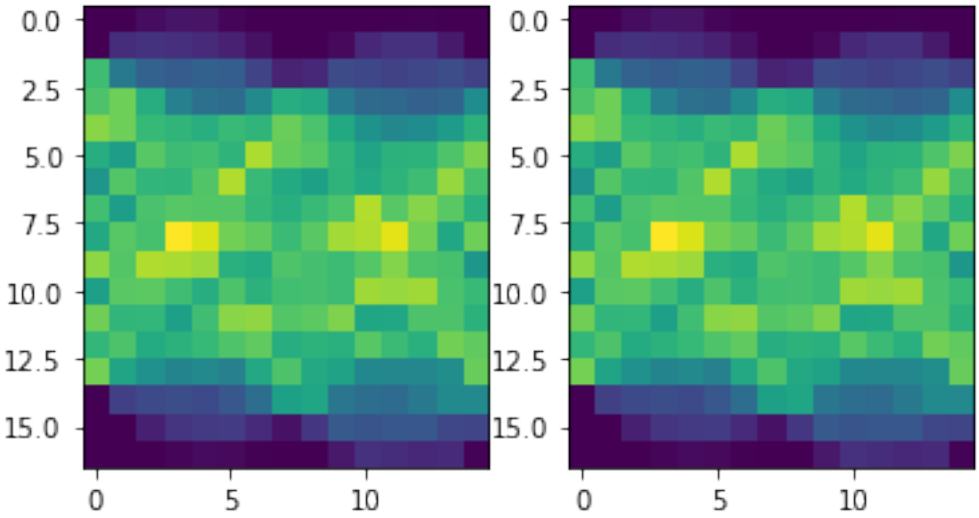
[ ]:
```
def projmtx(N, thetas):

    T = np.empty((int(np.ceil(N*np.sqrt(2))*len(thetas)),0))
    for i in range(N**2):
        row = np.mod(i,N)
        col = i//N
        basis_vec = np.zeros((N,N))
        basis_vec[row,col] = 1
        basis_vec_output = radon(basis_vec,thetas,circle=False)
        transformed_basis_vec = basis_vec_output.flatten(order="F")
        T = np.column_stack((T,transformed_basis_vec.flatten(order="F")))
    return T
```

[ ]:
```
for _ in range(100):
    N = np.random.randint(4,20,(1,1)).item()
    a = np.random.randint(1,100,(N,N))
    thetas = np.arange(0,180,N)
    T = projmtx(N,thetas)
    res = T.dot(a.flatten(order="F"))
    res = np.reshape(res,(res.shape[0],1))
    res2 = radon(a,thetas,circle=False, preserve_range=True)
    res = np.reshape(res,(res2.shape),order="F")
    assert np.sum(np.isclose(res,res2)) == res.size

print("[INFO] All tests are success!")
```

```
plt.subplot(1,2,1)
plt.imshow(res)
plt.subplot(1,2,2)
plt.imshow(res2)
```

```
[INFO] All tests are success!
```

[ ]: <matplotlib.image.AxesImage at 0x270ab38e880>

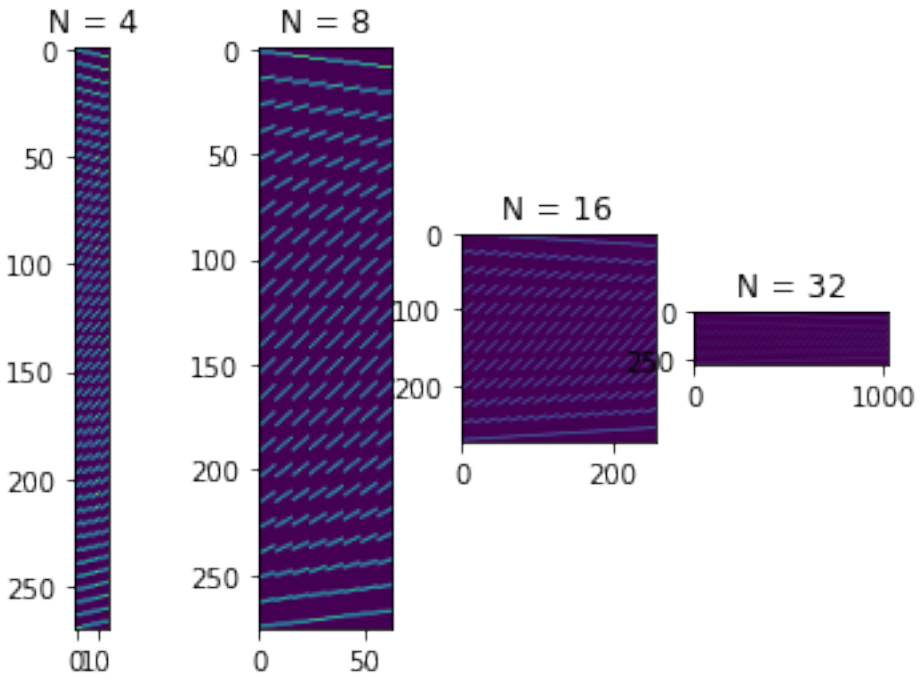


10 Q2 e

[ ]:
```
i = 1
for N in [1,4,8,16,32]:
    if not N == 1:
        X = resize(shepp_logan_phantom() ,(N,N))
    else:
        X = np.random.randint(1,100,(N,N))
    thetas = np.arange(0,180,N)
    T = projmtx(N,thetas)
    plt.subplot(1,4,i)
    plt.imshow(T)
    plt.title("N = " + str(N))
    if not N == 1:
        i += 1
    res = T.dot(X.flatten(order="F"))
    res = np.reshape(res,(res.shape[0],1))
    res2 = radon(X,thetas,circle=False, preserve_range=True)
    res = np.reshape(res,(res2.shape),order="F")
```

```
    assert np.sum(np.isclose(res,res2)) == res.size
print("[INFO] All tests are success!")
```

```
[INFO] All tests are success!
```



10.1 **Shepp-Logan and other random phantoms have the same projection via projection matrix and built in radon.**

10.2 **The resultant matrices are not Toeplitz matrices, hence the transform is not shift invariant.**

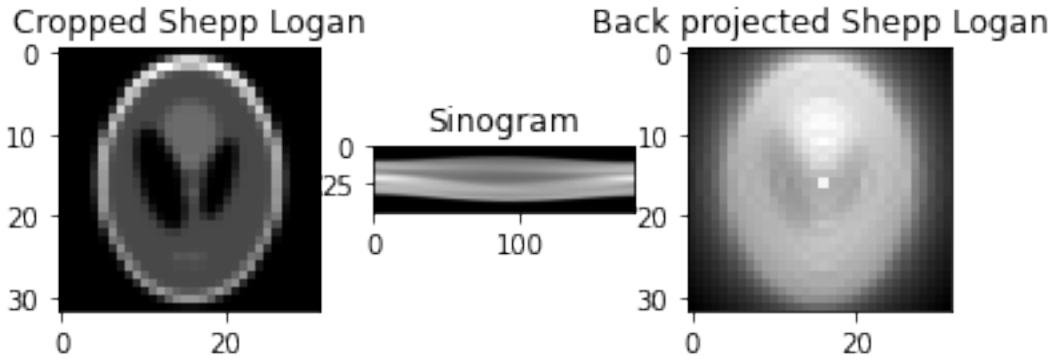
11 Q2 f

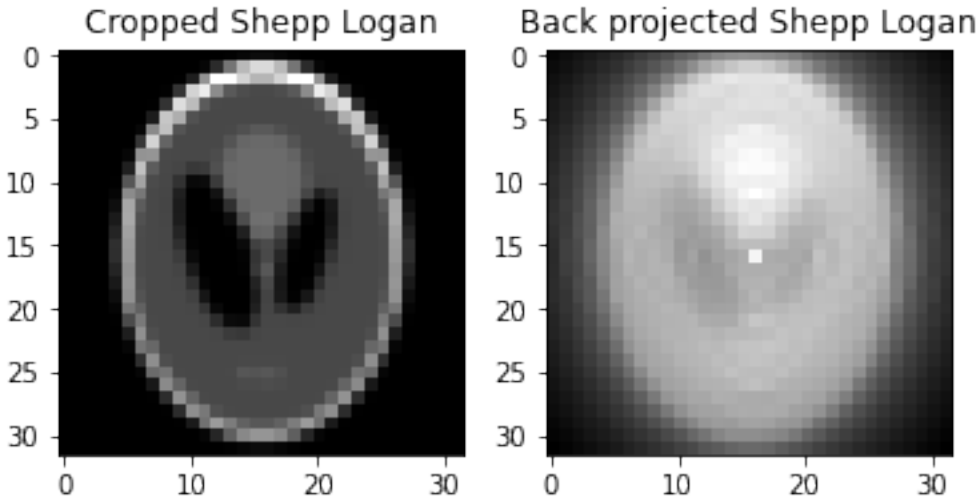
11.1 **Let A be the matrix representation of a linear system operator. Then the adjoint operator A^* is defined as A^T.**

[ ]:
```
X = resize(shepp_logan_phantom() ,(N,N))
thetas = np.arange(0,180,1)
T = projmtx(N,thetas)
projections = radon(X,theta=thetas,circle=False,preserve_range=True)
back_projected_X = np.reshape(T.T.dot(projections.flatten(order="F")),X.
 ↪shape,order="F")

plt.figure()
plt.subplot(1,3,1)
```

```
plt.imshow(X,cmap="gray")
plt.title("Cropped Shepp Logan")
plt.subplot(1,3,2)
plt.imshow(projections,cmap="gray")
plt.title("Sinogram")
plt.subplot(1,3,3)
plt.imshow(back_projected_X,cmap="gray")
plt.title("Back projected Shepp Logan")

plt.figure()
plt.subplot(1,2,1)
plt.imshow(X,cmap="gray")
plt.title("Cropped Shepp Logan")
plt.subplot(1,2,2)
plt.imshow(back_projected_X,cmap="gray")
plt.title("Back projected Shepp Logan")
```

[ ]: Text(0.5, 1.0, 'Back projected Shepp Logan')





11.2 The remaining questions are implemented in MATLAB.