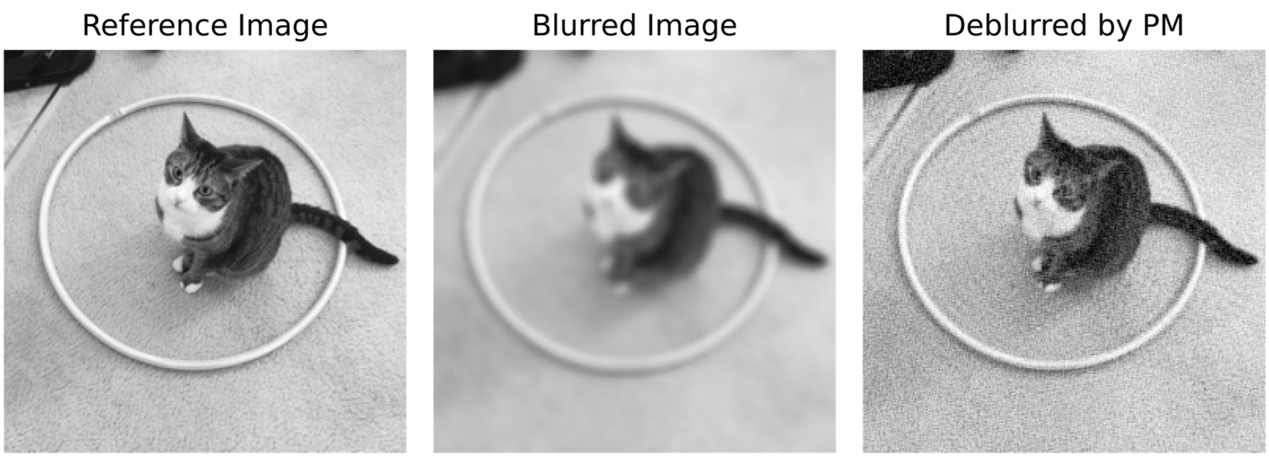
Introduction to Optimization - Computational Exercise

Due November 10th 2023, 9:00

Introduction

This computational exercise is aimed at deblurring images. Given a non-blurry image, we shall blur it, and apply different methods learned in class to deblur it. The following should give you a rough estimate of the final expected result.



Required Libraries

This lab will require some libraries. If you do not know some of them, their documentation is available and quite extensive.

```python
from PIL import Image, ImageOps          # For Image Handling
import numpy as np                       # NumPy
import scipy as sp                       # SciPy
import scipy.sparse.linalg               # Various Linear Algebra Tools
from scipy.fftpack import dct, idct      # For Image Blurring
import matplotlib.pyplot as plt          # For Plotting
import matplotlib.cm as cm               # ColorMaps for Plotting
```

Importing the Image

The following code imports the image and formats it as required. We are considering a 256×256 grayscale image. Make sure to download (or take) an image "cat.jpg", and to import it in the appropriate folder. Note that the `.jpg` extension is important here.

```python
# Load the image, Resize it, Grayscale it, and convert it to an array
img = Image.open('cat.jpeg').resize((256, 256))
X_ref = np.asarray(ImageOps.grayscale(img)).astype('float32')

# Show image
plt.imshow(X_ref, cmap=cm.Greys_r, vmin=0, vmax=255)
```

Alternatively, in Matlab:

```matlab
% Load the image, Resize it, Grayscale it, and convert it to an array
img = imread('cat.jpg');
img = imresize(img, [256, 256]);
X_ref = rgb2gray(img); % Assuming 'cat.jpg' is a RGB image

% Convert to float32
X_ref = single(X_ref);

% Show image
imshow(X_ref, [0, 255], 'Colormap', gray);
```

Blurring the Image

The following code blurs the image. The blurring process in itself is not important for the scope of this lab, and is thus kept as a black box. Note that the underlying idea is to do a convolution, which is a linear operator. The function `blackbox` returns a function, `R`, which is the blurring operator. Note that this operator is self-adjoint (symmetric).

```python
# Black Box, Not Important how it works
def blackbox():
    s = 4
    size_filter = size = 9
    PSF = np.array([[np.exp(-0.5*((i-4)/s)**2 - 0.5*((j-4)/s)**2)
                     for j in range(size)] for i in range(size)])
    PSF /= np.sum(PSF)
    def dctshift(PSF, center):
        m, n = PSF.shape
        i, j = center
        l = min(i, m-i-1, j, n-j-1)
        PP = PSF[i-l:i+l+1, j-l:j+l+1]
        Z1 = np.diag(np.ones(l+1), k=l  )
        Z2 = np.diag(np.ones(l)  , k=l+1)
        PP = Z1 @ PP @ Z1.T + Z1 @ PP @ Z2.T + Z2 @ PP @ Z1.T + \
             Z2 @ PP @ Z2.T
        Ps = np.zeros_like(PSF)
        Ps[0:2*l+1, 0:2*l+1] = PP
        return Ps
    dct2 = lambda a: dct(dct(a.T, norm='ortho').T, norm='ortho')
    idct2 = lambda a:  idct(idct(a.T,norm='ortho').T,norm='ortho')
    Pbig = np.zeros_like(X_ref)
    Pbig[0:size_filter , 0:size_filter] = PSF
    e1 = np.zeros_like(X_ref)
    e1[0][0] = 1
    S = np.divide( dct2(dctshift(Pbig, (4,4))), dct2(e1) )
    R = lambda X, S_matrix=S: idct2( np.multiply(S_matrix, dct2(X)) )
    return R

```

```python
# Retrieve blurring and adjoint of blurring operators
R = blackbox()

# Blur Image and add noise
np.random.seed(10)
n = np.random.normal(0, 0.5, size=X_ref.shape)
X_blur = R(X_ref) + n

# Show blurry image
plt.imshow(X_blur, cmap=cm.Greys_r, vmin=0, vmax=255)
```

In Matlab:

```matlab
% Black Box, Not Important how it works
function R = blackbox(X_ref)
    s = 4;
    size_filter = 9;
    PSF = zeros(size_filter, size_filter);
    for i = 1:size_filter
        for j = 1:size_filter
            PSF(i, j) = exp(-0.5*((i-4)/s)^2 - 0.5*((j-4)/s)^2);
        end
    end
    PSF = PSF / sum(PSF(:));

    function Ps = dctshift(PSF, center)
        [m, n] = size(PSF);
        i = center(1);
        j = center(2);
        l = min([i, m-i+1, j, n-j+1]);
        PP = PSF(i-l+1:i+l+1, j-l+1:j+l+1);
        Z1 = diag(ones(l+1,1), l);
        Z2 = diag(ones(l,1), l+1);
        PP = Z1 * PP * Z1' + Z1 * PP * Z2' + Z2 * PP * Z1' + Z2 * PP * Z2';
        Ps = zeros(size(PSF));
        Ps(1:2*l+1, 1:2*l+1) = PP;
    end

    dct2 = @(a) dct(dct(a')');
    idct2 = @(a) idct(idct(a')');

    Pbig = zeros(size(X_ref));
    Pbig(1:size_filter, 1:size_filter) = PSF;

    e1 = zeros(size(X_ref));
    e1(1,1) = 1;

    S = dct2(dctshift(Pbig, [4, 4])) ./ dct2(e1);

    R = @(X) idct2(S .* dct2(X));
end
```

Discrete Gradient

You will need the discrete gradient operator to describe the deblurring problem. The operator is given by the following code.

```python
# The Discrete Gradient Linear Operator
# grad: R^{256x256} -> R^{2x256x256}
def grad(X):
    G = np.zeros_like([X, X])
    G[0, :, :-1] = X[:, 1:] - X[:, :-1] # Horizontal Direction
    G[1, :-1, :] = X[1:, :] - X[:-1, :] # Vertical Direction
    return G

# The Adjoint of the Discrete Gradient Linear Operator, the Discrete Divergence
# grad_T: R^{2x256x256} -> R^{256x256}
def grad_T(Y):
    G_T = np.zeros_like(Y[0])

    G_T[:, :-1] += Y[0, :, :-1] # Corresponds to c[0]
    G_T[:-1, :] += Y[1, :-1, :] # Corresponds to c[1]
    G_T[:, 1:] -= Y[0, :, :-1] # Corresponds to c[0]
    G_T[1:, :] -= Y[1, :-1, :] # Corresponds to c[1]

    return G_T
```

And in Matlab:

```matlab
% The Discrete Gradient Linear Operator
% grad: R^{256x256} -> R^{2x256x256}
function G = grad(X)
    [m, n] = size(X);
    G = zeros(2, m, n);
    G(1, :, 1:end-1) = X(:, 2:end) - X(:, 1:end-1); % Horizontal Direction
    G(2, 1:end-1, :) = X(2:end, :) - X(1:end-1, :); % Vertical Direction
end

% The Adjoint of the Discrete Gradient Linear Operator, the Discrete Divergence
% grad_T: R^{2x256x256} -> R^{256x256}
function G_T = grad_T(Y)
    [c, m, n] = size(Y);
    G_T = zeros(m, n);
    G_T(:, 1:end-1) = G_T(:, 1:end-1) + Y(1, :, 1:end-1); % Corresponds to c[0]
    G_T(1:end-1, :) = G_T(1:end-1, :) + Y(2, 1:end-1, :); % Corresponds to c[1]
    G_T(:, 2:end) = G_T(:, 2:end) - Y(1, :, 1:end-1); % Corresponds to c[0]
    G_T(2:end, :) = G_T(2:end, :) - Y(2, 1:end-1, :); % Corresponds to c[1]
end
```

Primal-Dual Method

We write the problem as a Total Variation problem, using the 1-norm of the discrete graident as penalty function according to the ROF Model. Notice that instead of having the λ before the first term, we could have a factor r in front of the second term, representing its inverse. This would be analogous of course.

$$\min_{X \in \mathbb{R}^{256\times 256}} \left\{ \frac{\lambda}{2} \|R(X) - X_{blur}\|_2^2 + \|\text{grad}(X)\|_1 \right\}$$

Where R is the blur operator, X_{blur} is the blurred (and noisy) image, and $\text{grad}(X)$ is the discrete gradient of X (Which is a linear operator). Of course all matrix-norms are vector induced.

Apply the Primal-Dual method to the above problem. You may use without proof that $\|\text{grad}\| \leq 2\sqrt{2}$.

Test your algorithm for various values of λ and different number of iterations to see which one yields the best visual result.

ADMM

The above problem may be written as

$$\min_{X \in \mathbb{R}^{256\times 256}, Y \in \mathbb{R}^{2\times 256\times 256}} \left\{ \frac{\lambda}{2} \|R(X) - X_{blur}\|_2^2 + \|Y\|_1 \right\} \quad \text{subject to} \quad \text{grad}(X) = Y.$$

Apply the Alternating Direction Method of Multipliers (ADMM) to solve the above problem. As before, test your algorithm on different values of λ and select the one yielding the best results.

Deliverable

You are expected to turn in a single report in LaTeX including the following:

1. A short description of the computations required before running the algorithms. This includes, for instance, the values of the different proximal operators.
2. One (or multiple) figures representing the deblurred image for different values of λ and different number of iterations, for each method.
3. One final figure in which you compare the original image, the blurry image, and the best deblurred image for each method.
4. An appendix with all the code. You may include your code using a package such as `listings`.