Homework 1

Aasim Zahoor

September 10, 2020

1 Problems

Link `https://github.com/AasimZahoor/Comp_methods.git`

Problem 1

In the code for Problem 1 I have defined five functions. They are:

- **midpoint(f,a,b,n)**
- **trapezoid(f,a,b,n)**
- **simpson(f,a,b,n)**

  These functions have the same arguments and thet are f= 'the function being integrated', $[a, b]$='The range of integration', n='The number of steps'. All of them need to be inputted while calling the function.

- **diff(f,a,h=0.000001)**

  This function differentiates the function f at point a. h represents the step size and has a default value of 0.000001

- **g()**

  This is a test function and can be used to test these function. You would need to make changes in the code to test different function.

Testing the functions using sine function(the figure on the left), the results of the test(the figure on the right)

```
#testing
####
#test function
def g():
    def y(x):
        return(math.sin(x))
    return(y)
####
diff(g(),math.pi)
simpson(g(),0,math.pi,1000)
midpoint(g(),0,math.pi,1000)
trapezoid(g(),0,math.pi,1000)
```

```
   ...: #test function
   ...: def g():
   ...:     def y(x):
   ...:         return(math.sin(x))
   ...:     return(y)
   ...:
   ...: ####
   ...: diff(g(),math.pi)
   ...: simpson(g(),0,math.pi,1000)
   ...: midpoint(g(),0,math.pi,1000)
   ...: trapezoid(g(),0,math.pi,1000)
('The result of differentiation is', -1.0000000001396114)
('The value from simpson rule is', 2.000000000001107)
('The value from midpoint rule is', 2.000000822467294)
('The value from trapezoid rule is', 1.999998355065688)

In [396]:
```

Problem 2

Problem 3

In this problem I have made a matrix class in which I have defined eight instances.

- **init** It has self and the matrix array as the argument. Here I defined 3 instance attributes, self.g gives back the array and helps me handle the array in a better way.
- **add(self,other)** It takes self and the other(matrix) as the argument. I have included a check to see if addition is possible
- **mult(self,other)** It takes self and the other(matrix) as the argument. I have included a check to see if multiplication is possible
- **tran(self)**It takes self as argument and gives out Transpose as output.
- **trace(self)**It takes self as argument and gives out Trace as output.
- **Det(self)**It takes self as argument and gives out Determinant as output.

```

#test
x=[[1,2,3],[2,5,4],[3,15,5]]
y=[[4,5,6],[1,2,4],[8,9,10]]
m1=matrix(x)
m2=matrix(y)
m1.add(m2)
m1.mult(m2)
m1.tran()
m1.trace()
m1.Det()

```

```
   ...: #test
   ...: x=[[1,2,3],[2,5,4],[3,15,5]]
   ...: y=[[4,5,6],[1,2,4],[8,9,10]]
   ...: m1=matrix(x)
   ...: m2=matrix(y)
   ...: m1.add(m2)
   ...: m1.mult(m2)
   ...: m1.tran()
   ...: m1.trace()
   ...: m1.Det()
('the sum is', [[5, 7, 9], [3, 7, 8], [11, 24, 15]])
('The product is', [[30, 36, 44], [45, 56, 72], [67, 90, 128]])
('the transpose is', [[1, 2, 3], [2, 5, 15], [3, 4, 5]])
('The trace is', 11)
Out[414]: ('The determinant is', 14)

In [415]:
```

Testing the instances using a matrix(the figure on the left), the results of the test(the figure on the right)

- **LU(self)**Takes in self as argument and gives out lower and upper triangular matrix (in that order). In the figure I multiplies the L and U matrix and got the original matrix back.

Testing the LU Instance using a matrix

```
#test
x=[[1,2,3],[2,5,4],[3,15,5]]
#y=[[4,5,6],[1,2,4],[8,9,10]]
m1=matrix(x)
l,u=m1.LU()                    #LU returns l and u
m2=matrix(l)
m3=matrix(u)
#testing if their multiplication gives the original matrix
m2.mult(m3)
```

```
    ...:
    ...: #test
    ...: x=[[1,2,3],[2,5,4],[3,15,5]]
    ...: #y=[[4,5,6],[1,2,4],[8,9,10]]
    ...: m1=matrix(x)
    ...: l,u=m1.LU()                    #LU returns l and u
    ...: m2=matrix(l)
    ...: m3=matrix(u)
    ...: #testing if their multiplication gives the original
matrix
    ...: m2.mult(m3)
('lower Traingular matrix', [[1, 0, 0], [2, 1, 0], [3, 9, 1]])
('upper Triangular matrix', [[1, 2, 3], [0, 1, -2], [0, 0, 14]])
('The product is', [[1, 2, 3], [2, 5, 4], [3, 15, 5]])
Out[434]: [[1, 2, 3], [2, 5, 4], [3, 15, 5]]

In [435]:
```

Results of the test

- **Inv(self)** Takes self as the argument and gives out the inverse.

```
#test
o=[[2,3,4,11],[5,6,8,90],[9,10,20,40],[12,44,34,26]]
m1=matrix(o)
m1.Inv()


#test

```

Testing the Inv Instance using a matrix

```
    ....
    ...: #test
[[4.443021766965431, -0.2624839948783615, -0.5369184805804526,
-0.14511310285958173], [0.3956466069142124, -0.0032010243277848793,
-0.11630388390951772, 0.02262057191634656], [-2.007682458386684,
0.08258642765685037, 0.3339735381988904, 0.04972257789159198],
[-0.0947503201024328, 0.018565941101152374, 0.007895860008536067,
0.002134016218523261]]
Out[530]:
[[4.443021766965431,
  -0.2624839948783615,
  -0.5369184805804526,
  -0.14511310285958173],
 [0.3956466069142124,
  -0.0032010243277848793,
  -0.11630388390951772,
  0.02262057191634656],
 [-2.007682458386684,
  0.08258642765685037,
  0.3339735381988904,
  0.04972257789159198],
 [-0.0947503201024328,
  0.018565941101152374,
  0.007895860008536067,
  0.002134016218523261]]
```

Results of the test

I multiplied the result of the inverse with the original and got the identity matrix

```
#test
o=[[2,3,4,11],[5,6,8,90],[9,10,20,40],[12,44,34,26]]
m1=matrix(o)
p=[[4.443021766965431, -0.2624839948783615, -0.5369184805804526, -0.14511310285958173], [0.3956466069142124, -0.0032010243
m2=matrix(p)
m2.mult(m1)
```

```
Out[529]:
[[0.9999999999999998, 1.7763568394002505e-15, 0.0,
-3.108624468950438e-14],
 [-4.440892098500626e-16,
  0.9999999999999994,
  -6.661338147750939e-16,
  -6.661338147750939e-16],
 [7.771561172376096e-16,
  1.7763568394002505e-15,
  1.0000000000000027,
  1.3988810110276972e-14],
 [0.0, 0.0, 0.0, 1.0000000000000004]]
```