Алгоритм Кронекера и численно-анал метод

Задача. Дан многочлен $f \in \mathbb{Z}[x]$, трубется выяснить, является ли он прост

Алгоритм Кронекера

В [1]:

```
import itertools
def factors(n):
    ans=[]
    for (p,m)  in ZZ(abs(n)).factor():
        ans=ans+[p for mm in range(m)]
    return [(-1)^k*prod(a) for a in Combinations(ans) for k

def ipoly(points,x=x):
    m=1
    f=0
    for (xx,yy) in points:
        f=f+ (yy-f.subs([x==xx]))*m/m.subs([x==xx])
        m=m*(x-xx)
    return f
```

В [2]:

```
def Kronecker(f):
    n=f.degree()
    m=floor(n/2)
    L=[]
    ans=[]
    for k in range(m+1):
        a=f(k)
        if a==0:
            ans=[x-k, f.quo_rem(x-k)[0]]
            break
        else:
            L.append(factors(a))
    L=iter.product(*L)
    while ans==[]:
        try:
            l=next(L)
            points=zip(range(m+1),l)
            g=QQ[x](ipoly(points))
            if g.degree()>0:
                [u,r]=f.quo_rem(g)
                if r==0:
                    ans=[g,u]
        except StopIteration as err:
            break
    return ans
```

1.) Как исправить функцию Kronecker, чтобы она работа при любом выборе переменной?

В [3]:

```
var("t")
f=expand(QQ[t](t^3+1))
print(f)
Kronecker(f)
```

```
t^3 + 1

---------------------------------------------------------------------------
-
AttributeError                            Traceback (most recent
t)
<ipython-input-3-421704718701> in <module>
      2 f=expand(QQ[t](t**Integer(3)+Integer(1)))
      3 print(f)
----> 4 Kronecker(f)

<ipython-input-2-730ed2ef766a> in Kronecker(f)
     11         else:
     12             L.append(factors(a))
---> 13     L=iter.product(*L)
     14     while ans==[]:
     15         try:

AttributeError: 'builtin_function_or_method' object has no attri
uct'
```

Указание. Воспользуйтесь двумя методами:

В [4]:

```
f.parent()
```

Out[4]:

```
Univariate Polynomial Ring in t over Rational Field
```

В [5]:

```
f.variables()
```

Out[5]:

```
(t,)
```

В [6]:

```
def Kronecker(f):
    x=f.variables()[0]
    K=f.parent()
    n=f.degree()
    m=floor(n/2)
    ans=[]
# Цикл 1
    L=[]
    for k in range(m+1):
        a=f(k)
        if a==0:
            ans=[x-k, f.quo_rem(K(x-k))[0]]
            break
        else:
            L.append(factors(a))
    L=itertools.product(*L)
# Цикл 2
    while ans==[]:
        try:
            l=next(L)
            points=zip(range(m+1),l)
            g=K(ipoly(points,x=SR(x)))
            if g.degree()>0:
                [u,r]=f.quo_rem(g)
                if r==0:
                    ans=[g,u]
        except StopIteration as err:
            ans = 'poly is prime'
            break
    return ans
```

В [8]:

```
var("t")
f=expand(QQ[t](t^3+1))
print(f)
Kronecker(f)
```

```
t^3 + 1
```

Out[8]:

```
[t + 1, t^2 - t + 1]
```

В [8]:

```
var("t")
f=expand(QQ[t](t^3+t+1))
print(f)
Kronecker(f)
```

```
t^3 + t + 1
(1, 1)
(1, -1)
(1, 3)
(1, -3)
(-1, 1)
(-1, -1)
(-1, 3)
(-1, -3)
```

Out[8]:

```
'poly is prime'
```

2.) Почему используется конструкция ipoly(points,x=SR(x))? Что означает SR

Мы используем параметр x=SR(x), так как функция ipoly должна быть симво под другие выражения, а SR - символьные выражения)

Численно-аналитический метод

Теорема. Если многочлен $f \in \mathbb{Z}[x]$ делится на многочлен $g \in \mathbb{Q}[x]$, то он $lc(f)\prod(x-x_i) \in \mathbb{Z}[x]$, где x_i --- комплексные корни g.

Численно-аналитический алгоритм

Дано: $f \in \mathbb{Z}[x]$, ε > 0

Находим:

1. Старший коэффициент: $c = lc(c) \in \mathbb{Z}$
2. Комплексные корни: $x_1 \ldots x_2 \in \mathbb{C}$
3. Образвуем всевозможные многочлены: $c(x-x_i 1)\ldots(x-x_i r)$

Если один из них имеет коэффициенты, которые отличаются от целый чисе заменить эти коэффициенты на эти числа и поделить на f на этот многочле равен нулю, то многочлен f не простой. В противном случае многочлен прос

Теорема. Если многочлен $f \in \mathbb{Z}[\mathbb{x}]$ делился на многолчен $g \in \mathbb{Q}[\mathbb{x}]$, то он $lc(f)\Pi(x-x_i) \in \mathbb{Z}[\mathbb{x}]$, где x_i - комплексные корни g.

3.) Напишите функцию, которая позволяет выяснить, является ли комплекс точностью до заданного $\varepsilon > 0$.

В [12]:

```
def is_integer(a,eps): # данная функция проверяет является л
    b=floor(a.real())
    if abs(a-b)<eps or abs(a-b-1)<eps:
        return True
    else:
        return False

def almost_integer(a,eps):
    b=floor(a.real()) # округляем заданное число и делаем ег
    if abs(a-b)<eps: # возвращаем абсолютную величину числа
        return b # абсолютное значение a + bj вычисляется ка
    elif abs(a-b-1)<eps:
        return b+1
    else:
        return a
```

В [19]:

```
almost_integer(1.03+0.01*i,0.1)
```

Out[19]:

```
1
```

В [18]:

```
almost_integer(1.98+0.01*i,0.1)
```

Out[18]:

```
2
```

В [20]:

```
almost_integer(1.6+0.01*i,0.1)
```

Out[20]:

```
1.60000000000000 + 0.0100000000000000*I
```

В [21]:

```
is_integer(1.09+0.01*i,0.1)
```

Out[21]:

```
True
```

В [22]:

```
is_integer(1.1+0.01*i,0.1)
```

Out[22]:

```
False
```

4.) В Sage имеется численный способ отыскания комплексных корней многі
простоты, что его корни -- однократные.

В [23]:

```
def alt_Kronecker(f,eps=10^-9):
    x=f.variables()[0]
    R=Combinations(f.roots(CC,False))
    for r in R[1:]:
        g=CC[x](prod([x-xi for xi in r])*(f).lc())
        if prod([is_integer(a,eps) for a in g.coefficients()
            g=QQ[x](sum([almost_integer(a,eps)*x^m for (a,m)
            [u,r]=f.quo_rem(g)
            if r==0:
                ans=[g,u]
                break
    return ans
```

В [29]:

```
f=QQ[x](-16*x^4 - 7*x^4 + 2*x^2 + 1)
alt_Kronecker(f)
```

Out[29]:

```
[-23*x^4 + 2*x^2 + 1, 1]
```

В [28]:

```
var("t")
f=QQ[t](t^2-1)
alt_Kronecker(f)
```

Out[28]:

```
[t + 1, t - 1]
```

5.) Что представляет собой список R? Почему выкидывается его первый эл

В списке R представлены все множества, которые можно составить из корн
где все эти элементы не повторяются.

6.) Зачем произведение $\prod(x - x_i)$ домножается на $lc(f)$?

Чтобы образовать все многочлены, которые могут получиться.

7.) Как устроен список zip(g.coefficients(),g.exponents())?

В [52]:

```
eps = 10^(-9)
f = QQ[x](-14*x^4 - 7*x^2 + 2*x + 1)
x = f.variables()[0]
R = Combinations(f.roots(CC, False))
for r in R[1:]:
    g=CC[x](prod([x-xi for xi in r])*(f).lc())
    if prod([is_integer(a,eps) for a in g.coefficients()]):
        g=QQ[x](sum([almost_integer(a,eps)*x^m for (a,m) in
        [u,r]= f.quo_rem(g)
        if r == 0:
            ans = [g,u]
            break
L = zip(g.coefficients(),g.exponents())
zip_L = list(L)
print(zip_L)
```

```
[(1, 0), (2, 1), (-7, 2), (-14, 4)]
```

zip объединяет элементы из коэффициентов и степеней заданной перемен

Сравнение методов

8.) Какой из двух методов быстрее?

В [53]:

```
g=sum([(n+1)*x^n for n in range(4)])
h=sum([(-1)^n*(n+1)*x^n for n in range(4)])
f=QQ[x](g*h)
f
```

Out[53]:

```
-16*x^6 - 7*x^4 + 2*x^2 + 1
```

В [42]:

```
%timeit Kronecker(f)
```

```
1.31 s ± 61.2 ms per loop (mean ± std. dev. of 7 runs, 1 loop ea
```

В [43]:

```
%timeit alt_Kronecker(f)
```

```
11.3 ms ± 713 µs per loop (mean ± std. dev. of 7 runs, 100 loops
```

В [44]:

```
g=sum([(n+1)*x^n for n in range(10)])
h=sum([(-1)^n*(n+1)*x^n for n in range(5)])
f=QQ[x](g*h)
f
```

Out[44]:

```
50*x^13 + 5*x^12 + 34*x^11 + 10*x^10 + 18*x^9 + 15*x^8 + 12*x^7 -
6*x^5 + 3*x^4 + 2*x^2 + 1
```

В [46]:

```
alt_Kronecker(f)
```

Out[46]:

```
[50*x^4 - 40*x^3 + 30*x^2 - 20*x + 10,
 x^9 + 9/10*x^8 + 4/5*x^7 + 7/10*x^6 + 3/5*x^5 + 1/2*x^4 + 2/5*x
x^2 + 1/5*x + 1/10]
```

alt_Kronecker(f) быстрее.

9.) Как модифицировать функцию alt_Kronecker, чтобы она работала с крат

Замечание. См. конец л.р. № 2.

В [123]:

```
def alt_Kronecker(f,eps=10^-9):
    x=f.variables()[0]
    g=f.gcd(diff(f,x))
    if g.degree()>0:
        (u,r)=f.quo_rem(g)
        ans=[g,u]
    else:
        R=Combinations(f.roots(CC,False))
        for r in R[1:]:
            g=CC[x](prod([x-xi for xi in r])*(f).lc())
            if prod([is_integer(a,eps) for a in g.coefficien
                g=QQ[x](sum([almost_integer(a,eps)*x^m for (
                (u,r)=f.quo_rem(g)
                if r==0:
                    ans=[g,u]
                    break
    return ans
```

В [55]:

```
f=QQ[x](-16*x^6 - 7*x^4 + 2*x^2 + 1)
alt_Kronecker(f)
```

Out[55]:

```
[-16*x^3 - 12*x^2 - 8*x - 4, x^3 - 3/4*x^2 + 1/2*x - 1/4]
```

В [54]:

```
f=QQ[x]((x^2-1)^2*(x+5))
alt_Kronecker(f)
```

Out[54]:

```
[x + 5, x^4 - 2*x^2 + 1]
```