Izaan Mohtashim
P20-0613
Sec 5A
Numerical Computing

Assignment 2

Matrix Representation

Matrix Representation(Dense)

In [59]:
```
class Matrix:
    def __init__(self, dims,fill):
        self.rows=dims[0]
        self.cols=dims[1]
        self.A=[
            [fill]*self.cols
            for i in range(self.rows)
        ]
```

In [60]:
```
m=Matrix((3,4),2.0)
print(m)
```

```
<__main__.Matrix object at 0x0000012F2B30C130>
```

In [61]:
```
def __str__(self):
    rows=len(self.A)
    ret=''
    for i in range(rows):
        cols=len(self.A[i])
        for j in range(cols):
            ret+=str(self.A[i][j])+"\t"
        ret+="\n"
    return ret

Matrix.__str__=__str__
```

In [62]:
```
print (m)
```

In [62]:
```
print (m)
```

```
2.0     2.0     2.0     2.0
2.0     2.0     2.0     2.0
2.0     2.0     2.0     2.0
```

In [63]:
```
%time n=Matrix((100,100),0.0)
```

```
CPU times: total: 0 ns
Wall time: 0 ns
```

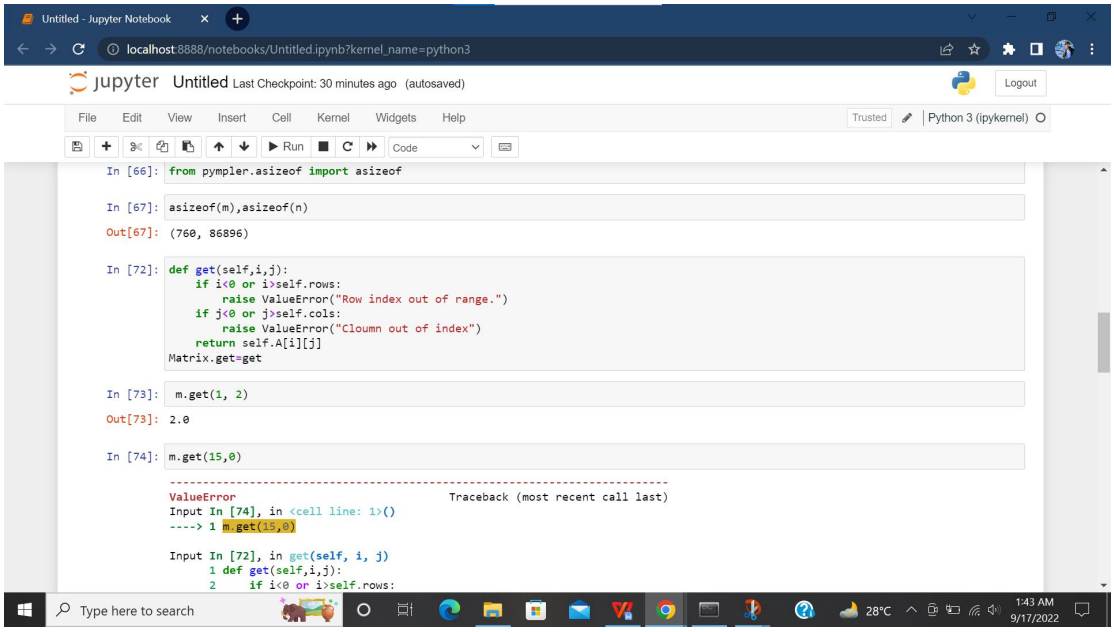
Memory usage

In [64]:
```
from sys import getsizeof
print(getsizeof(m))
print(getsizeof(n))
```

```
48
48
```

In [65]:
```
!pip install pympler
```

```
Requirement already satisfied: pympler in d:\new folder\lib\site-packages (1.0.1)
```




```
In [75]: m.get(1,10)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [75], in <cell line: 1>()
----> 1 m.get(1,10)

Input In [72], in get(self, i, j)
      3     raise ValueError("Row index out of range.")
      4 if j<0 or j>self.cols:
----> 5     raise ValueError("Cloumn out of index")
      6 return self.A[i][j]

ValueError: Cloumn out of index
```

Matrix Reprsentation(sparse)

```
In [76]: class Matrix:
             def __init__(self,dims):
                 self.rows=dims[0]
                 self.cols=dims[1]
                 self.vals={}
```

```
In [77]: def set(self,i,j, val):
             self.vals[(i,j)]=val
         Matrix.set=set
```

```
In [78]: def get(self,i,j):
             if i<0 or i>self.rows:
                 raise ValueError("Row index out of range.")
             if j<0 or j>self.cols:
                 raise ValueError("Cloumn out of index")

             if(i,j) in self.vals:
                 return self.vals[(i,j)]

             return 0.0
         Matrix.get=get
```

```
In [79]: m=Matrix((5,5))
```

```
In [80]: print(m.vals)
         {}
```

```
In [81]: m.get(1,1)
Out[81]: 0.0
```

In [77]:
```
def set(self,i,j, val):
    self.vals[(i,j)]=val
Matrix.set=set
```

In [78]:
```
def get(self,i,j):
    if i<0 or i>self.rows:
        raise ValueError("Row index out of range.")
    if j<0 or j>self.cols:
        raise ValueError("Cloumn out of index")

    if(i,j) in self.vals:
        return self.vals[(i,j)]

    return 0.0
Matrix.get=get
```

In [79]:
```
m=Matrix((5,5))
```

In [80]:
```
print(m.vals)
```

```
{}
```

In [81]:
```
m.get(1,1)
```

Out[81]:
```
0.0
```

NUMPY

NUMPY

In [87]:
```
import numpy as np
```

In [88]:
```
np.random.seed(1337)
```

In [89]:
```
x=np.array([1,4,3])
x
```

Out[89]:
```
array([1, 4, 3])
```

In [91]:
```
y=np.array([[1,4,3],
            [9,2,7]])
y
```

Out[91]:
```
array([[1, 4, 3],
       [9, 2, 7]])
```

In [98]:
```
x.shape
```

Out[98]:
```
(3,)
```

In [99]:
```
y.shape
```

Out[99]:
```
(2, 3)
```

In [101]:
```
z=np.array([[1,4,3]])
z.shape
```

Out[101]:
```
(1, 3)
```

In [107]:
```
z=np.arange(1,2000,1)
z[:10]
```

Out[107]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [109]:
```
np.arange(0.5,3,0.5).shape
```

Out[109]:
```
(5,)
```

In [110]:
```
np.arange(0.5,10,1).reshape(5,2).shape
```

Out[110]:
```
(5, 2)
```

In [112]:
```
np.linspace(3,9,10)
```

Out[112]:
```
array([3.        , 3.66666667, 4.33333333, 5.        , 5.66666667,
       6.33333333, 7.        , 7.66666667, 8.33333333, 9.        ])
```

In [113]:
```
print(x)
print(x[1])
print(x[1:])
```

```
[1 4 3]
4
[4 3]
```

In [115]:
```
print(y)
```

```
[[1 4 3]
 [9 2 7]]
```

Marrix Operation

In [116]:
```
np.zeros((3,5))
```

Out[116]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [117]:
```
np.ones((5,3))
```

Out[117]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

In [118]:
```
a=np.arange(1,7)
a
```

Out[118]:
```
array([1, 2, 3, 4, 5, 6])
```

In [119]:
```
a.shape
```

Out[119]:
```
(6,)
```

In [120]:
```
b=np.zeros((2,2))
```

In [121]:
```
b[0,0]=1
b[0,1]=2
b[1,1]=4
```

In [122]:
```
b
```

Out[122]:
```
array([[1., 2.],
       [0., 4.]])
```

In [123]:
```
b.shape
```

Out[123]:
```
(2, 2)
```

Array Operation

In [124]:
```
print(b)
```

```
[[1. 2.]
 [0. 4.]]
```

In [125]:
```
b+2
```

Out[125]:
```
array([[3., 4.],
       [2., 6.]])
```

```
In [126]: b**2

Out[126]: array([[ 1.,  4.],
                 [ 0., 16.]])

In [127]: sum(b)

Out[127]: array([1., 6.])

In [128]: b.sum()

Out[128]: 7.0

In [130]: b

Out[130]: array([[1., 2.],
                 [0., 4.]])

In [133]: b.sum(axis=0).shape

Out[133]: (2,)

In [132]: b.sum(axis=1).shape

Out[132]: (2,)
```

```
In [134]: b = np.array([[1, 2], [3, 4]])
          d = np.array([[3, 4], [5, 6]])

In [135]: print(b)
          print(d)

          [[1 2]
           [3 4]]
          [[3 4]
           [5 6]]

In [136]: b+d

Out[136]: array([[ 4,  6],
                 [ 8, 10]])

In [137]: b.T

Out[137]: array([[1, 3],
                 [2, 4]])

In [138]: a

Out[138]: array([1, 2, 3, 4, 5, 6])

In [139]: a.shape
```

```
In [140]: a.T.shape

Out[140]: (6,)

In [141]: print(np.sqrt(36))
          x=[1,4,9,16]
          np.sqrt(x)

          6.0

Out[141]: array([1., 2., 3., 4.])

In [142]: x = np.array([1, 2, 4, 5, 9, 3])
          y = np.array([0, 2, 3, 1, 2, 3])

In [144]: x>3

Out[144]: array([False, False,  True,  True,  True, False])

In [145]: x>y

Out[145]: array([ True, False,  True,  True,  True, False])
```

Mics Operation

In [146]:
```
import math
def basic_sigmoid(x):
    """
    Compute sigmoid of x.

    Arguments:
    x -- A scalar

    Return:
    s -- sigmoid(x)
    """

    s = 1./(1. + math.e ** (-x))

    return s
```

In [148]:
```
basic_sigmoid(-1)
```

Out[148]:
```
0.2689414213699951
```

In [149]:
```
basic_sigmoid(0)
```

Out[149]:
```
0.5
```

In [150]:
```
x=[-1,0,3]
```

In [151]:
```
basic_sigmoid(x)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [151], in <cell line: 1>()
----> 1 basic_sigmoid(x)

Input In [146], in basic_sigmoid(x)
      2 def basic_sigmoid(x):
      3     """
      4     Compute sigmoid of x.
      5
   (...)
     10     s -- sigmoid(x)
     11     """
---> 13     s = 1./(1. + math.e ** (-x))
     15     return s

TypeError: bad operand type for unary -: 'list'
```

In [154]:
```
import numpy as np
x=[-1,0,3]
x=np.array(x)
basic_sigmoid(x)
```

Out[154]:
```
array([0.26894142, 0.5       , 0.95257413])
```