COMP 576 Homework 0

Yinsen Miao

September 5, 2018

0.1 Task 1

```
In [1]: !conda info
```

```
     active environment : None
       user config file : C:\Users\yinse\.condarc
 populated config files :
          conda version : 4.5.4
    conda-build version : 3.10.5
         python version : 2.7.15.final.0
       base environment : C:\ProgramData\Anaconda2  (read only)
           channel URLs : https://repo.anaconda.com/pkgs/main/win-64
                          https://repo.anaconda.com/pkgs/main/noarch
                          https://repo.anaconda.com/pkgs/free/win-64
                          https://repo.anaconda.com/pkgs/free/noarch
                          https://repo.anaconda.com/pkgs/r/win-64
                          https://repo.anaconda.com/pkgs/r/noarch
                          https://repo.anaconda.com/pkgs/pro/win-64
                          https://repo.anaconda.com/pkgs/pro/noarch
                          https://repo.anaconda.com/pkgs/msys2/win-64
                          https://repo.anaconda.com/pkgs/msys2/noarch
          package cache : C:\ProgramData\Anaconda2\pkgs
                          C:\Users\yinse\AppData\Local\conda\conda\pkgs
       envs directories : C:\Users\yinse\AppData\Local\conda\conda\envs
                          C:\ProgramData\Anaconda2\envs
                          C:\Users\yinse\.conda\envs
               platform : win-64
             user-agent : conda/4.5.4 requests/2.18.4 CPython/2.7.15 Windows/10 Windows/10.0.162
          administrator : False
             netrc file : None
           offline mode : False
```

0.2 Task 2

```
In [2]: import numpy as np
        import scipy.linalg
```

```
        # let us sample a array 5 by 5 a from a standard normal distribution
        np.random.seed(2018)
        a = np.random.normal(0, 1, (5,5))
```

```
In [3]: a.ndim
```

```
Out[3]: 2
```

```
In [4]: a.size
```

```
Out[4]: 25
```

```
In [5]: a.shape
```

```
Out[5]: (5L, 5L)
```

```
In [6]: "nrows: %02d, ncols: %02d"%(a.shape[0], a.shape[1])
```

```
Out[6]: 'nrows: 05, ncols: 05'
```

```
In [7]: # samole a 2 by 2 matrix b and construct a block matrix
        b = np.random.normal(0, 1, (2, 2))
        c = np.block([[b, b], [b, b]])  # construct a block matrix
        print(c)
```

```
[[ 0.06371017  0.37062839  0.06371017  0.37062839]
 [-1.60454294 -2.16572937 -1.60454294 -2.16572937]
 [ 0.06371017  0.37062839  0.06371017  0.37062839]
 [-1.60454294 -2.16572937 -1.60454294 -2.16572937]]
```

```
In [8]: a[-1] # access the last element of matrix a
```

```
Out[8]: array([ 0.28916512,  1.28273322, -1.0656958 , -1.70663287, -0.17279739])
```

```
In [9]: a[1, 4] # access element of the second row and the fifth column
```

```
Out[9]: 0.4335640815380337
```

```
In [10]: a[1]     # acess the entire second row
```

```
Out[10]: array([ 0.8560293 , -0.14279008,  0.11007867, -0.68806479,  0.43356408])
```

```
In [11]: a[0:5]  # access the first five rows of a
```

```
Out[11]: array([[-0.2767676 ,  0.581851  ,  2.14839926, -1.279487  ,  0.50227689],
                [ 0.8560293 , -0.14279008,  0.11007867, -0.68806479,  0.43356408],
                [ 0.510221  , -0.16513097, -1.35177905,  0.54663075,  1.23065512],
                [ 1.0764461 , -1.21062488, -0.30667657, -1.05741884,  0.40205692],
                [ 0.28916512,  1.28273322, -1.0656958 , -1.70663287, -0.17279739]])
```

```
In [12]: a[-5:]  # access the last five rows of a

Out[12]: array([[-0.2767676 ,  0.581851  ,  2.14839926, -1.279487  ,  0.50227689],
                [ 0.8560293 , -0.14279008,  0.11007867, -0.68806479,  0.43356408],
                [ 0.510221  , -0.16513097, -1.35177905,  0.54663075,  1.23065512],
                [ 1.0764461 , -1.21062488, -0.30667657, -1.05741884,  0.40205692],
                [ 0.28916512,  1.28273322, -1.0656958 , -1.70663287, -0.17279739]])

In [13]: # read-only access
         a[0:3][:, 1:4]  # access rows from 1 to 3 and column from 2 to 4

Out[13]: array([[ 0.581851  ,  2.14839926, -1.279487  ],
                [-0.14279008,  0.11007867, -0.68806479],
                [-0.16513097, -1.35177905,  0.54663075]])

In [14]: a[np.ix_([1, 3, 4], [0, 2])]

Out[14]: array([[ 0.8560293 ,  0.11007867],
                [ 1.0764461 , -0.30667657],
                [ 0.28916512, -1.0656958 ]])

In [15]: # access ever other row of a starting from the third row
         # and going to the twenty-first
         a[2:21:2, :]

Out[15]: array([[ 0.510221  , -0.16513097, -1.35177905,  0.54663075,  1.23065512],
                [ 0.28916512,  1.28273322, -1.0656958 , -1.70663287, -0.17279739]])

In [16]: # access every other row of a starting from the first row
         a[::2, :]

Out[16]: array([[-0.2767676 ,  0.581851  ,  2.14839926, -1.279487  ,  0.50227689],
                [ 0.510221  , -0.16513097, -1.35177905,  0.54663075,  1.23065512],
                [ 0.28916512,  1.28273322, -1.0656958 , -1.70663287, -0.17279739]])

In [17]: # a with the rows in the reverse order
         a[::-1, :]

Out[17]: array([[ 0.28916512,  1.28273322, -1.0656958 , -1.70663287, -0.17279739],
                [ 1.0764461 , -1.21062488, -0.30667657, -1.05741884,  0.40205692],
                [ 0.510221  , -0.16513097, -1.35177905,  0.54663075,  1.23065512],
                [ 0.8560293 , -0.14279008,  0.11007867, -0.68806479,  0.43356408],
                [-0.2767676 ,  0.581851  ,  2.14839926, -1.279487  ,  0.50227689]])

In [18]: # append the firs row of array a to the last row of a
         a[np.r_[:len(a), 0]]

Out[18]: array([[-0.2767676 ,  0.581851  ,  2.14839926, -1.279487  ,  0.50227689],
                [ 0.8560293 , -0.14279008,  0.11007867, -0.68806479,  0.43356408],
                [ 0.510221  , -0.16513097, -1.35177905,  0.54663075,  1.23065512],
                [ 1.0764461 , -1.21062488, -0.30667657, -1.05741884,  0.40205692],
                [ 0.28916512,  1.28273322, -1.0656958 , -1.70663287, -0.17279739],
                [-0.2767676 ,  0.581851  ,  2.14839926, -1.279487  ,  0.50227689]])
```

In [19]:
```
# transpose of a
a.transpose()
```

Out[19]:
```
array([[-0.2767676 ,  0.8560293 ,  0.510221  ,  1.0764461 ,  0.28916512],
       [ 0.581851  , -0.14279008, -0.16513097, -1.21062488,  1.28273322],
       [ 2.14839926,  0.11007867, -1.35177905, -0.30667657, -1.0656958 ],
       [-1.279487  , -0.68806479,  0.54663075, -1.05741884, -1.70663287],
       [ 0.50227689,  0.43356408,  1.23065512,  0.40205692, -0.17279739]])
```

In [20]:
```
# conjugate transpose of a
a.conj().T
```

Out[20]:
```
array([[-0.2767676 ,  0.8560293 ,  0.510221  ,  1.0764461 ,  0.28916512],
       [ 0.581851  , -0.14279008, -0.16513097, -1.21062488,  1.28273322],
       [ 2.14839926,  0.11007867, -1.35177905, -0.30667657, -1.0656958 ],
       [-1.279487  , -0.68806479,  0.54663075, -1.05741884, -1.70663287],
       [ 0.50227689,  0.43356408,  1.23065512,  0.40205692, -0.17279739]])
```

In [21]:
```
# matrix multiply
a.dot(a)
```

Out[21]:
```
array([[ 0.43878239,  1.59437874, -3.57760471,  1.62390187,  2.15597554],
       [-0.91828227,  1.88942852,  1.42353846, -0.94921919,  0.151964  ],
       [-0.02799457,  1.46050968,  1.42614478, -3.95641846, -1.47177377],
       [-2.51272214,  2.64570759,  2.48974622, -0.27997824, -0.8562395 ],
       [-1.41277998,  2.00550808,  2.91056272,  0.26439878, -1.2664206 ]])
```

In [22]:
```
# element-wise multiply
a * a
```

Out[22]:
```
array([[0.0766003 , 0.33855059, 4.6156194 , 1.63708699, 0.25228207],
       [0.73278615, 0.02038901, 0.01211731, 0.47343316, 0.18797781],
       [0.26032547, 0.02726824, 1.82730659, 0.29880518, 1.51451203],
       [1.15873622, 1.4656126 , 0.09405052, 1.1181346 , 0.16164977],
       [0.08361647, 1.64540452, 1.13570754, 2.91259575, 0.02985894]])
```

In [23]:
```
# element-wise division
a / a
```

Out[23]:
```
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

In [24]:
```
# element-wise exponentiation
a ** 3
```

```
Out[24]: array([[-2.12004815e-02,  1.96985999e-01,  9.91619332e+00,
                 -2.09463153e+00,  1.26715455e-01],
                [ 6.27286416e-01, -2.91134765e-03,  1.33385761e-03,
                 -3.25752686e-01,  8.15004278e-02],
                [ 1.32823523e-01, -4.50283081e-03, -2.47011475e+00,
                  1.63336098e-01,  1.86384198e+00],
                [ 1.24731709e+00, -1.77430708e+00, -2.88430902e-02,
                 -1.18233660e+00,  6.49924080e-02],
                [ 2.41789660e-02,  2.11061504e+00, -1.21031876e+00,
                 -4.97073165e+00, -5.15954680e-03]])
```

```
In [25]: # element-wise comparison
         (a > 0.5)
```

```
Out[25]: array([[False,  True,  True, False,  True],
                [ True, False, False, False, False],
                [ True, False, False,  True,  True],
                [ True, False, False, False, False],
                [False,  True, False, False, False]])
```

```
In [26]: # find the index where (a > 0.5)
         np.nonzero(a > 0.5)
```

```
Out[26]: (array([0, 0, 0, 1, 2, 2, 2, 3, 4], dtype=int64),
          array([1, 2, 4, 0, 0, 3, 4, 0, 1], dtype=int64))
```

```
In [27]: # extract the columns of a where vector v > 0.5
         np.random.seed(2018)
         v = np.random.normal(0, 1, (5))
         v
```

```
Out[27]: array([-0.2767676 ,  0.581851  ,  2.14839926, -1.279487  ,  0.50227689])
```

```
In [28]: a[:, np.nonzero(v > 0.5)[0]]
```

```
Out[28]: array([[ 0.581851  ,  2.14839926,  0.50227689],
                [-0.14279008,  0.11007867,  0.43356408],
                [-0.16513097, -1.35177905,  1.23065512],
                [-1.21062488, -0.30667657,  0.40205692],
                [ 1.28273322, -1.0656958 , -0.17279739]])
```

```
In [29]: a[:, v.T > 0.5]
```

```
Out[29]: array([[ 0.581851  ,  2.14839926,  0.50227689],
                [-0.14279008,  0.11007867,  0.43356408],
                [-0.16513097, -1.35177905,  1.23065512],
                [-1.21062488, -0.30667657,  0.40205692],
                [ 1.28273322, -1.0656958 , -0.17279739]])
```

```
In [30]: # zero out element of a less than 0.5
         a[a < 0.5] = 0
         a

Out[30]: array([[0.        , 0.581851  , 2.14839926, 0.        , 0.50227689],
                [0.8560293 , 0.        , 0.        , 0.        , 0.        ],
                [0.510221  , 0.        , 0.        , 0.54663075, 1.23065512],
                [1.0764461 , 0.        , 0.        , 0.        , 0.        ],
                [0.        , 1.28273322, 0.        , 0.        , 0.        ]])

In [31]: # zero out element of a less than 0.5
         a * (a > 0.5)

Out[31]: array([[0.        , 0.581851  , 2.14839926, 0.        , 0.50227689],
                [0.8560293 , 0.        , 0.        , 0.        , 0.        ],
                [0.510221  , 0.        , 0.        , 0.54663075, 1.23065512],
                [1.0764461 , 0.        , 0.        , 0.        , 0.        ],
                [0.        , 1.28273322, 0.        , 0.        , 0.        ]])

In [32]: # set a to the same scalar value 3
         a[:] = 3
         a

Out[32]: array([[3., 3., 3., 3., 3.],
                [3., 3., 3., 3., 3.],
                [3., 3., 3., 3., 3.],
                [3., 3., 3., 3., 3.],
                [3., 3., 3., 3., 3.]])

In [33]: # numpy assign by reference
         np.random.seed(2018)
         x = np.random.normal(0, 1, (2, 2))
         y = x.copy()
         y

Out[33]: array([[-0.2767676 ,  0.581851  ],
                [ 2.14839926, -1.279487  ]])

In [34]: # numpy slices by reference
         y = x[1,:].copy()
         y

Out[34]: array([ 2.14839926, -1.279487  ])

In [35]: # turn array to a vector and this operation forces a copy
         y = x.flatten()
         y

Out[35]: array([-0.2767676 ,  0.581851  ,  2.14839926, -1.279487  ])
```

```
In [36]: # create an increasing vector
         np.r_[1:11.]

Out[36]: array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])

In [37]: # create an increasing vector
         np.arange(1., 11.)

Out[37]: array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])

In [38]: # create an increasing column vector
         np.arange(1., 11.)[:, np.newaxis]

Out[38]: array([[ 1.],
                [ 2.],
                [ 3.],
                [ 4.],
                [ 5.],
                [ 6.],
                [ 7.],
                [ 8.],
                [ 9.],
                [10.]])

In [39]: # create two dimensional zero arrays
         np.zeros((3, 4))

Out[39]: array([[0., 0., 0., 0.],
                [0., 0., 0., 0.],
                [0., 0., 0., 0.]])

In [40]: # create three dimensional zero arrays
         np.zeros((3, 4, 5))

Out[40]: 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 [41]: # 3 by 3 identity matrix
         np.eye(3)
```

```
Out[41]: array([[1., 0., 0.],
                [0., 1., 0.],
                [0., 0., 1.]])
```

```
In [42]: # diagnal of matrix a
         np.diag(a)
```

```
Out[42]: array([3., 3., 3., 3., 3.])
```

```
In [43]: # main diagnal of matrix a
         np.diag(a, 0)
```

```
Out[43]: array([3., 3., 3., 3., 3.])
```

```
In [44]: # create a random matrix of 3 by 4
         np.random.rand(3, 4)
```

```
Out[44]: array([[0.8371111 , 0.69780061, 0.80280284, 0.10721508],
                [0.75709253, 0.99967101, 0.725931  , 0.14144824],
                [0.3567206 , 0.94270411, 0.61016189, 0.22757747]])
```

```
In [45]: # equally spaced elements between 1 and 3 inclusive
         np.linspace(1, 3, 4)
```

```
Out[45]: array([1.        , 1.66666667, 2.33333333, 3.        ])
```

```
In [46]: # create two 2D arrays: one of x values and the other of y values
         np.mgrid[0:9.0, 0:6.0]
```

```
Out[46]: 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 [47]: np.ogrid[0:9.0, 0:6.0]
```

```
Out[47]: [array([[0.],
                [1.],
                [2.],
                [3.],
                [4.],
                [5.],
                [6.],
                [7.],
                [8.]]), array([[0., 1., 2., 3., 4., 5.]])]

In [48]: np.meshgrid([1,2,4], [2,4,5])

Out[48]: [array([[1, 2, 4],
                [1, 2, 4],
                [1, 2, 4]]), array([[2, 2, 2],
                [4, 4, 4],
                [5, 5, 5]])]

In [49]: # evaluate a function on a grid
         np.ix_([1,2,4], [2,4,5])

Out[49]: (array([[1],
                [2],
                [4]]), array([[2, 4, 5]]))

In [50]: # create a 2 by 2 copies of x
         np.tile(x, (2, 2))

Out[50]: array([[-0.2767676 ,  0.581851  , -0.2767676 ,  0.581851  ],
               [ 2.14839926, -1.279487  ,  2.14839926, -1.279487  ],
               [-0.2767676 ,  0.581851  , -0.2767676 ,  0.581851  ],
               [ 2.14839926, -1.279487  ,  2.14839926, -1.279487  ]])

In [51]: # concatenate the columns of x and x
         np.concatenate((x, x), 1)

Out[51]: array([[-0.2767676 ,  0.581851  , -0.2767676 ,  0.581851  ],
               [ 2.14839926, -1.279487  ,  2.14839926, -1.279487  ]])

In [52]: # concatenate the rows of x and x
         np.concatenate((x, x), 0)

Out[52]: array([[-0.2767676 ,  0.581851  ],
               [ 2.14839926, -1.279487  ],
               [-0.2767676 ,  0.581851  ],
               [ 2.14839926, -1.279487  ]])

In [53]: # find maximum element of array x
         x.max()
```

Out[53]: 2.14839926388642

In [54]:
```
# maximum element of each column of matrix a
x.max(0)
```

Out[54]: array([2.14839926, 0.581851])

In [55]:
```
# maximum element of each row of matrix a
x.max(1)
```

Out[55]: array([0.581851 , 2.14839926])

In [56]:
```
# compare a and b element-wise and return the maximum value per pair
np.random.seed(2018)
b = np.random.normal(3, 2, (5, 5))
np.maximum(a, b)
```

Out[56]:
```
array([[3.        , 4.163702  , 7.29679853, 3.        , 4.00455378],
       [4.71205859, 3.        , 3.22015733, 3.        , 3.86712816],
       [4.02044201, 3.        , 3.        , 4.0932615 , 5.46131024],
       [5.15289221, 3.        , 3.        , 3.        , 3.80411384],
       [3.57833024, 5.56546645, 3.        , 3.        , 3.        ]])
```

In [57]:
```
# L2 norm of vector v
np.linalg.norm(v)
```

Out[57]: 2.630615774983453

In [58]:
```
# element-wise logic and
np.logical_and(a, b)
```

Out[58]:
```
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]])
```

In [59]:
```
# element-wise logic or
np.logical_or(a, b)
```

Out[59]:
```
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]])
```

In [60]:
```
# bitwise logic and
5 & 5
```

Out[60]: 5

```
In [61]: # bitwise logic or
         5 | 12
```

```
Out[61]: 13
```

```
In [62]: # inverse of matrix
         np.linalg.inv(x)
```

```
Out[62]: array([[1.42811409, 0.64943967],
                [2.39796047, 0.30891733]])
```

```
In [63]: # pseudo-inverse of matrix
         np.linalg.pinv(x)
```

```
Out[63]: array([[1.42811409, 0.64943967],
                [2.39796047, 0.30891733]])
```

```
In [64]: # rank of a 2D matrix
         np.linalg.matrix_rank(x)
```

```
Out[64]: 2
```

```
In [65]: # least square solver
         np.random.seed(2018)
         a = np.random.normal(0, 1, (3, 3))
         b = np.random.normal(0, 1, (3, 1))
         np.linalg.solve(a, b)
```

```
Out[65]: array([[-0.37725662],
                [-0.38317169],
                [ 0.25698226]])
```

```
In [66]: # svd of a
         U, S, Vh = np.linalg.svd(a)
         U, S, Vh
```

```
Out[66]: (array([[-0.83687903,  0.41080513,  0.36176324],
                 [-0.50613326, -0.83243348, -0.22557398],
                 [ 0.20847689, -0.37187854,  0.90456826]]),
          array([2.62228763, 1.11300036, 0.20905204]),
          array([[ 0.32393224, -0.27388646, -0.90556839],
                 [ 0.9025072 , -0.19768243,  0.38262568],
                 [ 0.28381096,  0.94122679, -0.18314879]]))
```

```
In [67]: # upper Cholesky factor of matrix a
         np.linalg.cholesky(a + 2*np.eye(3)).T
```

```
Out[67]: array([[ 1.31271947, -0.97468426, -0.10877425],
                [ 0.        ,  1.24590027,  0.00325718],
                [ 0.        ,  0.        ,  1.1402161 ]])
```

In [68]:
```
# eigenvalues and eigenvectors of a
D, V = np.linalg.eig(a)
D, V
```

Out[68]:
```
(array([ 0.12854271+0.91176604j,  0.12854271-0.91176604j,
        -0.71964092+0.j        ]),
 array([[-0.27807545+0.54936403j, -0.27807545-0.54936403j,
         -0.5324129 +0.j        ],
        [-0.78441944+0.j        , -0.78441944-0.j        ,
         -0.78293363+0.j        ],
        [-0.07316288-0.01437202j, -0.07316288+0.01437202j,
          0.32179409+0.j        ]]))
```

In [69]:
```
# QR decomposition of matrix a
Q, R = scipy.linalg.qr(a)
Q, R
```

Out[69]:
```
(array([[-0.21017504,  0.97111016, -0.11301111],
        [-0.97163193, -0.22029868, -0.08602259],
        [-0.10843361,  0.09172541,  0.98986292]]),
 array([[ 1.31684331, -0.62225505, -1.20867595],
        [ 0.        ,  0.46448749,  1.8346372 ],
        [ 0.        ,  0.        , -0.99752068]]))
```

In [70]:
```
# LU decomposition of matrix a
P, L, U = scipy.linalg.lu(a)
P, L, U
```

Out[70]:
```
(array([[0., 1., 0.],
        [1., 0., 0.],
        [0., 0., 1.]]), array([[1.        , 0.        , 0.        ],
        [0.21631138, 1.        , 0.        ],
        [0.11159947, 0.11416845, 1.        ]]), array([[-1.279487  ,  0.50227689,  0.85
        [ 0.        ,  0.4732028 ,  1.96323039],
        [ 0.        ,  0.        , -1.00773618]]))
```

In [71]:
```
# sort the matrix by row
np.sort(x)
```

Out[71]:
```
array([[-0.2767676 ,  0.581851  ],
       [-1.279487  ,  2.14839926]])
```

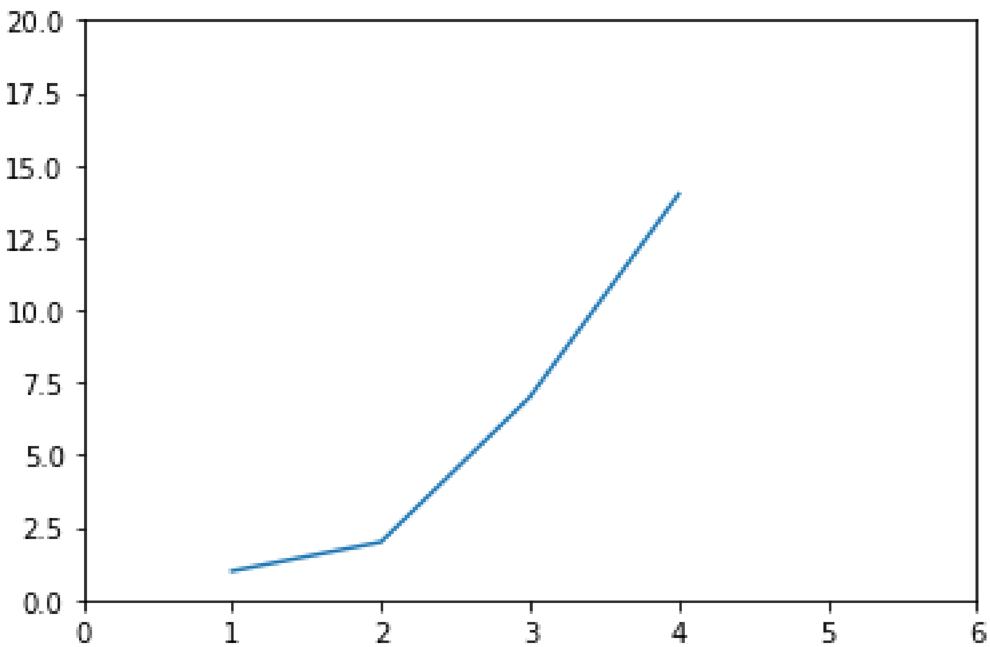
In [72]:
```
# multilinear regression
np.linalg.lstsq(a, b, rcond=None)
```

Out[72]:
```
(array([[-0.37725662],
        [-0.38317169],
        [ 0.25698226]]),
 array([], dtype=float64),
 3,
 array([2.62228763, 1.11300036, 0.20905204]))
```

```
In [73]: # downsample with low-pass filtering
         np.random.seed(2018)
         x = np.random.normal(0, 1, (1000, 1))
         #scipy.signal.resample(x, 20)

In [74]: # unique element in array a
         np.unique(a)

Out[74]: array([-1.279487  , -0.68806479, -0.2767676 , -0.14279008,  0.11007867,
                 0.50227689,  0.581851  ,  0.8560293 ,  2.14839926])

In [75]: # squeeze
         s = np.array([[1],[2],[3]])
         s

Out[75]: array([[1],
                [2],
                [3]])

In [76]: s.squeeze()

Out[76]: array([1, 2, 3])
```

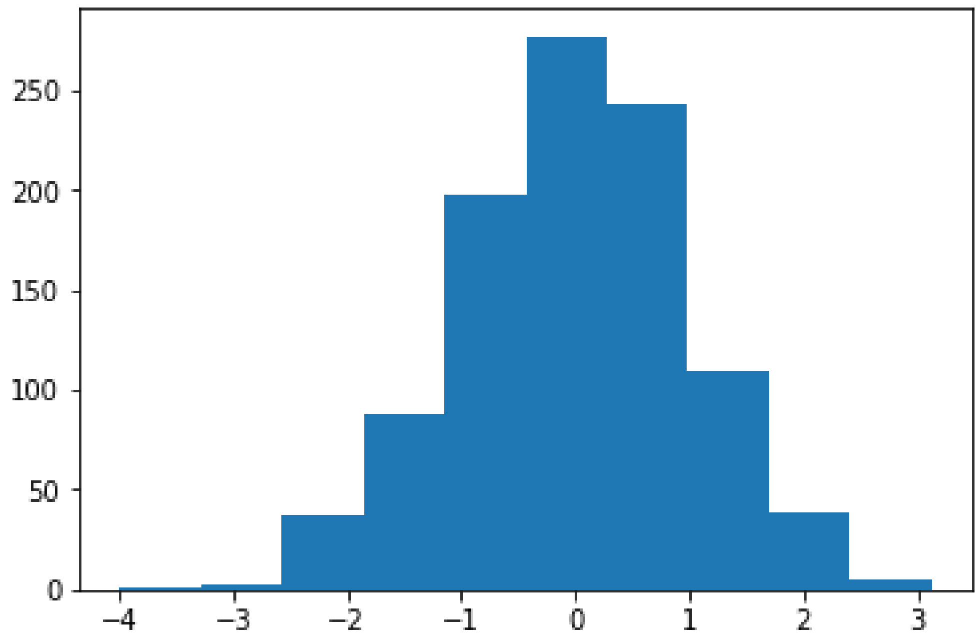
0.3 Task 3

```
In [77]: %matplotlib inline
         import matplotlib.pyplot as plt
         # plot task 3
         plt.plot([1,2,3,4], [1,2,7,14])
         plt.axis([0, 6, 0, 20])
         plt.show()
```



0.4 Task 4

```
In [78]: # plot the histogram of a normal distribution
         samples = np.random.normal(0, 1, 1000)
         plt.hist(samples)
         plt.show()
```



0.5 Task 5

My github account is **yinsenm**.

0.6 Task 6

The github link to my reposistory is https://github.com/yinsenm/COMP576 .