Упражнение 6 – Объектно-ориентированное программирование и алгоритмы

В этом упражнении мы строим class вещественных многочленов от одной переменной и напишем для него функции (называемые methods) для различных алгебраических вычислений.

Представление:

Полином от одной переменной может быть описан его коэффициентами следующим образом:

$$\mathbf{P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0}$$

Когда ведущий коэффициент a_n отличается от нуля. Например, для полинома x^3+4x^2-5

коэффициенты $a_0=-5$, $a_1=0$, $a_2=4$, $a_3=1$ степень 3.

Фактический многочлен n-уровня всегда имеет ровно n корней ($r_1 \ldots r_n$), то есть $P(r_j)=0$. В общем, некоторые корни могут быть составными. Например, для полинома x^2+1 корни i, -i. Мы будем ссылаться только на фактические корни многочленов. Также могут быть корни с множественностью, что означает, что один и тот же корень будет появляться более одного раза. Полином всегда можно представить как умножение линейных функций, в которых корни выглядят следующим образом:

$$\mathbf{P(x) = a_n(x-r_1)(x-r_2)\ldots(x-r_n)}$$

Например:

$$\mathbf{4(x-2)^2(x-1) = 4x^3-16x^2+20x-8}$$

Мы будем строить class Polynomial когда многочлен представлен списком коэффициентов a в длину n+1, член i в списке будет коэффициент x^i в многочлене.

Например:

$$\mathbf{x^3+4x^2-5}$$

Будет представлен списком: a = [-5, 0, 4, 1].

Список будет внутренним по отношению к полиному, то есть он будет представлен **self.__a.** Последний элемент в списке, который представляет коэффициент самого высокого удержания, всегда будет отличаться от нуля, за исключением случая, когда полином является нулевым полиномом, и тогда список будет равен [0]. Какой бы метод не требовался для возврата полинома, вы должны убедиться, что возвращаемый полином отвечает этим требованиям **self.__a**.

Методы реализации:

Все методы, которые вы должны написать, будут methods с Polyomial class. Все должно быть записано в один файл с именем **polynomial.py.**

Помимо методов, которые вам понадобятся в упражнении, вы можете написать простые вспомогательные функции, которые можно использовать как Polyomial class.

Примечание. Требуется написать функции, которые вычисляют численные вычисления (например, умножение 2 полиномов на реальные коэффициенты), для которых компьютер имеет конечный уровень точности. Иногда ошибки округления из-за уровня точности могут изменить вывод функции (например, есть ли фактический корень или нет) - тесты не требуют, чтобы функция возвращала правильный ответ в этих крайних случаях.

Примечание: Нет необходимости проверять законность ввода, если это явно не требуется.

1. Реализовать метод constructor полинома.

def __init__ (*self, a, data='coef'*)

Метод получает в качестве входных данных список действительных чисел и запускает полином в одном из следующих двух режимов:

- Если переменная **data 'coef'** то список ввода **a** будет списком коэффициентов многочлена. По умолчанию (т. Е. Если значение не введено для переменной **data**) '**coef**'.
- Если переменная **data 'roots'** то список ввода **a** будет списком корней многочлена. То есть полином, который нужно инициализировать (x-a[0])*(x-a[1])*...*(x-a[n-1]) когда **n** длина списка **a**. В этом случае коэффициент самой высокой степени в полиноме будет равен 1.
- Для другого значения **data** вернется **None**.

  Например:

  для команды:

  P = Polynomial ([2, 2, 0, -3])

  вы получаете полином:

  P(x)=-3x^3+2x+2

  для команды:

  Q = Polynomial ([2, 2, 0, -3], 'roots')

  вы получаете полином:

  Q(x)=(x-2)^2x(x-3)=x^4-x^3-8x^2+12x

2. a) Реализуйте метод, который возвращает список коэффициентов полиномов:

   def get_coef(self):

   должна быть возвращена переменная типа **list** содержащая упорядоченный список коэффициентов.

   b) Реализуйте метод, который возвращает степень полинома:

   def get_deg(self):

   должна быть возвращена переменная типа **int**

3. Заполните следующие операторы для двух полиномов:

   a) **def __eq__(self, other):**

      Оператор равенства ('==') вернет **True** если два полинома равны, то есть коэффициенты равны. В противном случае будет возвращен **False**.

Например:

>>> Polynomial([3,4]) == Polynomial([3.0,4,0])

True

Потому что оба случая представляют полином 4x+3

>>> Polynomial([1,2]) == Polynomial([2,1])

False

2x+1≠ x+2

b) def __add__(self, other):

Оператор суммы (‘+’). Возвращает полином суммы двух полиномов. Обратите внимание, что когда полиномы складываются, коэффициенты некоторых могут быть сброшены. Например:

>>> P = Polynomial([1,2,3])

>>> Q = Polynomial([-5,-4,-3])

>>> print(P,Q,P+Q)

3x^2+2x+1 -3x^2-4x-5 -2x-4

c) **def __sub__(self, other):**

Оператор разности (‘-’). Например, для полиномов из раздела B:

>>> print(P-Q)

6x^2+6x+6

d) def __mul__(self, other):

Оператор умножения (‘*’). Вернет произведение полиномов. Например, для полиномов из раздела b:

>>> print(P*Q)

-9x^4-18x^3-26x^2-14x-5

e) **def __pow__(self, k):**

Оператор степени (‘**’). Получит в качестве входных данных целое неотрицательное число k и вернет полином P(x)^k. Метод должен быть рекурсивно реализован. Например:

>>> R = Polynomial([-2,1])

>>> print(R**0, R, R**2, R**3, R**4)

1 x-2 x^2-4x+4 x^3-6x^2+12x-8 x^4-8x^3+24x^2-32x+16

f) def __str__(self):

Оператор преобразования в строковый тип (‘str’). После правильной реализации этого оператора эта строка может быть напечатана при вызове **print** с **instance** полиномов.

Пожалуйста, обратите внимание на следующее:

- Коэффициенты должны быть напечатаны в порядке убывания справа налево, начиная с максимального, когда после каждого коэффициента должен быть напечатан ‘x^i’, когда на месте i, соответствующее значение степени.
- Округлите каждый коэффициент до 2 самых значащих цифр - используйте функцию **sig** предоставленную вам.
- «+» Или «-» должны использоваться перед каждым коэффициентом (кроме первого) в соответствии с знаком коэффициента.
- Если коэффициент равен 0, пропустите этот элемент.
- Если коэффициент равен 1 или -1, он не должен быть записан, а только x с соответствующим знаком.
- Если степень равна 1, она не должна быть написана, а только x.
- Если степень равна 0, должен быть записан только коэффициент.

Примеры:

```
>>> print(Polynomial([8]))
8

>>> print(Polynomial([0]))
0

>>> print(Polynomial([1, -4.35636, 0.143525]))
0.14x^2-4.4x+1

>>> print(Polynomial([0, 2, 0, -3]))
-3x^3+2x

>>> print(Polynomial([-4.0, 2.0, -1.7345, 1, 0, 0.0]))
x^3-1.7x^2+2x-4
```

g) **def __floordiv__(self, other):**

Оператор делителя (‘//’).

Возвращает деление многочлена на другой многочлен с остатком. Приведенный здесь алгоритм Евклида должен использоваться: https://en.wikipedia.org/wiki/Polynomial_greatest_common_divisor#Euclidean_division

h) **def __mod__(self, other):**

Оператор по модулю (‘%’). Возвращает остаток от деления полинома на полином, где деление выполняется в соответствии с оператором в разделе f.

Пример:

P(x)=2x^3+5x^2-x+7

Q(x)=x^2-3x+5

H(x)=P(x)//Q(x)=x+4

R(x)=P(x)%Q(X)=6x-13

P(x)=H(x)*Q(x)+R(x)

Python скрипт:

```
>>> P = Polynomial([7,-1,5,2])
>>> Q = Polynomial([5,-3,2])
>>> print('P =', P, '\nQ =', Q, '\nP//Q =', P//Q, '\nP%Q =', P%Q)
P = 2x^3+5x^2-x+7
Q = 2x^2-3x+5
P//Q = x+4
P%Q = 6x-13
```

4. Реализуйте метод, который вычисляет полиномиальное значение для определенного значения x:

   def evaluate(self, x):

   Метод должен вернуть число.

   $$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

Пример:

```
>>> print(Polynomial([1, -4, 2]).evaluate(3))
7
```

Имеет место потому что: 2*3^2-4*3+1=7

>>> print(Polynomial([0.5, 0, 2, 1]).evaluate(-0.5))

0.875

Имеет место потому что: (-0.5)^3+2*(-0.5)^2+0.5=0.875

5. Реализуйте метод, который вычисляет k-производную полинома:

   def diffrentiate(self,k=1):

   Метод должен быть рекурсивно реализован, когда рекурсивная база (для значения по умолчанию) должна вычислить первую производную. Метод заменит исходный полином на полином производной k.

6. Реализуйте метод, который вычисляет неопределенный интеграл порядка k многочлена:

   def integrate(self, a, b)

   Используйте рекурсию, когда вычисляется первый интеграл. Метод изменит полином на его k-интеграл и ничего не вернет. Метод трансформируется в метод из вопроса 5, Так что, например, для каждого объекта **p** с **Polynomial class: p == p.differentiate(7).integrate(7)**

   Возвращаемое значение: True

7. Реализуйте метод, который вычисляет определенный интеграл полинома:

   def definite_integral(self, a,b):

   Метод принимает пределы интеграла a, b. То есть метод должен рассчитать $\int_a^b P(x)dx$ для полинома P(x). Например, для полинома $\mathbf{P(x)=2x^2-4x+1}$ получаем:

   >>> print(Polynomial([1, -4, 2]).definite_integral(-2, 4))

   29.99999999999999

   потому что $\int_{-4}^{2}(2x^2-4x+1)dx=30$

8. Используйте метод, который вычисляет и возвращает один корень многочлена:

   def get_root(self, x0=0, iters=100, epsilon=0.000000001)

   Метод получает начальное предположение **x0**, количество максимальных итераций **iters** и максимальную ошибку **epsilon**. Метод должен численно найти корень многочлена, то есть число **x** для **P(x)=0.** Если корень не найден, вернуть **None.**

Алгоритм, который мы используем, - это алгоритм Ньютона-Рафсона, чтобы найти корни функции. https://en.wikipedia.org/wiki/Newton%27s_method

Один из следующих 3 случаев должен быть остановлен:

I. найден точный корень

II. мы достигли максимального количества итераций

III. мы достигли точки **x** для которой $P'(x)=0$ тогда следующая точка не может быть рассчитана.

Примечание: есть два разных способа измерения точности, когда мы хотим найти корень функции f. Предположим, что истинный корень **r** при **f(r)=0**.

I. должно быть выполнено условие для **x** так что $|x-r|<\varepsilon$. То есть расстояние от x, возвращаемое от реального корня r, мало (это то, что мы делали в алгоритме двоичного поиска).

II. должно быть выполнено условие для **x** так что $|f(x)|<\varepsilon$.

9. Используйте метод, который вычисляет корни многочлена:

def get_roots(self):

Метод должен возвращать список действительных чисел, содержащих все действительные полиномиальные корни (с кратностью), из n корней (некоторые из которых могут быть комплексными) n-полиномиального полинома, то есть, если заданный корень имеет кратность 3 в полиноме, возвращенный список будет содержать его 3 раза. Корни можно вернуть в любом порядке по вашему выбору. Если вы не нашли реальных корней, должен быть возвращен пустой список.

Алгоритм: один корень должен быть найден с использованием предыдущего метода при его инициализации **x0=0**, Сделайте не более 100 шагов и остановитесь, если мы найдем корень с точностью 10^{-9}. Если мы нашли корень r, он должен быть добавлен в корневой список и теперь рекурсивно найти корни полинома **P(x)/(x-r)** пока не найдем все корни.

Примечание. Для больших полиномов алгоритм может страдать от численной нестабильности. Поскольку для каждого корня найдена конечная точность, при делении на r-x для его удаления накапливается ошибка, которая может возрасти для следующих корней. Для заданных вами полиномов метод будет работать должным образом с разумной точностью, которая будет учтена в тесте.

Пример:

>>> P = Polynomial([-2, 4, 3, 3], 'roots')

>>> print(P)

x^4-8x^3+13x^2+30x-72

>>> P.get_roots()

[3.000396299417213 ,-1.999999994792993 ,3.999999869164775 ,2.999603826211005]

Пример:

>>> Polynomial([1, 0, 1]).get_roots()

[]

Потому что полином x^2+1 не имеет реального корня.

10. Реализуйте метод, который вычисляет и возвращает наибольший общий делитель двух полиномов:

def gcd(self, other):

Наибольший общий знаменатель двух многочленов P(x), Q(x) определяется как многочлен R(x) выполняющий условия:

P(x)=R(x)U(x)

Q(x)=R(x)V(x)

Gcd полиномов должен быть найден с использованием алгоритма Евклида, который мы изучили для натуральных чисел, где все операции (деление, вычитание и т. Д.) Выполняются для полиномов, использующих операторы Вопроса 2 вместо аналогичных операций над натуральными числами, останавливаясь, когда один из полиномов является нулевым полиномом.

https://en.wikipedia.org/wiki/Polynomial_greatest_common_divisor#Euclid's_algorithm

Пример:

```
>>> P = Polynomial([-2, 4, 3, 3], 'roots')
>>> Q = Polynomial([-2, 3, 5], 'roots')
>>> print('GCD of ', P, ', ', Q, ' is: ', P.gcd(Q))
```

GCD of x^4-8x^3+13x^2+30x-72 , x^3-6x^2-x+30 is: x^2-x-6

Самый большой общий делитель многочленов $P(x)=(x+2)(x-4)(x-3)^2$, $Q(x)=(x+2)(x-3)(x-5)$: $R(x)=(x+2)(x-3)=x^2-x-6$

11. Вот метод, который составляет многочлен с многочленом:

def compose(self, other):

То есть для полиномов P(x), Q(x) метод изменит полином P(x) так что сборочный полином P(Q(x)) ничего не возвращает.

Пример:

$P(x)=2x^2-4x+1$, $Q(x)= x^2+1$

$P(Q(x))= 2(x^2+1)^2-4(x^2+1)+1=2x^4-1$

Правила сдачи:

Отправить файл ex6.zip включающий файлы:

1. polynomial.py
2. README

Функции должны пройти тест предоставленный отдельно:

Console:

```
import test_stat

test_stat.run_all_tests('ex6tests')
```