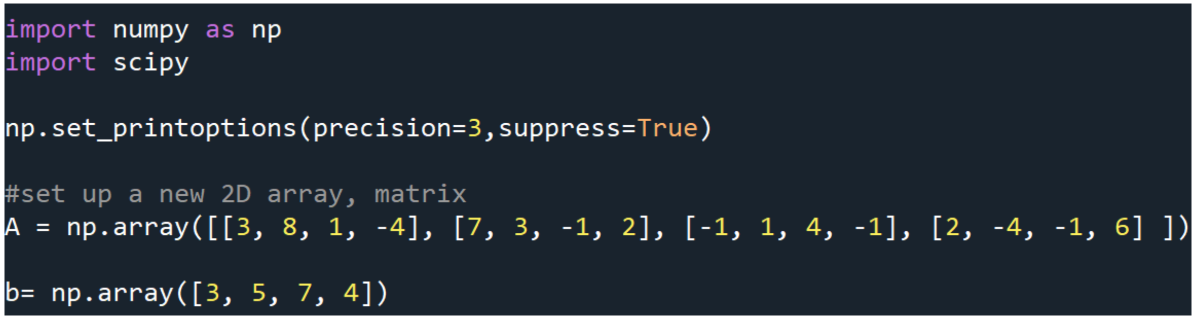
Linear algebra in Python

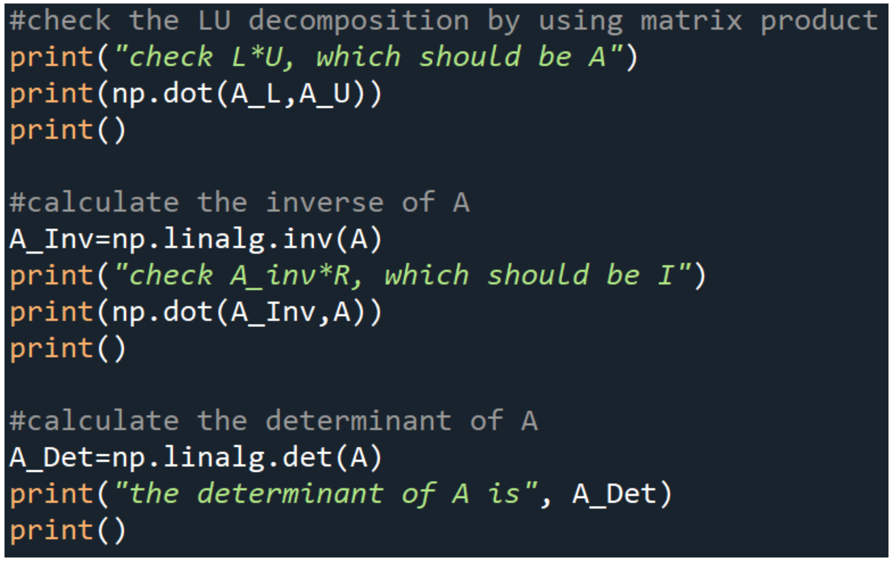
- Most of the functions about matrix operations are included in numpy and scipy. You can find lots of tutorials online, e.g., the following links.
- https://github.com/lijin-THU/notes-python/tree/master/03-numpy
- https://github.com/lijin-THU/notes-python/tree/master/04-scipy

- Basic operations like matrix multiplication, LU decomposition, determinant, inverse, eigenvalue and eigenvector, solving linear equation, etc.

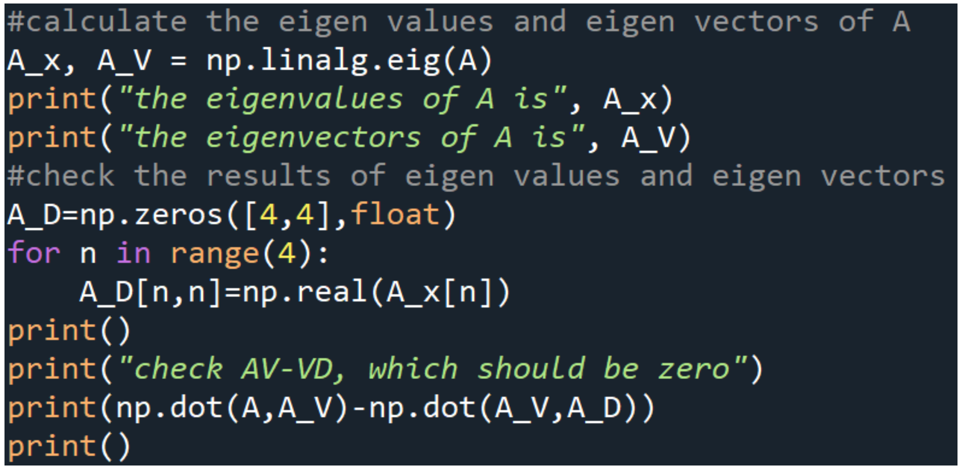
Example of codes



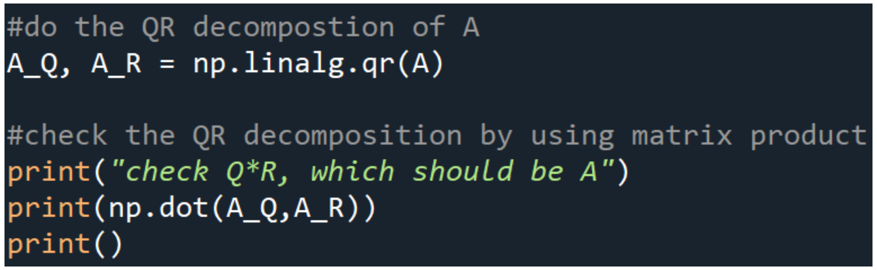
```python
import numpy as np
import scipy

np.set_printoptions(precision=3,suppress=True)

#set up a new 2D array, matrix
A = np.array([[3, 8, 1, -4], [7, 3, -1, 2], [-1, 1, 4, -1], [2, -4, -1, 6] ])

b= np.array([3, 5, 7, 4])
```



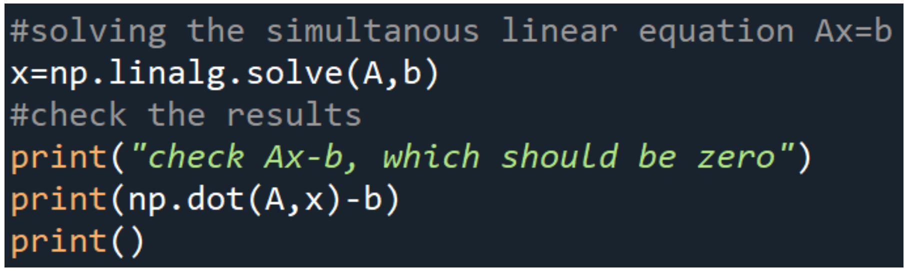
```python
#check the LU decomposition by using matrix product
print("check L*U, which should be A")
print(np.dot(A_L,A_U))
print()

#calculate the inverse of A
A_Inv=np.linalg.inv(A)
print("check A_inv*R, which should be I")
print(np.dot(A_Inv,A))
print()

#calculate the determinant of A
A_Det=np.linalg.det(A)
print("the determinant of A is", A_Det)
print()
```



```python
#calculate the eigen values and eigen vectors of A
A_x, A_V = np.linalg.eig(A)
print("the eigenvalues of A is", A_x)
print("the eigenvectors of A is", A_V)
#check the results of eigen values and eigen vectors
A_D=np.zeros([4,4],float)
for n in range(4):
    A_D[n,n]=np.real(A_x[n])
print()
print("check AV-VD, which should be zero")
print(np.dot(A,A_V)-np.dot(A_V,A_D))
print()
```

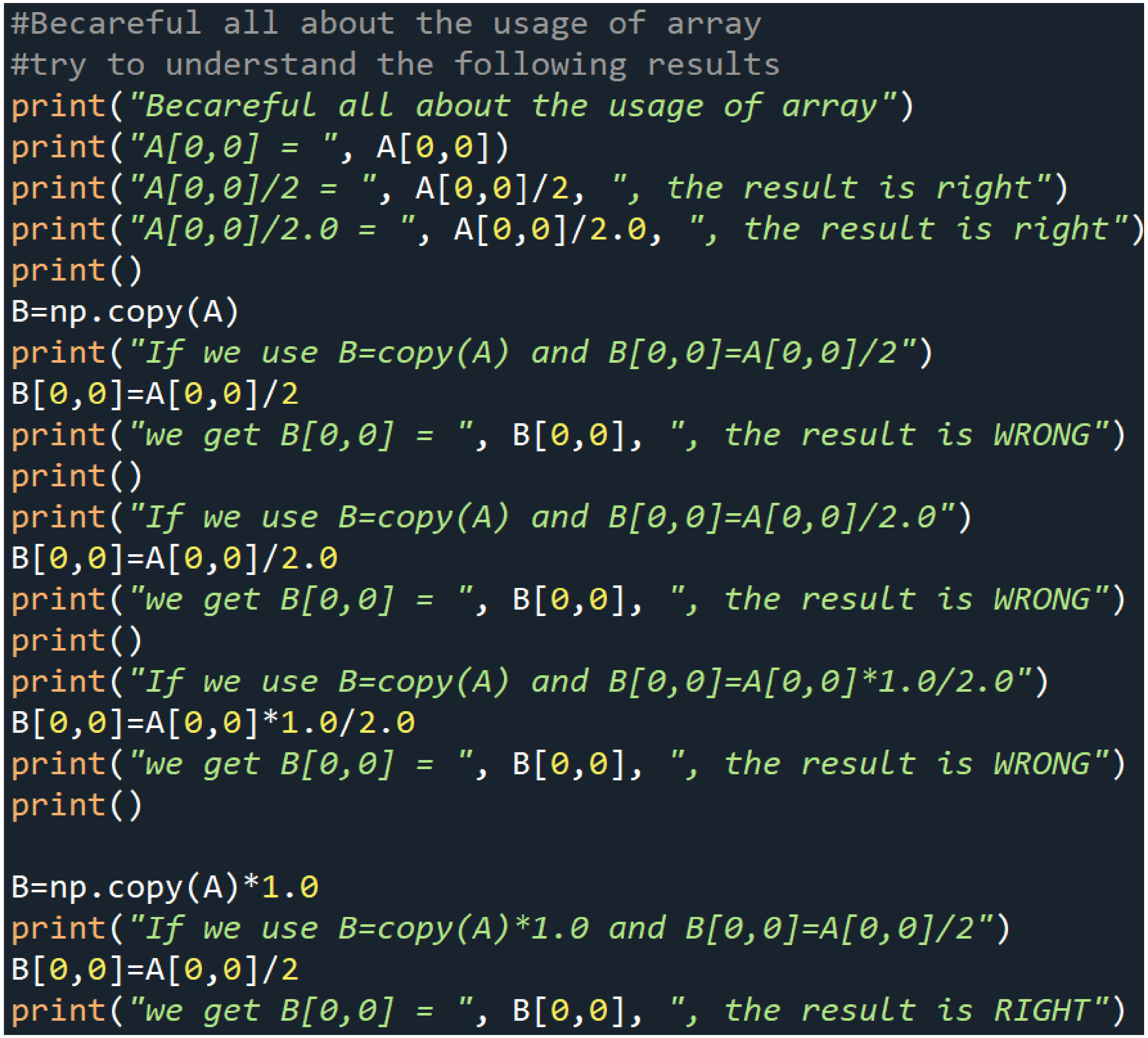


```python
#do the QR decompostion of A
A_Q, A_R = np.linalg.qr(A)

#check the QR decomposition by using matrix product
print("check Q*R, which should be A")
print(np.dot(A_Q,A_R))
print()
```



```python
#solving the simultanous linear equation Ax=b
x=np.linalg.solve(A,b)
#check the results
print("check Ax-b, which should be zero")
print(np.dot(A,x)-b)
print()
```

Be careful about the variable type



```
#Becareful all about the usage of array
#try to understand the following results
print("Becareful all about the usage of array")
print("A[0,0] = ", A[0,0])
print("A[0,0]/2 = ", A[0,0]/2, ", the result is right")
print("A[0,0]/2.0 = ", A[0,0]/2.0, ", the result is right")
print()
B=np.copy(A)
print("If we use B=copy(A) and B[0,0]=A[0,0]/2")
B[0,0]=A[0,0]/2
print("we get B[0,0] = ", B[0,0], ", the result is WRONG")
print()
print("If we use B=copy(A) and B[0,0]=A[0,0]/2.0")
B[0,0]=A[0,0]/2.0
print("we get B[0,0] = ", B[0,0], ", the result is WRONG")
print()
print("If we use B=copy(A) and B[0,0]=A[0,0]*1.0/2.0")
B[0,0]=A[0,0]*1.0/2.0
print("we get B[0,0] = ", B[0,0], ", the result is WRONG")
print()

B=np.copy(A)*1.0
print("If we use B=copy(A)*1.0 and B[0,0]=A[0,0]/2")
B[0,0]=A[0,0]/2
print("we get B[0,0] = ", B[0,0], ", the result is RIGHT")
```

Eigenvalues and eigenvectors

- One common problem is the calculation of the eigenvalues and/or eigenvectors of a matrix.

- For a matrix $\boldsymbol{A}$, an eigenvector $\boldsymbol{v}$ is a vector satisfying $\boldsymbol{A}\boldsymbol{v} = \lambda\boldsymbol{v}$, where λ is the corresponding eigenvalues. For a $N \times N$ matrix there are N eigenvectors with eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_N$. The eigenvectors have the property that they are orthogonal to one another $\boldsymbol{v}_i \cdot \boldsymbol{v}_j = 0$ if $i \neq j$, and we will assume they are normalized to have unite length $\boldsymbol{v}_i \cdot \boldsymbol{v}_i = 1$.

- If we wish, we can consider the eigenvectors to be the columns of a single $N \times N$ matrix $\boldsymbol{V}$ and combine all the equations $\boldsymbol{A}\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i$ into a single matrix $\boldsymbol{A}\boldsymbol{V} = \boldsymbol{V}\boldsymbol{D}$, where $\boldsymbol{D}$ is the diagonal matrix with the eigenvalues λ_i as its diagonal entries. Notice that the matrix $\boldsymbol{V}$ is ***orthogonal***, meaning that its transpose $\boldsymbol{V}^T$is equal to its inverse $\boldsymbol{V}^{-1}$, so that $\boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{I}$.

How to get the eigenvalues and eigenvectors?

- Based o the definition, for a matrix $\boldsymbol{A}$, an eigenvalue λ and an eigenvector $\boldsymbol{v}$ obey the equation $\boldsymbol{Av} = \lambda\boldsymbol{v}$, and it can be write as $(\boldsymbol{A} - \lambda\boldsymbol{I})\boldsymbol{v} = \boldsymbol{0}$, which is a set of simultaneous linear equations with the form $\boldsymbol{Bv} = \boldsymbol{0}$ (we take $\boldsymbol{B} = \boldsymbol{A} - \lambda\boldsymbol{I}$). It seems that we can use Gaussian elimination to solve it. Can we? why?
- The answer is NO, because there are unknown variables in the matrix $\boldsymbol{B}$ but not only in the vector $\boldsymbol{v}$. And there is always a trivial solution $\boldsymbol{v} = (0,0,\cdots,0)$. So if there is only solution of the equation, then $\boldsymbol{v} = (0,0,\cdots,0)$. Thus, the actual question should be what values of λ can make the equation $(\boldsymbol{A} - \lambda\boldsymbol{I})\boldsymbol{v} = \boldsymbol{0}$ have non-zero solution $\boldsymbol{v}$.
- The answer is that if and only if the determinant of $(\boldsymbol{A} - \lambda\boldsymbol{I})$ is zero, then there will non-zero solution $\boldsymbol{v}$ for $(\boldsymbol{A} - \lambda\boldsymbol{I})\boldsymbol{v} = \boldsymbol{0}$.
- Finally, we can get the eigenvalues by solving the following nonlinear equation $\det(\boldsymbol{A} - \lambda\boldsymbol{I}) = \boldsymbol{0}$

An example

- For a diagonal matrix, the answer is very simple. The eigenvalues are just the diagonal elements.
- Consider a matrix $\boldsymbol{A} = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$
- Taking the determinant to find the characteristic polynomial of $\boldsymbol{A}$

$$|\boldsymbol{A} - \lambda \boldsymbol{I}| = \left| \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} - \lambda \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right| = \begin{vmatrix} 2-\lambda & 1 \\ 1 & 2-\lambda \end{vmatrix} = 3 - 4\lambda + \lambda^2 = 0$$

- Then we get the eigenvalues are $\lambda = 1$ and $\lambda = 3$
- Next, we will find the eigenvectors. For $\lambda = 1$, $(\boldsymbol{A} - \lambda \boldsymbol{I})\boldsymbol{v} = \boldsymbol{0}$ becomes $\begin{pmatrix} 2-1 & 1 \\ 1 & 2-1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, and we can easily get

$$\boldsymbol{v}_{\lambda=1} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

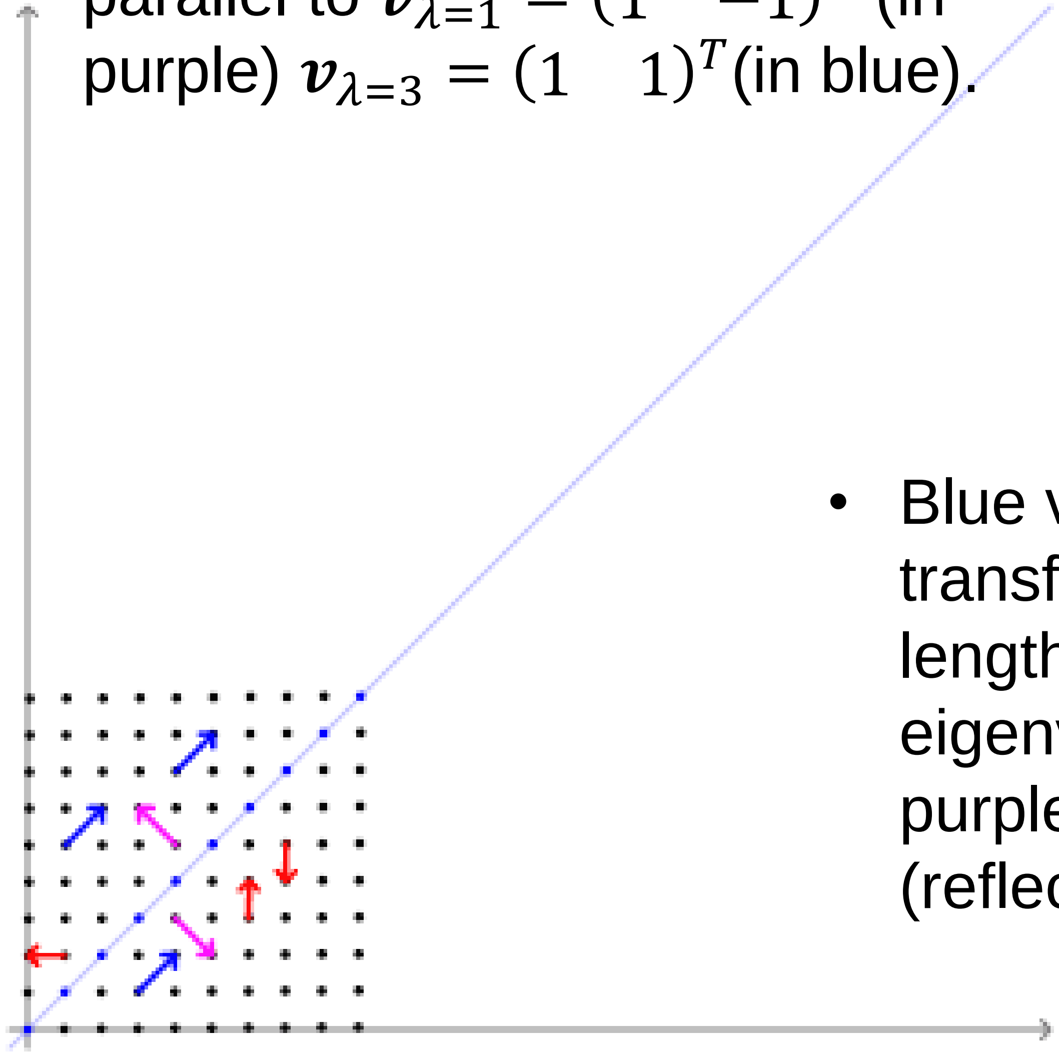
- Similarly, we can get the eigenvector corresponding to $\lambda = 3$

$$\boldsymbol{v}_{\lambda=3} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

The meaning of eigenvalues and eigenvectors

- The transformation matrix $\boldsymbol{A}$ preserves the direction of vectors parallel to $\boldsymbol{v}_{\lambda=1} = (1 \quad -1)^T$ (in purple) $\boldsymbol{v}_{\lambda=3} = (1 \quad 1)^T$(in blue).

- Vectors in red are not parallel to either eigenvector, so their directions are changed by the transformation.



- Blue vectors after the transformation are three times the length of the original (their eigenvalue is 3), while lengths of purple vectors are unchanged (reflecting an eigenvalue of 1).

Solve eigenequation by QR decomposition

- The calculation of eigenvalues and eigenvectors based on the determinant involves solving nonlinear equations, and it is hard to implement in codes.
- There are many other algorithms to do the calculations. One of the most used method is QR decomposition. QR decomposition is a variant on the same ideas as LU decomposition, but now the matrix is written as the product $\boldsymbol{QR}$ of an orthogonal matrix $\boldsymbol{Q}$ and an upper-triangular matrix $\boldsymbol{R}$. Any square matrix can be written in this form.
- Suppose we have a real, square matrix $\boldsymbol{A}$ and let us break it down into its decomposition as $\boldsymbol{A} = \boldsymbol{Q}_1\boldsymbol{R}_1$, then we get

$$\boldsymbol{Q}_1^T\boldsymbol{A} = \boldsymbol{Q}_1^T\boldsymbol{Q}_1\boldsymbol{R}_1 = \boldsymbol{R}_1$$

, where we have used the fact that $\boldsymbol{Q}_1$ is orthogonal ($\boldsymbol{Q}_1^T\boldsymbol{Q}_1 = \boldsymbol{I}$)

- Now we define a new matrix $\boldsymbol{A}_1 = \boldsymbol{R}_1\boldsymbol{Q}_1$, which is just the product of the same two matrices, but in the reverse order. Due to$\boldsymbol{R}_1 = \boldsymbol{Q}_1^T\boldsymbol{A}$, we have $\boldsymbol{A}_1 = \boldsymbol{Q}_1^T\boldsymbol{A}\boldsymbol{Q}_1$

Solve eigenequation by QR decomposition

- Using a orthogonal matrix $\boldsymbol{Q}_1$, we get a new matrix $\boldsymbol{A}_1 = \boldsymbol{Q}_1^T \boldsymbol{A} \boldsymbol{Q}_1$
- Now we do the QR decomposition for matrix $\boldsymbol{A}_1 = \boldsymbol{Q}_2 \boldsymbol{R}_2$, and get another new matrix $\boldsymbol{A}_2 = \boldsymbol{R}_2 \boldsymbol{Q}_2 = \boldsymbol{Q}_2^T \boldsymbol{A}_1 \boldsymbol{Q}_2 = \boldsymbol{Q}_2^T \boldsymbol{Q}_1^T \boldsymbol{A} \boldsymbol{Q}_1 \boldsymbol{Q}_2$
- We can do it again and again, repeatedly forming the QR decomposition of the current matrix, then multiplying $\boldsymbol{R}$ and $\boldsymbol{Q}$ in the opposite order to get a new matrix. If we do a total of steps, we generate the sequence of matrices

$$\begin{aligned}
\boldsymbol{A}_1 &= \boldsymbol{Q}_1^T \boldsymbol{A} \boldsymbol{Q}_1 \\
\boldsymbol{A}_2 &= \boldsymbol{Q}_2^T \boldsymbol{Q}_1^T \boldsymbol{A} \boldsymbol{Q}_1 \boldsymbol{Q}_2 \\
\boldsymbol{A}_3 &= \boldsymbol{Q}_3^T \boldsymbol{Q}_2^T \boldsymbol{Q}_1^T \boldsymbol{A} \boldsymbol{Q}_1 \boldsymbol{Q}_2 \boldsymbol{Q}_3 \\
&\vdots \\
\boldsymbol{A}_k &= \left(\boldsymbol{Q}_k^T \cdots \boldsymbol{Q}_1^T\right) \boldsymbol{A} (\boldsymbol{Q}_1 \cdots \boldsymbol{Q}_k)
\end{aligned}$$

- It can be proven that if you continue this process long enough, the matrix $\boldsymbol{A}_k$ will eventually become diagonal. The off-diagonal entries of the matrix get smaller and smaller the more iterations of the process you do.

Eigenvalue decomposition (EVD) of a matrix

- Suppose we continue until all the off-diagonal elements become so small that, to whatever accuracy we choose, the matrix $\boldsymbol{A}_k$ approximates a diagonal matrix $\boldsymbol{D}$. Let us define

$$\boldsymbol{V} = \boldsymbol{Q}_1\boldsymbol{Q}_2\boldsymbol{Q}_3\cdots\boldsymbol{Q}_k = \prod_{i=1}^{k}\boldsymbol{Q}_k$$

, which is an orthogonal matrix since a product of orthogonal matrices is always another orthogonal matrix. Then we have

$$\boldsymbol{D} = \boldsymbol{A}_k = \left(\boldsymbol{Q}_k^T\cdots\boldsymbol{Q}_1^T\right)\boldsymbol{A}(\boldsymbol{Q}_1\cdots\boldsymbol{Q}_k) = \boldsymbol{V}^T\boldsymbol{A}\boldsymbol{V}$$

- And multiplying on the left by $\boldsymbol{V}$, we get

$$\boldsymbol{A}\boldsymbol{V} = \boldsymbol{V}\boldsymbol{D}$$

- This is precisely the definition of the eigenvalues and eigenvectors of $\boldsymbol{A}$ in the matrix form. Thus, the columns of $\boldsymbol{V}$ are the eigenvectors of $\boldsymbol{A}$ and the diagonal elements of $\boldsymbol{D}$ are the corresponding eigenvalues (in the same order as the eigenvectors).

Detailed procedures of the complete algorithm

- For a symmetric $N \times N$ matrix $\boldsymbol{A}$, the algorithm is as follows:

1. Create an $N \times N$ matrix $\boldsymbol{V}$ to hold the eigenvalues and initially set it equal to the identity matrix $\boldsymbol{I}$. Also choose a target accuracy ϵ for the off-diagonal elements of the eigenvalue matrix.

2. Calculate the QR decomposition $\boldsymbol{A} = \boldsymbol{QR}$.

3. Update $\boldsymbol{A}$ to the new value $\boldsymbol{A} = \boldsymbol{RQ}$.

4. Multiply $\boldsymbol{V}$ on the right by $\boldsymbol{Q}$.

5. Check the off-diagonal elements of $\boldsymbol{A}$. If they are all less thanϵ, we are done. Otherwise go back to step 2.

- When the algorithm ends, the diagonal elements of $\boldsymbol{A}$ contain the eigenvalues and the columns of $\boldsymbol{V}$ contain the eigenvectors.
- The calculations for a general matrix are very subtle. More details in https://www.andreinc.net/2021/01/25/computing-eigenvalues-and-eigenvectors-using-qr-decomposition.

Gram-Schmidt algorithm for QR decomposition

- Consider a matrix $\boldsymbol{A} = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}$. We can think of the matrix as a set of N column vectors $\vec{\boldsymbol{a}}_1, \vec{\boldsymbol{a}}_2, \cdots, \vec{\boldsymbol{a}}_n$, thus $\boldsymbol{A} = (\vec{\boldsymbol{a}}_1, \vec{\boldsymbol{a}}_2, \cdots, \vec{\boldsymbol{a}}_n)$ with $\vec{\boldsymbol{a}}_i = (a_{1i}, a_{2i}, \cdots, a_{ni})^T$
- We can define two new sets of vectors $\vec{\boldsymbol{u}}_1, \vec{\boldsymbol{u}}_2, \cdots, \vec{\boldsymbol{u}}_n$ and $\vec{\boldsymbol{q}}_1, \vec{\boldsymbol{q}}_2, \cdots, \vec{\boldsymbol{q}}_n$ as follows:

$$\vec{\boldsymbol{u}}_1 = \vec{\boldsymbol{a}}_1, \qquad \vec{\boldsymbol{q}}_1 = \frac{\vec{\boldsymbol{u}}_1}{|\vec{\boldsymbol{u}}_1|}$$

$$\vec{\boldsymbol{u}}_2 = \vec{\boldsymbol{a}}_2 - (\vec{\boldsymbol{q}}_1 \cdot \vec{\boldsymbol{a}}_2)\vec{\boldsymbol{q}}_1, \qquad \vec{\boldsymbol{q}}_2 = \frac{\vec{\boldsymbol{u}}_2}{|\vec{\boldsymbol{u}}_2|}$$

$$\vec{\boldsymbol{u}}_3 = \vec{\boldsymbol{a}}_3 - (\vec{\boldsymbol{q}}_1 \cdot \vec{\boldsymbol{a}}_3)\vec{\boldsymbol{q}}_1 - (\vec{\boldsymbol{q}}_2 \cdot \vec{\boldsymbol{a}}_3)\vec{\boldsymbol{q}}_2, \qquad \vec{\boldsymbol{q}}_3 = \frac{\vec{\boldsymbol{u}}_3}{|\vec{\boldsymbol{u}}_3|}$$

- The general formulas for calculating $\vec{\boldsymbol{u}}_i$ and $\vec{\boldsymbol{q}}_i$ are

$$\vec{\boldsymbol{u}}_i = \vec{\boldsymbol{a}}_i - \sum_{j=1}^{i-1} (\vec{\boldsymbol{q}}_j \cdot \vec{\boldsymbol{a}}_i)\vec{\boldsymbol{q}}_j\,, \qquad \vec{\boldsymbol{q}}_i = \frac{\vec{\boldsymbol{u}}_i}{|\vec{\boldsymbol{u}}_i|}$$

Gram-Schmidt algorithm for QR decomposition

- We can demonstrate that, all the vectors $\vec{\boldsymbol{q}}_i$ are orthonormal, i.e., that they satisfy $\vec{\boldsymbol{q}}_i \cdot \vec{\boldsymbol{q}}_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$
- Now, rearranging the definitions of the vectors, we have

$$\begin{aligned}
\vec{\boldsymbol{a}}_1 &= |\vec{\boldsymbol{u}}_1|\vec{\boldsymbol{q}}_1, \\
\vec{\boldsymbol{a}}_2 &= |\vec{\boldsymbol{u}}_2|\vec{\boldsymbol{q}}_2 + (\vec{\boldsymbol{q}}_1 \cdot \vec{\boldsymbol{a}}_2)\vec{\boldsymbol{q}}_1, \\
\vec{\boldsymbol{a}}_3 &= |\vec{\boldsymbol{u}}_3|\vec{\boldsymbol{q}}_3 + (\vec{\boldsymbol{q}}_1 \cdot \vec{\boldsymbol{a}}_3)\vec{\boldsymbol{q}}_1 + (\vec{\boldsymbol{q}}_2 \cdot \vec{\boldsymbol{a}}_3)\vec{\boldsymbol{q}}_2,
\end{aligned}$$

- We can group the vectors together as the columns of a matrix and write all of these equations as a single matrix equation

$$\boldsymbol{A} = (\vec{\boldsymbol{a}}_1, \vec{\boldsymbol{a}}_2, \vec{\boldsymbol{a}}_3, \cdots) = (\vec{\boldsymbol{q}}_1, \vec{\boldsymbol{q}}_2, \vec{\boldsymbol{q}}_3 \cdots) \begin{pmatrix} |\vec{\boldsymbol{u}}_1| & (\vec{\boldsymbol{q}}_1 \cdot \vec{\boldsymbol{a}}_2) & (\vec{\boldsymbol{q}}_1 \cdot \vec{\boldsymbol{a}}_3) & \cdots \\ 0 & |\vec{\boldsymbol{u}}_2| & (\vec{\boldsymbol{q}}_2 \cdot \vec{\boldsymbol{a}}_3) & \cdots \\ 0 & 0 & |\vec{\boldsymbol{u}}_3| & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

$$= \boldsymbol{QR}$$

Gram-Schmidt algorithm for QR decomposition

- For $\boldsymbol{A} = (\vec{\boldsymbol{a}}_1, \vec{\boldsymbol{a}}_2, \cdots, \vec{\boldsymbol{a}}_n)$ with $\vec{\boldsymbol{a}}_i = (a_{1i}, a_{2i}, \cdots, a_{ni})^T$
- We define $\vec{\boldsymbol{u}}_i$ and $\vec{\boldsymbol{q}}_i$ as

$$\vec{\boldsymbol{u}}_i = \vec{\boldsymbol{a}}_i - \sum_{j=1}^{i-1} (\vec{\boldsymbol{q}}_j \cdot \vec{\boldsymbol{a}}_i)\vec{\boldsymbol{q}}_j, \qquad \vec{\boldsymbol{q}}_i = \frac{\vec{\boldsymbol{u}}_i}{|\vec{\boldsymbol{u}}_i|}$$

- We can demonstrate

$$\boldsymbol{A} = (\vec{\boldsymbol{a}}_1, \vec{\boldsymbol{a}}_2, \vec{\boldsymbol{a}}_3, \cdots) = (\vec{\boldsymbol{q}}_1, \vec{\boldsymbol{q}}_2, \vec{\boldsymbol{q}}_3 \cdots) \begin{pmatrix} |\vec{\boldsymbol{u}}_1| & (\vec{\boldsymbol{q}}_1 \cdot \vec{\boldsymbol{a}}_2) & (\vec{\boldsymbol{q}}_1 \cdot \vec{\boldsymbol{a}}_3) & \cdots \\ 0 & |\vec{\boldsymbol{u}}_2| & (\vec{\boldsymbol{q}}_2 \cdot \vec{\boldsymbol{a}}_3) & \cdots \\ 0 & 0 & |\vec{\boldsymbol{u}}_3| & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

- Thus $\boldsymbol{Q} = (\vec{\boldsymbol{q}}_1, \vec{\boldsymbol{q}}_2, \vec{\boldsymbol{q}}_3 \cdots)$ and $\boldsymbol{R} = \begin{pmatrix} |\vec{\boldsymbol{u}}_1| & (\vec{\boldsymbol{q}}_1 \cdot \vec{\boldsymbol{a}}_2) & (\vec{\boldsymbol{q}}_1 \cdot \vec{\boldsymbol{a}}_3) & \cdots \\ 0 & |\vec{\boldsymbol{u}}_2| & (\vec{\boldsymbol{q}}_2 \cdot \vec{\boldsymbol{a}}_3) & \cdots \\ 0 & 0 & |\vec{\boldsymbol{u}}_3| & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$,

  and $\boldsymbol{A} = \boldsymbol{QR}$.

An example

$$\boldsymbol{A} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ -1 & 0 & 1 & 0 \end{pmatrix}$$

- Based on the definition of $\vec{\boldsymbol{u}}_i$ and $\vec{\boldsymbol{q}}_i$

$$\vec{\boldsymbol{u}}_i = \vec{\boldsymbol{a}}_i - \sum_{j=1}^{i-1} (\vec{\boldsymbol{q}}_j \cdot \vec{\boldsymbol{a}}_i)\vec{\boldsymbol{q}}_j, \quad \vec{\boldsymbol{q}}_i = \frac{\vec{\boldsymbol{u}}_i}{|\vec{\boldsymbol{u}}_i|}$$

- First, we get all the $\vec{\boldsymbol{u}}_i$ and $\vec{\boldsymbol{q}}_i$ by the definition

$$\begin{cases} \vec{\boldsymbol{u}}_1 = \vec{\boldsymbol{a}}_1 = (1,1,1,-1)^{\boldsymbol{T}} \\ \vec{\boldsymbol{q}}_1 = \vec{\boldsymbol{u}}_1/|\vec{\boldsymbol{u}}_1| = (1,1,1,-1)^T/2 \end{cases}$$

$$\begin{cases} \vec{\boldsymbol{u}}_2 = \vec{\boldsymbol{a}}_2 - (\vec{\boldsymbol{q}}_1 \cdot \vec{\boldsymbol{a}}_2)\vec{\boldsymbol{q}}_1 = (2,2,0,0)^{\boldsymbol{T}} - (1,1,1,-1)^{\boldsymbol{T}} \\ \quad = (1,1,-1,1)^{\boldsymbol{T}} \\ \vec{\boldsymbol{q}}_2 = \vec{\boldsymbol{u}}_2/|\vec{\boldsymbol{u}}_2| = (1,1,-1,1)^T/2 \end{cases}$$

$$\begin{cases} \vec{\boldsymbol{u}}_3 = \vec{\boldsymbol{a}}_3 - (\vec{\boldsymbol{q}}_2 \cdot \vec{\boldsymbol{a}}_3)\vec{\boldsymbol{q}}_2 - (\vec{\boldsymbol{q}}_1 \cdot \vec{\boldsymbol{a}}_3)\vec{\boldsymbol{q}}_1 \\ \quad = (3,1,1,1)^{\boldsymbol{T}} - (1,1,-1,1)^T - (1,1,1,-1)^{\boldsymbol{T}} \\ \quad = (1,-1,1,1)^{\boldsymbol{T}} \\ \vec{\boldsymbol{q}}_3 = \vec{\boldsymbol{u}}_3/|\vec{\boldsymbol{u}}_3| = (1,-1,1,1)^T/2 \end{cases}$$

$$\begin{cases} \vec{\boldsymbol{u}}_4 = \vec{\boldsymbol{a}}_4 - (\vec{\boldsymbol{q}}_3 \cdot \vec{\boldsymbol{a}}_4)\vec{\boldsymbol{q}}_3 - (\vec{\boldsymbol{q}}_2 \cdot \vec{\boldsymbol{a}}_4)\vec{\boldsymbol{q}}_2 - (\vec{\boldsymbol{q}}_1 \cdot \vec{\boldsymbol{a}}_4)\vec{\boldsymbol{q}}_1 \\ \quad = (4,0,0,0)^{\boldsymbol{T}} - (1,-1,1,1)^{\boldsymbol{T}} - (1,1,-1,1)^T \\ \quad -(1,1,1,-1)^{\boldsymbol{T}} = (1,-1,-1,-1)^{\boldsymbol{T}} \\ \vec{\boldsymbol{q}}_4 = \vec{\boldsymbol{u}}_4/|\vec{\boldsymbol{u}}_4| = (1,-1,-1,-1)^T/2 \end{cases}$$

- Finally, we can get $\boldsymbol{Q}$ and $\boldsymbol{R}$

$$\boldsymbol{Q} = (\vec{\boldsymbol{q}}_1, \vec{\boldsymbol{q}}_2, \vec{\boldsymbol{q}}_3 \cdots) = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & -0.5 & -0.5 \\ 0.5 & -0.5 & 0.5 & -0.5 \\ -0.5 & 0.5 & 0.5 & -0.5 \end{pmatrix}$$

$$\boldsymbol{R} = \begin{pmatrix} |\vec{\boldsymbol{u}}_1| & (\vec{\boldsymbol{q}}_1 \cdot \vec{\boldsymbol{a}}_2) & (\vec{\boldsymbol{q}}_1 \cdot \vec{\boldsymbol{a}}_3) & (\vec{\boldsymbol{q}}_1 \cdot \vec{\boldsymbol{a}}_4) \\ 0 & |\vec{\boldsymbol{u}}_2| & (\vec{\boldsymbol{q}}_2 \cdot \vec{\boldsymbol{a}}_3) & (\vec{\boldsymbol{q}}_2 \cdot \vec{\boldsymbol{a}}_4) \\ 0 & 0 & |\vec{\boldsymbol{u}}_3| & (\vec{\boldsymbol{q}}_3 \cdot \vec{\boldsymbol{a}}_4) \\ 0 & 0 & 0 & |\vec{\boldsymbol{u}}_4| \end{pmatrix} = \begin{pmatrix} 2 & 2 & 2 & 2 \\ 0 & 2 & 2 & 2 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

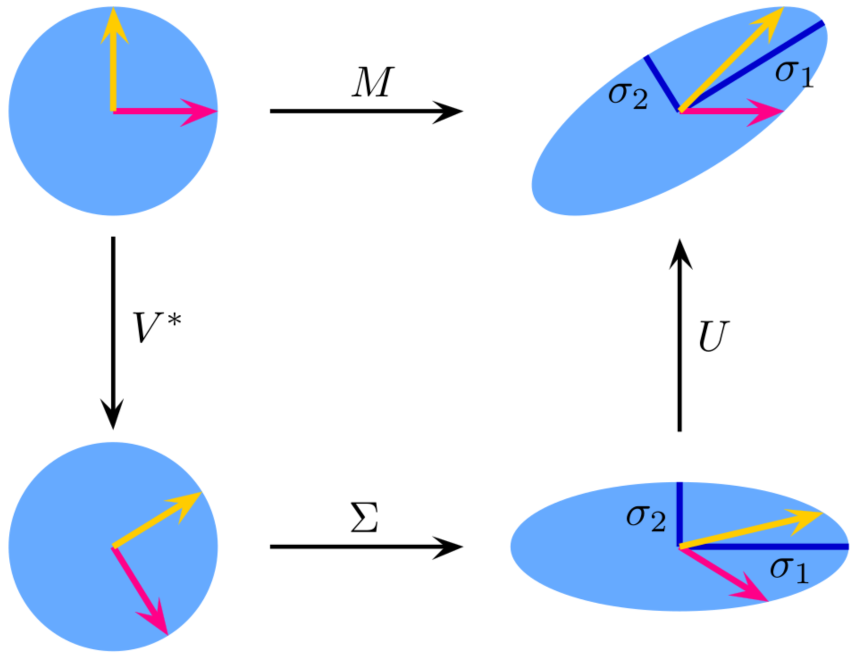
SVD and PCA

- **SVD** and **PCA** are two concepts closely related to EVD, which are very useful in many areas, e.g. numerical stabilization, machine learning, signal processing, etc.

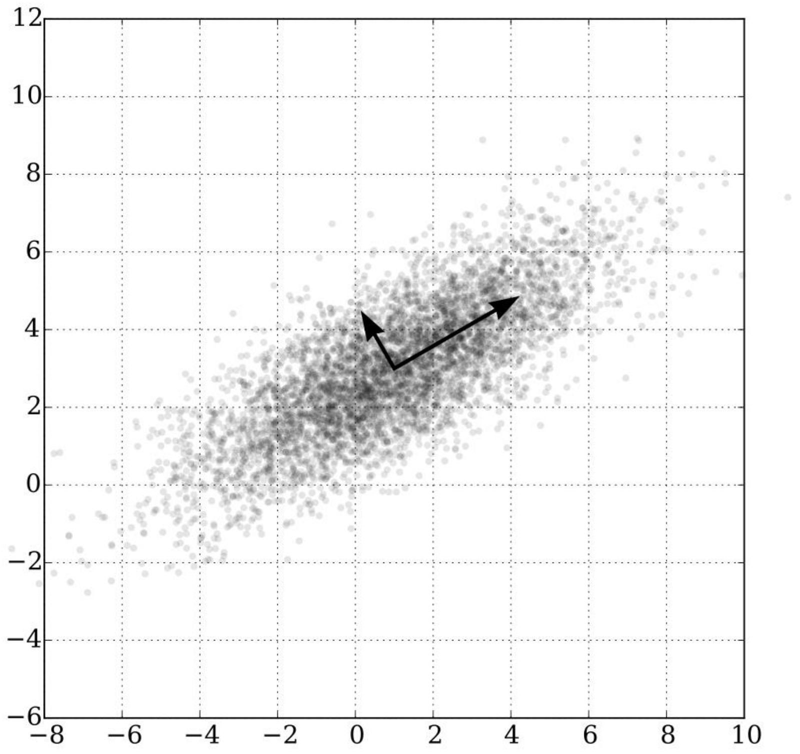
- **SVD** is the singular value decomposition, which can be regarded as an extension of EVD for non-square matrix of size $m \times n$. $\boldsymbol{M} = \boldsymbol{U\Sigma V}^*$, where the diagonal entries of $\boldsymbol{D}$ are the singular values of $\boldsymbol{\Sigma}$, and the first $p = \min(m, n)$ columns of $\boldsymbol{U}$ and $\boldsymbol{V}$ are left- and right-singular vectors for the corresponding singular values.

- **PCA** refers to the principal component analysis, which is used to convert a set of possibly correlated variables into as set of linearly uncorrelated variables (principal components).

SVD and PCA



$$M = U \cdot \Sigma \cdot V^*$$



Singular Value Decomposition (SVD)

- In linear algebra, the singular value decomposition (SVD) is a factorization of a real or complex matrix that generalizes the eigendecomposition of a square normal matrix to any $m \times n$ matrix via an extension of the polar decomposition.

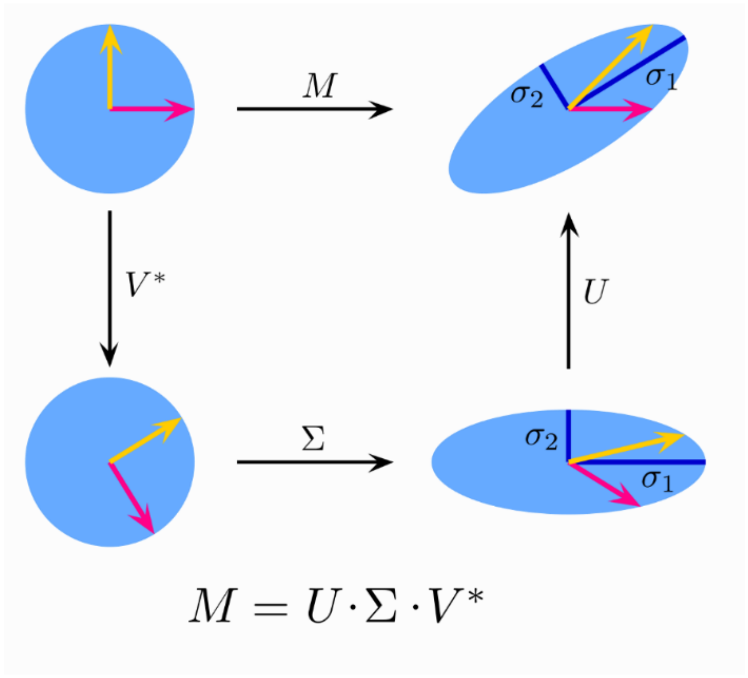
- Specifically, the singular value decomposition of an $m \times n$ real or complex matrix $\boldsymbol{M}$ is a factorization of the form $\boldsymbol{M} = \boldsymbol{U\Sigma V}^*$, where $\boldsymbol{U}$ is an $m \times m$ real or complex **unitary matrix**, $\boldsymbol{\Sigma}$ is an $m \times n$ rectangular **diagonal matrix** with ***non-negative real numbers*** on the diagonal, and $\boldsymbol{V}$ is an $n \times n$ real or complex **unitary matrix**. If $\boldsymbol{M}$ is real, $\boldsymbol{U}$ and $\boldsymbol{V}^* = \boldsymbol{V}^T$ are real orthogonal matrices.

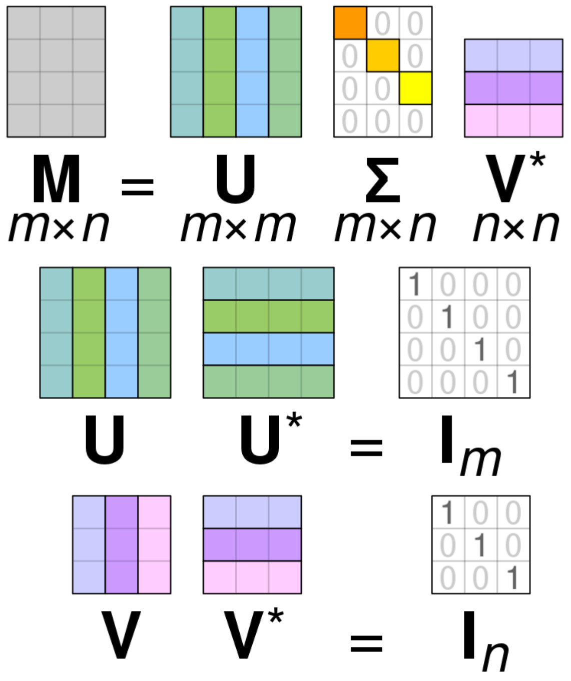
- The diagonal entries $\sigma_i = \boldsymbol{\Sigma}_{ii}$ of $\boldsymbol{\Sigma}$ are known as the ***singular values*** of $\boldsymbol{M}$. The number of non-zero singular values is equal to the rank of $\boldsymbol{M}$. The columns of $\boldsymbol{U}$ and the columns of $\boldsymbol{V}$ are called the ***left-singular vectors*** and ***right-singular vectors*** of $\boldsymbol{M}$, respectively.

- The SVD is not unique. It is always possible to choose the decomposition so that the singular values $\boldsymbol{\Sigma}_{ii}$ are in descending order. In this case, $\boldsymbol{\Sigma}$ (but not always $\boldsymbol{U}$ and $\boldsymbol{V}$) is uniquely determined by $\boldsymbol{M}$.

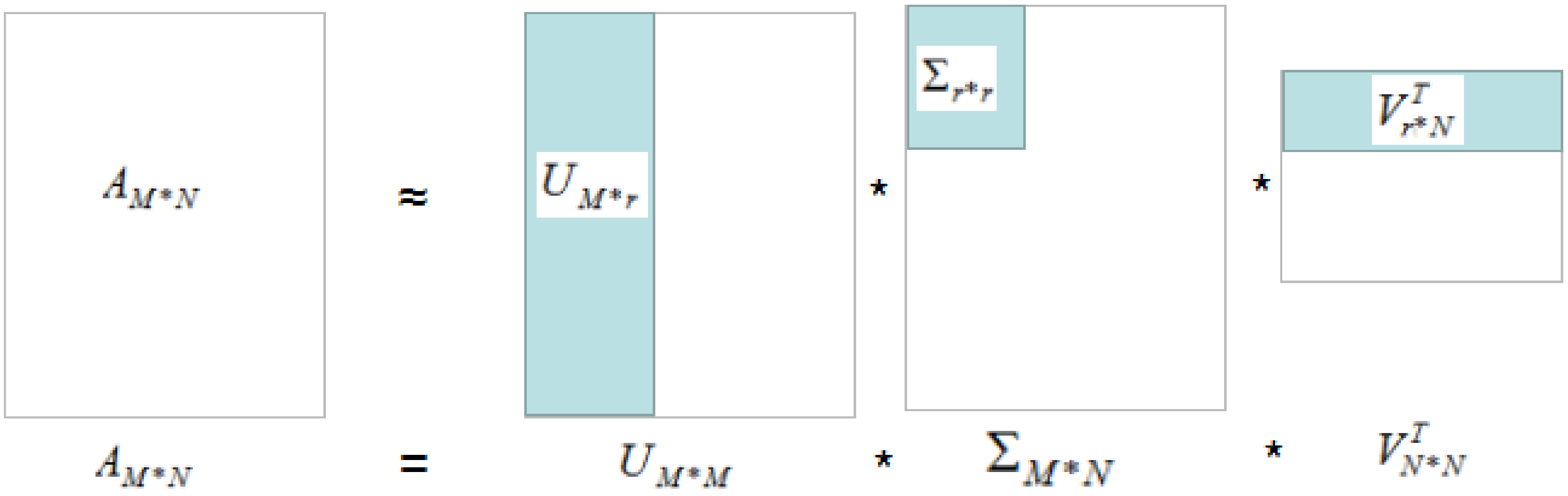
Singular value decomposition

- SVD sometimes also refers to the compact SVD, a similar decomposition $\boldsymbol{M} = \boldsymbol{U\Sigma V}^*$ in which $\boldsymbol{\Sigma}$ is square diagonal of size $r \times r$, where $r \leq \min\{m, n\}$ is the rank of $\boldsymbol{M}$, and has only the non-zero singular values. In this variant, $\boldsymbol{U}$ is an $m \times r$ semi-unitary matrix and $\boldsymbol{V}$ is an $n \times r$ semi-unitary matrix, such that $U * U = V * V = I_{r \times r}$.
- The SVD is also extremely useful in all areas of science, engineering, and statistics, such as signal processing, least squares fitting of data, and process control.





Data compression

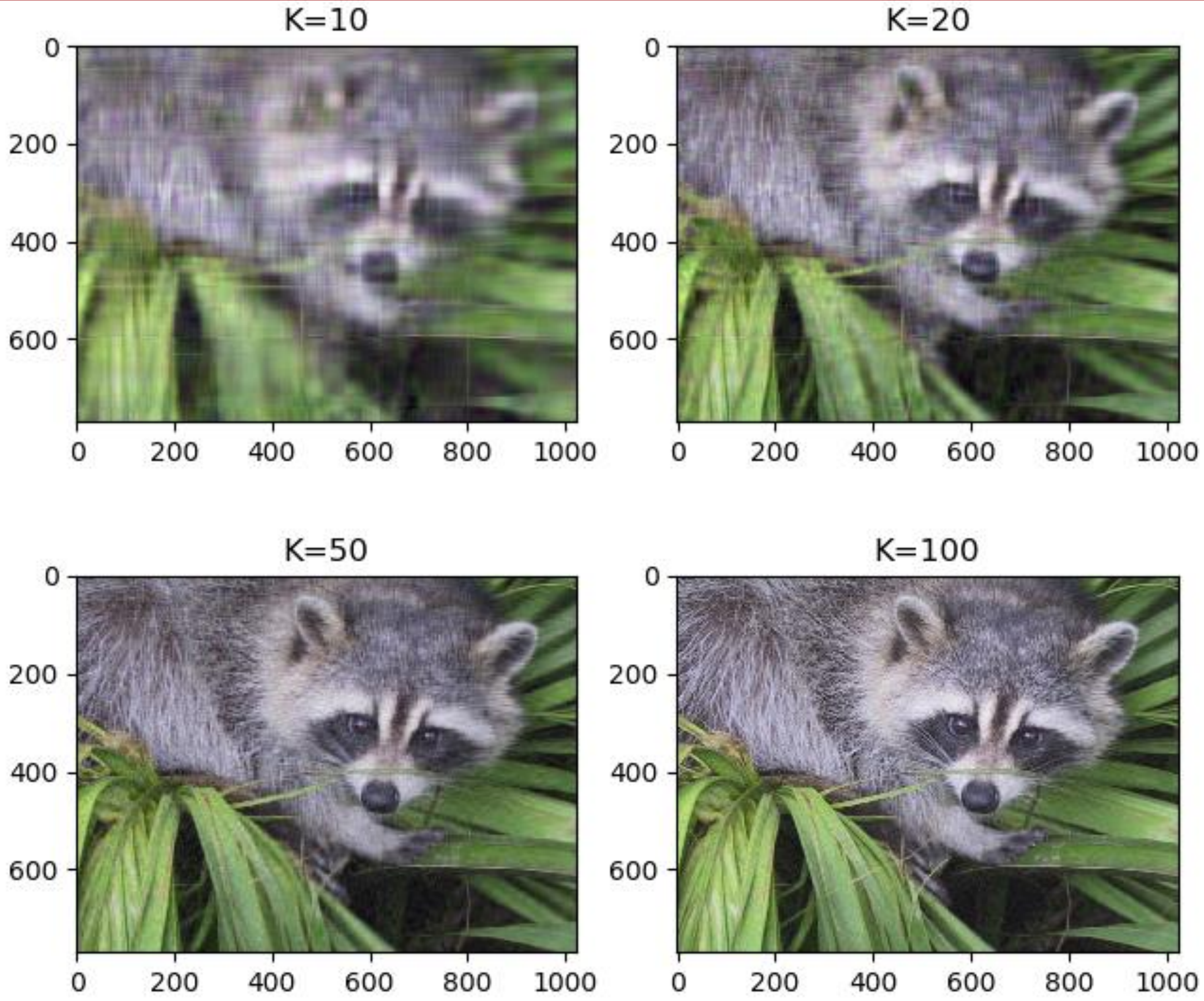


- The number of data you need to save is now $r * (M + N + 1)$, which is much smaller than the original $M * N$
- It is very useful in the image processing.

The example codes

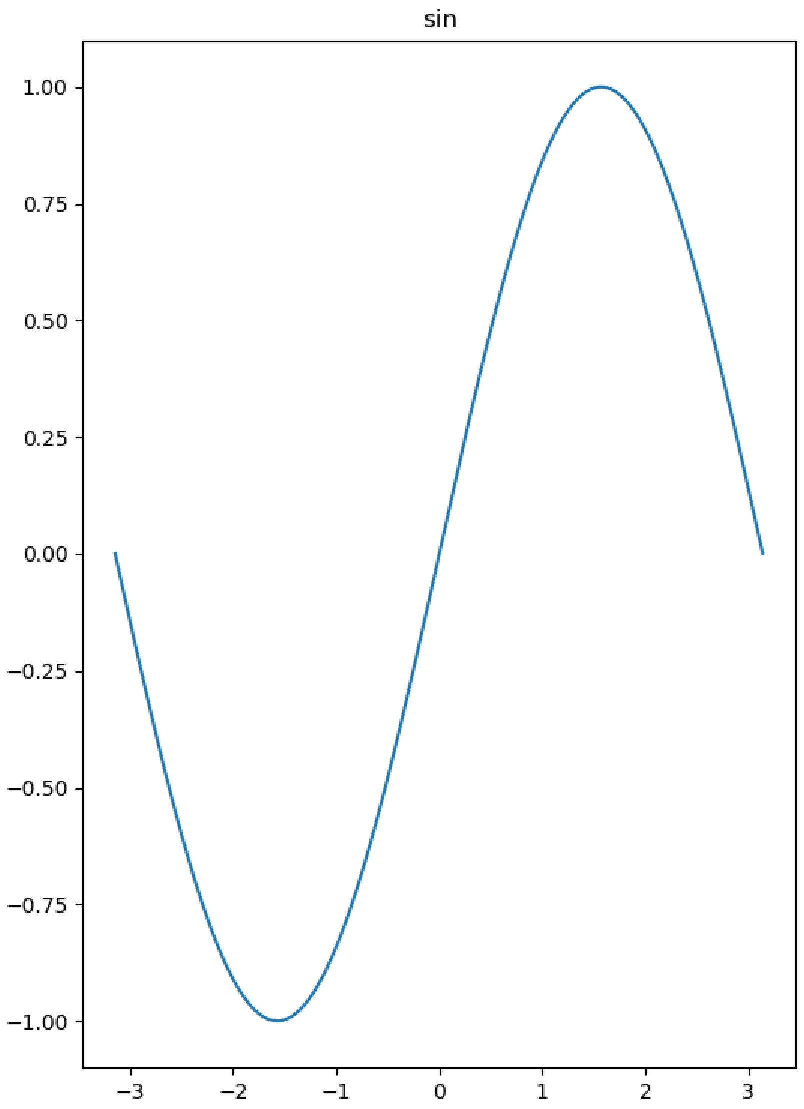
```python
from scipy import misc
import matplotlib.pyplot as plt
from numpy import linalg
import numpy as np

plt.figure()
img = misc.face()
img_array=img/img.max()


#If our array has more than two dimensions, then the SVD can be applied to
#all axes at once. However, the linear algebra functions in NumPy expect to
#see an array of the form (N, :, :), where the first axis represents
#the number of matrices
img_array_transposed = np.transpose(img_array, (2, 0, 1))

Ua, sa, Vta = linalg.svd(img_array_transposed)

Sigmaa = np.zeros((3, 768, 1024))
for j in range(3):
    np.fill_diagonal(Sigmaa[j,:,:], sa[j,:])

for k in (10,20,30,40,100):
    approx_img = Ua @ Sigmaa[:,:,0:k] @ Vta[:, 0:k, :]

    new_img2=np.transpose(approx_img, (1, 2, 0))
    new_img2=new_img2-new_img2.min()
    new_img2=new_img2/new_img2.max()

    plt.figure()
    plt.imshow(new_img2)
    plt.imsave('k'+str(k)+'.pdf', new_img2)
```

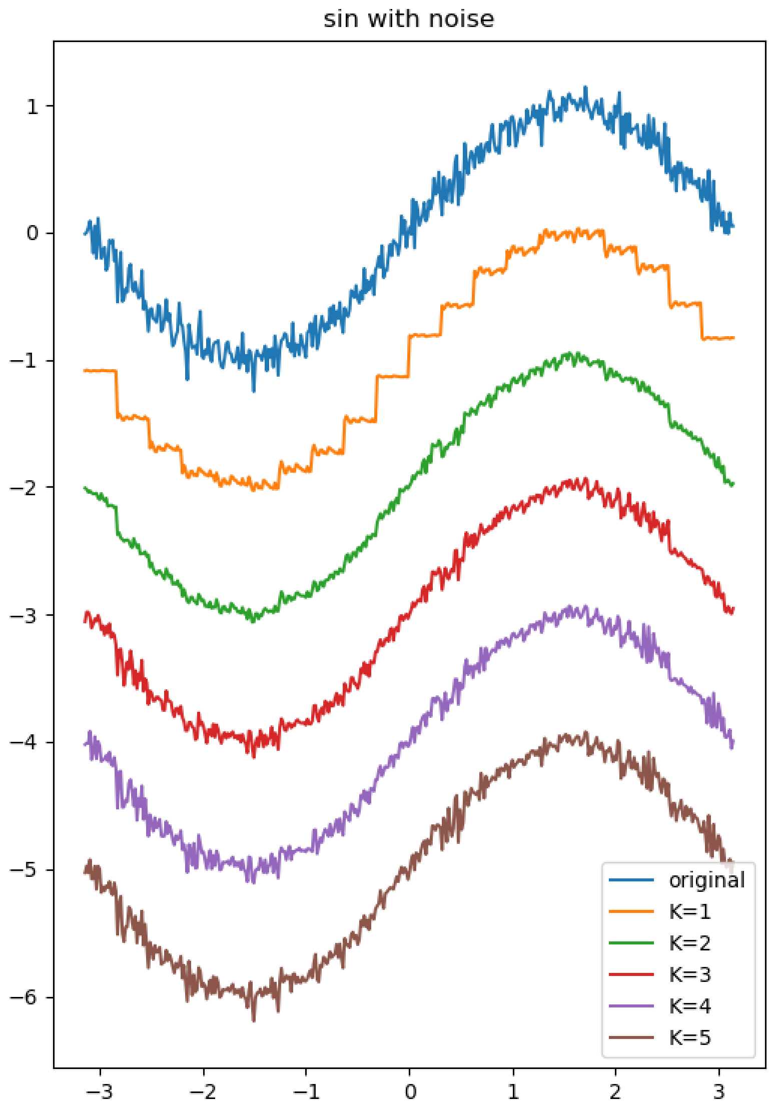
The results



Noise reduction

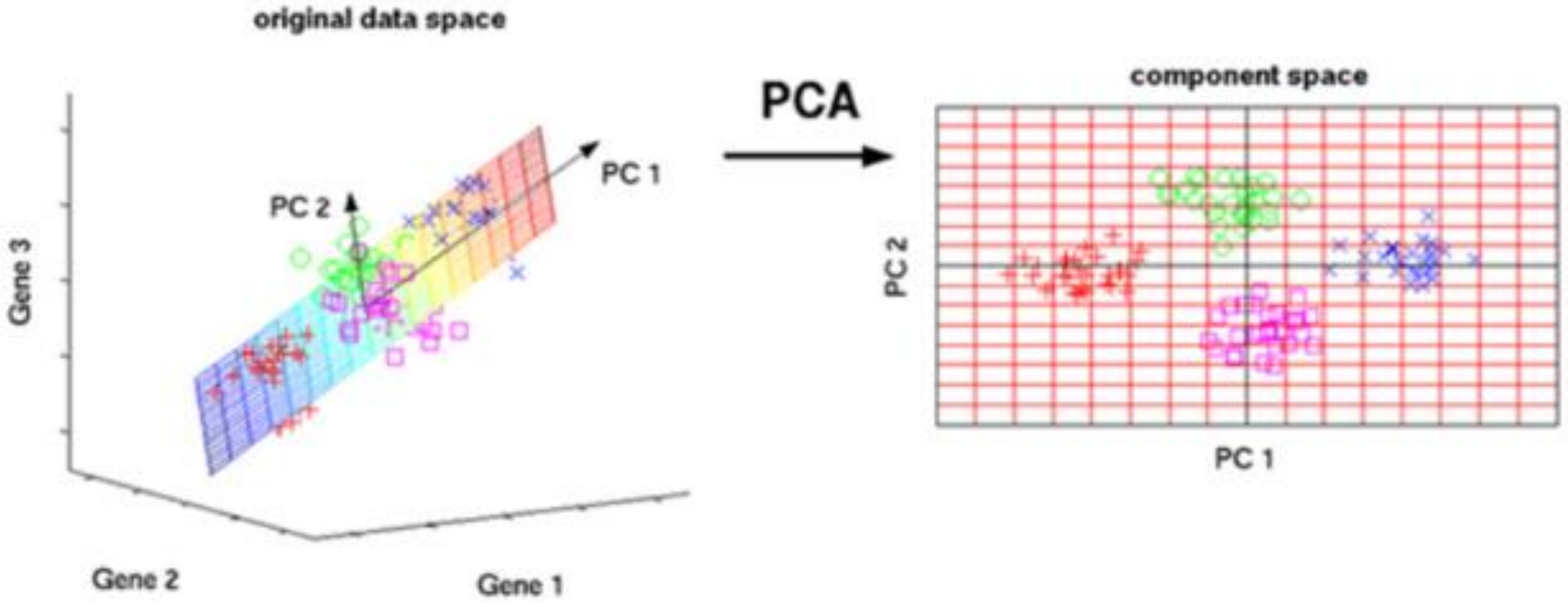
- In some cases, the noise in the signal corresponds to the states with smaller singular values. Then we can use SVD to reduce the noise.





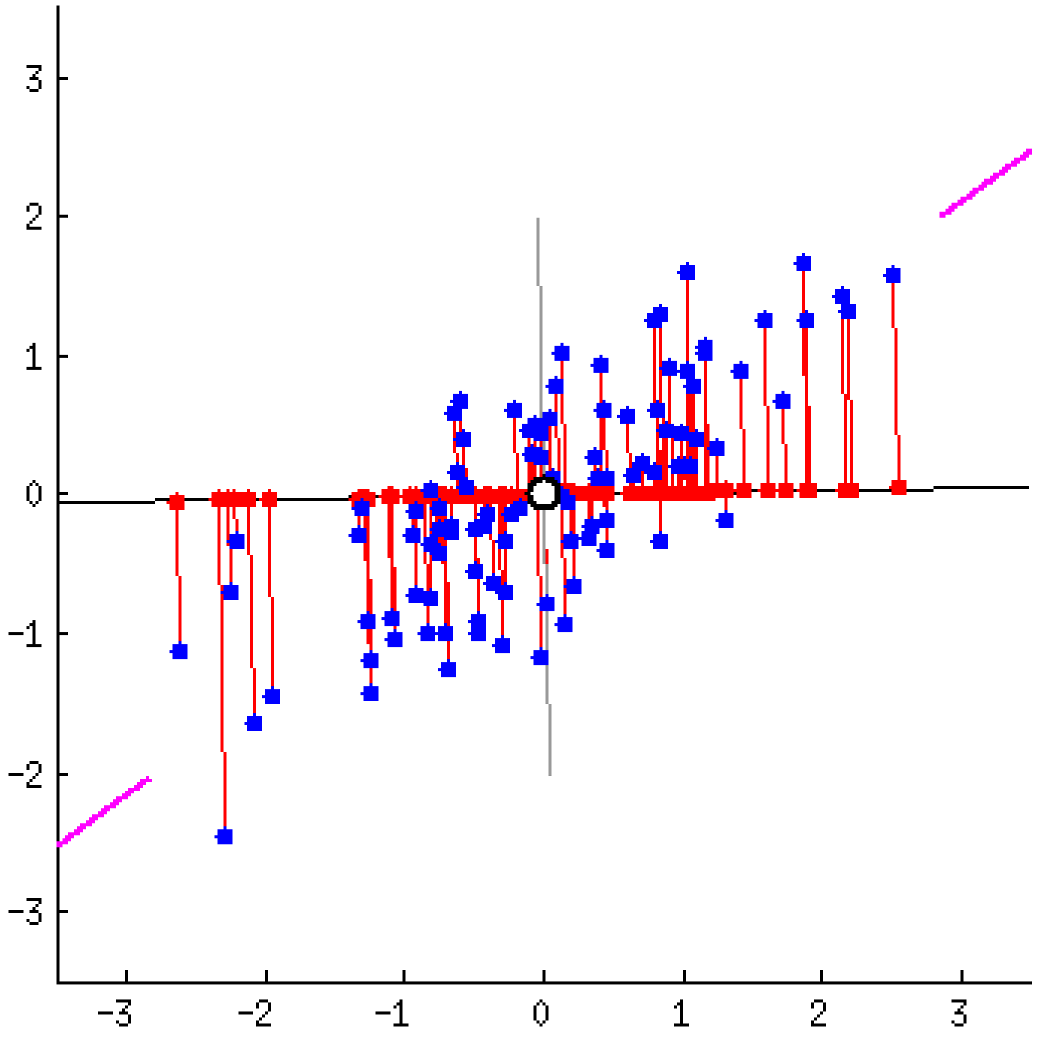
Principal Component Analysis (PCA)

- Principal component analysis, or PCA, is a statistical procedure that allows you to summarize the information content in large data tables by means of a smaller set of “summary indices” that can be more easily visualized and analyzed.



Principal Component Analysis (PCA)

- Statistically, PCA finds lines, planes and hyper-planes in the K-dimensional space that approximate the data as well as possible in the least squares sense. A line or plane that is the least squares approximation of a set of data points makes the variance of the coordinates on the line or plane as large as possible.



Variance and covariance

- Variance of one sample X

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

- Covariance of two samples X and Y

$$\begin{aligned} Cov(X,Y) &= E\left[(X - E(X))(Y - E(Y))\right] \\ &= \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) \end{aligned}$$

- Covariance matrix of multiple samples labelled by X

$$X = \begin{pmatrix} a_1 & a_2 & \cdots & a_m \\ b_1 & b_2 & \cdots & b_m \end{pmatrix}$$

$$\frac{1}{m}XX^\mathsf{T} = \begin{pmatrix} \frac{1}{m}\sum_{i=1}^{m} a_i^2 & \frac{1}{m}\sum_{i=1}^{m} a_i b_i \\ \frac{1}{m}\sum_{i=1}^{m} a_i b_i & \frac{1}{m}\sum_{i=1}^{m} b_i^2 \end{pmatrix} = \begin{pmatrix} Cov(a,a) & Cov(a,b) \\ Cov(b,a) & Cov(b,b) \end{pmatrix}$$

PCA and EVD

- In PCA, we want to find new basis/principal component, in which all the covariance are minimized.

$$\frac{1}{m}XX^{\mathsf{T}} = \begin{pmatrix} \frac{1}{m}\sum_{i=1}^{m} a_i^2 & \frac{1}{m}\sum_{i=1}^{m} a_i b_i \\ \frac{1}{m}\sum_{i=1}^{m} a_i b_i & \frac{1}{m}\sum_{i=1}^{m} b_i^2 \end{pmatrix} = \begin{pmatrix} Cov(a,a) & Cov(a,b) \\ Cov(b,a) & Cov(b,b) \end{pmatrix}$$

- Relation of covariance in different basis

$$\begin{aligned} D &= \frac{1}{m}YY^T \\ &= \frac{1}{m}(PX)(PX)^T \\ &= \frac{1}{m}PXX^TP^T \\ &= P(\frac{1}{m}XX^T)P^T \\ &= PCP^T \end{aligned}$$

- Definition of EVD

$$E^TCE = \Lambda = \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix}$$

- **PCA is to just find the EVD of the covariance matrix of all the samples.**

Application of PCA

- **Data Visualization:** When working on any data related problem, Considering that there are a large number of variables or dimensions along which the data is distributed, visualization can be a challenge and almost impossible. Hence, PCA can do that for you since it projects the data into a lower dimension, thereby allowing you to visualize the data in a 2D or 3D space with a naked eye.

- **Machine Learning :** Since PCA's main idea is dimensionality reduction, you can leverage that to speed up your machine learning algorithm's training and testing time considering your data has a lot of features, and the ML algorithm's learning is too slow. PCA can also be used as a machine learning algorithm and it is widely used in quantitative finance, image compression, facial recognition, etc.

Fourier series

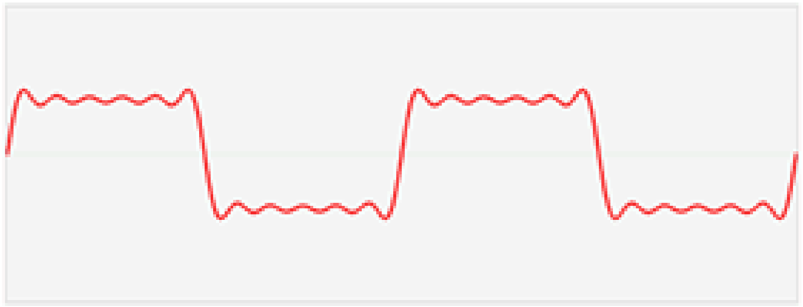
- Consider a periodic real-valued function, $f(x)$, that is integrable on an interval of length P (one period), then $f(x)$ can always be expressed as Fourier series as following

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{N} \left(a_n \cos\left(\frac{2\pi}{P} nx\right) + b_n \sin\left(\frac{2\pi}{P} nx\right) \right)$$

$$a_0 = \frac{2}{P} \int_P f(x) dx$$

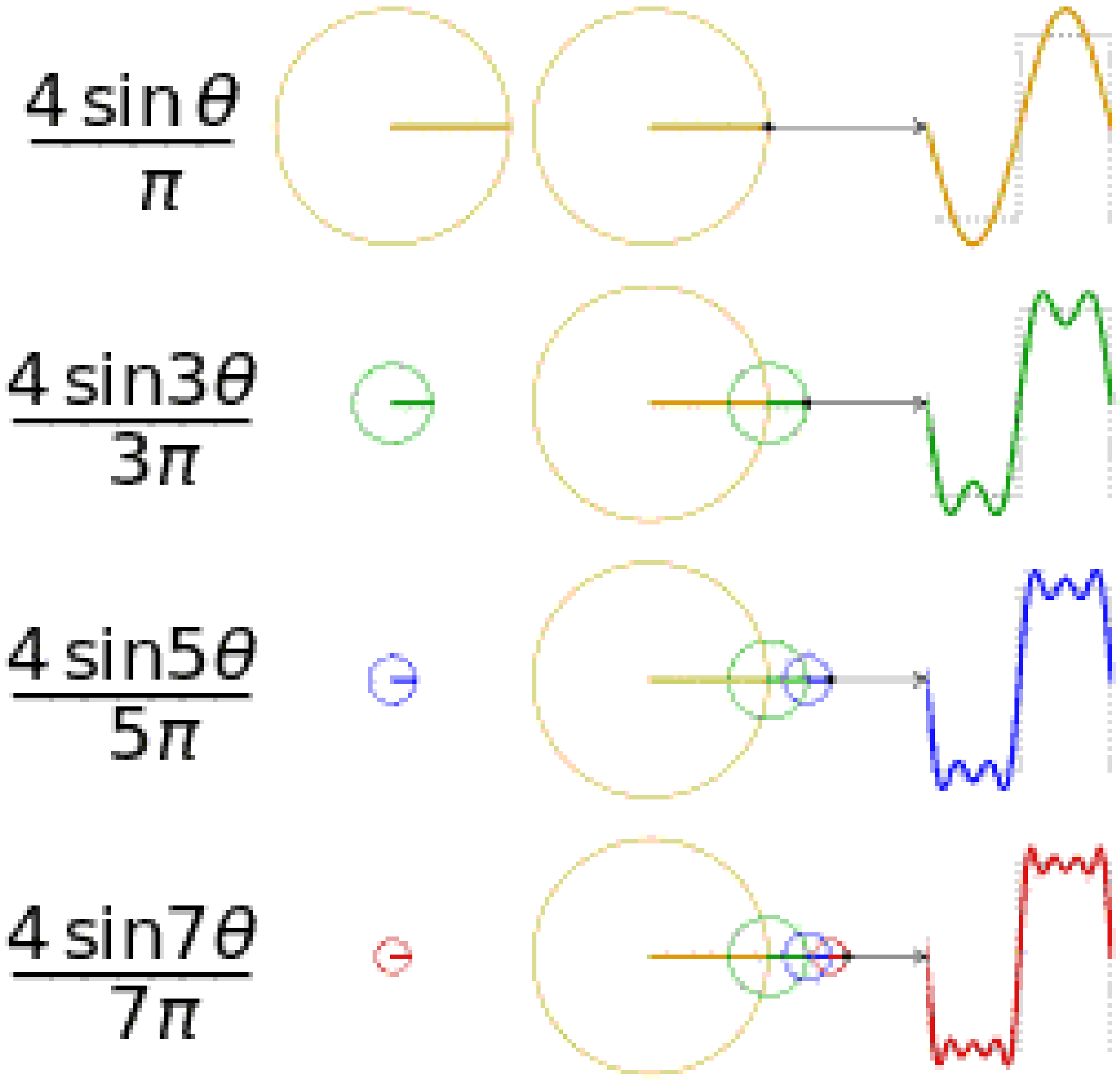
$$a_n = \frac{2}{P} \int_P f(x) \cos\left(\frac{2\pi}{P} nx\right) dx$$

$$b_n = \frac{2}{P} \int_P f(x) \sin\left(\frac{2\pi}{P} nx\right) dx$$



- Usually, we choose $P = [-\pi, \pi]$ o $P = [0,1]$ to simply the analysis.

Summation of sin(nx) → circular movement



$\frac{4\sin\theta}{\pi}$
$\frac{4\sin 3\theta}{3\pi}$
$\frac{4\sin 5\theta}{5\pi}$
$\frac{4\sin 7\theta}{7\pi}$

Exponential form (Complex number expression)

- Fourier series can be expressed in simple exponential form

$$f(x) = \sum_{n=-N}^{N} c_n e^{\frac{i2\pi}{P}nx}$$

$$c_n = \frac{1}{P}\int_P f(x) e^{-\frac{i2\pi}{P}nx}\, dx$$

- If $f(x)$ is real-valued, then c_n and c_{-n} are complex conjugates, otherwise they are not.

- We can also define the Fourier series for functions of two variables in the square with length P_x and P_y

$$f(x,y) = \sum_{n=-N}^{N} c_{nm} e^{\frac{i2\pi}{P_x}nx} e^{\frac{i2\pi}{P_y}ny}$$

$$c_{nm} = \frac{1}{P_x P_y}\int_{P_x,P_y} f(x,y) e^{-\frac{i2\pi}{P_x}nx} e^{-\frac{i2\pi}{P_y}ny}\, dxdy$$

Hilbert space interpretation

- In the language of Hilbert space, the set of function $\{e_n =$

- One can demonstrate
$\frac{1}{\pi}\int_{-\pi}^{\pi}\cos(mx)\sin(nx)\,dx = 0,$
$\frac{1}{\pi}\int_{-\pi}^{\pi}\cos(mx)\cos(nx)\,dx = \delta_{mn},$
$\frac{1}{\pi}\int_{-\pi}^{\pi}\sin(mx)\sin(nx)\,dx = \delta_{mn}$

Fourier transform

- The Fourier transform is an extension of the Fourier series that results when the period of the represented function is lengthened and allowed to approach infinity

$$f(x) = \int_{-\infty}^{\infty} c(k) e^{2\pi i k x} dk$$

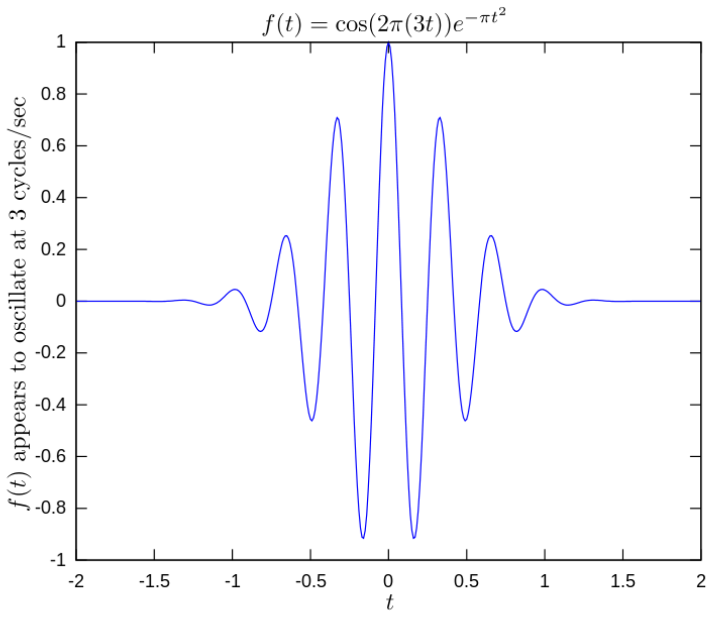
$$c(k) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i k x} dx$$

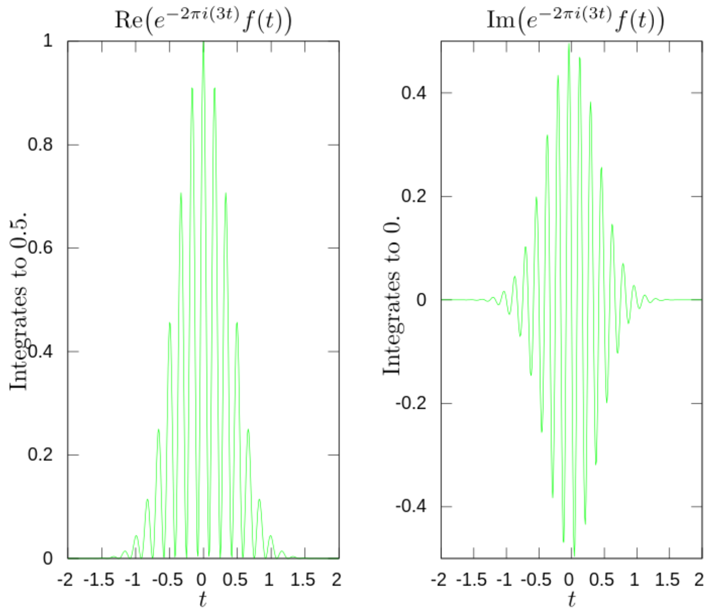
- In some cases, we write $\omega = 2\pi k$, then we have

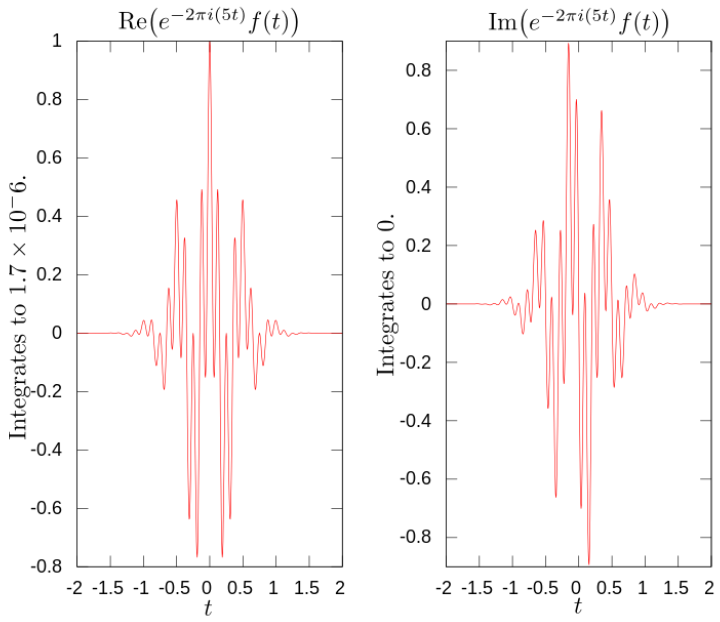
$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} c(\omega) e^{i\omega x} d\omega$$

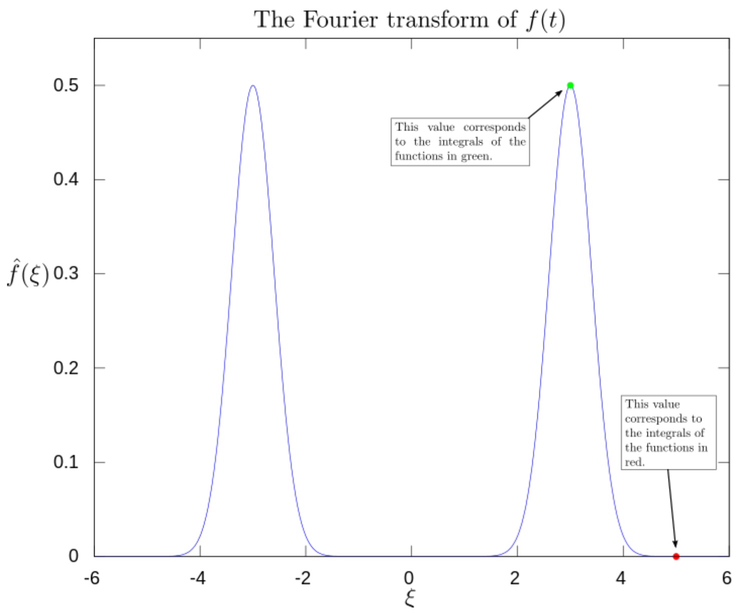
$$c(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i\omega x} dx$$

One example


$f(t) = \cos(2\pi(3t))e^{-\pi t^2}$
$f(t)$ appears to oscillate at 3 cycles/sec
t


$\mathrm{Re}\left(e^{-2\pi i(3t)} f(t)\right)$
Integrates to 0.5.
$\mathrm{Im}\left(e^{-2\pi i(3t)} f(t)\right)$
Integrates to 0.
t


$\mathrm{Re}\left(e^{-2\pi i(5t)} f(t)\right)$
Integrates to 1.7×10^-6.
$\mathrm{Im}\left(e^{-2\pi i(5t)} f(t)\right)$
Integrates to 0.
t


The Fourier transform of $f(t)$
$\hat{f}(\xi)$
ξ
This value corresponds to the integrals of the functions in green.
This value corresponds to the integrals of the functions in red.

Discrete Fourier Transform

- The DFT is the equivalent of the continuous Fourier Transform for signals known only at discrete points (usually the case in real life).

- Let $f(x)$ can only exist at given points $x = x_0, x_2, \dots, x_{N-1}$, and we label $f(x_n)$ to be $f(n)$ then

$$f(n) = \frac{1}{N}\sum_{k=0}^{N-1} c(k)e^{i\frac{2\pi}{N}kn} = \frac{1}{N}\sum_{k=0}^{N-1} c(k)w_N^{-kn}$$

$$c(k) = \sum_{n=0}^{N-1} f(n)e^{-i\frac{2\pi}{N}kn} = \sum_{n=0}^{N-1} f(n)w_N^{kn}$$

Where $w_N = e^{-\frac{i2\pi}{N}}$, $n = 0,1,\dots,N-1$ and $k = 0,1,\dots,N-1$

Matrix form and FFT

- DFT can be expressed in simple matrix form

$$\begin{bmatrix} c(0) \\ c(1) \\ c(2) \\ c(3) \\ \vdots \\ c(N-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & w_N^1 & w_N^2 & w_N^3 & \dots & w_N^{(N-1)} \\ 1 & w_N^2 & w_N^4 & w_N^6 & \dots & w_N^{2(N-1)} \\ 1 & w_N^3 & w_N^6 & w_N^9 & \dots & w_N^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w_N^{N-1} & w_N^{2(N-1)} & w_N^{3(N-1)} & \dots & w_N^{(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} f(0) \\ f(1) \\ f(2) \\ f(3) \\ \vdots \\ f(N-1) \end{bmatrix}$$

- It is discovered that the Fourier matrix W_N has the following properties due to its high symmetry

$$W_{2N} = \begin{bmatrix} I_N & D_N \\ I_N & -D_N \end{bmatrix} \begin{bmatrix} W_N & 0 \\ 0 & W_N \end{bmatrix} P_{2N}$$

Where I_N is an identity matrix, D_N is a diagonal matrix with diagonal element to be $w_{2N}^0, w_{2N}^1, \dots, w_{2N}^{N-1}$ and P_{2N} is a permutation matrix.

- By continuously doing this matrix factorization, you can get the Fast Fourier transform with complexity as $N\log N$ instead of N^2.

Typical Application of FFT

The FFT is used in digital recording, sampling, additive synthesis and pitch correction software. The FFT's importance derives from the fact that it has made working in the frequency domain equally computationally feasible as working in the temporal or spatial domain. Some of the important applications of the FFT include:

- Filter, denoise, and compress data
- Solving difference equations
- Fast large-integer and polynomial multiplication
- Efficient matrix–vector multiplication for structured matrices
- Fast algorithms for discrete cosine or sine transforms (e.g. fast DCT used for JPEG and MPEG/MP3 encoding and decoding)
- Modulation and demodulation of complex data symbols for 5G, LTE, Wi-Fi, DSL, and other modern communication systems