BICK-hw5

October 11, 2022

1 Problem 1

Create a 5×5 matrix A using the command `hilb(5)` in Matlab, or `scipy.linalg.hilbert(5)` in Python. Generate a random vector x, and compute $b = Ax$. Add a tiny amount of noise to b, call it $\hat{b}$. Then recover $\hat{x}$ from $A\hat{x} = \hat{b}$.

How accurate is the recovered solution? Why did this happen? You don't need to provide any code or console output, just describe what you did and what you got in a few sentences.

[75]:
```
import scipy
import pandas as pd
import numpy as np
import random
```

What I did was generate A, a random x, and then calculate b. I see that if we make a tiny amount of noise change to b, there is a very large change to x. That is, the difference between x and x-hat is very large. This means that this system is very unstable.

Below we compare x to x-hat.

[7]:
```
A = scipy.linalg.hilbert(5)

x = np.random.rand(5,1)
print("The following is x")
print(x)

b = np.matmul(A,x)
print("The following is b")
print(b)

epsilon = np.random.uniform(-1,1,(5,1)) / 1000

b_hat = b + epsilon

print("The following is b-hat")
print(b_hat)

x_hat = np.linalg.solve(A, b_hat)
print("The following is x-hat")
```

```
print(x_hat)
```

```
The following is x
[[0.19166311]
 [0.95892318]
 [0.37570231]
 [0.52487727]
 [0.32844751]]
The following is b
[[0.99326762]
 [0.6691149 ]
 [0.51315958]
 [0.41835587]
 [0.35392874]]
The following is b-hat
[[0.9936499 ]
 [0.66831323]
 [0.51268276]
 [0.4177889 ]
 [0.35325633]]
The following is x-hat
[[ 0.31121013]
 [-0.7598628 ]
 [ 6.62956426]
 [-7.78092293]
 [ 3.98873572]]
```

2 Problem 2

Construct any 3×3 invertible symmetric matrix with no entry equal to 0.

a) Using the function `eig` in Matlab (or equivalent) to find the dominant eigenvalue λ^*_{max} and its corresponding eigenvector v^*.

b) Use the Power Method to find the approximate dominate eigenvector $v^{(k)}$ and eigenvalue μ_k of this matrix for different stopping criteria.

$$\frac{||v^{(k)} - v^*||}{||v^*||_2} \leq \epsilon$$

Record these data in the folloiwing table for given different values.

Note that in practice, we don't know the exact eigenvalues and eigenvectors. So the stopping criteria needs to be replaced by

$$\frac{||v^{(k)} - v^{(k-1)}||}{||v^{(k-1)}||_2}$$

First we respond to part a. We use the eig function on a constructed matrix. And we present the dominant eigenvector and eigenvalue pair.

[71]:
```
A = np.array([
  [1,2,3],
  [2,5,6],
  [3,6,9]
])

A_eig = np.linalg.eig(A)

dominant_e_val = A_eig[0][0]
dominant_e_vec = A_eig[1].transpose()[0]*1

print("Dominant eigenvalue is:")
print(dominant_e_val)

print("Dominant eigenvector is:")
print(dominant_e_vec)
```

```
Dominant eigenvalue is:
14.30073525436773
Dominant eigenvector is:
[0.2614964  0.56231339 0.78448919]
```

For part b, we implement the power method. Following the desription of the method in the lecture, we first get the $v^{(i-1)}$ vector and proceed from there.

[77]:
```
for stop_point in [1e-3,1e-6,1e-9]:
    iter = 0
    x = np.array([1,1,1])
    v = x/np.linalg.norm(x)
    eigval = np.matmul(v.transpose(),x)
    # define stopping criteria
    stopping_criterion = abs(eigval - dominant_e_val)

    while stopping_criterion > stop_point:
        iter += 1
        v_old = v
        x = np.matmul(A,v)
        v = x/np.linalg.norm(x)
        eigval = np.matmul(v_old.transpose(),x)
        stopping_criterion = abs(eigval - dominant_e_val)
    print(stop_point)
    print(iter)
    print(x)
    print(v)
    print(eigval)
    print(stopping_criterion)
```

```
0.001
```

```
2
[ 3.73913043  8.04347826 11.2173913 ]
[0.26146582 0.56245555 0.78439746]
14.300567107750473
0.000168146617257392
1e-06
3
[ 3.73956931  8.04159416 11.21870792]
[0.2614949  0.56232034 0.78448471]
14.300734852336273
4.0203145701411813e-07
1e-09
4
[ 3.7395897   8.04149973 11.21876909]
[0.26149632 0.56231373 0.78448897]
14.300735253406495
9.612346474341393e-10
```

[73]:
```
for stop_point in [1e-3,1e-6,1e-9]:
    iter = 0
    x = np.array([1,1,1])
    v = x/np.linalg.norm(x)
    eigval = np.matmul(v.transpose(),x)
    # define stopping criteria
    stopping_criterion = np.linalg.norm(v - dominant_e_vec)/np.linalg.
 ↪norm(dominant_e_vec)

    while stopping_criterion > stop_point:
        iter += 1
        v_old = v
        x = np.matmul(A,v)
        v = x/np.linalg.norm(x)
        eigval = np.matmul(v_old.transpose(),x)
        stopping_criterion = np.linalg.norm(v - dominant_e_vec)/np.linalg.
 ↪norm(dominant_e_vec)
    print(stop_point)
    print(iter)
    print(x)
    print(v)
    print(eigval)
    print(stopping_criterion)
```

```
0.001
2
[ 3.73913043  8.04347826 11.2173913 ]
[0.26146582 0.56245555 0.78439746]
14.300567107750473
0.00017192422741231806
```

```
1e-06
4
[ 3.7395897   8.04149973 11.21876909]
[0.26149632 0.56231373 0.78448897]
14.300735253406495
4.1105848686689787e-07
1e-09
6
[ 3.73959074  8.04149488 11.21877222]
[0.2614964  0.56231339 0.78448919]
14.30073525436772
9.828113264581863e-10
```

[74]:
```
for stop_point in [1e-3,1e-6,1e-9]:
    iter = 0
    x = np.array([1,1,1])
    v = x/np.linalg.norm(x)
    eigval = np.matmul(v.transpose(),x)
    # define stopping criteria
    stopping_criterion = np.linalg.norm(v - dominant_e_vec)/np.linalg.
 ↪norm(dominant_e_vec)

    while stopping_criterion > 0.001:
        iter += 1
        v_old = v
        x = np.matmul(A,v)
        v = x/np.linalg.norm(x)
        eigval = np.matmul(v_old.transpose(),x)
        stopping_criterion = np.linalg.norm(v - v_old)/np.linalg.norm(v_old)
    print(stop_point)
    print(iter)
    print(x)
    print(v)
    print(stopping_criterion)
```

```
0.001
3
[ 3.73956931  8.04159416 11.21870792]
[0.2614949  0.56232034 0.78448471]
0.00016351762803709012
1e-06
3
[ 3.73956931  8.04159416 11.21870792]
[0.2614949  0.56232034 0.78448471]
0.00016351762803709012
1e-09
3
[ 3.73956931  8.04159416 11.21870792]
```

```
[0.2614949  0.56232034 0.78448471]
0.00016351762803709012
```

We get the following results table:

$|\mu_k - \lambda^*_{max}|$

ϵ	iteration	value
1e-3	2	0.000168146617257392
1e-6	3	4.0203145701411813e-07
1e-9	4	9.612346474341393e-10

$\frac{||v^{(k)}-v^*||}{||v^*||_2} \leq \epsilon$

ϵ	iteration	value
1e-3	2	0.00017192422741231806
1e-6	3	4.1105848686689787e-07
1e-9	4	9.828113264581863e-10

$\frac{||v^{(k)}-v^{(k-1)}||}{||v^{(k-1)}||_2}$

ϵ	iteration	value
1e-3	3	0.0.00016351762803709012
1e-6	3	0.00016351762803709012
1e-9	3	0.00016351762803709012

3 Problem 3

Build a connected network graph of 5 nodes, that is, a network with 5 pages. Determine the highest rated web page using the page rank approach discussed in the lecture.

In the following code, we follow the example given in the lecture slides. The eigenvalue calculatoin shows us that the node with the largest page rank is the 4th node, out of 5.

We first start with an adjacency matrix, and create a transition probability matrix. We find the eigen pairs of its transpose, and then can calcualte the value needed for page rank algorithm.

[81]:
```
# define adjacency matrix with 5 nodes
A = np.array([
    [0,1,1,1,0],
    [0,0,0,1,0],
    [0,0,0,0,1],
    [1,0,1,0,1],
    [0,0,0,1,0]
])
```

```
# turn this previous matrix into transition probability matrix
B = A/A.sum(axis=1)[:,None]

# then B transpose
B_T = B.transpose()

# get the eigenpairs
B_T_eig = np.linalg.eig(B_T)

# get the e-vals and e-vecs distinctly
B_T_e_vals = B_T_eig[0]
B_T_e_vecs = B_T_eig[1].transpose()

B_T_e_vecs[0]/B_T_e_vecs[0].sum(axis=0)
```

```
[[0.         0.33333333 0.33333333 0.33333333 0.        ]
 [0.         0.         0.         1.         0.        ]
 [0.         0.         0.         0.         1.        ]
 [0.33333333 0.         0.33333333 0.         0.33333333]
 [0.         0.         0.         1.         0.        ]]
```

[81]:
```
array([0.125     +0.j, 0.04166667+0.j, 0.16666667+0.j, 0.375     +0.j,
       0.29166667+0.j])
```