Power Iteration and its Variants

In [2]:

```
#keep
import numpy as np
import numpy.linalg as la
```

Let's prepare a matrix with some random or deliberately chosen eigenvalues:

In [6]:

```
#keep
n = 6

if 1:
    np.random.seed(70)
    eigvecs = np.random.randn(n, n)
    eigvals = np.sort(np.random.randn(n))
    # Uncomment for near-duplicate largest-magnitude eigenvalue
    # eigvals[1] = eigvals[0] + 1e-3

    A = eigvecs.dot(np.diag(eigvals)).dot(la.inv(eigvecs))
    print(eigvals)

else:
    # Complex eigenvalues
    np.random.seed(40)
    A = np.random.randn(n, n)
    print(la.eig(A)[0])
```

```
[-2.667651   -0.95797093 -0.33019549 -0.29151942 -0.18635343 -0.1441
8093]
```

Let's also pick an initial vector:

In [4]:

```
x0 = np.random.randn(n)
x0
```

Out[4]:

```
array([ 2.26930477,  0.66356156,  0.8991019 , -0.36580094,  0.462690
04,
        0.079874  ])
```

Power iteration

In [5]:

```
#keep
x = x0
```

Now implement plain power iteration.

Run the below cell in-place (Ctrl-Enter) many times.

In [6]:

```
x = np.dot(A, x)
x
```

Out[6]:

```
array([-6.40496816,  7.1851682 ,  1.97149585,  4.77787616,  3.090991
27,
        4.33054803])
```

- What's the problem with this method?
- Does anything useful come of this?
- How do we fix it?

Normalized power iteration

Back to the beginning: Reset to the initial vector.

In [7]:

```
#keep
x = x0/la.norm(x0)
```

Implement normalized power iteration.

Run this cell in-place (Ctrl-Enter) many times.

In [8]:

```
x = np.dot(A, x)
nrm = la.norm(x)
x = x/nrm

print(nrm)
print(x)
```

```
4.67639723405
[-0.52706768  0.59127069  0.16223527  0.39317355  0.25435905  0.3563
6272]
```

- What do you observe about the norm?
- What about the sign?
- What is the vector x now?

Extensions:

- Now try the matrix variants above.
- Suggest a better way of estimating the eigenvalue. Hint (https://en.wikipedia.org/wiki/Rayleigh_quotient)

What if we want the smallest eigenvalue (by magnitude)?

Once again, reset to the beginning.

In [9]:

```
#keep
x = x0/la.norm(x0)
```

Run the cell below in-place many times.

In [10]:

```
x = la.solve(A, x)
nrm = la.norm(x)
x = x/nrm

print(1/nrm)
print(x)
```

```
0.0464410512401
[ 0.07260214 -0.82392938  0.17918176 -0.52684691  0.07819739  0.0089
8414]
```

- What's the computational cost per iteration?
- Can we make this method search for a specific eigenvalue?
- What is this method (https://en.wikipedia.org/wiki/Inverse_iteration) called?

Can we feed an estimate of the current approximate eigenvalue back into the calculation? (Hint: Rayleigh quotient)

Reset once more.

In [11]:

```
#keep
x = x0/la.norm(x0)
```

Run this cell in-place (Ctrl-Enter) many times.

In [12]:

```
sigma = np.dot(x, np.dot(A, x))/np.dot(x, x)
x = la.solve(A-sigma*np.eye(n), x)
x = x/la.norm(x)

print(sigma)
print(x)
```

```
-1.17969302281
[-0.06324034 -0.54447197  0.4628917  -0.37622458  0.41482856  0.4143
1213]
```

- What's this method (https://en.wikipedia.org/wiki/Rayleigh_quotient_iteration) called?
- What's a reasonable stopping criterion?
- Computational downside of this iteration?