HW8, Dept.: 수리과학과, NAME: 국윤범

Problem 1

(a)

Power Iteration

```
function [ev, ew, count] = pw_ite(A, v)
% input : m*m real symmetric matrix A and initial vector v
% output : (usually) the largest eigenvalue, its eigenvector and iteration
% count
count = 0;
ev = v;
%sprintf("power_iteration")
while 1
    ew_previous = ev' * A * ev;
    ev = A * ev; ev = ev / norm(ev);
    ew = ev' * A * ev;
    if count <= 2
        sprintf("Iteration %d : %.8f", count+1, ew)
    end
    count = count + 1;
    if norm(ew - ew_previous) < 1e-12
       break
    end
end
end
```

Inverse Iteration

```
function [ev, ew, count] = inv_ite(A, v, mu)
% input : m*m real symmetric matrix A,
%         initial vector v and mu
% output : (usually) the largest eigenvalue, its eigenvector and iteration
% counter
[~, m] = size(A); count = 0;
ev = v / norm(v); A_trans = inv(A - mu * eye(m));
%sprintf("inverse_iteration")
while 1
    ew_previous = ev' * A * ev;
    ev = A_trans * ev;
    ev = ev / norm(ev);
    ew = ev' * A * ev;
    if count <= 2
        sprintf("Iteration %d : %.8f", count+1, ew)
    end
    count = count + 1;
    if norm(ew - ew_previous) < 1e-12
       break
    end
end
end
```

Rayleigh Quotient Iteration

```
function [ev, ew, count] = rq_ite(A, v, mu)
% input : m*m real symmetric matrix A,
%         initial vector v and mu
% output : (usually) the largest eigenvalue, its eigenvector and iteration
% counter
[~, m] = size(A); count = 0;
ev = v / norm(v); ew = mu;
%sprintf("rayleigh_quotient_iteration")
while 1
    ew_previous = ew;
    ev = (A-ew * eye(m))\ev;
    ev = ev / norm(ev);
    ew = ev' * A * ev;
    if count <= 2
        sprintf("Iteration %d : %.8f", count+1, ew)
    end
    count = count + 1;
    if norm(ew - ew_previous) < 1e-12
       break
    end
end
end
```

(b)

```
A = [2 1 1; 1 3 1; 1 1 4];

% (b)
[V, D] = eig(A)
```

yields

```
V =

    0.8877    0.2332    0.3971
   -0.4271    0.7392    0.5207
   -0.1721   -0.6318    0.7558


D =

    1.3249         0         0
         0    2.4608         0
         0         0    5.2143
```

1.324869129433354
2.460811127189110
5.214319743377535

(c)

```
% (c)
v = [1, 1, 1]' / sqrt(3);
mu = v' * A * v;

[pw_ev, pw_ew, pw_count] = pw_ite(A, v);
[inv_ev, inv_ew, inv_count] = inv_ite(A, v, mu);
[rq_ev, rq_ew, rq_count] = rq_ite(A, v, mu);
```

From above, I obtain the following results.

```
pw_count =  pw_ew =

    18         5.2143

inv_count = inv_ew =

     7         5.2143

rq_count =  rq_ew =

     4         5.2143
```

and (Power, Inverse, Rayleight)'s \lambda^1, 2, 3 is

```
ans =                              ans =                              ans =

    "Iteration 1 : 5.18181818"     "Iteration 1 : 5.21311475"     "Iteration 1 : 5.21311475"

ans =                              ans =                              ans =

    "Iteration 2 : 5.20819277"     "Iteration 2 : 5.21431262"     "Iteration 2 : 5.21431974"

ans =                              ans =                              ans =

    "Iteration 3 : 5.21302889"     "Iteration 3 : 5.21431970"     "Iteration 3 : 5.21431974"
```

The true largest eigenvalue is 5.2143197431⋯.

In terms of iteration number, rq_count, inv_count, ane pw_count are in increasing order. Hence, Rayleigh Quotient Iteration is the best among them.

Comparing \lambda^1, 2, 3, we can observe that RQ iteration reaches the true value up to 8 digits only in 2^{nd} step. Inverse iteration reaches up to 7 digits in 3^{rd} step. However, Power iteration yields the result that only matches only up to 2-digits in 3^{rd} step. Hence, RQ > Inverse > Power iteration in terms of error.

(d)

```
%(d)
sprintf("Power Iteration")
tic
it_count=0; S=A;
for i=1:3
    [pw_ev, pw_ew, pw_count] = pw_ite(S, v);
    S = S - pw_ew * pw_ev * pw_ev';
    it_count = it_count + pw_count;
    pw_ew
end
it_count
toc

sprintf("Inverse Iteration")
tic
mu = v'*A*v; it_count = 0; S = A;
for i=1:3
    [inv_ev, inv_ew, inv_count] = inv_ite(S, v, mu);
    S = S - inv_ew * inv_ev * inv_ev';
    it_count= it_count + inv_count;
    inv_ew
end
it_count
toc
sprintf("Rayleigh Quotient Iteration")
tic
it_count = 0; S = A;
mu=[1,2,3];
for i=1:3
    [rq_ev, rq_ew, rq_count] = rq_ite(S, v, mu(i));
    S = S - rq_ew * rq_ev * rq_ev';
    it_count= it_count + rq_count;
    rq_ew
end
it_count
toc
```

By deflation technique, I transformed A into A-ew*ev*ev', which replaces ew by 0 and maintain the other eigenvalues.

My basic assumption for this problem is we have no idea of how the eigenvalues of A are distributed. Hence, I test \mu = 1, 2, ⋯ until given methods come to find all eigenvalues.

```
"Power Iteration"

pw_ew =

    5.2143


pw_ew =

    2.4608


pw_ew =

    1.3249


it_count =

    43
```

```
"Inverse Iteration"

inv_ew =

    5.2143


inv_ew =

    2.4608


inv_ew =

    1.3249


it_count =

    96
```

```
"Rayleigh Quotient Iteration"
> In rq_ite (line 11)
Warning: Matrix is close to singu

rq_ew =

    1.3249

> In rq_ite (line 11)
Warning: Matrix is close to singu

rq_ew =

    2.4608

> In rq_ite (line 11)
Warning: Matrix is close to singu

rq_ew =

    5.2143


it_count =

    17
```

Based on convergence speed (iteration number), RQ iteration stands out among three methods. Compared to the other two methods, RQ iteration requires significantly small number of iteration.

To compare for error, let's see if superiority (RQ > Inverse > Power) among them still retains for the other eigenvalues (2^{nd}, 3^{rd}).

```
ans =

    "Iteration 1 : 1.32486913"

ans =

    "Iteration 2 : 1.32486913"
```

```
    "Inverse Iteration"

ans =

    "Iteration 1 : 2.00000000"

ans =

    "Iteration 2 : 1.33333333"

ans =

    "Iteration 3 : 1.32508834"

ans =

    "Iteration 4 : 1.32487878"
```

```
    "Rayleigh Quotient Iteration"

ans =

    "Iteration 1 : 2.00000000"

ans =

    "Iteration 2 : 1.50000000"

ans =

    "Iteration 3 : 1.33050847"

ans =

    "Iteration 4 : 1.32486927"
```

```
ans =

    "Iteration 1 : 2.26543687"

ans =

    "Iteration 2 : 2.39630053"

ans =

    "Iteration 3 : 2.44132623"
```

```
ans =

    "Iteration 1 : 3.00000000"

ans =

    "Iteration 2 : 2.33333333"

ans =

    "Iteration 3 : 2.38461538"
```

```
ans =

    "Iteration 1 : 3.00000000"

ans =

    "Iteration 2 : 2.50000000"

ans =

    "Iteration 3 : 2.46078431"
```

Power Iteration / Inverse Iteration / Rayleigh Quotient Iteration

As indicated in (b), the other accurate eigenvalues are 1.324869129433354, 2.460811127189110. For the first eigenvalue, RQ is rapidly approaching the accurate value compared to Inverse iteration. However, Power iteration is the best in this case. This seem weird at first glance, however it's quite evident from the way I took to calculate this eigenvalue. That is, I removed all other components except the component associated with this eigenvalue. Hence, possibly, the power iteration happens to estimate almost accurate value at very first iteration.

For the second eigenvalue, RQ iteration is also the best among three methods. Considering all cases, RQ is the best in estimating eigenvalues accurately.

In order to check elapsed time, I wrap whole code (d) like below.

```
time = zeros(3,50);
for k=1:50
    tic
    it_count=0; v0=v;
    [pw ev, pw ew, pw
```

```
    a3 = toc
    time(1,k)=a1;
    time(2,k)=a2;
    time(3,k)=a3;
end
mean(time')'
```

What I'm doing here is repeat (d) 50 times and save elapsed time for each method in the matrix 'time'. After 50 iteration, calculate the mean of elapsed time. Then,

(Power, Inverse, RQ)'s elapsed time is

```
ans =

    0.0013
    0.0042
    0.0077
```

Hence, Power iteration is the fastest, and Inverse iteration and RQ iteration follow in order. Namely, in terms of time, Power iteration is the best.