Machine Learning for Computer Vision

Exercise 7

Kodai Matsuoka, Yuyan Li

June 12, 2017

1 Implementation

Our implementation of the Gaussian Graphical Model (GGM) is shown below in listing 1.

2 Comparison

We compared our algorithm to some of the algorithms available on ipol.im.
In particular we used the *Non-Local Means Denoising* to do a qualitative comparision. For that we used their program to first produce a noisy image which we denoised with both their and our algorithm. Then we used their program *img_mse_ipol* to calculate the RMSE and the PSNR. The better denoising has a smaller RMSE and larger PSNR value. We can see that the GGM is still far from achieving the results of the *nlmeans* algorithm.

Table 1: RMSE and PSNR values of different algorithm with noise value $\sigma = 30$.

	GGM	nlmeans
RMSE	13.66	8.00
PSNR	25.42	30.07

Figure 1: From left to right the pictures show: the noisy image, denoised with *nlmeans*, denoised with our algorithm.

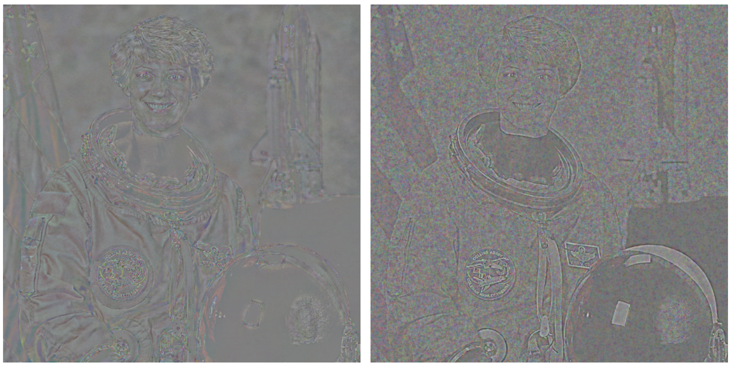

Figure 2: The differences to the original image. On the left is *nlmeans* and on the right our algorithm.

Listing 1: denoising.py

```
import numpy as np
import matplotlib.pyplot as plt
import skimage as ski
from skimage import data
import scipy.sparse
import time

# normalize images to [0,1]
def norm01(data):
    res = data.astype('float')
    res -= res.min()
    res /= res.max()
    return res

# difference between two images using the 2-norm
def performance(p1, p2):
    return np.linalg.norm(p1 - p2)

# build a simple Q matrix with constant off-diagonal elements beta
def buildConstQ(imshape, beta = -1.):
    start = time.time()

    nrows, ncols = imshape
    size = nrows * ncols

    Q = scipy.sparse.lil_matrix((size,size), dtype='float')

    diagElem = 4 * abs(beta)

    # vectorized index
    def getInd(x, y):
        return x + y * ncols

    for x in range(nrows):
        for y in range(ncols):

            i = getInd(x,y)
            Q[i,i] = diagElem

            if x < ncols-1:
                j = getInd(x+1, y)
                Q[i,j] = beta
                Q[j,i] = beta

            if y < nrows-1:
                j = getInd(x, y+1)
                Q[i,j] = beta
                Q[j,i] = beta

    print("Simple Q built in", time.time()-start)
```

```python
    return Q.tocsc()

# build a conditional Q matrix
def buildFancyQ(im, gamma, alpha=-1.):
    start = time.time()

    nrows, ncols, ch = im.shape
    size = nrows * ncols

    Q = scipy.sparse.lil_matrix((size,size), dtype='float')

    # off diagonal elements
    def getBeta(c0, c1):
        return alpha * np.exp(-gamma * np.linalg.norm(c0 - c1))

    # vectorized index
    def getInd(x, y):
        return x + y * ncols

    for x in range(nrows):
        for y in range(ncols):

            c0 = im[x,y]
            i = getInd(x,y)

            if x < ncols-1:
                j = getInd(x+1, y)

                beta = getBeta(c0, im[x+1, y])
                Q[i,j] = beta
                Q[j,i] = beta

                # add values to corresponding diagonal elements
                Q[i,i] += abs(beta)
                Q[j,j] += abs(beta)

            if y < nrows-1:
                j = getInd(x, y+1)

                beta = getBeta(c0, im[x, y+1])
                Q[i,j] = beta
                Q[j,i] = beta

                Q[i,i] += abs(beta)
                Q[j,j] += abs(beta)

    print("Fancy Q built in", time.time()-start)

    return Q.tocsc()

# denoise an image with a given Q matrix
def denoise(img, Q, sigma = 1.0):
```

```
    shape = img.shape
    size = shape[0] * shape[1]

    # init result image
    result = np.zeros(shape)

    # (I + sigma^2 Q) z = x
    # A = I + sigma^2 Q
    A = scipy.sparse.identity(size, format='csc') + sigma**2 * Q

    for c in range(3):
        start = time.time()

        x = img[:,:,c].ravel()

        # z = scipy.sparse.linalg.spsolve(A, x)

        result[:,:,c] = scipy.sparse.linalg.spsolve(A, x).reshape(shape
            [0:2])

        print("Solved for color", c, "in", time.time()-start)

    return result


# time it
start = time.time()

# parameters
noise = 0.15
sigma = 5.0
gamma = 2.0

print(f"Parameters: Noiselevel {noise}, sigma {sigma}, gamma {gamma}")


# load image
gt_img = ski.data.astronaut()
gt_img = norm01(gt_img)

# get image parameters
shape = gt_img.shape
size = shape[0] * shape[1]

# add noise
noisy_img = ski.util.random_noise(gt_img, var=noise**2)
noisy_img = np.clip(noisy_img, 0, 1)

# load noisy image
# noisy_img = ski.io.imread("noisy.png")
# noisy_img = norm01(noisy_img)
```

```python
# constQ = buildConstQ(shape[0:2], -.5)
fancyQ = buildFancyQ(noisy_img, gamma)

# use simple Q
# result_img = denoise(noisy_img, constQ, sigma)
# or
# use fancyQ
result_img = denoise(noisy_img, fancyQ, sigma)
result_img = norm01(result_img)


print("Time from loading image to denoised image:", time.time() - start)


quality = performance(gt_img, result_img)
print("Quality of the denoising:", quality)



# Do something with the images
plt.imsave("noisy.png", noisy_img)
plt.imsave("denoised.png", result_img)

plt.figure(figsize=(10, 6))
# plt.subplot(221)
# plt.imshow(gt_img)
# plt.title("Original")
# plt.axis('off')

plt.subplot(121)
plt.imshow(noisy_img)
plt.title("Noisy")
plt.axis('off')

plt.subplot(122)
plt.imshow(result_img)
plt.title("Denoised")
plt.axis('off')

plt.suptitle(performance(gt_img, result_img))

plt.savefig("result.png")
```