Numerical Analysis Homework 5
APPM 4650

Zachary Vogel

December 10, 2015

Problem 1

Here we consider:

$$\begin{pmatrix} 4 & 3 & 0 \\ 3 & 4 & -1 \\ 0 & -1 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 24 \\ 30 \\ -24 \end{pmatrix}$$

Solve this system of equations using Gauss-Seidel with relaxation. Determine the optimum relaxation factor ω where $x_{i+1} = x_i + \omega \frac{r_i}{a_{ii}}$ to get the solution within 6 decimal places.
Here, I found the optimal relaxation factor to be 1.25. The iterative solutions to this problem can be seen below, and the code in the Appendix 1.

```
current val: [ 0.  0.  0.]
0
current val: [ 7.5         2.34375    -6.76757812]
1
current val: [ 3.42773438  3.46069336 -4.72663879]
2
current val: [ 3.39866638  3.8465023  -5.11630833]
3
current val: [ 3.04423749  3.96055542 -4.98324935]
4
current val: [ 3.02591992  3.9907958  -5.00706398]
5
current val: [ 3.00214896  3.99807891 -4.99883435]
6
current val: [ 3.00126378  3.99965974 -5.00039774]
7
current val: [ 3.00000305  3.99995791 -4.99991372]
8
current val: [ 3.00003869  4.00000121 -5.00002119]
9
current val: [ 2.99998919  4.00000321 -4.9999937 ]
10
current val: [ 2.9999997   4.00000145 -5.00000112]
11
current val: [ 2.99999871  4.00000049 -4.99999957]
12
current val: [ 2.99999986  4.00000014 -5.00000006]
13
current val: [ 2.9999999   4.00000004 -4.99999997]
14
current val: [ 2.99999999  4.00000001 -5.        ]
15
Solution
[ 2.99999999  4.00000001 -5.        ]
b:
[ 23.99999999  30.00000001 -24.00000003]
```

Problem 2

Consider the matrix:

$$A = \begin{pmatrix} 3 & 0 & 1 \\ 0 & 5 & 0 \\ -1 & 1 & -1 \end{pmatrix}$$

(a)

Calculate A^{-1} exactly using the cofactor method.
The output of my script that calculated this is below, the code is in appendix 2.

```
[zap@WIN hw5]$ python cofactor.py
A inverse is:
[[ 0.5 -0.1  0.5]
 [-0.   0.2 -0. ]
 [-0.5  0.3 -1.5]]
[zap@WIN hw5]$
```

(b)

Start with the inital approximation to A^{-1} of:

$$x_0 = \begin{pmatrix} 0.5 & -0.1 & 0.4 \\ 0 & 0.2 & 0 \\ -0.4 & 0.3 & -1.5 \end{pmatrix}$$

and use the iterative method $x_{i+1} = x_i(2I - Ax_i)$ to calculate the next approximation x_1.
The solution produced by my code can be seen below, the code is in appendix 2.

```
x1 is
[[ 0.49 -0.1   0.51]
 [ 0.    0.2   0.  ]
 [-0.51  0.3  -1.47]]
```

(c)

Calculate the deviations of x_0 and x_1 from the true inverse matrix A^{-1}.
The actual inverse is:

$$A^{-1} = \begin{pmatrix} 0.5 & -0.1 & 0.5 \\ 0 & 0.2 & 0 \\ -0.5 & 0.3 & -1.5 \end{pmatrix}$$

The deviations can be seen below:

```
A^(-1)-x0
[[  0.00000000e+00   1.38777878e-17   1.00000000e-01]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00]
 [ -1.00000000e-01   0.00000000e+00   2.22044605e-16]]
A^(-1)-x1
[[  1.00000000e-02   1.38777878e-17  -1.00000000e-02]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00]
 [  1.00000000e-02  -5.55111512e-17  -3.00000000e-02]]
```

Problem 3

Use the power method to find the dominant eigenvalue λ and the corresponding eigenvector V for the matrix:

$$A = \begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix}$$

(a)

Start the procedure with the initial vector $x_0 = (1, 1, 1, 1)^T$.
It's interesting that this converged to the second largest eigenvalue. The code is in appendix 3, the output can be seen below.

```
iteration=0

[[ 0.70710678]
 [ 0.        ]
 [ 0.        ]
 [ 0.70710678]]
[[ 1.41421356]]
iteration=1

[[ 0.63245553]
 [-0.31622777]
 [-0.31622777]
 [ 0.63245553]]
[[ 2.23606798]]
iteration=2

[[ 0.60633906]
 [-0.36380344]
 [-0.36380344]
 [ 0.60633906]]
[[ 2.60768096]]
iteration=3

[[ 0.60221337]
 [-0.37059284]
 [-0.37059284]
 [ 0.60221337]]
[[ 2.61781229]]
iteration=4
```

```
iteration=4

[[ 0.60160503]
 [-0.37157958]
 [-0.37157958]
 [ 0.60160503]]
[[ 2.61802927]]
iteration=5

[[ 0.60151614]
 [-0.37172346]
 [-0.37172346]
 [ 0.60151614]]
[[ 2.61803389]]
iteration=6

[[ 0.60150317]
 [-0.37174445]
 [-0.37174445]
 [ 0.60150317]]
[[ 2.61803399]]
iteration=7

[[ 0.60150128]
 [-0.37174751]
 [-0.37174751]
 [ 0.60150128]]
[[ 2.61803399]]
iteration=8

[[ 0.601501  ]
 [-0.37174796]
 [-0.37174796]
 [ 0.601501  ]]
[[ 2.61803399]]
```

(b)

Now, repeat the calculations starting with $x_0 = (1, 1, 5, 1)^T$.
Again, the output is below, code in appendix 3.

```
iteration=0

[[ 0.10540926]
 [-0.42163702]
 [ 0.84327404]
 [-0.31622777]]
[[ 9.48683298]]
iteration=1

[[ 0.18516402]
 [-0.52463139]
 [ 0.70979541]
 [-0.43204938]]
[[ 3.41565026]]
iteration=2

[[ 0.2510846 ]
 [-0.54545966]
 [ 0.66667291]
 [-0.44156258]]
[[ 3.56437398]]
iteration=3

[[ 0.29147743]
 [-0.55886581]
 [ 0.64558637]
 [-0.43119388]]
[[ 3.59420239]]
iteration=4
```

```
iteration=30

[[ 0.3717367 ]
 [-0.60149395]
 [ 0.60150796]
 [-0.37175937]]
[[ 3.61803399]]
iteration=31

[[ 0.37173983]
 [-0.60149589]
 [ 0.60150602]
 [-0.37175623]]
[[ 3.61803399]]
iteration=32

[[ 0.3717421 ]
 [-0.60149729]
 [ 0.60150462]
 [-0.37175397]]
[[ 3.61803399]]
iteration=33

[[ 0.37174374]
 [-0.6014983 ]
 [ 0.60150361]
 [-0.37175233]]
[[ 3.61803399]]
iteration=34

[[ 0.37174493]
 [-0.60149903]
 [ 0.60150288]
 [-0.37175114]]
[[ 3.61803399]]
```

(c)

Commment on the results from parts (a) and (b).

Problem 4

Consider the matrix and initial vector:

$$A = \begin{pmatrix} 2 & -1 & 1 \\ -1 & 3 & 2 \\ 1 & 2 & 3 \end{pmatrix} \quad x_0 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

(a)

Calculate the Rayleigh quotient and the error estimate using x_2 and x_3.
Didn't know how to do the error estimate, but the first few values can be seen below. Code is in appendix 4.

```
[zap@WIN hw5]$ python Rayleigh.py
iteration= 0

x is:
[[-0.41294832]
 [-0.59648091]
 [-0.6882472 ]]
the eigenvalue is:
[[ 4.]]
iteration= 1

x is:
[[ 0.33814787]
 [-0.78123819]
 [-0.52471222]]
the eigenvalue is:
[[ 4.54736842]]
iteration= 2

x is:
[[-0.10069692]
 [-0.65602718]
 [-0.74798962]]
the eigenvalue is:
[[ 4.69884432]]
iteration= 3

x is:
[[ 0.01764433]
 [-0.71548541]
 [-0.69840483]]
the eigenvalue is:
[[ 4.97118688]]
```

(b)

Use the power method to find $\lambda_{\max}$ and the corresponding $V_{\max}$.
Values can be seen below, code is in appendix 4.

```
[zap@WIN hw5]$ python Pow2.py
iteration=0

[[ 0.26726124]
 [ 0.53452248]
 [ 0.80178373]]
[[ 7.48331477]]
iteration=1

[[ 0.16615463]
 [ 0.60923365]
 [ 0.77538829]]
[[ 4.82552736]]
iteration=2

[[ 0.10104072]
 [ 0.65115128]
 [ 0.75219199]]
[[ 4.93329741]]
iteration=3

[[ 0.06092372]
 [ 0.67467375]
 [ 0.73559747]]
[[ 4.97543753]]
iteration=4

[[ 0.03661953]
 [ 0.68808549]
 [ 0.72470502]]
[[ 4.99108397]]
iteration=5

[[ 0.02198587]
 [ 0.69585745]
 [ 0.71784332]]
[[ 4.99678059]]
```

```
iteration=9

[[ 0.00285039]
 [ 0.70567728]
 [ 0.70852767]]
[[ 4.99994584]]
iteration=10

[[ 0.00171024]
 [ 0.70625011]
 [ 0.70796035]]
[[ 4.9999805]]
iteration=11

[[ 0.00102614]
 [ 0.70659315]
 [ 0.70761929]]
[[ 4.99999298]]
iteration=12

[[  6.15686845e-04]
 [  7.06798737e-01]
 [  7.07414424e-01]]
[[ 4.99999747]]
iteration=13

[[  3.69412174e-04]
 [  7.06922003e-01]
 [  7.07291415e-01]]
[[ 4.99999909]]
iteration=14

[[  2.21647319e-04]
 [  7.06995931e-01]
 [  7.07217579e-01]]
[[ 4.99999967]]
```

A Appendix 1

```
import numpy as np

A=np.array([[4., 3., 0.],[3., 4., -1.],[0., -1., 4.]])
b=np.array([24.,30.,-24.])

n=A.size

xold=np.array([2.,2.,2.])

x=np.zeros_like(xold);
j=0
w=1.25


while ~(np.allclose(xold,x,rtol=1e-14)):
    xold=x
    print ("current val:",xold)
    x=np.zeros_like(xold)

    for i in range(A.shape[0]):
        a1=np.dot(A[i,:i],x[:i])
        a2=np.dot(A[i,i+1:],xold[i+1:])
        s=(b[i]-a1-a2)/A[i,i]
        x[i]=(1-w)*xold[i]+w*s
    print(j)
    j=j+1

print("Solution")
print(xold)
print("b:")
print(np.dot(A,xold))
```

B Appendix 2

```
import numpy as np

#dat cofactor method for inverse matrix finding
#inverse is 1/det*adjugate which is checkerboard of +,- on the og matrix transposed.

A=np.array([[3,0,1],[0,5,0],[-1,1,-1]])
n=A[:,1].size
Ain=np.zeros_like(A)


def twobytwo(A,i,j):
    a=(i+1)%n
    b=(i+2)%n
    c=(j+1)%n
    d=(j+2)%n
    if (i+j)%2==0:
        return A[a,c]*A[b,d]-A[a,d]*A[b,c]
    else:
        return A[a,d]*A[b,c]-A[a,c]*A[b,d]

B=np.zeros_like(A)
for i in range(n):
```

```
    for j in range(n):
        B[i,j]=twobytwo(A,i,j)

#find the adjunct matrix
for i in range(n):
    for j in range(n):
        if (i+j)%2==0:
            Ain[j,i]=B[i,j]
        else:
            Ain[j,i]=-B[i,j]

det=B[0,0]*A[0,0]-B[0,1]*A[0,1]+B[0,2]*A[0,2]
Ain=1/det*Ain

print("A inverse is:")
print(Ain)

import numpy as np


A=np.array([[3,0,1],[0,5,0],[-1,1,-1]])
n=A[:,1].size
x1=np.zeros_like(A)
x0=np.array([[0.5,-0.1,0.4],[0,0.2,0],[-0.4,0.3,-1.5]])
I=np.identity(3)

x1=np.dot(x0,(2*I-np.dot(A,x0)))

print("x1 is")
print(x1)

Ain=np.linalg.inv(A)

print("A^(-1)-x0")
print(Ain-x0)

print("A^(-1)-x1")
print(Ain-x1)
```

C Appendix 3

```
import numpy as np

A=np.array([[2,-1,0,0],[-1,2,-1,0],[0,-1,2,-1],[0,0,-1,2]])
x0=np.array([[1],[1],[1],[1]])
x1=np.array([[1],[1],[5],[1]])
tol=10**(-6)


xold=np.zeros_like(x0)
n=0
while (np.abs(xold-x0)>tol).all():
    xold=x0
    s=np.dot(A,x0)
    x0=s/np.linalg.norm(s)
    xt=np.transpose(x0)
    lam=np.dot(xt,s)/(np.dot(xt,x0))
    s="iteration=%d\n"% n
```

```
    print(s)
    print(x0)
    print(lam)
    n=n+1


print()
n=0
while (np.abs(xold-x1)>tol).all():
    xold=x1
    s=np.dot(A,x1)
    x1=s/np.linalg.norm(s)
    xt=np.transpose(x1)
    lam=np.dot(xt,s)/(np.dot(xt,x1))
    s="iteration=%d\n"% n
    print(s)
    print(x1)
    print(lam)
    n=n+1
```

D Appendix 4

```
import numpy as np

A=np.array([[2,-1,1],[-1,3,2,],[1,2,3]])
x0=np.array([[1],[1],[1]])
lam=10
I=np.identity(3)

n=0
while n<4:
    f=np.linalg.inv(A-lam*I)
    s=np.dot(f,x0)
    xt=np.transpose(x0)
    lam=np.dot(xt,np.dot(A,x0))/np.dot(xt,x0)
    x0=s/np.linalg.norm(s)
    r=("iteration= %d\n")%(n)
    print(r)
    print("x is:")
    print(x0)
    print("the eigenvalue is:")
    print(lam)
    n=n+1

import numpy as np

A=np.array([[2,-1,1],[-1,3,2,],[1,2,3]])
x0=np.array([[1],[1],[1]])
tol=10**(-4)


xold=np.zeros_like(x0)
n=0
while (np.abs(xold-x0)>tol).all():
    xold=x0
    s=np.dot(A,x0)
    x0=s/np.linalg.norm(s)
    xt=np.transpose(x0)
```

```
lam=np.dot(xt,s)/(np.dot(xt,x0))
s="iteration=%d\n"% n
print(s)
print(x0)
print(lam)
n=n+1
```