Homework 4

Michael Brodskiy
Professor: M. Fanaei

April 4, 2023

Listing 1: Problem 1

```
"""
 * ============================================================================
 *
 *       Filename:  HW4Prob1Brodskiy.py
 *       Assignment: Homework 4 Problem 1
 *       Title: Matrix Common Term Finder
 *
 *    Description:  Takes in a matrix and returns terms common to
 *                  all rows of the matrix
 *
 *        Version:  1.0
 *        Created:  04/01/2023
 *       Revision:  N/A
 *         Python:  Python 3.9.2
 *
 *         Author:  M. Brodskiy
 *
 * ============================================================================
"""

import numpy as np

def comMatTerms(A):

    uniqMatTerms = np.unique(A)
    comTerms = []

    for uniqVal in uniqMatTerms:

        B = [A == uniqVal]
        addTerm = True

        for row in B:
            for j in row:
                if (True not in j):
                    addTerm = False

        if addTerm:
            comTerms.append(uniqVal)

    return np.array(comTerms)
```

The same can be done for columns if the transpose of the inputted matrix, `A.T`, were used in the same function.

Listing 2: Problem 2

```
"""
 * ============================================================================
 *
 *       Filename:  HW4Prob2Brodskiy.py
 *       Assignment: Homework 4 Problem 2
 *       Title: Plot Generator
 *
 *    Description:  Generates two types of plots for given functions
 *
 *        Version:  1.0
 *        Created:  04/01/2023
 *       Revision:  N/A
 *         Python:  Python 3.9.2
 *
 *         Author:  M. Brodskiy
 *
 * ============================================================================
"""

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-1.5,1.5,1000)
y = np.linspace(-1,1,1000)

def onePlot():
    plt.figure()
    plt.plot(y, np.sin(np.exp(2*x)), y, np.cos(np.exp(-2*x)), '-r')
    plt.grid(axis='both', color='grey', linewidth=0.5, ls='dotted', alpha=0.7)
    plt.title("Two Sinusoidal Functions on One Graph")
    plt.xlabel("$x$-axis from -1.5 to 1.5")
    plt.ylabel("$y$-axis from -1 to 1")
    plt.legend(["$y_1=\sin(e^{2x})$", "$y_2=\cos(e^{-2x})$"])
    plt.text(.1, 1.05,'$\sin(e^{2x})$', horizontalalignment='center', verticalalignment='center')
    plt.text(-.6, 1.05,'$\cos(e^{-2x})$', horizontalalignment='center', verticalalignment='center')

def twoPlot():
    plt.figure()
```

```
39      plt.subplot(2, 1, 1)
40      plt.plot(y, np.sin(np.exp(2*x)), '-b')
41      plt.title("Two Sinusoidal Functions on Two Graphs")
42      plt.ylabel("$\sin(e^{2x})$")
43      plt.grid(axis='both', color='grey', linewidth=0.5, ls='dotted'
            , alpha=0.7)
44      plt.text(-.1, .9,'$\sin(e^{2x})$', horizontalalignment='center
            ', verticalalignment='center')
45      plt.legend(["$y_1=\sin(e^{2x})$"])
46      plt.subplot(2, 1, 2)
47      plt.plot(y, np.cos(np.exp(-2*x)), '-r')
48      plt.xlabel("$x$-axis")
49      plt.ylabel("$\cos(e^{-2x})$")
50      plt.legend(["$y_2=\cos(e^{-2x})$"])
51      plt.text(-.1, .75,'$\cos(e^{-2x})$', horizontalalignment='
            center', verticalalignment='center')
52      plt.grid(axis='both', color='grey', linewidth=0.5, ls='dotted'
            , alpha=0.7)
53
54  onePlot()
55  twoPlot()
56  plt.show()
```

Listing 3: Problem 3

```
"""
 * ============================================================================
 *
 *        Filename:  HW4Prob3Brodskiy.py
 *        Assignment: Homework 4 Problem 3
 *        Title: Plot Generator 2.0
 *
 *     Description:  Generates a plot from parametric function
 *
 *         Version:  1.0
 *         Created:  04/01/2023
 *        Revision:  N/A
 *          Python:  Python 3.9.2
 *
 *          Author:  M. Brodskiy
 *
 * ============================================================================
"""

import numpy as np
import matplotlib.pyplot as plt

t = np.linspace(-4 * np.pi, 4 * np.pi, 1000)
x = np.cos(t) + 2 * np.cos(t / 4)
y = np.sin(t) - 2 * np.sin(t / 4)

def parametrize():
    plt.figure()
    plt.plot(y, x)
    plt.grid(axis='both', color='grey', linewidth=0.5, ls='dotted', alpha=0.7)
    plt.title("$\sin(t)-2\sin\left(\dfrac{t}{4}\\right)$ vs. $\cos(t)+2\cos\left(\dfrac{t}{4}\\right)$")
    plt.xlabel("$\cos(t)+2\cos\left(\dfrac{t}{4}\\right)$")
    plt.ylabel("$\sin(t)-2\sin\left(\dfrac{t}{4}\\right)$")
    plt.text(0, 0, "$y=\sin(t)-2\sin\left(\dfrac{t}{4}\\right)$\n$x=\cos(t)+2\cos\left(\dfrac{t}{4}\\right)$",
        horizontalalignment='center', verticalalignment='center')

parametrize()
plt.show()
```

Listing 4: Problem 4

```
"""
 * ============================================================================
 *
 *        Filename:  HW4Prob4Brodskiy.py
 *        Assignment: Homework 4 Problem 4
 *        Title: Point Class Creator
 *
 *     Description:  Defines a Point Class
 *
 *         Version:  1.0
 *         Created:  04/01/2023
 *        Revision:  N/A
 *          Python:  Python 3.9.2
 *
 *          Author:  M. Brodskiy
 *
 * ============================================================================
"""

import numpy as np

class Point:

    def __init__(self, x, y, z = 0):
        self.x = x
        self.y = y
        self.z = z

    @property
    def _r(self):
        return np.sqrt((self.x ** 2) + (self.y ** 2) + (self.z **
            2))

    def __str__(self):
        return str((self.x, self.y, self.z))

    def __eq__(self, other):
        return (self._r == other._r)

    def __lt__(self, other):
        return (self._r < other._r)

    def __le__(self, other):
```

```
43          return self == other or self < other
44
45      def __gt__(self, other):
46          return not self <= other
47
48      def __ge__(self, other):
49          return not self < other
50
51      def __ne__(self, other):
52          return not self == other
53
54      def __add__(self, other):
55          return Point(self.x + other.x, self.y + other.y, self.z +
                other.z)
56
57      def __iadd__(self, other):
58          return Point(self.x + other.x, self.y + other.y, self.z +
                other.z)
59
60      def asdict(self):
61          return {'x':self.x, 'y':self.y, 'z':self.z}
62
63  cur_point = Point(1, 2, 3)
64  print(cur_point._r)
65  cur_point._r = 5 # Read-only attribute modification causes error
66  print(cur_point._r)
```

Listing 5: Problem 5

```
"""
 * ============================================================================
 *
 *        Filename:  HW4Prob5Brodskiy.py
 *        Assignment: Homework 4 Problem 5
 *        Title: Circle Class
 *
 *     Description:  Defines a Circle Class, based on Point
 *
 *         Version:  1.0
 *         Created:  04/01/2023
 *        Revision:  N/A
 *          Python:  Python 3.9.2
 *
 *          Author:  M. Brodskiy
 *
 * ============================================================================
"""

import matplotlib.pyplot as plt
import numpy as np

class Point:

    def __init__(self, x, y, z = 0):
        self.x = x
        self.y = y
        self.z = z

    @property
    def _r(self):
        return np.sqrt((self.x ** 2) + (self.y ** 2) + (self.z **
            2))

    def __str__(self):
        return str((self.x, self.y, self.z))

    def __eq__(self, other):
        return (self._r == other._r)

    def __lt__(self, other):
        return (self._r < other._r)

```

```
43      def __le__(self, other):
44          return self == other or self < other
45
46      def __gt__(self, other):
47          return not self <= other
48
49      def __ge__(self, other):
50          return not self < other
51
52      def __ne__(self, other):
53          return not self == other
54
55      def __add__(self, other):
56          return Point(self.x + other.x, self.y + other.y, self.z +
                other.z)
57
58      def __iadd__(self, other):
59          return Point(self.x + other.x, self.y + other.y, self.z +
                other.z)
60
61      def asdict(self):
62          return {'x':self.x, 'y':self.y, 'z':self.z}
63
64
65  class Circle():
66
67      def __init__(self, center, radius):
68          self.center = center
69          self._radius = radius
70
71      @property
72      def radius(self):
73          return self._radius
74
75      @radius.setter
76      def radius(self, rad):
77          if (rad < 0): raise ValueError("Negative numbers are not
                permitted for radius values")
78          else: self._radius = rad
79
80      def draw(self):
81          t = np.linspace(-np.pi, np.pi, 1000)
82          y = (self._radius * np.sin(t)) + self.center.x
83          x = (self._radius * np.cos(t)) + self.center.y
84
```

```
85          plt.figure()
86          plt.plot(y, x)
87          plt.grid(axis='both', color='grey', linewidth=0.5, ls='
               dotted', alpha=0.7)
88          plt.title(f"Circle of radius {self._radius} centered at ({
               self.center.x}, {self.center.y})")
89          plt.xlabel("$x$-axis")
90          plt.ylabel("$y$-axis")
91          plt.legend(['$(x-$' + str(self.center.x) + '$)^2+(y-$' +
               str(self.center.y) + '$)^2=$' + str(self._radius ** 2)
               ])
92          plt.text(self.center.x, self.center.y, '$(x-$' + str(self.
               center.x) + '$)^2+(y-$' + str(self.center.y) + '$)^2=$'
                + str(self._radius ** 2), horizontalalignment='center'
               , verticalalignment='center')
93          plt.show()
94
95  c = Circle(Point(1,4), 2)
96  c.draw()
```

Listing 6: Problem 6

```
"""
 * ============================================================================
 *
 *        Filename:  HW4Prob6Brodskiy.py
 *        Assignment: Homework 4 Problem 6
 *        Title: Sphere Class
 *
 *     Description:  Defines a Sphere Class, based on Point
 *
 *         Version:  1.0
 *         Created:  04/01/2023
 *        Revision:  N/A
 *          Python:  Python 3.9.2
 *
 *          Author:  M. Brodskiy
 *
 * ============================================================================
"""

import matplotlib.pyplot as plt
import numpy as np

class Point:

    def __init__(self, x, y, z = 0):
        self.x = x
        self.y = y
        self.z = z

    @property
    def _r(self):
        return np.sqrt((self.x ** 2) + (self.y ** 2) + (self.z **
            2))

    def __str__(self):
        return str((self.x, self.y, self.z))

    def __eq__(self, other):
        return (self._r == other._r)

    def __lt__(self, other):
        return (self._r < other._r)

```

```
43      def __le__(self, other):
44          return self == other or self < other
45
46      def __gt__(self, other):
47          return not self <= other
48
49      def __ge__(self, other):
50          return not self < other
51
52      def __ne__(self, other):
53          return not self == other
54
55      def __add__(self, other):
56          return Point(self.x + other.x, self.y + other.y, self.z +
                other.z)
57
58      def __iadd__(self, other):
59          return Point(self.x + other.x, self.y + other.y, self.z +
                other.z)
60
61      def asdict(self):
62          return {'x':self.x, 'y':self.y, 'z':self.z}
63
64  class Sphere:
65
66      def __init__(self, center, radius):
67          self.center = center
68          self._radius = radius
69
70      @property
71      def radius(self):
72          return self._radius
73
74      @radius.setter
75      def radius(self, rad):
76          if (rad < 0): raise ValueError("Negative numbers are not
                permitted for radius values")
77          else: self._radius = rad
78
79      @property
80      def _vol(self):
81          return (4.0 / 3.0) * np.pi * self._radius ** 3
82
83      def __str__(self):
84          return f"The volume of the sphere centered at {self.center
```

```
            } with a radius of {self._radius} is equal to {self.
            _vol:.2f}."

class centeredSphere:

    def __init__(self, radius):
        self.center = Point(0,0,0)
        self._radius = radius

    @property
    def radius(self):
        return self._radius

    @radius.setter
    def radius(self, rad):
        if (rad < 0): raise ValueError("Negative numbers are not
            permitted for radius values")
        else: self._radius = rad

    @property
    def _vol(self):
        return (4.0 / 3.0) * np.pi * self._radius ** 3

    def __str__(self):
        return f"The volume of the sphere centered at the origin
            with a radius of {self._radius} is equal to {self._vol
            :.2f}."

notOrigin = [Sphere(Point(np.random.randint(0,6),np.random.randint
    (0,6),np.random.randint(0,6)), np.random.randint(1,6)), Sphere(
    Point(np.random.randint(0,6),np.random.randint(0,6),np.random.
    randint(0,6)), np.random.randint(1,6)), Sphere(Point(np.random.
    randint(0,6),np.random.randint(0,6),np.random.randint(0,6)), np
    .random.randint(1,6)), Sphere(Point(np.random.randint(0,6),np.
    random.randint(0,6),np.random.randint(0,6)), np.random.randint
    (1,6))]

Origin = [centeredSphere(np.random.randint(5,11)), centeredSphere(
    np.random.randint(5,11))]

allSphere = notOrigin + Origin
for i in allSphere:
    print(i)
```