Machine Learning for Computer Vision

Exercise 8

Kodai Matsuoke, Yuyan Li

June 27, 2017

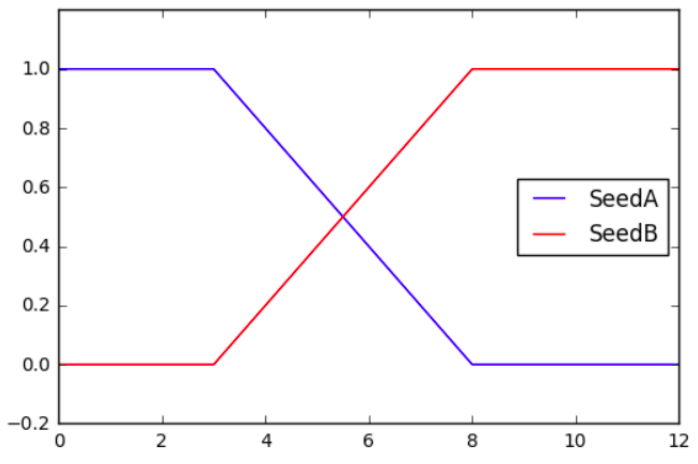
1 Random Walker

1.1 1D Random walker

Suppose x_i is the probability that a random walk particle started from position i first reaches to Seed A. Obviously, for i $\in \{0,1,2,3\}, x_i = 1$. also for i $\in \{8,9,10,11,12\}, x_i = 0$.
So now, we want to calculate x_4, x_5, x_6, x_7. We formulate this problem as follows.

$$x = (x_4, x_5, x_6, x_7) \quad x_m = (x_3, x_8) \quad A = \begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix} \quad B = \begin{pmatrix} -1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & -1 \end{pmatrix}$$

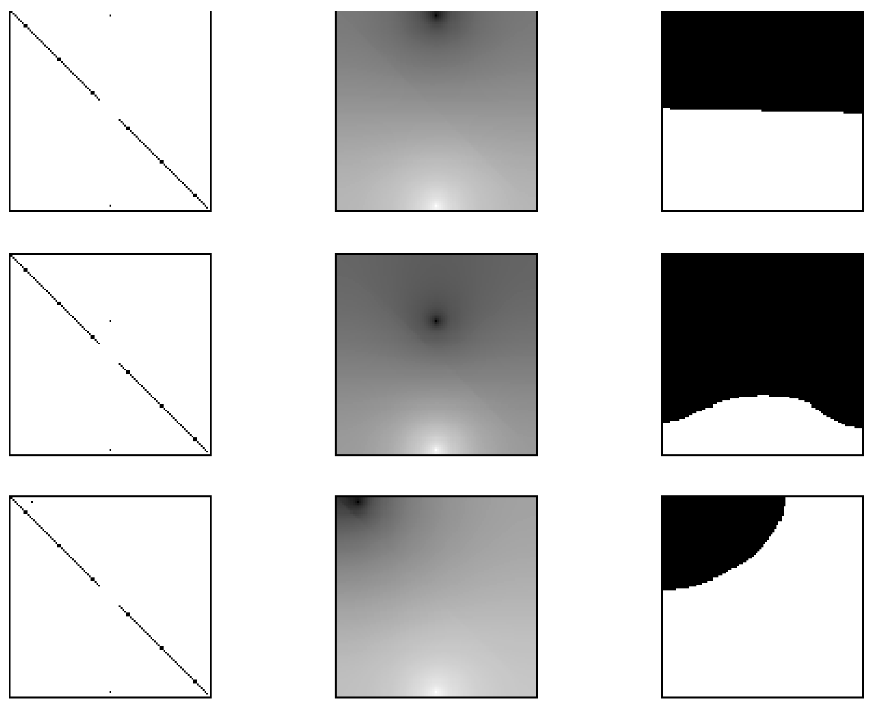
By solving $Ax = -Bx_m$, we obtain $x = (0.8, 0.6, 0.4, 0.2)$. As well we can calculate the probability for Seed B. The overall probability is as shown below.



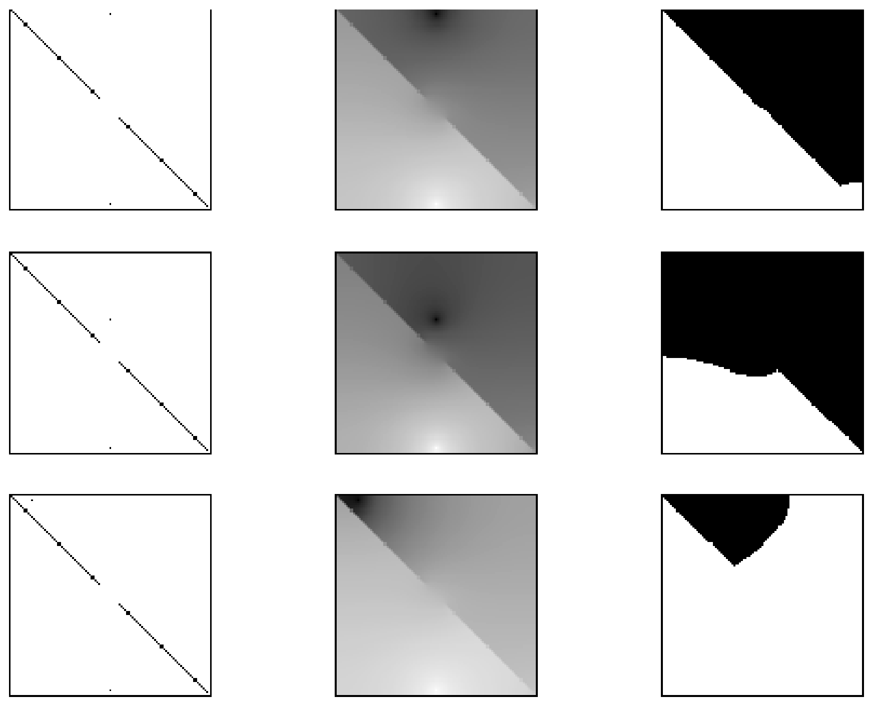
1.2 Implementation

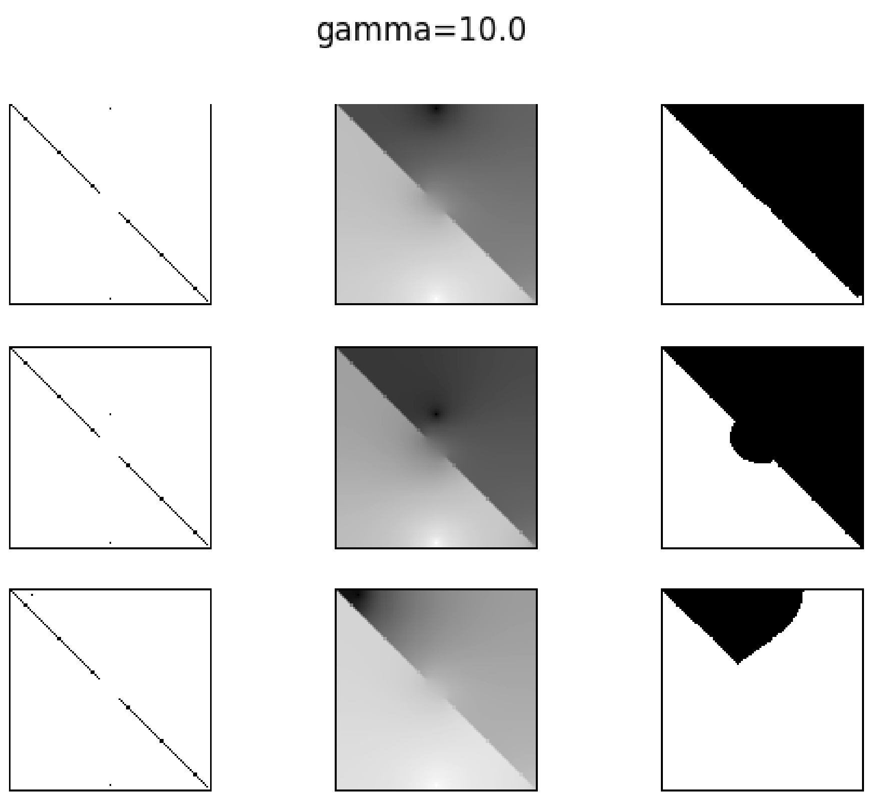
Below is our simple implementation of the random walker. We experimented with different seed positions and different values of γ. One can see that for small γ the random walker doesn't properly respect the borders in the picture. For sufficiently large values of γ the algorithm gives a satisfying result.

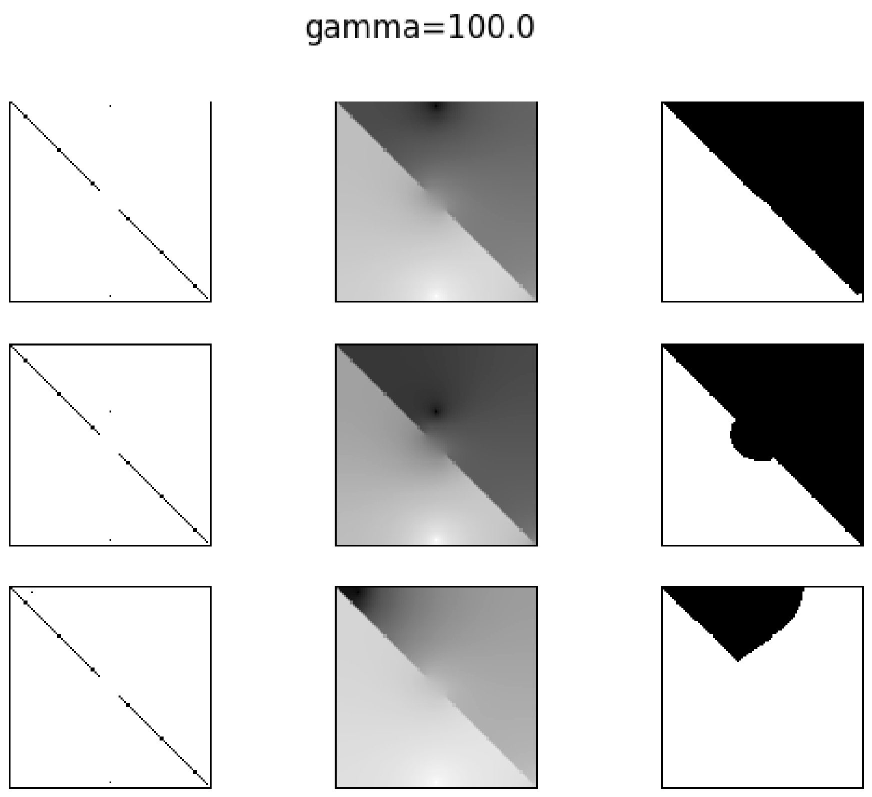
gamma=1.1



gamma=5.0




gamma=10.0


gamma=100.0

Listing 1: random_walker.py

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.sparse
import scipy.sparse.linalg
import time


def buildFancyQ(im, gamma):
    start = time.time()

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

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

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

    # vectorized index
    def getInd(x, y):
        return y + x * 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 diagonal elements in same row
                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()


def simple_random_walker(img, fg, bg, gamma=1.0):
    """
    A simple random walker that takes one seed for foreground and
        background each.
    img: image array
    fg: indices of foreground seed
    bg: indices of background seed
    gamma: weight parameter

    return: probability image, segmentation image

    Using buildFancyQ a laplacian qmat is made
    Bringing qmat into the proper form one can just solve the equation
    A x = B xm
    for x
    (in the code A=a, B=b)
    """
    nrows, ncols = shape = img.shape
    size = nrows * ncols


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


    # Seed indices
    fg_ind = getInd(fg[0], fg[1])
    bg_ind = getInd(bg[0], bg[1])

    label = np.zeros(size)
    label[fg_ind] = 1
    label[bg_ind] = 2

    indizes = np.arange(size)
    unseeded_indizes = indizes[label == 0]
    seeded_indizes = indizes[label > 0]


    qmat = buildFancyQ(img, gamma)
    a = qmat[unseeded_indizes][:,unseeded_indizes]
    b = qmat[unseeded_indizes][:,seeded_indizes]


    xms = []
    for i in range(1,3):
        mask = (label[seeded_indizes] == i)
        xm = scipy.sparse.csr_matrix(mask)
        xm = xm.transpose()
        xms.append(xm)


    xs = []
    for xm in xms:
        x = scipy.sparse.linalg.spsolve(a, -b.dot(xm))
        xs.append(x)

    label[bg_ind] = 0

    probs = []
    for x in xs:
        prob = np.zeros(size)
        prob[seeded_indizes] = label[seeded_indizes]
        prob[unseeded_indizes] = x
        probs.append(prob)

    pred = np.zeros(size)
    pred[seeded_indizes] = label[seeded_indizes]
    fg_indizes = indizes[probs[0] > probs[1]]
    pred[fg_indizes] = 1


    return probs[0].reshape(img.shape), pred.reshape(img.shape)


def experiment(gamma):
    """
    Experiment with different foreground seed positions

    return: figure with images
    """
    n = 100
    n2 = int(n/2)
    image = np.zeros((n+1, n+1))

    # Add a diagonal line
    for i in range(n):
        image[i,i] = 1

    # Add an opening in the line
    o = int(n/20)
    for i in range(n2-o, n2+o):
        image[i,i] = 0

    # Foreground and background seed
    fgs = [(2, n2), (int(n/3), n2), (2,10)]
    bg = (n-2, n2)

    probs, preds = [], []
    images = []
    for fg in fgs:
        prob, pred = simple_random_walker(image, fg, bg, gamma)

        # Add a border
        prob = np.lib.pad(prob, ((1,1),(1,1)), 'constant', constant_values
            =(1))
        pred = np.lib.pad(pred, ((1,1),(1,1)), 'constant', constant_values
            =(1))
        probs.append(prob)
        preds.append(pred)

        # Draw seeds
        img = image.copy()
        img[fg] = 1
        img[bg] = 1
        img = np.lib.pad(img, ((1,1),(1,1)), 'constant', constant_values
            =(1))
        images.append(img)


    fig, ax = plt.subplots(3, 3)

    ax[0,0].imshow(images[0], cmap='Greys', interpolation='nearest')
    ax[0,0].axis('off')

    ax[0,1].imshow(probs[0], cmap='Greys')
    ax[0,1].axis('off')

    ax[0,2].imshow(preds[0], cmap='Greys')
    ax[0,2].axis('off')

    ax[1,0].imshow(images[1], cmap='Greys', interpolation='nearest')
    ax[1,0].axis('off')

    ax[1,1].imshow(probs[1], cmap='Greys')
    ax[1,1].axis('off')

    ax[1,2].imshow(preds[1], cmap='Greys')
    ax[1,2].axis('off')

    ax[2,0].imshow(images[2], cmap='Greys', interpolation='nearest')
    ax[2,0].axis('off')

    ax[2,1].imshow(probs[2], cmap='Greys')
    ax[2,1].axis('off')

    ax[2,2].imshow(preds[2], cmap='Greys')
    ax[2,2].axis('off')

    fig.suptitle(f"gamma={gamma}")

    return fig
```

```
if __name__ == "__main__":
    for gamma in [1.1, 5.0, 50.0, 100.0]:
        fig = experiment(gamma)
        g = int(gamma)
        fig.savefig(f"walker{g}")
```

2 Random walker as Anisotropic Diffusion

$$\frac{\partial\phi}{\partial t} = D\frac{\partial^2\phi}{\partial x^2}$$

When a large number of particles make randomwalk, diffusion equation describes the time developement of the particle density. I am going to show it.
Let's suppose that the particle at position x makes a random walk such that after time Δt, it moves to x+Δx with probability 1/2, x-Δx with probability 1/2.
Now, the following equation goes with regard to the particle density ϕ

$$\phi(x, t+\Delta t) = \frac{1}{2}\phi(x-\Delta x, t) + \frac{1}{2}\phi(x+\Delta x, t)$$

Using Taylor expansion, it can be transformed as follows.

$$\frac{\partial\phi}{\partial t} + O(\Delta t) = \frac{\Delta x^2}{\Delta t}\frac{\partial^2\phi}{\partial x^2} + O(\frac{\Delta x^3}{\Delta t})$$

Take colloids in the water as a example of diffusion. They do brownian movement and their mean travel distance is proportional to the square root of time. For this reason, we can say that $\frac{\Delta x^2}{\Delta t}$ is constant if we take Δx and Δt in proper range. Call constant D, by bringing Δx and Δt close to 0, we obtain a diffusion equation.