Section: Power method - Lecture 02

In this lecture we will explaine how to find the second eigenvalue and how to find several eigenvalues together, but not all of them.

[4] Deflation (For symmetric matrix)

So, suppose we have found the first eigenvalue, how to find the second one? We can use deflation.

Method 1

Given a symmetric metrix B, suppose we found the eigenvalue λ_1 and eigenvector v_1, we can define a new matrix $B_2 = B - \lambda_1 v_1 v_1^T$. Then, one can show that the eigenvalues of $B2$ and B are exactly the same except λ_1 had been shifted to 0.

Note that $v_1^\top v_1 = 1$

Example

In the following example we show how the deflation works.

Julia

```
## Define a 3-by-3 matrix A
A = [2 1 5; 5 7 9; 4 6 1];

## Evaluate the eigenvalues
using LinearAlgebra
e_A = eigvals(A);

## Take one of them that to be shifted to 0
lambda = e_A[1];
println(lambda)

## Get the corresponding eigenvector
v_A = eigvecs(A);
v1 = v_A[:,1];
```

```
13.78378635197844
```

```julia
A2 = A - lambda*v1*v1';
e_A2 = eigvals(A2); e_A2 = sort(e_A2);
println("Eigenvalues of A","\t","Eigenvalues of A2")
for ii=1:3
    println(e_A[ii], "\t", e_A2[ii])
end
```

```
Eigenvalues of B    Eigenvalues of B2
13.78378635197844   -4.616660526085693
0.8328741741072435  7.993605777301127e-15
-4.616660526085692  0.8328741741072436
```

Do you see the difference of eigenvalues of A and A_2 ? The first eigenvalue has been shifted to 0 (or almost 0).

So if we apply power method to A_2 we should be able to obtain the second eigenvalue.

Method 2

In fact, the deflation procedure is not unique. In general, we want to choose x such that $x^\top v_1 = 1$, then $A_3 = A - \lambda_1 v_1 x^\top$ should work.

For example, define $x^\top = (1/v_1[1], 0, 0, \ldots, 0)$, and $A_3 = A - \lambda_1 v_1 x^\top$.

(Where $v_1[1]$ is the first component of vector v_1.)

```julia
x = zeros(3);
x[1] = 1/v1[1];
A3 = A - lambda*v1*x';
e_A3 = eigvals(A3); e_A3 = sort(e_A3);
println("Eigenvalues of A","\t","Eigenvalues of A3")
for ii=1:3
    println(e_A[ii], "\t", e_A3[ii])
end
```

```
Eigenvalues of A    Eigenvalues of A3
13.78378635197844   -4.616660526085702
0.8328741741072435  -2.0372592501871623e-14
-4.616660526085692  0.8328741741072727
```

Again, as you can see, the first eigenvalue has been shifted to 0 in A_3.

Method 3 - Wielandt Deflation

There is a very special choice of the vector x. Define $x = A[1,:]/(\lambda_1 v_1[1])$ and $A_4 = A - \lambda v_1 x^\top$.

(Where $A[1,:]$ is the first row vector of A.)

A little bit of bonus in Wielandt Deflation is that the size of the new matrix can be reduced by one, one of its row is entirely zero.

```julia
x = A[1,:]/(lambda*v1[1]);
A4 = A - lambda*v1*x';
e_A4 = eigvals(A4); e_A4 = sort(e_A4);
println("Eigenvalues of B","\t","Eigenvalues of B4")
for ii=1:3
    println(e_A[ii], "\t", e_A4[ii])
end

println()
println("The matrix A4")
for ii=1:3
    println(A4[ii,:])
end
```

```
Eigenvalues of B    Eigenvalues of B4
13.78378635197844   -4.616660526085691
0.8328741741072435  0.0
-4.616660526085692  0.8328741741072467

The matrix A4
[0.0, 0.0, 0.0]
[-1.10705, 3.94647, -6.26763]
[0.507896, 4.25395, -7.73026]
```

We see that the first row of $A4$ is indeed exactly zero.

Example - Wielandt Deflation

In the following, we show a small test explaining that the size of the matrix can be reduced by one.

Start with 4×4 symmetric matrix C, define $x = C[1,:]/(\lambda_1 v_1[1])$, $C_2 = C - \lambda_1 v_1 x^\top$ and $C_3 = C_2[2:4, 2:4]$.

```julia
## Construct a 4-by-4 matrix
n = 4;
C = rand(n,n) .- 0.5; C = C+C';
e_C = eigvals(C);

## Find the smallest eigenvalue and corresponding eigenvector
u = rand(n); u=u/norm(u,2);
sigma = 0; sigma0 = 1;
mu = 1;
eps = 1.0e-10; ii=0;
while abs.(sigma-sigma0)>eps
    sigma0 = sigma;
    global v = (C-sigma*UniformScaling(1))\u;
    v = v/norm(v,2);
    sigma = v'*C*v;
    ii = ii+1;
end
lambda_1 = sigma; v1 = v;

## Use deflation to define the matrix C2
x = C[1,:]/(lambda_1*v1[1]);
C2 = C - lambda_1*v1*x';
e_C2 = eigvals(C2); e_C2 = sort(e_C2);

## As we know, the first row of C2 is zero
## So we define a smaller matrix C3 and show its eigenvalues
C3 = C2[2:n, 2:n];
e_C3 = eigvals(C3); e_C3 = sort(e_C3);
println("Eigenvalues of C:", "\t", e_C)
println("The smallest eigenvalue lambda_1 is","\t", sigma)
println("Eigenvalues of C2:", "\t", e_C2)
println("Eigenvalues of C3:", "\t", e_C3)
```

```
Eigenvalues of C:   [-0.401399, 0.244015, 0.993442, 1.39261]
The smallest eigenvalue lambda_1 is -0.40139877365078946
Eigenvalues of C2:  [0.0, 0.244015, 0.993442, 1.39261]
Eigenvalues of C3:  [0.244015, 0.993442, 1.39261]
```

[5] Subspace iteration

Can we find several eigenvalues in the same time? Yes!

Algorithm:

1. Construct a matrix $Q_0 \in \mathbb{R}^{n\times p}$ of orthonormal column vectors.
2. for $k = 1, 2, \ldots m$ (1) $Z_k = AQ_{k-1}$ (2) $Q_kR_k = Z_k$ (QR-decomposition)
3. The p largest (absolute) eigenvalues are $\|Q_m(:,1)\|_2, \cdots, \|Q_m(:,p)\|_2$. (Where $Q_m(:,l)$ is the l

column vector of Q_m.)

Note that when $p = 1$ this is just the power method.

```julia
# Construct a symmetric matrix
n=1000;
A = rand(n, n).-0.5;
A = A+A';
```

Try to find 4 largest (absolute) eigenvalues

```julia
# Constructing a set of orthonormal vectors
u2 = zeros(n,4);
u = rand(n,1); u = u/norm(u,2); u2[:,1] = u;
v = rand(n,1); v = v/norm(u,2);
v = v - (v'*u).*u; u2[:,2] = v;
w = rand(n,1); w = w/norm(u,2);
w = w - (w'*u).*u; w = w - (w'*v).*v;
u2[:,3] = w;
x = rand(n,1); x = x/norm(u,2);
x = x - (x'*u).*u; x = x - (x'*v).*v; x = x - (x'*w).*w;
u2[:,4] = x;

# Initialize the eigenvalue
mu = ones(1,4); mu2 = zeros(1,4);

# Define epsilon
eps = 1.0e-10; ii=0;

# The loop
while norm(mu2-mu)>eps
    mu2[:] = mu[:];
    v2 = A*u2;
    u = v2[:,1]; mu[1,1] = norm(u,2); u = u/mu[1,1];
    v = v2[:,2]; mu[1,2] = norm(v,2); v = v/mu[1,2];
    w = v2[:,3]; mu[1,3] = norm(w,2); w = w/mu[1,3];
    x = v2[:,4]; mu[1,4] = norm(x,2); x = x/mu[1,4];
    v = v - (v'*u).*u;
    w = w - (w'*u).*u; w = w - (w'*v).*v;
    x = x - (x'*u).*u; x = x - (x'*v).*v; x = x - (x'*w).*w;
    u2[:,1] = u; u2[:,2] = v; u2[:,3] = w; u2[:,4] = x;
    ii=ii+1;
end
println("The 1st eigenvalue is  ", mu[1,1])
println("The 2nd eigenvalue is  ", mu[1,2])
println("The 3rd eigenvalue is  ", mu[1,3])
println("The 4th eigenvalue is  ", mu[1,4])
println("# of iterations= ", ii)
```

```
The 1st eigenvalue is  25.625496745162256
The 2nd eigenvalue is  25.55390053598934
The 3rd eigenvalue is  25.492736253154703
The 4th eigenvalue is  25.32194814725078
# of iterations= 3813
```

```julia
eigvals(A)
```

```
1000-element Array{Float64,1}:
 -25.55390055071426
 -25.49273623846501
 -25.07297226022513
 -25.031529206139744
 -24.751845169690455
 -24.667147082617593
 -24.640702160694268
 -24.410154441981888
 -24.346425364819023
 -24.168058539544013
 -24.056576493620646
 -24.0384983111136
 -23.812732259714835
   ⋮
  23.99097703142358
  24.109124650709415
  24.15419482437092
  24.400006176865723
  24.50990445656496
  24.569429013037578
  24.82285026049988
  24.930559137774665
  25.140264352233473
  25.21460425766708
  25.321948147251813
  25.62549674516199
```

Try to find 4 smallest (absolute) eigenvalues. (By timing A^{-1}, similar to the inverse power method)

```julia
# Constructing a set of orthonormal vectors
u2 = zeros(n,4);
u = rand(n,1); u = u/norm(u,2); u2[:,1] = u;
v = rand(n,1); v = v/norm(u,2);
v = v - (v'*u).*u; u2[:,2] = v;
w = rand(n,1); w = w/norm(u,2);
w = w - (w'*u).*u; w = w - (w'*v).*v;
u2[:,3] = w;
x = rand(n,1); x = x/norm(u,2);
x = x - (x'*u).*u; x = x - (x'*v).*v; x = x - (x'*w).*w;
u2[:,4] = x;

# Initialize the eigenvalue
mu = ones(1,4); mu2 = zeros(1,4);

# Define epsilon
eps = 1.0e-10; ii=0;

# The loop
while norm(mu2-mu)>eps
    mu2[:] = mu[:];
    v2 = A\u2;
    u = v2[:,1]; mu[1,1] = norm(u,2); u = u/mu[1,1];
    v = v2[:,2]; mu[1,2] = norm(v,2); v = v/mu[1,2];
    w = v2[:,3]; mu[1,3] = norm(w,2); w = w/mu[1,3];
    x = v2[:,4]; mu[1,4] = norm(x,2); x = x/mu[1,4];
    v = v - (v'*u).*u;
    w = w - (w'*u).*u; w = w - (w'*v).*v;
    x = x - (x'*u).*u; x = x - (x'*v).*v; x = x - (x'*w).*w;
    u2[:,1] = u; u2[:,2] = v; u2[:,3] = w; u2[:,4] = x;
    ii=ii+1;
end
println("The 1st eigenvalue is  ", 1/mu[1,1])
println("The 2nd eigenvalue is  ", 1/mu[1,2])
println("The 3rd eigenvalue is  ", 1/mu[1,3])
println("The 4th eigenvalue is  ", 1/mu[1,4])
println("# of iterations= ", ii)
```

```
The 1st eigenvalue is  0.03311930379088369
The 2nd eigenvalue is  0.05074606701304302
The 3rd eigenvalue is  0.06706882622958633
The 4th eigenvalue is  0.10242229420618781
# of iterations= 169
```

Subspace iteration - Test for non-symmetric matrix

Julia

```
B = zeros(3,3);
B[:,:] = Diagonal([1;1;2]); B[1,3]=1;
C = rand(3,3); Q, R = qr(C); B = Q'*B*Q;
println(eigvals(B))
B
```

```
[2.0, 1.0, 1.0]
```

```
3×3 Array{Float64,2}:
  1.56537     -0.465441    -0.735656
 -0.523134     1.43067      0.680702
 -0.00304395   0.00250594   1.00396
```

Julia

```
n=3; m=2;
# Constructing a set of orthonormal vectors
u2 = zeros(n,m);
u = rand(n,1); u = u/norm(u,2); u2[:,1] = u;
v = rand(n,1); v = v/norm(u,2);
v = v - (v'*u).*u; u2[:,2] = v;

# Initialize the eigenvalue
mu = ones(1,m); mu2 = zeros(1,m);

# Define epsilon
eps = 1.0e-10; ii=0;

# The loop
while norm(mu2-mu)>eps
    mu2[:] = mu[:];
    v2 = B*u2;
    u = v2[:,1]; mu[1,1] = norm(u,2); u = u/mu[1,1];
    v = v2[:,2]; mu[1,2] = norm(v,2); v = v/mu[1,2];
    v = v - (v'*u).*u;
    u2[:,1] = u; u2[:,2] = v;
    ii=ii+1;
end
println("The 1st eigenvalue is  ", mu[1,1])
println("The 2nd eigenvalue is  ", mu[1,2])
println("# of iterations= ", ii)
```

```
The 1st eigenvalue is  2.000000000078225
The 2nd eigenvalue is  1.0000000000131641
# of iterations= 34
```

QR algorithm

Algorithm:

1. $Q_0 R_0 = qr(B)$ (QR-decomposition)
2. for $i = 1, 2, \ldots, k$
    1. $B_i = R_{i-1} Q_{i-1}$
    2. $Q_i R_i = B_i$
    3. $\mu_i = diag(B_i)$
3. components of μ_k are the desired eigenvalues

Julia
```
n=3;
Q, R = qr(B);

# Define epsilon
eps = 1.0e-10; ii=0;
mu = ones(n,1); mu2 = zeros(n,1);

# The loop
while norm(mu2-mu)>eps
    global A2
    mu2 = mu;
    A2 = R*Q;
    Q, R = qr(A2);
    mu = diag(A2);
    ii=ii+1;
end
println("The eigenvalues are  ", mu)
println("# of iterations= ", ii)
```

```
The eigenvalues are  [2.0, 1.0, 1.0]
# of iterations= 30
```

Julia
```
A2
```

```
3×3 Array{Float64,2}:
  2.0           0.0579215    -0.998321
 -8.21161e-10   1.0           8.19783e-10
 -3.69499e-12  -2.13865e-13   1.0
```

Eigenvalue problem of Laplace operator

$$u_{xx} = \lambda u, \qquad u(0) = u(1) = 0.$$

We can solve this numerically by the difference method.

$$u_{xx} \approx \frac{u(x-h) - 2u(x) + u(x+h)}{h^2}$$

$$A := \frac{1}{h^2}\begin{bmatrix} -2 & 1 & & & 0 \\ 1 & -2 & 1 & & \\ & 1 & -2 & \ddots & \\ & & \ddots & \ddots & 1 \\ 0 & & & 1 & -2 \end{bmatrix}$$

```julia
n = 1000; A = zeros(n-1,n-1);
for ii=1:n-1
    A[ii,ii] = -2;
end
for ii=1:n-2
    A[ii, ii+1]=1;
    A[ii+1, ii]=1;
end
h = 1/n; A = A/h^2;
```

Solve the 4 smallest (absolute) eigenvalues by subspace iteration.

```julia
n = n-1;
# Constructing a set of orthonormal vectors
u2 = zeros(n,4);
u = rand(n,1); u = u/norm(u,2); u2[:,1] = u;
v = rand(n,1); v = v/norm(u,2);
v = v - (v'*u).*u; u2[:,2] = v;
w = rand(n,1); w = w/norm(u,2);
w = w - (w'*u).*u; w = w - (w'*v).*v;
u2[:,3] = w;
x = rand(n,1); x = x/norm(u,2);
x = x - (x'*u).*u; x = x - (x'*v).*v; x = x - (x'*w).*w;
u2[:,4] = x;

# Initialize the eigenvalue
mu = ones(1,4); mu2 = zeros(1,4);

# Define epsilon
eps = 1.0e-13; ii=0;

# The loop
while norm(mu2-mu)>eps
    mu2[:] = mu[:];
    v2 = A\u2;
    u = v2[:,1]; mu[1,1] = norm(u,2); u = u/mu[1,1];
    v = v2[:,2]; mu[1,2] = norm(v,2); v = v/mu[1,2];
    w = v2[:,3]; mu[1,3] = norm(w,2); w = w/mu[1,3];
    x = v2[:,4]; mu[1,4] = norm(x,2); x = x/mu[1,4];
    v = v - (v'*u).*u;
    w = w - (w'*u).*u; w = w - (w'*v).*v;
    x = x - (x'*u).*u; x = x - (x'*v).*v; x = x - (x'*w).*w;
    u2[:,1] = u; u2[:,2] = v; u2[:,3] = w; u2[:,4] = x;
    ii=ii+1;
end
println("The 1st eigenvalue is  ", 1/mu[1,1])
println("The 2nd eigenvalue is  ", 1/mu[1,2])
println("The 3rd eigenvalue is  ", 1/mu[1,3])
println("The 4th eigenvalue is  ", 1/mu[1,4])
println("# of iterations= ", ii)
```

```
The 1st eigenvalue is  9.86959628366954
The 2nd eigenvalue is  39.47828772573998
The 3rd eigenvalue is  88.82578210039149
The 4th eigenvalue is  157.91159236855148
# of iterations= 28
```

Conclusion

In this two lectures we have explain quit a bit of the methods of finding eigenvalues. All the algorithm are explained using test examples but there is no proof or anything rigorous which shall be complete by reading textbook.