MIDDLE EAST TECHNICAL UNIVERSITY (METU)
Department of Electrical and Electronics Engineering

EE 798 Theory of Remote Image Formation
Spring 2022

Problem Set 3

Issued: Thursday, May 26, 2022 **Due:** 23:59, Sunday, June 5, 2022

Problem 3.1 (Matrix representation of 2D convolution)

In this problem we extend the 1-D convolutional results developed previously to the 2-D, imaging, case. For images, such discrete LSI convolutional problems are of the form:

$$y(n_1, n_2) = \sum_{k_1=1}^{L_1} \sum_{k_2=1}^{L_2} h(n_1 - k_1, n_2 - k_2) x(k_1, k_2) \tag{1}$$

where $x(n_1, n_2)$ is a $L_1 \times L_2$ pixel image and $h(n_1, n_2)$ is a $P_1 \times P_2$ pixel FIR filter. The image $h(n_1, n_2)$ of the impulse response is often called the "point spread function", since it indicates how the system responds to, and spreads, a point input.

(a) It is clear that $x(n_1, n_2)$ and $y(n_1, n_2)$ are related by a linear transformation, which thus can be captured by a matrix-vector product. Unlike l-D, the difficultly in 2-D is that there is no obvious ordering of the points in an image into a vector. By choosing one such ordering we are effectively fixing a set of coordinate vectors, and thus a mapping of the image pixels to a vector. The most common mapping of a set of image pixels $x(n_1, n_2)$ to a vector x, and the one we will use here, is obtained by *stacking* one column on top of another, as follows:

$$[x(n_1, n_2)] = \begin{bmatrix} x(1,1) & x(1,2) & \dots & x(1,L_2) \\ x(2,1) & x(2,2) & \dots & x(2,L_2) \\ \vdots & \vdots & \vdots & \vdots \\ x(L_1,1) & x(L_1,2) & \dots & x(L_1,L_2) \end{bmatrix} \Longrightarrow x = \begin{bmatrix} x(1,1) \\ \vdots \\ x(L_1,1) \\ - - - - - \\ x(1,2) \\ \vdots \\ x(L_1,2) \\ - - - - - \\ \vdots \end{bmatrix} \tag{2}$$

In Python this operation can performed using the **numpy.reshape** function, with the `order='F'` argument. The opposite operation taking a vector back to an image can also

be done using the same function. In MATLAB this operation is performed using the colon operator ":". The opposite operation of converting a vector back to an image is done using the MATLAB function "reshape". To practice this, read the 'cameraman.tif' image (using the function imread in MATLAB) and make sure that it is in grayscale. Display the resulting image (using imshow in Matlab). Then convert this image into a vector as described above, and then convert it back to an image. Display the resulting image again and also provide your code.

Let the vectors x and y be defined this way for the images $[x(n_1, n_2)]$ and $[y(n_1, n_2)]$, respectively. If the elements of y and x are related through linear convolution with $h(n_1, n_2)$ as described in (2), then what are the entries of the matrix C which captures this linear convolution as $y = Cx$? What is the form of the C matrix in this 2-D case? How does it appear related to a Toeplitz matrix? How is the size of C related to the sizes of the image $x(n_1, n_2)$ and the filter impulse response kernel $h(n_1, n_2)$? What practical problems does this matrix representation of such a convolutional problem pose?

(b) Since the underlying problem is again convolutional, we again expect Fourier techniques to be of use. This time the appropriate tool is the 2-D DFT, which may be efficiently implemented using the 2-D FFT. As before, however, the product of the DFTs of two images is related to the 2-D *circular convolution* of the images. What dimensions N_1, N_2 in a 2-D circular convolution must be used to ensure that the circular convolution of $h(n_1, n_2)$ and $x(n_1, n_2)$ produces the same results as the linear convolution? How are the corresponding vectors of periodic sequences $\tilde{h}$ and $\tilde{x}$ related to h and x? Write a Python/Matlab function **cconv2** to perform the N_1, N_2-point circular convolution of two images (Hint: Use the **numpy.fft.fft2** routine in Phyton and **fft2** routine in Matlab). Note: Be careful of numerical roundoff introducing complex components into your answer.

(c) Given the point spread function $h(n_1, n_2)$, what is the matrix $\tilde{C}$ that performs the N_1, N_2-point *circular* convolution of this point spread function with an image of size $L_1 \times L_2$ (assume $L_i > P_i$)? What special form does it have? How is it related to C? Using your routine **cconv2**, write a Python/Matlab function **cconvmtx2** to create $\tilde{C}$ for an arbitrary h and N_1, N_2 (Hint: Consider the relationship between 2-D circular convolution with the unit coordinate images and the columns of $\tilde{C}$. The unit coordinate images are all zero except for a single 1. Be careful to access the columns of $\tilde{C}$ in the correct order). Note that $y = \tilde{C}\tilde{x}$. You now have a way of generating a matrix representation for arbitrary 2-D LSI convolutional problems.

(d) The 2-D DFT of size N_1, N_2 of an image is typically defined as:

$$X(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) e^{-j2\pi(k_1 n_1/N_1 + k_2 n_2/N_2)}; k_1 = 0, ..., N_1 - 1, k_2 = 0, ..., N_2 - 1 \quad (3)$$

$$x(n_1,n_2)=\frac{1}{N_1N_2}\sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1}X(k_1,k_2)e^{j2\pi(k_1n_1/N_1+k_2n_2/N_2)};n_1=0,...,N_1-1,n_2=0,...,N_2-1 \tag{4}$$

Let X denote the vector of 2-D DFT coefficients $X(k_1,k_2)$ and x the vector corresponding to the image $x(n_1,n_2)$. Then X and x are related by a matrix F through $X=Fx$, i.e. the matrix F takes the 2-D DFT of a sequence. The Python function **numpy.fft.fft2** and Matlab function fft2 generates the N_1,N_2-point 2-D DFT $X(k_1,k_2)$ of the matrix $x(n_1,n_2)$. Using this function, write another Python/Matlab function **dftmtx2** to generate the matrix F which performs the 2-D DFT for an arbitrary size N_1,N_2 (Hint: Use the same approach as in part (c) and relate the 2-D DFT of the unit coordinate images to the columns of F). What is F^{-1} (the inverse DFT matrix) in terms of F itself (Hint: different columns of F are orthogonal). Be careful of scalings!

(e) If we define $Y=Fy$, $H=Fh$, and $X=Fx$ to be vectors of the 2-D DFT coefficients corresponding to the images $y(n_1,n_2)$, $h(n_1,n_2)$, $x(n_1,n_2)$, argue that the matrix $\tilde{\mathbf{C}}=F\tilde{C}F^H/(N_1N_2)$ is a diagonal matrix. Similar to the 1-D case this shows that block circulant matrices with circulant blocks are diagonalized by the 2-D DFT transform operator. In terms of operations on the rows and columns of $\tilde{C}$, what does the matrix operation $F\tilde{C}F^H/(N_1N_2)$ represent in the 2-D case? Using this insight, suggest an alternative procedure for diagonalizing $\tilde{C}$ using 2-D FFT operations. Verify that $Fy=(FCF^H/N)F\tilde{x}=\tilde{H}.*\tilde{X}$, and that $F\tilde{C}F^H/N$ is indeed diagonal for a small numerical example.

(f) What is the relationship between the elements on the diagonal of $\tilde{\mathbf{C}}$ and the 2-D DTFT of the original impulse response $h(n_1,n_2)$? How are the diagonal elements of $\tilde{\mathbf{C}}$ related to the eigenvalues of $\tilde{C}$? The significance of this result is that *independent of the system* $h(n_1,n_2)$ you know how to quickly diagonalize the (circular) convolutional operator $\tilde{C}$. Thus problems that can be cast in this form are much easier to analyze than general problems.

(g) Again, if the problem were *not* shift-invariant, how would things change? In parts (b) through (f) we went to considerable lengths to cast the standard linear convolution problem as a circular convolution problem. Is this warranted in the shift-variant case? Explain.

Problem 3.2 (Matrix representation for 2D tomography)

Consider the *Radon Transform*, relating the data $y(\theta,t)$ to the object $x(s_1,s_2)$, via a linear transformation of the form:

$$y(\theta,t)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\delta(t-s_1\cos\theta-s_2\sin\theta)x(s_1,s_2)ds_1ds_2 \tag{5}$$

(a) In (s_1,s_2) space, plot $\delta(t-s_1\cos\theta-s_2\sin\theta)$. How does this function change as the

variable t changes? How does it change as a function of θ? In what sense is $y(\theta, t)$ a *projection* of $x(s_1, s_2)$?

(b) Derive an explicit expression for the adjoint of the Radon transform operator. Note: 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. The operation performed by the adjoint is "backprojection". What is the physical interpretation of this operation?

(c) In practice, we usually manipulate discrete representations of the object as well as the projection data. Toward that end, suppose that the object is represented using a square, flat-top pixel basis of the form

$$x(s_1, s_2) = \sum_{j=1}^{N} x_j b_j(s_1, s_2) \tag{6}$$

where $b_j(s_1, s_2)$ is 1 over the j-th pixel and zero elsewhere, and we assume that the pixels are ordered column-wise as usual. Furthermore, assume that $y(\theta, t)$ is sampled at a finite number of points in θ-t space: $y_i = y(\theta_i, t_i), i = 1, ..., M$. Assume we order these pairs first by t then by θ. Under this discretization scheme, show that the vector y of projection samples may be related to the vector x of object coefficients x_i via a matrix-vector equation of the form:

$$y = Tx. \tag{7}$$

What is the analytical expression for the ij-th element of T? Assume that $x(s_1, s_2)$ is supported in a square region centered about the origin.

(d) The Python routine **skimage.transform.radon** and the MATLAB routine radon.m generate the projection of a 2-D image at a given set of projection angles `theta`. Using this routine write a Python/Matlab function **projmtx** that generates the matrix T for a given square image size: `N`, and a vector of angles: `theta`. For this, you should notice that the columns of T are given by the application of the projection operation to the unit coordinate vectors. The unit coordinate vectors are all zero except for a single 1, i.e. e1 = [10...0], e2 = [010...0], and such. Hence in 2D form, each unit coordinate vector corresponds to a 2D image that is zero everywhere except a single pixel. Be careful to access the columns of T in the correct order.

In Matlab, radon.m automatically determines the field of view of the output to fit the largest diagonal of the image, so the units of length are not the same in s and p in general. To avoid running out of memory, your routine should generate its output in MATLAB's sparse matrix format. Arrays in this format need to be converted using full.m to perform certain MATLAB operations correctly.

In Phyton, set the argument `circle=False`, which results in $\lceil N\sqrt{2} \rceil$ projections per each projection angle for square images.

Generate some projections using the Shepp-Logan phantom image which can be obtained using the MATLAB function phantom.m or in Phyton by:

```
from skimage.data import shepp_logan_phantom
from skimage.transform import resize
N = 32 # pick whatever size
X = resize(shepp_logan_phantom(), (N,N))
```

Make sure that your T matrix produces the same result as the built-in **radon** function.

(e) Using **projmtx** generate T for the case of $N \times N$ images and N uniformly spaced angles from 0 to 180 degrees for $N = 4, 8, 16$. Look at the matrices using **matplotlib.pyplot.imshow** in Phyton and **spy.m** in Matlab. Is the Radon transform a shift-invariant system?

(f) Based on your answer in (b), what is the backprojection operation in the data y in terms of the matrix T? Backproject and view your data.

Problem 3.3 (Deconvolution)

In this problem we will study the recovery of a noisy blurred image. The image will be the cameraman image (available in Matlab by simply using I = imread('cameraman.tif'); x = im2double(I); x=x(50:80,105:138).) Working on a small-size image is recommended to avoid memory errors.

(a) Create a 2-D FIR blurring kernel $h(n_1, n_2)$ of size 9×9 pixels, which consists of samples of a centered Gaussian with a standard deviation of σ_h pixel. For this, you can use the provided MATLAB file **gauss2d.m** if you are using Matlab. Choose $\sigma_h = 1, 2, 4$. Using this blurring kernel and the built-in function **convmtx2**, generate the matrix C that performs the 2-D convolution with this blur function. Using this matrix, generate a blurred version y of the image, which will serve as the clean data. In addition, generate a noisy version $\tilde{y}$ of the observation y according to the formula: $\tilde{y} = y + n$ where $n \sim N(0, \sigma_n^2 I)$. The signal-to-noise ratio (SNR) of an image is often defined as the variance of the image over the variance of the noise:

$$SNR(dB) = 10 \log_{10} \frac{var(y)}{\sigma_n^2}$$

Choose σ_n to yield an SNR of 30dB and 40dB. Are these noise levels large in $\tilde{y}$? Plot the original image, the noiseless blurred image, and the noisy blurred image (in Matlab you can use the function **imagesc** together with **reshape**). Can you notice the noise?

(b) We will first examine the analytical approach that we discuss in the class: *inverse filtering*. Apply inverse filtering approach to both noiseless data y and noisy data $\tilde{y}$. Plot the true image x, the solution based on the noiseless data, and the solution based on the noisy data, and compare. (Hint: Note that to apply inverse filtering you can either multiply in the frequency domain with the frequency response of the inverse filter, or equivalently apply C^{-1} to the measurement vectors.)

(c) We will now examine the least-squares approach. Find the least-squares solutions for both the noiseless data and the noisy data. Plot the true image x, the LS solution based on the noiseless data and the LS solution based on the noisy data and compare. Can you explain the distinct checkerboard textured appearance to the noisy reconstruction based on SVD?

(d) Now generate truncated SVD (TSVD) reconstructions for both the noiseless and noisy data. Plot the true image x, the solution based on the noiseless data, and the solution based on the noisy data, and compare. What seems to be the optimal threshold for this problem? Show your results for different thresholds.

Problem 3.4 (Design of reconstruction algorithms using optimization)

In generating estimated answers to inverse problems we have focused mainly on "direct" methods of solution, involving such tools as the SVD or direct matrix inverses in Python/-Matlab. For very large problems such direct approaches are not tractable and more effcient iterative methods must be employed for the solution of the resulting sets of equations. In this problem we will study how to effciently use such iterative approaches to solve standard regularization problems. In the process we will generate both approaches and code that will be of use for a wide variety of problems.

To fix ideas, let us consider the following standard Tikhonov formulation:

$$_x\|y - Cx\|_2^2 + \lambda\|Dx\|_2^2 \tag{8}$$

where D is a discrete derivative operator. The solution of such a problem is given by the solution of the following set of linear *normal equations*:

$$\underbrace{(C^TC + \lambda D^TD)}_{A}x = \underbrace{C^Ty}_{b} \tag{9}$$

$$Ax = b \tag{10}$$

where the matrix A is symmetric and positive semi-definite.

There exist a number of methods for effciently solving such linear problems of the form $Ax = b$. The method which we will use here is the conjugate gradient iterative method, given in Algorithm 1. A detailed derivation of this algorithm is beyond the scope of this problem. The inputs to the algorithm are the operator A, the right hand side b, and an initial guess x_0 for x.

The iterations of the CG algorithm can be seen to be quite efficient. Most of the computations involve only scalar products of vectors and multiplications by scalars. The computationally intensive tasks in the algorithm is the repeated computation of the quantity Az for some z. Note that this is just repeated application of the forward operator to different inputs. Thus the CG method solves a linear inverse problem through repeated solution of the forward problem and the method will be attractive if we can solve the forward problem, as represented by the product Az, efficiently.

Algorithm 1 Conjugate Gradient

procedure CG($\mathbf{A}, \mathbf{b}, \mathbf{x}_0$)
  Compute $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ for some initial guess $\mathbf{x}_0$
  $\mathbf{p}_0 = \mathbf{r}_0$
  for $k = 0, 1, 2, \ldots$ **do**
    $\alpha_k = \mathbf{r}_k^T \mathbf{r}_k / (\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k)$
    $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$
    $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A} \mathbf{p}_k$
    $\beta_k = \mathbf{r}_{k+1}^T \mathbf{r}_{k+1} / (\mathbf{r}_k^T \mathbf{r}_k)$
    $\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$
    check convergence; continue if necessary
  end for
  return $\mathbf{x}_{k+1}$
end procedure

(a) One simple and direct way to exploit the CG algorithm to solve Tikhonov regularized problems is to just define the operator/matrix $A = (C^T C + \lambda D^T D)$ and right hand side $b = C^T y$, as suggested in (2). Suppose we did this for a deconvolution problem involving an $L \times L$ image with a $P \times P$ blurring kernel (assuming $P \ll L$). If we use N_{iter} iterations of CG, what is the approximate computational cost of using the CG method vs direct matrix inversion? Keep only dominant terms and assume that the support of the regularization operator D is much smaller than the support of C. In general, when N_{iter} is kept "small" the CG method requires much less computation than a direct method.

(b) While straightforward application of the CG method to regularized problems can be efficient, unfortunately in some cases the operator $A = (C^T C + \lambda D^T D)$ may be too large to explicitly form due to memory limitations. Further, it may be that we simply want to exploit fast forward solver codes that are available for a given problem. In either case we can exploit the structure of the CG algorithm to avoid having to explicitly form A.

To this end, write an algorithm CGTik (not code yet) based on the original CG algorithm in Algorithm 1 to iteratively find the solution of the Tikhonov regularized problem in (1). Your algorithm should take as input the operators C and D together with their adjoints C^T and D^T, and the usual components of the observation y, an initial guess x_0 for x. Write your algorithm so that only forward operator evaluations Cz and Dz and adjoint operator evaluations $C^T z$ and $D^T z$ are needed (make this clear in your algorithm). In particular, do not simply define a new operator $A = (C^T C + \lambda D^T D)$ and apply CG to it! The point is to write an algorithm that just needs to perform forward and adjoint evaluations.

(c) To fix ideas we will focus on the problem of Tikhonov regularized deblurring. In particular, assume that the noisy observation Y corresponds to the output of a convolution of the $L \times L$ image X with $P \times P$ blurring kernel H, and corrupted by additive white Gaussian noise.

Further, assume that the "regularizer" Dx in (1) is the stacked output of filtering by first derivative kernels D_1 and D_2 along the first and second coordinate directions, respectively. That is, denoting R_1 and R_2 the convolution matrices corresponding to the kernels D_1 and D_2:

$$Dx = \begin{bmatrix} R_1 \\ R_2 \end{bmatrix} x = \begin{bmatrix} R_1 x \\ R_2 x \end{bmatrix} \quad \text{and} \quad D^T z = \begin{bmatrix} R_1^T & R_2^T \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = R_1^T z_1 + R_2^T z_2$$

For simplicity we will ignore edge effects – that is, we will assume that the output of forward blurring is given by **scipy.signal.convolve(X,H,'same')** in Phyton or **conv2(X,H,'same')** in Matlab while the output of the regularizer is given by the pair **scipy.signal.convolve(X,D1,'same')** and **scipy.signal.convolve(X,D2,'same')** in Phyton or by the equivalent operations in Matlab. Further, again for simplicity, assume that all kernel dimensions are odd (i.e. that P is odd as well as the dimensions of D_1 and D_2). When the kernels are odd the outputs of the adjoint operation can be easily expressed and obtained using 2D convolution In particular, what are the operations $C^T z$ and $D^T z$ in terms of the kernels H, D_1, D_2 and the 2D convolution operation? Hint: we covered this in Homework 2.

Based on your algorithm in part (b) write a Python/Matlab function **cgconvtik** to solve for the Tikhonov regularized solution of this deblurring problem. Your function should take as input the blurring kernel H, the regularizer kernels D_1 and D_2, the noisy data image Y, the initial condition X_0, and termination criteria (e.g. max iteration number and/or a convergence criterion). Inside the code you should only use calls to **scipy.signal.convolve** in Phyton or **conv2** in Matlab as the primary computation step and thus will avoid any large matrix-vector products.

(d) Now let us apply the code. Let X represent the 256×256 cameraman image, and C represent the distortion due to blurring with a 15×15 Gaussian kernel that has a standard deviation of $\sigma = 2$ pixels. This kernel can be obtained using the provided **gauss2d** code in Matlab or in Phyton using the following code:

```
import numpy as np
from skimage.transform import resize
from skimage import data

X = resize(data.camera(), (256,256))

def gauss2D(shape=(15,15),sigma=2):
"""
2D Gaussian kernel
"""
m,n = [(ss-1.)/2. for ss in shape]
y,x = np.ogrid[-m:m+1,-n:n+1]
h = np.exp(-(x*x + y*y)/(2.*sigma*sigma))
h[h < np.finfo(h.dtype).eps*h.max()] = 0
sumh = h.sum()
```

$if sumh! = 0:$
$h/ = sumh$
return h

You can use $D_1 = [-1\ 0\ 1]^T$ and $D_2 = [-1\ 0\ 1]$ as the first derivative kernels.

Assume that the observation y is the blurred image Cx corrupted by additive, white Gaussian noise; that is $y = Cx + w$ with $w \sim \mathcal{N}(0, \sigma_w^2)$, with the signal-to-noise ratio (SNR) of 30 dB. The SNR of an image is often defined as the variance of the clean image over the variance of the noise:

$$SNR(dB) = 10 \log_{10} \frac{\text{var}(Cx)}{\sigma_w^2}$$

Try different regularization parameters λ and display your best (visually) reconstruction along with the measurement and the true image. Make sure you use enough iterations of CG so you are seeing the effects of the regularization and not the premature truncation of the iterates. Can you solve this problem using a "matrix approach"? Explain.

(e) (BONUS) Apply CG to the Tikhanov regularized 2D tomography problem and repeat the previous parts. We will keep the same formulation in Eq. (1), the only difference from the deblurring problem will be that the operator C will be the Radon transform this time. To efficiently implement the operators C and C^T, we will use the built-in radon and inverse radon routines in Phyton/Matlab.