In [3]:
```
conda info
```

```
     active environment : base
    active env location : /Users/chagall/anaconda3
            shell level : 1
       user config file : /Users/chagall/.condarc
 populated config files : /Users/chagall/.condarc
          conda version : 23.7.3
    conda-build version : 3.26.0
         python version : 3.11.4.final.0
       virtual packages : __archspec=1=arm64
                          __osx=13.5.2=0
                          __unix=0=0
       base environment : /Users/chagall/anaconda3  (writable)
      conda av data dir : /Users/chagall/anaconda3/etc/conda
  conda av metadata url : None
          channel URLs : https://repo.anaconda.com/pkgs/main/osx-arm64
(https://repo.anaconda.com/pkgs/main/osx-arm64)
                         https://repo.anaconda.com/pkgs/main/noarch (htt
ps://repo.anaconda.com/pkgs/main/noarch)
                         https://repo.anaconda.com/pkgs/r/osx-arm64 (htt
ps://repo.anaconda.com/pkgs/r/osx-arm64)
                         https://repo.anaconda.com/pkgs/r/noarch (http
s://repo.anaconda.com/pkgs/r/noarch)
         package cache : /Users/chagall/anaconda3/pkgs
                         /Users/chagall/.conda/pkgs
       envs directories : /Users/chagall/anaconda3/envs
                         /Users/chagall/.conda/envs
               platform : osx-arm64
             user-agent : conda/23.7.3 requests/2.31.0 CPython/3.11.4 Dar
win/22.6.0 OSX/13.5.2 aau/0.3.0 c/QVr5NLwlTVyC4iPadkftyw s/0LkFlYj1SnWSUV
UVs-puyA e/ALQyCJQQQ2m033AiLYCUHw
                UID:GID : 501:20
             netrc file : /Users/chagall/.netrc
           offline mode : False


Note: you may need to restart the kernel to use updated packages.
```

In [4]:
```
import numpy as np
from scipy import io, integrate, linalg, signal
from scipy.sparse.linalg import cg, eigs
from numpy.random import default_rng
```

In [5]:
```
a = np.array([[1., 2., 3.], [4., 5., 6.]])
```

In [6]:
```
np.ndim(a)  #number of dimensions
```

Out[6]: 2

In [7]:
```
np.size(a)  #number of elements
```

Out[7]:
```
6
```

In [8]:
```
np.shape(a)    #tuple of array dimensions
```

Out[8]:
```
(2, 3)
```

In [9]:
```
a.shape[1]
```

Out[9]:
```
3
```

In [10]:
```
a.shape[0]  #number of rows
```

Out[10]:
```
2
```

In [11]:
```
a.shape[2]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[11], line 1
----> 1 a.shape[2]

IndexError: tuple index out of range
```

In [12]:
```
b=c=d=a
```

In [13]:
```
np.block([[a, b], [c, d]])  #block matrix
```

Out[13]:
```
array([[1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.]])
```

In [14]:
```
a[-1]#access last row/last element
```

Out[14]:
```
array([4., 5., 6.])
```

In [15]:
```
# block two a together in a row
a = np.array([[1., 2., 3.], [4., 5., 6.]])
a=np.block([[a], [a]])
a
```

Out[15]:
```
array([[1., 2., 3.],
       [4., 5., 6.],
       [1., 2., 3.],
       [4., 5., 6.]])
```

In [16]:
```
# block two a together in a column
a = np.array([[1., 2., 3.], [4., 5., 6.]])
a=np.block([a,a])
a
```

Out[16]:
```
array([[1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.]])
```

In [17]:
```
a[1, 4]#access 2nd row, 5th element
```

Out[17]:
```
5.0
```

In [18]:
```
a[1, :]#access 2nd row
```

Out[18]:
```
array([4., 5., 6., 4., 5., 6.])
```

In [19]:
```
a[1]#access 2nd row
```

Out[19]:
```
array([4., 5., 6., 4., 5., 6.])
```

In [20]:
```
a[:,:4]   #access first 4 columns
```

Out[20]:
```
array([[1., 2., 3., 1.],
       [4., 5., 6., 4.]])
```

In [21]:
```
a[-1:]    #access last row
```

Out[21]:
```
array([[4., 5., 6., 4., 5., 6.]])
```

In [22]:
```
a=np.block([[a], [a], [a],[a]])
a
```

Out[22]:
```
array([[1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.]])
```

In [23]:
```
a[4:9,0:3]#access 1st to 3rd row, 5th to 9th element
```

Out[23]:
```
array([[1., 2., 3.],
       [4., 5., 6.],
       [1., 2., 3.],
       [4., 5., 6.]])
```

In [24]:
```
a[np.ix_([1, 3, 4], [0, 2])]
#access 2nd, 4th, 5th row, 1st, 3rd column
```

Out[24]:
```
array([[4., 6.],
       [4., 6.],
       [1., 3.]])
```

In [25]:
```
a[2:9:2,:]
#access 3rd to 9th row, every other row
```

Out[25]:
```
array([[1., 2., 3., 1., 2., 3.],
       [1., 2., 3., 1., 2., 3.],
       [1., 2., 3., 1., 2., 3.]])
```

In [26]:
```
a[::2, :]
#access every other row
```

Out[26]:
```
array([[1., 2., 3., 1., 2., 3.],
       [1., 2., 3., 1., 2., 3.],
       [1., 2., 3., 1., 2., 3.],
       [1., 2., 3., 1., 2., 3.]])
```

In [27]:
```
a[::-1,:]
#access every  row in reverse order
```

Out[27]:
```
array([[4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.]])
```

In [31]:
```
a[np.r_[:len(a),0]]
#a with copy of the first row appended to the end
#复制的第一行附加到末尾，不会对a做出改变
```

Out[31]:
```
array([[1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.]])
```

In [32]:
```
a.T
```

Out[32]:
```
array([[1., 4., 1., 4., 1., 4., 1., 4.],
       [2., 5., 2., 5., 2., 5., 2., 5.],
       [3., 6., 3., 6., 3., 6., 3., 6.],
       [1., 4., 1., 4., 1., 4., 1., 4.],
       [2., 5., 2., 5., 2., 5., 2., 5.],
       [3., 6., 3., 6., 3., 6., 3., 6.]])
```

In [33]:
```
a.conj().T
#conjugate transpose 共轭
```

Out[33]:
```
array([[1., 4., 1., 4., 1., 4., 1., 4.],
       [2., 5., 2., 5., 2., 5., 2., 5.],
       [3., 6., 3., 6., 3., 6., 3., 6.],
       [1., 4., 1., 4., 1., 4., 1., 4.],
       [2., 5., 2., 5., 2., 5., 2., 5.],
       [3., 6., 3., 6., 3., 6., 3., 6.]])
```

In [37]:
```
b=a.T
```

In [38]:
```
a@b
```

Out[38]:
```
array([[ 28.,  64.,  28.,  64.,  28.,  64.,  28.,  64.],
       [ 64., 154.,  64., 154.,  64., 154.,  64., 154.],
       [ 28.,  64.,  28.,  64.,  28.,  64.,  28.,  64.],
       [ 64., 154.,  64., 154.,  64., 154.,  64., 154.],
       [ 28.,  64.,  28.,  64.,  28.,  64.,  28.,  64.],
       [ 64., 154.,  64., 154.,  64., 154.,  64., 154.],
       [ 28.,  64.,  28.,  64.,  28.,  64.,  28.,  64.],
       [ 64., 154.,  64., 154.,  64., 154.,  64., 154.]])
```

In [39]:
```
a*2
```

Out[39]:
```
array([[ 2.,  4.,  6.,  2.,  4.,  6.],
       [ 8., 10., 12.,  8., 10., 12.],
       [ 2.,  4.,  6.,  2.,  4.,  6.],
       [ 8., 10., 12.,  8., 10., 12.],
       [ 2.,  4.,  6.,  2.,  4.,  6.],
       [ 8., 10., 12.,  8., 10., 12.],
       [ 2.,  4.,  6.,  2.,  4.,  6.],
       [ 8., 10., 12.,  8., 10., 12.]])
```

In [40]:
```
a**2
```

Out[40]:
```
array([[ 1.,  4.,  9.,  1.,  4.,  9.],
       [16., 25., 36., 16., 25., 36.],
       [ 1.,  4.,  9.,  1.,  4.,  9.],
       [16., 25., 36., 16., 25., 36.],
       [ 1.,  4.,  9.,  1.,  4.,  9.],
       [16., 25., 36., 16., 25., 36.],
       [ 1.,  4.,  9.,  1.,  4.,  9.],
       [16., 25., 36., 16., 25., 36.]])
```

In [41]:
```
a/2
```

Out[41]:
```
array([[0.5, 1. , 1.5, 0.5, 1. , 1.5],
       [2. , 2.5, 3. , 2. , 2.5, 3. ],
       [0.5, 1. , 1.5, 0.5, 1. , 1.5],
       [2. , 2.5, 3. , 2. , 2.5, 3. ],
       [0.5, 1. , 1.5, 0.5, 1. , 1.5],
       [2. , 2.5, 3. , 2. , 2.5, 3. ],
       [0.5, 1. , 1.5, 0.5, 1. , 1.5],
       [2. , 2.5, 3. , 2. , 2.5, 3. ]])
```

In [44]:
```
a>2
```

Out[44]:
```
array([[False, False,  True, False, False,  True],
       [ True,  True,  True,  True,  True,  True],
       [False, False,  True, False, False,  True],
       [ True,  True,  True,  True,  True,  True],
       [False, False,  True, False, False,  True],
       [ True,  True,  True,  True,  True,  True],
       [False, False,  True, False, False,  True],
       [ True,  True,  True,  True,  True,  True]])
```

In [46]:
```
np.nonzero(a > 2)
#find the indices where (a > 0.5)
```

Out[46]:
```
(array([0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 3, 3, 3, 3, 3, 3, 4, 4, 5, 5, 5, 5,
        5, 5, 6, 6, 7, 7, 7, 7, 7, 7]),
 array([2, 5, 0, 1, 2, 3, 4, 5, 2, 5, 0, 1, 2, 3, 4, 5, 2, 5, 0, 1, 2, 3,
        4, 5, 2, 5, 0, 1, 2, 3, 4, 5]))
```

In [47]:
```
#a(:,find(v > 0.5))
#extract the columns of a where vector v > 0.5
a[:,np.nonzero(b > 2)[0]]
```

Out[47]:
```
array([[1., 1., 1., 1., 2., 2., 2., 2., 3., 3., 3., 3., 3., 3., 3., 3.,
        1., 1., 1., 1., 2., 2., 2., 2., 3., 3., 3., 3., 3., 3., 3., 3.],
       [4., 4., 4., 4., 5., 5., 5., 5., 6., 6., 6., 6., 6., 6., 6., 6.,
        4., 4., 4., 4., 5., 5., 5., 5., 6., 6., 6., 6., 6., 6., 6., 6.],
       [1., 1., 1., 1., 2., 2., 2., 2., 3., 3., 3., 3., 3., 3., 3., 3.,
        1., 1., 1., 1., 2., 2., 2., 2., 3., 3., 3., 3., 3., 3., 3., 3.],
       [4., 4., 4., 4., 5., 5., 5., 5., 6., 6., 6., 6., 6., 6., 6., 6.,
        4., 4., 4., 4., 5., 5., 5., 5., 6., 6., 6., 6., 6., 6., 6., 6.],
       [1., 1., 1., 1., 2., 2., 2., 2., 3., 3., 3., 3., 3., 3., 3., 3.,
        1., 1., 1., 1., 2., 2., 2., 2., 3., 3., 3., 3., 3., 3., 3., 3.],
       [4., 4., 4., 4., 5., 5., 5., 5., 6., 6., 6., 6., 6., 6., 6., 6.,
        4., 4., 4., 4., 5., 5., 5., 5., 6., 6., 6., 6., 6., 6., 6., 6.],
       [1., 1., 1., 1., 2., 2., 2., 2., 3., 3., 3., 3., 3., 3., 3., 3.,
        1., 1., 1., 1., 2., 2., 2., 2., 3., 3., 3., 3., 3., 3., 3., 3.],
       [4., 4., 4., 4., 5., 5., 5., 5., 6., 6., 6., 6., 6., 6., 6., 6.,
        4., 4., 4., 4., 5., 5., 5., 5., 6., 6., 6., 6., 6., 6., 6., 6.]])
```

In [50]:
```
b=np.array([1., 2., 3., 4., 5., 6.])
a[:, b.T > 0.5]
#   extract the columns of a where column vector v > 0.5
```

Out[50]:
```
array([[1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.]])
```

In [52]:
```
a[a < 0.5]=0
a
#   a with elements less than 0.5 zeroed out
```

Out[52]:
```
array([[1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.]])
```

In [53]:
```
a * (a > 0.5)
a
#   a with elements less than 0.5 zeroed out
```

Out[53]:
```
array([[1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.]])
```

In [55]:
```
b[:] = 3
b
```

Out[55]:
```
array([3., 3., 3., 3., 3., 3.])
```

In [57]:
```
b=a.copy()
b
```

Out[57]:
```
array([[1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.]])
```

In [59]:
```
b=a[1,:].copy()
b
```

Out[59]:
```
array([4., 5., 6., 4., 5., 6.])
```

In [61]:
```
b=a.flatten()
b
```

Out[61]:
```
array([1., 2., 3., 1., 2., 3., 4., 5., 6., 4., 5., 6., 1., 2., 3., 1.,
2.,
       3., 4., 5., 6., 4., 5., 6., 1., 2., 3., 1., 2., 3., 4., 5., 6.,
4.,
       5., 6., 1., 2., 3., 1., 2., 3., 4., 5., 6., 4., 5., 6.])
```

In [62]:
```
np.arange(1., 11.)
```

Out[62]:
```
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
```

In [63]:
```
np.arange(10.)
```

Out[63]:
```
array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
```

In [64]:
```
np.arange(1.,11.)[:, np.newaxis]
```

Out[64]:
```
array([[ 1.],
       [ 2.],
       [ 3.],
       [ 4.],
       [ 5.],
       [ 6.],
       [ 7.],
       [ 8.],
       [ 9.],
       [10.]])
```

In [65]:
```
np.zeros((3, 4))
```

Out[65]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

In [66]:
```
np.zeros((3, 4, 5))
```

Out[66]:
```
array([[[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]],

       [[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]],

       [[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]]])
```

In [67]:
```
np.ones((3, 4))
```

Out[67]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

In [68]:
```
np.eye(3)
```

Out[68]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [69]:
```
np.diag(a)
```

Out[69]:
```
array([1., 5., 3., 4., 2., 6.])
```

In [73]:
```
#returns a square diagonal matrix whose nonzero values are the elements of
np.diag(a, 0)
```

Out[73]:
```
array([1., 5., 3., 4., 2., 6.])
```

In [77]:
```
from numpy.random import default_rng#random number generator
rng = default_rng(42)
rng.random((3,4))
```

Out[77]:
```
array([[0.77395605, 0.43887844, 0.85859792, 0.69736803],
       [0.09417735, 0.97562235, 0.7611397 , 0.78606431],
       [0.12811363, 0.45038594, 0.37079802, 0.92676499]])
```

In [78]:
```
np.linspace(1,3,4)
```

Out[78]:
```
array([1.        , 1.66666667, 2.33333333, 3.        ])
```

In [79]:
```
np.mgrid[0:9.,0:6.]
```

Out[79]:
```
array([[[0., 0., 0., 0., 0., 0.],
        [1., 1., 1., 1., 1., 1.],
        [2., 2., 2., 2., 2., 2.],
        [3., 3., 3., 3., 3., 3.],
        [4., 4., 4., 4., 4., 4.],
        [5., 5., 5., 5., 5., 5.],
        [6., 6., 6., 6., 6., 6.],
        [7., 7., 7., 7., 7., 7.],
        [8., 8., 8., 8., 8., 8.]],

       [[0., 1., 2., 3., 4., 5.],
        [0., 1., 2., 3., 4., 5.],
        [0., 1., 2., 3., 4., 5.],
        [0., 1., 2., 3., 4., 5.],
        [0., 1., 2., 3., 4., 5.],
        [0., 1., 2., 3., 4., 5.],
        [0., 1., 2., 3., 4., 5.],
        [0., 1., 2., 3., 4., 5.],
        [0., 1., 2., 3., 4., 5.]]])
```

In [85]:
```
c=np.ix_(np.r_[0:9.],np.r_[0:6.])
c
```

Out[85]:
```
(array([[0.],
        [1.],
        [2.],
        [3.],
        [4.],
        [5.],
        [6.],
        [7.],
        [8.]]),
 array([[0., 1., 2., 3., 4., 5.]]))
```

In [86]:
```
d=np.meshgrid([1,2,4],[2,4,5])
d
```

Out[86]:
```
[array([[1, 2, 4],
        [1, 2, 4],
        [1, 2, 4]]),
 array([[2, 2, 2],
        [4, 4, 4],
        [5, 5, 5]])]
```

In [88]:
```
np.tile(a, (2, 2))
```

Out[88]:
```
array([[1., 2., 3., 1., 2., 3., 1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6., 4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3., 1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6., 4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3., 1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6., 4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3., 1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6., 4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3., 1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6., 4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3., 1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6., 4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3., 1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6., 4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3., 1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6., 4., 5., 6., 4., 5., 6.]])
```

In [90]:
```
np.c_[a,a]
```

Out[90]:
```
array([[1., 2., 3., 1., 2., 3., 1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6., 4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3., 1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6., 4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3., 1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6., 4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3., 1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6., 4., 5., 6., 4., 5., 6.]])
```

In [91]:
```
np.r_[a,a]
```

Out[91]:
```
array([[1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.]])
```

In [92]:
```
a.max()
```

Out[92]:
```
6.0
```

In [93]:
```
a.max(0)    # 0th dimension
```

Out[93]:
```
array([4., 5., 6., 4., 5., 6.])
```

In [94]:
```
a.max(1)# 1st dimension
```

Out[94]:
```
array([3., 6., 3., 6., 3., 6., 3., 6.])
```

In [96]:
```
np.maximum(a, a)
```

Out[96]:
```
array([[1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.]])
```

In [97]:
```
np.linalg.norm(a)
#L2 norm of vector v
```

Out[97]:
```
26.981475126464083
```

In [99]:
```
np.logical_and(a,a)
```

Out[99]:
```
array([[ True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True]])
```

In [100]:
```
np.logical_or(a,a)
```

Out[100]:
```
array([[ True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True]])
```

In [105]:
```
4&5
```

Out[105]:
```
4
```

In [106]:
```
4|5
```

Out[106]:
```
5
```

In [108]:
```
linalg.inv( [[1., 2.], [3., 4.]] )
```

Out[108]:
```
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

In [109]:
```
c=np.array([[1., 2.], [3., 4.]])
linalg.pinv(c)
# pinv(a)       pseudo-inverse of matrix a
# it is a generalization of the inverse matrix
```

Out[109]:
```
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

In [110]:
```
np.linalg.matrix_rank(a)
```

Out[110]:
```
2
```

In [111]:
```
linalg.solve(a, b)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[111], line 1
----> 1 linalg.solve(a, b)

File ~/anaconda3/lib/python3.11/site-packages/scipy/linalg/_basic.py:151, in solve(a, b, sym_pos, lower, overwrite_a, overwrite_b, check_finite, assume_a, transposed)
    148 overwrite_b = overwrite_b or _datacopied(b1, b)
    150 if a1.shape[0] != a1.shape[1]:
--> 151     raise ValueError('Input a needs to be a square matrix.')
    153 if n != b1.shape[0]:
    154     # Last chance to catch 1x1 scalar a and 1-D b arrays
    155     if not (n == 1 and b1.size != 0):

ValueError: Input a needs to be a square matrix.
```

In [112]:
```
linalg.lstsq(a, b)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[112], line 1
----> 1 linalg.lstsq(a, b)

File ~/anaconda3/lib/python3.11/site-packages/scipy/linalg/_basic.py:1165, in lstsq(a, b, cond, overwrite_a, overwrite_b, check_finite, lapack_driver)
   1163     nrhs = 1
   1164 if m != b1.shape[0]:
-> 1165     raise ValueError('Shape mismatch: a and b should have the same number'
   1166                      ' of rows ({} != {}).'.format(m, b1.shape[0]))
   1167 if m == 0 or n == 0:  # Zero-sized problem, confuses LAPACK
   1168     x = np.zeros((n,) + b1.shape[1:], dtype=np.common_type(a1, b1))

ValueError: Shape mismatch: a and b should have the same number of rows (8 != 48).
```

In [119]:
```
b=np.block([[c,c],[c,c]])
b
```

Out[119]:
```
array([[1., 2., 1., 2.],
       [3., 4., 3., 4.],
       [1., 2., 1., 2.],
       [3., 4., 3., 4.]])
```

In [121]:
```
b=np.block([[b,b],[b,b]])
```

In [123]:
```
linalg.lstsq(a,b)
```

Out[123]:
```
(array([[ 0.19444444, -0.05555556,  0.19444444, -0.05555556,  0.19444444,
         -0.05555556,  0.19444444, -0.05555556],
        [ 0.11111111,  0.11111111,  0.11111111,  0.11111111,  0.11111111,
          0.11111111,  0.11111111,  0.11111111],
        [ 0.02777778,  0.27777778,  0.02777778,  0.27777778,  0.02777778,
          0.27777778,  0.02777778,  0.27777778],
        [ 0.19444444, -0.05555556,  0.19444444, -0.05555556,  0.19444444,
         -0.05555556,  0.19444444, -0.05555556],
        [ 0.11111111,  0.11111111,  0.11111111,  0.11111111,  0.11111111,
          0.11111111,  0.11111111,  0.11111111],
        [ 0.02777778,  0.27777778,  0.02777778,  0.27777778,  0.02777778,
          0.27777778,  0.02777778,  0.27777778]]),
 array([], dtype=float64),
 2,
 array([2.68927756e+01, 2.18600544e+00, 7.27940913e-16, 6.90694916e-17,
        1.61724041e-32, 2.18683269e-34]))
```

In [124]:
```
#singular value decomposition of a
#奇异值分解
U, S, Vh = linalg.svd(a)
V = Vh.T
V
```

Out[124]:
```
array([[-0.30311344, -0.56990255, -0.02283233,  0.75803221, -0.08218663,
         0.03800718],
       [-0.40043946, -0.07946637, -0.47248736, -0.3192523 , -0.6470219 ,
         0.29921507],
       [-0.49776549,  0.41096981,  0.2631071 ,  0.11861028,  0.30251179,
         0.63916055],
       [-0.30311344, -0.56990255,  0.54904652, -0.52081164,  0.09360838,
        -0.04328916],
       [-0.40043946, -0.07946637, -0.57994104, -0.15518883,  0.62417841,
        -0.2886511 ],
       [-0.49776549,  0.41096981,  0.2631071 ,  0.11861028, -0.29109004,
        -0.64444253]])
```

In [131]:
```
#give me a symmetric positive definite matrix
d = np.array([[4,2,0],[2,4,2],[0,2,4]])
linalg.cholesky(d)
#a必须是实对称正定矩阵
```

Out[131]:
```
array([[2.        , 1.        , 0.        ],
       [0.        , 1.73205081, 1.15470054],
       [0.        , 0.        , 1.63299316]])
```

In [135]:
```
D,V = linalg.eig(d)
D,V
```

Out[135]:
```
(array([6.82842712+0.j, 4.        +0.j, 1.17157288+0.j]),
 array([[-5.00000000e-01,  7.07106781e-01,  5.00000000e-01],
        [-7.07106781e-01,  6.67337348e-16, -7.07106781e-01],
        [-5.00000000e-01, -7.07106781e-01,  5.00000000e-01]]))
```

In [137]:
```
D,V = linalg.eig(d,2*d)
D,V
```

Out[137]:
```
(array([0.5+0.j, 0.5+0.j, 0.5+0.j]),
 array([[-1.        , -0.22923716,  0.04614145],
        [-0.        , -0.87060913,  0.4596367 ],
        [-0.        ,  0.43530457,  0.88690759]]))
```

In [140]:
```
D,V = eigs(b, k=3)
D,V
```

Out[140]:
```
(array([ 2.14891253e+01+0.j,  3.20087767e-16+0.j, -1.48912529e+00+0.j]),
 array([[-0.20798678+0.j,  0.20500347+0.j,  0.41228242+0.j],
        [-0.45468835+0.j, -0.20229001+0.j, -0.28288373+0.j],
        [-0.20798678+0.j,  0.39268159+0.j,  0.41228242+0.j],
        [-0.45468835+0.j,  0.48383761+0.j, -0.28288373+0.j],
        [-0.20798678+0.j, -0.406418  +0.j,  0.41228242+0.j],
        [-0.45468835+0.j, -0.51982621+0.j, -0.28288373+0.j],
        [-0.20798678+0.j, -0.19126707+0.j,  0.41228242+0.j],
        [-0.45468835+0.j,  0.23827861+0.j, -0.28288373+0.j]]))
```

In [141]:
```
Q,R = linalg.qr(a)
Q,R
```

Out[141]:
```
(array([[-1.21267813e-01,  4.85071250e-01,  8.65406187e-01,
         -3.27434245e-02, -5.47583570e-17,  2.48784615e-19,
          1.02241991e-17, -2.16216926e-17],
        [-4.85071250e-01, -1.21267813e-01,  3.27434245e-02,
          8.65406187e-01, -2.23214369e-15,  1.83537270e-17,
         -1.89168450e-17,  6.88107471e-18],
        [-1.21267813e-01,  4.85071250e-01, -2.88468729e-01,
          1.09144748e-02,  8.16220608e-01, -2.12270080e-02,
          5.92553690e-18,  3.61835724e-18],
        [-4.85071250e-01, -1.21267813e-01, -1.09144748e-02,
         -2.88468729e-01,  2.12270080e-02,  8.16220608e-01,
         -6.73022559e-15, -2.54821775e-16],
        [-1.21267813e-01,  4.85071250e-01, -2.88468729e-01,
          1.09144748e-02, -4.08110304e-01,  1.06135040e-02,
         -7.06782556e-01, -2.14107157e-02],
        [-4.85071250e-01, -1.21267813e-01, -1.09144748e-02,
         -2.88468729e-01, -1.06135040e-02, -4.08110304e-01,
          2.14107157e-02, -7.06782556e-01],
        [-1.21267813e-01,  4.85071250e-01, -2.88468729e-01,
          1.09144748e-02, -4.08110304e-01,  1.06135040e-02,
          7.06782556e-01,  2.14107157e-02],
        [-4.85071250e-01, -1.21267813e-01, -1.09144748e-02,
         -2.88468729e-01, -1.06135040e-02, -4.08110304e-01,
         -2.14107157e-02,  7.06782556e-01]]),
 array([[-8.24621125e+00, -1.06715675e+01, -1.30969238e+01,
         -8.24621125e+00, -1.06715675e+01, -1.30969238e+01],
        [ 0.00000000e+00,  1.45521375e+00,  2.91042750e+00,
          2.23763068e-16,  1.45521375e+00,  2.91042750e+00],
        [ 0.00000000e+00,  0.00000000e+00,  9.15516323e-16,
          3.99211338e-16,  4.34616879e-17,  9.15516323e-16],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         -1.51045215e-17, -3.10560590e-16, -6.33005299e-32],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
          0.00000000e+00,  7.93466240e-31, -1.67497090e-31],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
          0.00000000e+00,  0.00000000e+00,  4.35600624e-33],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
          0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
          0.00000000e+00,  0.00000000e+00,  0.00000000e+00]]))
```

In [144]:
```
linalg.lu(a)
#P,L,U
```

Out[144]:
```
(array([[0., 1., 0., 0., 0., 0., 0., 0.],
        [1., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 1., 0., 0., 0., 0., 0.],
        [0., 0., 0., 1., 0., 0., 0., 0.],
        [0., 0., 0., 0., 1., 0., 0., 0.],
        [0., 0., 0., 0., 0., 1., 0., 0.],
        [0., 0., 0., 0., 0., 0., 1., 0.],
        [0., 0., 0., 0., 0., 0., 0., 1.]]),
 array([[1.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
        [0.25, 1.  , 0.  , 0.  , 0.  , 0.  ],
        [0.25, 1.  , 1.  , 0.  , 0.  , 0.  ],
        [1.  , 0.  , 0.  , 1.  , 0.  , 0.  ],
        [0.25, 1.  , 0.  , 0.  , 1.  , 0.  ],
        [1.  , 0.  , 0.  , 0.  , 0.  , 1.  ],
        [0.25, 1.  , 0.  , 0.  , 0.  , 0.  ],
        [1.  , 0.  , 0.  , 0.  , 0.  , 0.  ]]),
 array([[4.  , 5.  , 6.  , 4.  , 5.  , 6.  ],
        [0.  , 0.75, 1.5 , 0.  , 0.75, 1.5 ],
        [0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
        [0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
        [0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
        [0.  , 0.  , 0.  , 0.  , 0.  , 0.  ]]))
```

In [145]:
```
np.fft.fft(a)
```

Out[145]:
```
array([[12.+0.j        ,  0.+0.j        , -3.+1.73205081j,
         0.+0.j        , -3.-1.73205081j,  0.+0.j        ],
       [30.+0.j        ,  0.+0.j        , -3.+1.73205081j,
         0.+0.j        , -3.-1.73205081j,  0.+0.j        ],
       [12.+0.j        ,  0.+0.j        , -3.+1.73205081j,
         0.+0.j        , -3.-1.73205081j,  0.+0.j        ],
       [30.+0.j        ,  0.+0.j        , -3.+1.73205081j,
         0.+0.j        , -3.-1.73205081j,  0.+0.j        ],
       [12.+0.j        ,  0.+0.j        , -3.+1.73205081j,
         0.+0.j        , -3.-1.73205081j,  0.+0.j        ],
       [30.+0.j        ,  0.+0.j        , -3.+1.73205081j,
         0.+0.j        , -3.-1.73205081j,  0.+0.j        ],
       [12.+0.j        ,  0.+0.j        , -3.+1.73205081j,
         0.+0.j        , -3.-1.73205081j,  0.+0.j        ],
       [30.+0.j        ,  0.+0.j        , -3.+1.73205081j,
         0.+0.j        , -3.-1.73205081j,  0.+0.j        ]])
```

In [146]:
```
np.fft.ifft(a)
```

Out[146]:
```
array([[ 2. +0.j        ,  0. +0.j        , -0.5-0.28867513j,
         0. +0.j        , -0.5+0.28867513j,  0. +0.j        ],
       [ 5. +0.j        ,  0. +0.j        , -0.5-0.28867513j,
         0. +0.j        , -0.5+0.28867513j,  0. +0.j        ],
       [ 2. +0.j        ,  0. +0.j        , -0.5-0.28867513j,
         0. +0.j        , -0.5+0.28867513j,  0. +0.j        ],
       [ 5. +0.j        ,  0. +0.j        , -0.5-0.28867513j,
         0. +0.j        , -0.5+0.28867513j,  0. +0.j        ],
       [ 2. +0.j        ,  0. +0.j        , -0.5-0.28867513j,
         0. +0.j        , -0.5+0.28867513j,  0. +0.j        ],
       [ 5. +0.j        ,  0. +0.j        , -0.5-0.28867513j,
         0. +0.j        , -0.5+0.28867513j,  0. +0.j        ],
       [ 2. +0.j        ,  0. +0.j        , -0.5-0.28867513j,
         0. +0.j        , -0.5+0.28867513j,  0. +0.j        ],
       [ 5. +0.j        ,  0. +0.j        , -0.5-0.28867513j,
         0. +0.j        , -0.5+0.28867513j,  0. +0.j        ]])
```

In [147]:
```
np.sort(a)
```

Out[147]:
```
array([[1., 1., 2., 2., 3., 3.],
       [4., 4., 5., 5., 6., 6.],
       [1., 1., 2., 2., 3., 3.],
       [4., 4., 5., 5., 6., 6.],
       [1., 1., 2., 2., 3., 3.],
       [4., 4., 5., 5., 6., 6.],
       [1., 1., 2., 2., 3., 3.],
       [4., 4., 5., 5., 6., 6.]])
```

In [148]:
```
np.sort(a, axis=1)
```

Out[148]:
```
array([[1., 1., 2., 2., 3., 3.],
       [4., 4., 5., 5., 6., 6.],
       [1., 1., 2., 2., 3., 3.],
       [4., 4., 5., 5., 6., 6.],
       [1., 1., 2., 2., 3., 3.],
       [4., 4., 5., 5., 6., 6.],
       [1., 1., 2., 2., 3., 3.],
       [4., 4., 5., 5., 6., 6.]])
```

In [150]:
```
I = np.argsort(a[:, 0]); b = a[I,:]
b
```

Out[150]:
```
array([[1., 2., 3., 1., 2., 3.],
       [1., 2., 3., 1., 2., 3.],
       [1., 2., 3., 1., 2., 3.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [4., 5., 6., 4., 5., 6.],
       [4., 5., 6., 4., 5., 6.],
       [4., 5., 6., 4., 5., 6.]])
```

In [152]:
```
e = linalg.lstsq(a, b)
e
```

Out[152]:
```
(array([[-6.25000000e-01, -8.75000000e-01, -1.12500000e+00,
         -6.25000000e-01, -8.75000000e-01, -1.12500000e+00],
        [ 4.99600361e-16,  5.55111512e-16,  5.55111512e-16,
          4.99600361e-16,  5.55111512e-16,  5.55111512e-16],
        [ 6.25000000e-01,  8.75000000e-01,  1.12500000e+00,
          6.25000000e-01,  8.75000000e-01,  1.12500000e+00],
        [-6.25000000e-01, -8.75000000e-01, -1.12500000e+00,
         -6.25000000e-01, -8.75000000e-01, -1.12500000e+00],
        [-5.27595974e-17, -1.57997401e-16, -2.01652255e-16,
         -5.27595974e-17, -1.57997401e-16, -2.01652255e-16],
        [ 6.25000000e-01,  8.75000000e-01,  1.12500000e+00,
          6.25000000e-01,  8.75000000e-01,  1.12500000e+00]]),
 array([], dtype=float64),
 2,
 array([2.68927756e+01, 2.18600544e+00, 7.27940913e-16, 6.90694916e-17,
        1.61724041e-32, 2.18683269e-34]))
```

In [154]:
```
signal.resample(b, np.ceil(len(b)/3))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[154], line 1
----> 1 signal.resample(b, np.ceil(len(b)/3))

File ~/anaconda3/lib/python3.11/site-packages/scipy/signal/_signaltools.py:3145, in resample(x, num, t, axis, window, domain)
   3143 else:
   3144     newshape[axis] = num
-> 3145 Y = np.zeros(newshape, X.dtype)
   3147 # Copy positive frequency components (and Nyquist, if present)
   3148 N = min(num, Nx)

TypeError: 'numpy.float64' object cannot be interpreted as an integer
```

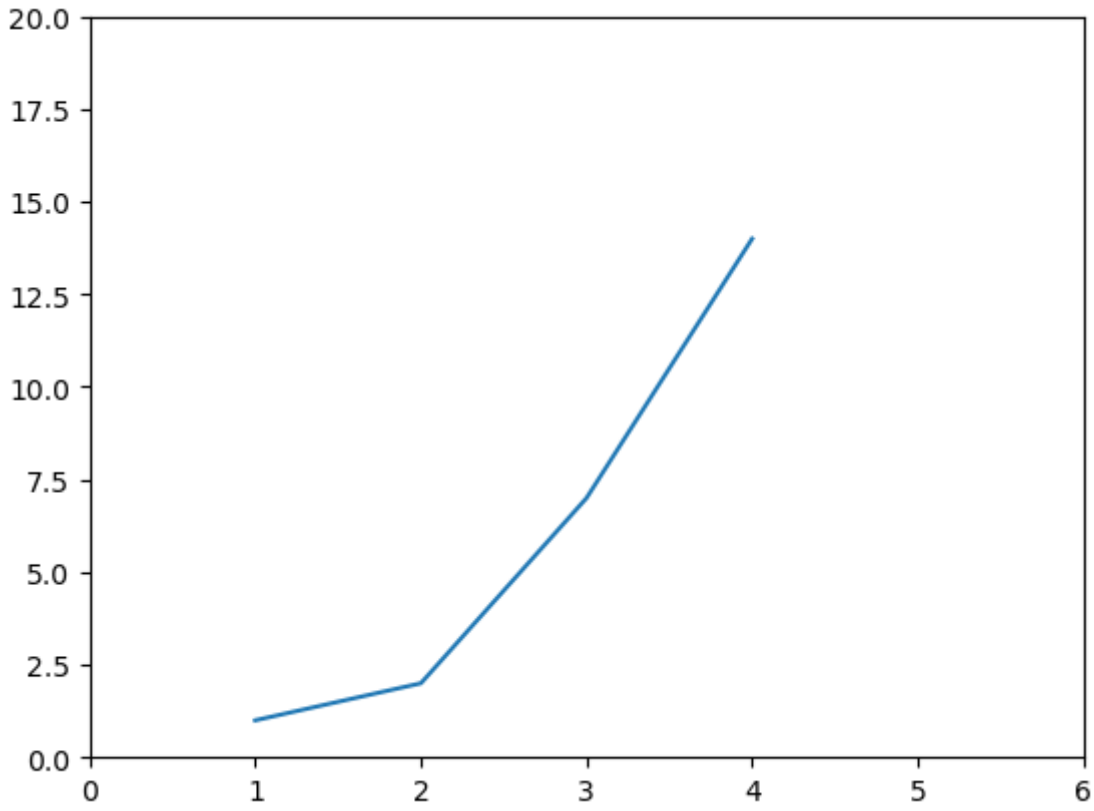
In [155]:
```
np.unique(a)
```

Out[155]:
```
array([1., 2., 3., 4., 5., 6.])
```

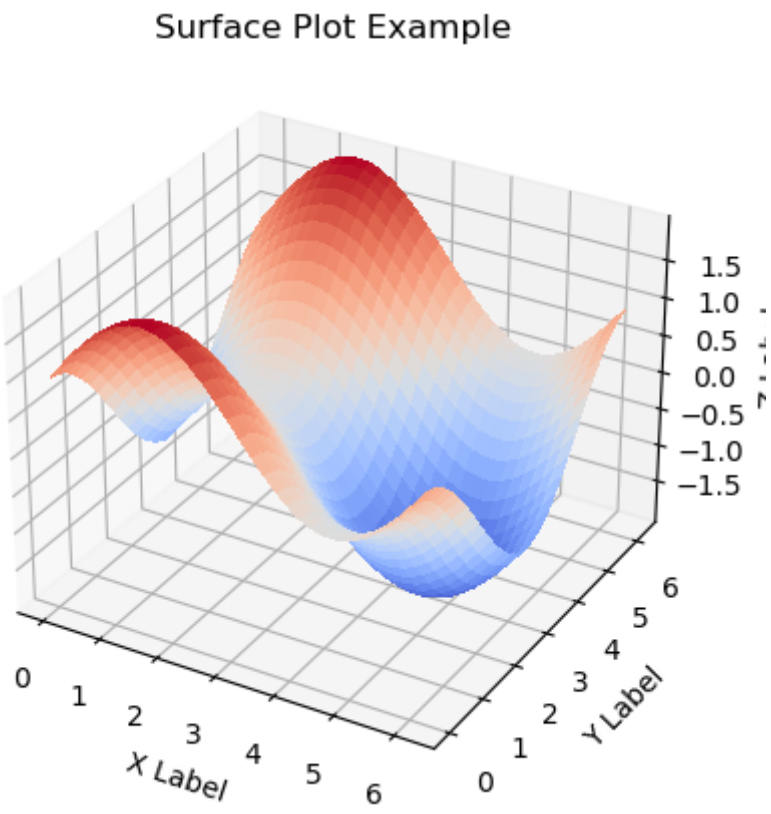
In [156]:
```
a.squeeze()
```

Out[156]:
```
array([[1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.],
       [1., 2., 3., 1., 2., 3.],
       [4., 5., 6., 4., 5., 6.]])
```

In [157]:
```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4], [1,2,7,14])
plt.axis([0, 6, 0, 20])
plt.show()
```



In [161]:
```
import matplotlib.pyplot as plt
from matplotlib import cm
import numpy as np

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# generate some X, Y data points
X, Y = np.meshgrid(np.arange(0, 2*np.pi, 0.2), np.arange(0, 2*np.pi, 0.2))
Z = np.sin(X) + np.cos(Y)


surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)

#set labels and title
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')
ax.set_title('Surface Plot Example')


plt.show()
ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.jet)
plt.show()
```



In [162]:
```
#github account https://github.com/FitzFitzFitz
#repository:https://github.com/FitzFitzFitz/ELEC576
#jupyter nbconvert notebook.ipynb --to pdf
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[162], line 3
      1 #github account https://github.com/FitzFitzFitz (https://github.com/FitzFitzFitz)
      2 #repository:https://github.com/FitzFitzFitz/ELEC576
----> 3 c.NotebookApp.export_output_types = {'pdf': 'nbconvert'}

AttributeError: 'numpy.ndarray' object has no attribute 'NotebookApp'
```

In [ ]: