第三次计算物理作业

吴远清 2018300001031

第一题

复数类实现:

```
import math

class Komplex:
    def __init__(self, x, y, typec=1):
        assert (typec in [0,1]), "typec could only be 1 or 0, not 
{}".format(typec)
        self.typec = typec
        if typec == 1:
            self.re = x
            self.im = y
        elif typec == 0:
            self.re = x*math.cos(y)
            self.im = x/math.sin(y)

    def __add__(self, another):
        if self.typec == another.typec:
            return Komplex(self.re + another.re, self.im + another.im, 
self.typec)
        else:
            print("Warning:two complex with different type were given, set type 
to cartesian")
            return Komplex(self.re + another.re, self.im + another.im)

    def __sub__(self, another):
        if self.typec == another.typec:
            return Komplex(self.re - another.re, self.im - another.im, 
self.typec)
        else:
            print("Warning:two complex with different type were given, set type 
to cartesian")
            return Komplex(self.re - another.re, self.im - another.im)

    def __mul__(self, another):
        if self.typec == another.typec:
            return Komplex(self.re * another.re - self.im * another.im, self.im 
* another.re + another.im * self.re, self.typec)
        else:
            print("Warning:two complex with different type were given, set type 
to cartesian")
            return Komplex(self.re * another.re - self.im * another.im, self.im 
* another.re + another.im * self.re)

    def __truediv__(self, another):
        factor = (another.re**2 + another.im**2)
        if self.typec == another.typec:
```

```python
                return Komplex((self.re * another.re + self.im * another.im)/factor,
(self.im * another.re - self.re * another.im)/factor, self.typec)
            else:
                print("Warning:two complex with different type were given, set type
to cartesian")
                return Komplex((self.re * another.re + self.im * another.im)/factor,
(self.im * another.re - self.re * another.im)/factor)

    def __repr__(self):
        if self.typec == 1:
            return "{}+{}i".format(self.re, self.im)
        else:
            return "{}*e^({}i)".format(math.sqrt(self.re**2+self.im**2),
math.atan(self.im/self.re))

    def setType(self, typec):
        assert (typec in [0,1]), "typec could only be 1 or 0, not
{}".format(typec)
        self.typec = typec

    def conj(self):
        return(Komplex(self.re, -self.im, self.typec))

    def getRe(self):
        return self.re

    def getIm(self):
        return self.im
```

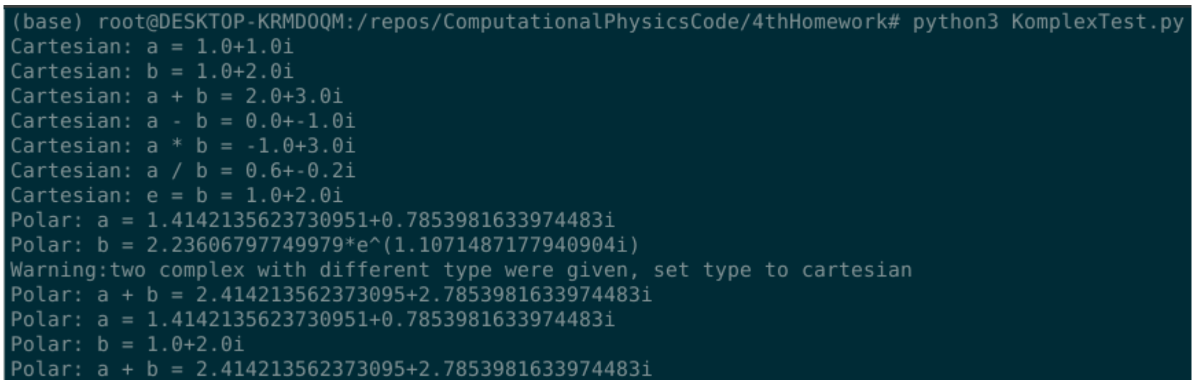
测试文件:

```python
from Komplex import Komplex
import math

a = Komplex(1.0, 1.0)
b = Komplex(1.0, 2.0)
e = b

print("Cartesian: a = {}".format(a))
print("Cartesian: b = {}".format(b))
print("Cartesian: a + b = {}".format(a+b))
print("Cartesian: a - b = {}".format(a-b))
print("Cartesian: a * b = {}".format(a*b))
print("Cartesian: a / b = {}".format(a/b))
print("Cartesian: e = b = {}".format(e))

a = Komplex(math.sqrt(2.0), math.pi/4.0)
b = Komplex(1.0, 2.0)
b.setType(0)

print("Polar: a = {}".format(a))
print("Polar: b = {}".format(b))
print("Polar: a + b = {}".format(a+b))
a.setType(1)
b.setType(1)
print("Polar: a = {}".format(a))
print("Polar: b = {}".format(b))
print("Polar: a + b = {}".format(a+b))
```

复数在类内均以$x + yi$的形式储存和计算, 仅在输出时转换为相应的形式.

测试文件中, 我测试了复数的四则运算, 不同形式复数的输入与输出, 不同类型复数之间的运算, 以及复数类型的转换.

运行结果



```
(base) root@DESKTOP-KRMDOQM:/repos/ComputationalPhysicsCode/4thHomework# python3 KomplexTest.py
Cartesian: a = 1.0+1.0i
Cartesian: b = 1.0+2.0i
Cartesian: a + b = 2.0+3.0i
Cartesian: a - b = 0.0+-1.0i
Cartesian: a * b = -1.0+3.0i
Cartesian: a / b = 0.6+-0.2i
Cartesian: e = b = 1.0+2.0i
Polar: a = 1.4142135623730951+0.7853981633974483i
Polar: b = 2.23606797749979*e^(1.1071487177940904i)
Warning:two complex with different type were given, set type to cartesian
Polar: a + b = 2.414213562373095+2.7853981633974483i
Polar: a = 1.4142135623730951+0.7853981633974483i
Polar: b = 1.0+2.0i
Polar: a + b = 2.414213562373095+2.7853981633974483i
```

第二题

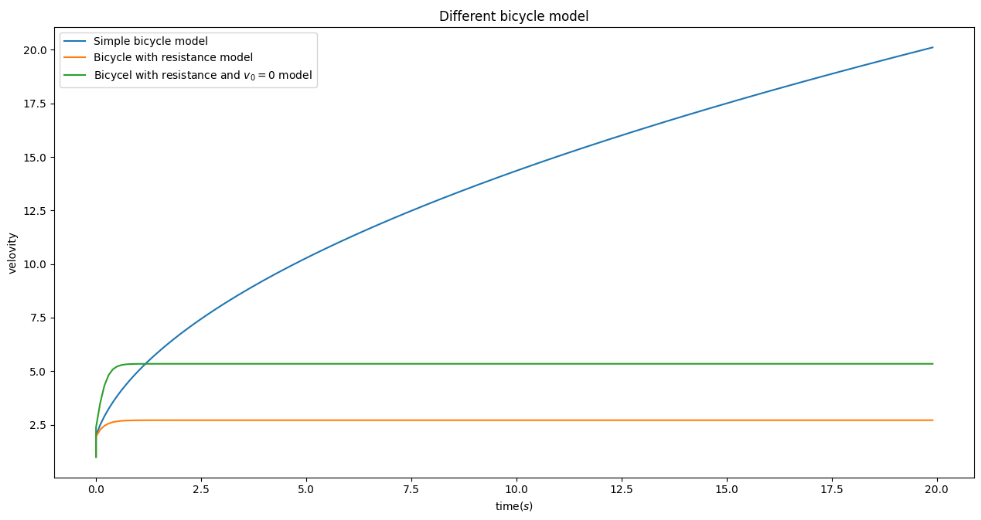
```python
import matplotlib.pyplot as plt

class bicycle():
    def __init__(self, power = 10, mass = 1, timeStep = 0.1, totalTime = 20, v0
= 1):
        self.v = [v0]
        self.t = [0]
        self.m = mass
        self.p = power
        self.dt = timeStep
        self.time = totalTime
    def run(self):
        _time = 0
        while(_time < self.time):
            self.v.append(self.v[-1] + self.dt * self.p / (self.m * self.v[-1]))
            self.t.append(_time)
            _time += self.dt
    def showResult(self):
        plt.plot(self.t, self.v, label = 'velocity with time')
        plt.xlabel('time($s$)')
        plt.ylabel('velovity')
        plt.title('Bicycling velocity with time')
        plt.legend()
        plt.show()

class bicycleWithResistance(bicycle):
    def __init__(self, C = 1, rho = 1, A = 1):
        super().__init__()
        self.C = C
        self.rho = rho
        self.A = A
    def run(self):
        _time = 0
        while(_time < self.time):
            self.v.append(self.v[-1] + self.dt * self.p / (self.m * self.v[-1])
- (self.C * self.rho * self.A * self.v[-1]**2)/(2*self.m)*self.dt)
            self.t.append(_time)
            _time += self.dt


class bicycleWeNeed(bicycleWithResistance):
    def __init__(self, vStar = 7):
        super().__init__()
        self.v0 = 0
        self.vStar = vStar
    def run(self):
        _time = 0
        f0 = self.p / self.vStar
        while _time < self.time:
            if self.v[-1] < self.vStar:
                self.v.append(self.v[-1] + f0/self.m - (self.C * self.rho *
self.A * self.v[-1]**2)/(2*self.m)*self.dt)
            else:
                self.v.append(self.v[-1] + self.dt * self.p / (self.m *
self.v[-1]) - (self.C * self.rho * self.A * self.v[-1]**2)/(2*self.m)*self.dt)
            self.t.append(_time)
            _time += self.dt

a = bicycle()
b = bicycleWithResistance()
c = bicycleWeNeed()
a.run()
b.run()
c.run()
plt.plot(a.t, a.v, label = 'Simple bicycle model')
plt.plot(b.t, b.v, label = 'Bicycle with resistance model')
plt.plot(c.t, c.v, label = 'Bicycel with resistance and $v_0=0$ model')
plt.xlabel('time($s$)')
plt.ylabel('velovity')
plt.title('Different bicycle model')
plt.legend()
plt.show()
```

考虑空气阻力的模型(bicycleWithResistance)继承自bicycle,重写了父类的构造函数(使得可以输入参数C, ρ, A)和run函数(将空气阻力的影响加入考虑).

使初始速度可以为0的模型(bicycleWeNeed)继承自bicycleWithResistance,重写了父类的构造函数(使得可以输入参数v^*)和run函数(使得在v较小时可以使用恒定$F = \frac{P}{v^*}$条件).

最后,我们没有使用showResult方法,而是将三种模型的结果绘制在一张图内.

运行结果


Different bicycle model
Simple bicycle model
Bicycle with resistance model
Bicycel with resistance and $v_0 = 0$ model
velovity
time(s)
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0