# Section: Power method - Lecture 01

Very often we want to find eigenvalues of a given matrix, not all of them, but only some of the largest/smallest or those close to zero. Power method (or power iteration) is a very simple algorithm to accomplish the task.

---

The followings are from wiki: power iteration

In mathematics, power iteration (also known as the power method) is an eigenvalue algorithm: given a diagonalizable matrix $A$, the algorithm will produce a number $\lambda$ , which is the greatest (in absolute value) eigenvalue of $A$, and a nonzero vector $v$, the corresponding eigenvector of $\lambda$, such that $Av = \lambda v$.

---

In this document we want to show you how to find the largest eigenvalue (in magnitude) by utilizing the power iteration. Furthermore, we will show you how the power iteration can be modified to find eigenvalues in more general settings.

## [1] Basic Power method

The idea of power method is the following: Suppose we have a matrix $A$ with corresponding eigenvalues $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \ldots$ and eigenvectors $v_1$, $v_2$, $\cdots$. Then, given a vector $u$, it can be expressed as a linear combination of the eigenvectors

$$u = \sum_{i=1}^{n} c_i v_i.$$

If we multiply both side by $A$ we get

$$Au = \sum_{i=1}^{n} c_i A v_i = \sum_{i=1}^{n} c_i \lambda_i v_i$$

and, if we multiply it $k$ times,

$$A^k u = \sum_{i=1}^{n} c_i \lambda_i^k v_i.$$

Since $|\lambda_1|$ is bigger than all the rest, we find, as $k$ get bigger,

$$\frac{1}{\lambda_1^k} A^k u = c_1 v_1 + \sum_{i=2}^{n} c_i \frac{\lambda_i^k}{\lambda_1^k} v_i \to c_1 v_1.$$

As a result, if we just randomly choose a vector and multiply it by $A$ many times, we should get closer and closer the the first eigenvector, and in this way we can find the first (largest) eigenvalue.

### Algorithm:

1. Start with a vector $u := u^{(0)}$
2. for $i = 1, 2, \ldots, k$ Compute
    1. $v^{(i)} = Au^{(i-1)}$
    2. $\mu^{(i)} = \|v^{(i)}\|_2$ or $\mu^{(i)} = v_2^{(i)}$
    3. $u^{(i)} = v^{(i)}/\mu^{(i)}$
3. $\mu^{(k)}$ is the largest eigenvalue, and $u^{(k)}$ is the corresponding eigenvector

Note: $v_2^{(i)}$ is the second component of vector $v^{(i)}$

Moreover, in fact, we can choose $\mu^{(i)} = l(v^{(i)})$, where $l$ is any linear function, for example, $\mu^{(i)} = v_2^{(i)}$ or $\mu^{(i)} = v_1^{(i)}$.

### Example 1

As a first example, we consider a 3-by-3 matrix A

```julia
A = [2 1 5; 5 7 9; 4 6 1]
```

```
3×3 Array{Int64,2}:
 2  1  5
 5  7  9
 4  6  1
```

All of its eigenvalues can be evaluated by the comment "eigvals".

Note: We use the julia package "LinearAlgebra" to find the eigenvalues.

```julia
using LinearAlgebra
e_A = eigvals(A);
println(e_A[1])
println(e_A[2])
println(e_A[3])
```

```
13.78378635197844
0.8328741741072435
-4.616660526085692
```

### Attemp 1

Given a random vector u, we compute A*u several times, here, 10 times.

```julia
# Start with a random vector u
u = rand(3);

# The loop.
for ii=1:10
    v = A*u;
    global mu = norm(v,2);
    u = v/mu;
end
println("After 10 iterations, \n")
println("the largest eigenvalue is approximately ", mu)
```

```
After 10 iterations,

the largest eigenvalue is approximately 13.783772000377752
```

### Attemp 2

The second attempt, we define an epsilon $\varepsilon$ to test if the process converges or not.

Given $\varepsilon$, if $|\mu_i - \mu_{i-1}| < \varepsilon$, stop the loop.

```julia
# Start with a random vector u
u = rand(3);
mu0 = 0;
mu = 1;

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

# The loop
while abs.(mu-mu0)>eps
    mu0 = mu;
    v = A*u;

    # Take the second component of v
    mu = v[2];

    u = v/mu;
    ii=ii+1;
end
println("Largest eigenvalue is ", mu)
println("# of iterations= ", ii)
```

```
Largest eigenvalue is 13.783786351956529
# of iterations= 23
```

# [2] Inverse power method

Similarly, if we apply the power method to $A^{-1}$ , we should obtain the smallest (absolute)eigenvalue.

Of course, we need to assume that $A$ is non-singular, otherwise the smallest (absolute)eigenvalue should be $0$.

### Algorithm:

1. Start with a vector $u := u^{(0)}$ with $\|u\|_2 = 1$
2. for i=1,2,...,k Compute
    1. $v^{(i)} = A^{-1}u^{(i-1)}$
    2. $\mu^{(i)} = v_2^{(i)}$
    3. $u^{(i)} = v^{(i)}/\mu^{(i)}$
3. $\mu^{(k)}$ is the smallest eigenvalue, and $u^{(k)}$ is the corresponding eigenvector

### Remark:

We compute $A^{-1}u$ by solving the linear system $Ax = u$.

### Example

To find the smallest eigenvalue we compute $A^{-1}u$ several times until the process converges.

```julia
# Start with a normalized vector
u = rand(3); u=u/norm(u,2);

mu0 = 0;
mu = 1;
eps = 1.0e-10;
ii=0;
while abs.(mu-mu0)>eps
    mu0 = mu;

    # Compute A^{-1}u by A\u
    v = A\u;

    mu = v[2];
    u = v/mu;
    ii=ii+1;
end
println("Smallest eigenvalue is ", 1/mu)
println("# of iterations= ", ii)
```

```
Smallest eigenvalue is 0.8328741741048754
# of iterations= 16
```

## [3.1] Shift-inverse power method

If we apply power method to $(A-\sigma I)^{-1}$, we should get the eigenvalue that is closest to $\sigma$.

### Algorithm:

Given $\sigma$ 1. Start with a vector $u := u^{(0)}$ with $\|u\|_2 = 1$ 2. for i=1,2,...,k Compute 1. $v^{(i)} = (A-\sigma I)^{-1}u^{(i-1)}$ 2. $\mu^{(i)} = \|v^{(i)}\|_2$ or $\mu^{(i)} = v_2^{(i)}$ 3. $u^{(i)} = v^{(i)}/\mu^{(i)}$ 3. $\mu^{(k)}$ is the eigenvalue closest to $\sigma$, and $u^{(k)}$ is the corresponding eigenvector

### Example

To find the eigenvalue that is closest to $\sigma$ we compute $(A-\sigma I)^{-1}u$ several times until the process converges.

Julia
```
u = rand(3); u=u/norm(u,2);
sigma = -10;
mu0 = 0;
mu = 1;
eps = 1.0e-10;
ii=0;
while abs.(mu-mu0)>eps
    mu0 = mu;
    v = (A-sigma*UniformScaling(1))\u;
    mu = v[2];
    u = v/mu;
    ii = ii+1;
end
println("The eigenvalue that is closest to ", sigma, " is ", sigma + 1/mu)
println("# of iterations= ", ii)
```

```
The eigenvalue that is closest to -10 is -4.616660523472157
# of iterations= 34
```

## [3.2] Shift-inverse power method - Algorithm 2

In the previous example we found the eigenvalue that is closest to $\sigma$. In fact, one can show that the coefficient of the convergence in algorithm 1 is about $\left|\frac{\lambda_1-\sigma}{\lambda_2-\sigma}\right|$, where $\lambda_1$ is the one that is closest to $\sigma$ and $\lambda_2$ is the second closest one. So in principle, if we can choose $\sigma$ such that it is very close to $\lambda_1$, the iteration should converge to it faster.

But of course we don't know in advance what is $\lambda_1$, so what we can do is to adjust $\sigma$ along with the iteration, this is the idea of the following algorithm.

### Algorithm:

Given $\sigma$ 1. Start with a vector $u := u^{(0)}$ with $\|u\|_2 = 1$. $\sigma := \sigma^{(0)}$ 2. for i=1,2,...,k Compute 1. $v^{(i)} = (A - \sigma^{(i-1)}I)^{-1}u^{(i-1)}$ 2. $\mu^{(i)} = v_2^{(i)}$ 3. $u^{(i)} = v^{(i)}/\mu^{(i)}$ 4. $\sigma^{(i)} = \sigma^{(i-1)} + 1/\mu^{(i)}$ 3. $\mu^{(k)}$ is the eigenvalue closest to $\sigma$, and $u^{(k)}$ is the corresponding eigenvector

### Example

We take variant shifts $\sigma^{(i)} = \sigma^{(i-1)} + 1/\mu^{(i)}$ to get a faster convergence.

```julia
u = rand(3); u=u/norm(u,2);
sigma = 0;
sigma0 = 1;
mu = 1;
eps = 1.0e-10;
ii=0;
while abs.(sigma-sigma0)>eps
    sigma0 = sigma;
    v = (A-sigma*UniformScaling(1))\u;
    mu = v[2];
    u = v/mu;
    sigma = sigma+ 1/mu;
    ii = ii+1;
end
println("The eigenvalue is ", sigma + 1/mu)
println("# of iterations= ", ii)
```

```
The eigenvalue is 0.8328741741049885
# of iterations= 7
```

## [3.3] Shift-inverse power method - Algorithm 3

The eigenvalue can be evaluate using Rayleigh quotient. So in algorithm 3 we use Rayleigh quotient.

### Algorithm:

Given $\sigma$ 1. Start with a vector $u := u^{(0)}$ with $\|u\|_2 = 1$. $\sigma := \sigma^{(0)}$ 2. for i=1,2,...,k Compute 1. $v^{(i)} = (A - \sigma^{(i-1)}I)^{-1}u^{(i-1)}$ 2. $\mu^{(i)} = \|v^{(i)}\|_2$
3. $u^{(i)} = v^{(i)}/\mu^{(i)}$
4. $\sigma^{(i)} = (v^{(i)})^\top A v^{(i)}$ 3. $\mu^{(k)}$ is the eigenvalue closest to $\sigma$, and $u^{(k)}$ is the corresponding eigenvector

### Example

Here we take variant shifts $\sigma^{(i)} = (v^{(i)})^\top A v^{(i)}$

```julia
u = rand(3); u=u/norm(u,2);
sigma = u'*A*u;
sigma0 = 1;
mu = 1;
eps = 1.0e-10;
ii=0;
while abs.(sigma-sigma0)>eps
    sigma0 = sigma;
    v = (A-sigma*UniformScaling(1))\u;
    v = v/norm(v,2);
    sigma = v'*A*v;
    ii = ii+1;
end
println("The eigenvalue is ", sigma)
println("# of iterations= ", ii)
```

```
The eigenvalue is 13.783786351974396
# of iterations= 8
```

## Example

Let's try a bigger symmetric matrix $B$, with size $n = 10$. We'll compute the eigenvalue closet to -1 ( $\sigma := -1$) with all 3 algorithms.

```julia
n = 10;
# Construct a symmectic matrix B
B = rand(n,n) .- 0.5; B = B+B';

e_B = eigvals(B);
for ii=1:n
    println(e_B[ii])
end
```

```
-2.3131285864106017
-1.4742155969831732
-1.1337919820848548
-0.4835431741064402
-0.287076080806612
-0.05695603462221564
0.6497956881700901
1.0053010938359734
1.2141422043828263
2.364659609645527
```

## Example

Shift-inverse power method - Algorithm 1

```julia
u = rand(n); u=u/norm(u,2);
sigma = -1;
mu0 = 0; mu = 1;
eps = 1.0e-10; ii=0;
while abs.(mu-mu0)>eps
    mu0 = mu;
    v = (B-sigma*UniformScaling(1))\u;
    mu = v[2];
    u = v/mu;
    ii = ii+1;
end
println("The eigenvalue is ", sigma + 1/mu)
println("# of iterations= ", ii)
```

```
The eigenvalue is -1.1337919820850129
# of iterations= 23
```

## Example

Shift-inverse power method - Algorithm 2

```julia
u = rand(n); u=u/norm(u,2);
sigma = -1; sigma0 = 1;
mu = 1;
eps = 1.0e-10; ii=0;
while abs.(sigma-sigma0)>eps
    sigma0 = sigma;
    v = (B-sigma*UniformScaling(1))\u;
    mu = v[2];
    u = v/mu;
    sigma = sigma + 1/mu;
    ii = ii+1;
end
println("The eigenvalue is ", sigma + 1/mu)
println("# of iterations= ", ii)
```

```
The eigenvalue is -1.1337919820848545
# of iterations= 13
```

## Example

Shift-inverse power method - Algorithm 3

```julia
u = rand(n); u=u/norm(u,2);
sigma = -1; sigma0 = 1;
mu = 1;
eps = 1.0e-10; ii=0;
while abs.(sigma-sigma0)>eps
    sigma0 = sigma;
    global v = (B-sigma*UniformScaling(1))\u;
    v = v/norm(v,2);
    sigma = v'*B*v;
    ii = ii+1;
end
println("The eigenvalue is ", sigma)
println("# of iterations= ", ii)
```

```
The eigenvalue is -0.4835431741064395
# of iterations= 6
```

Do you see the differences in the numbers of iterations? Also, we see that the results are different. All of the algorithms converges to a eigenvalue, but some of them might converges to a wrong one.

# Conclusion

As a brief summary, we have shown how to find the first (or largest in magnitude) eigenvalue,