Московский авиационный институт
(Национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа № 3

по курсу «Криптография»

Студент: Гаврилов М.С.

Группа: 80-306б

Преподаватель: Борисов А. В.

Оценка:

Москва, 2022

1. Постановка задачи

Подобрать такую эллиптическую кривую, порядок точки которой полным перебором находится за 10 минут на ПК. Упомянуть в отчёте результаты замеров работы программы, характеристики вычислителя. Также указать какие алгоритмы и/или теоремы существуют для облегчения и ускорения решения задачи полного перебора.
Рассмотреть для случая конечного простого поля Z_p.

2. Выполнение работы

Каноническая форма эллиптической кривой:
$$y^2 = x^3 + ax + b$$

Эллиптические кривые над двумерным рациональным полем

Понимаю, что задание не об этом, но я не могу не включить в отчет то, на что целый день потратил, когда только приступил к выполнению работы. Кривые над конечными полями с шестой страницы.

Порядок точки P – это такое число k, что kP = O, где O – бесконечно удаленная точка пространства. Умножение точки, лежащей на кривой на число k можно представить, как серию из k сложений точки с самой собой, где операция сложения определена так:

```
eps = 0.01
def pt_sum(pt1,pt2,a,b): #обе точки должны лежать на эллиптической кривой
    if(abs(pt1[0] - pt2[0]) < eps and abs(pt1[1] + pt2[1]) < eps):
        return (0,0)
    if(pt1 != pt2):
        m = (pt1[1] - pt2[1]) / (pt1[0] - pt2[0])
    else:
        m = (3*(pt2[0])**2 + a)/(2*pt2[1])
    x = m*m - pt1[0] - pt2[0]
    y = - pt1[1] + m*(pt1[0] - x)
    return (x,y)
```

Операция сложения основывается на том, что, если провести прямую через три точки эллиптической кривой, то сумма всех трех точек будет равна 0. Тогда, если провести прямую через две точки, лежащие на кривой и найти третью точку пересечения, то точка, обратная ей будет суммой этих двух точек. Если же такая кривая вертикальна, то сумма этих двух точек будет бесконечно удаленной точкой. То есть, наша задача – сложить некую точку с собой столько раз, чтобы получилась бесконечно удаленная точка, при этом, за шаг до достижения бесконечно удаленной точки, результатом k-1 предыдущих шагов будет точка,

лежащая под начальной (с такой же координатой x координатой y равной -y начальной).

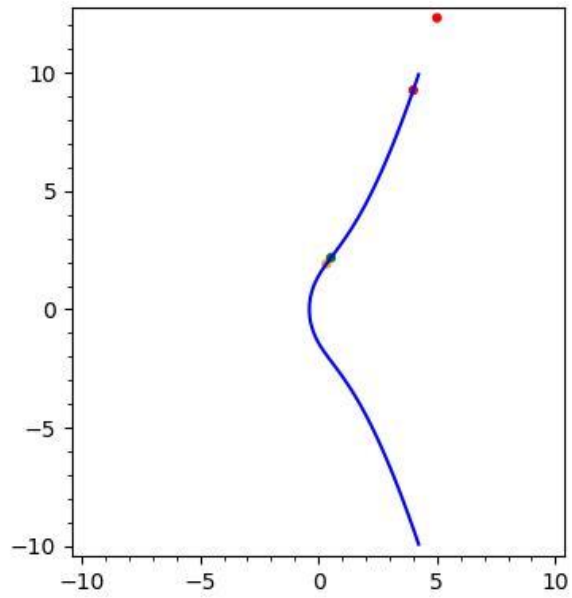
Для отслеживания и ограничения времени используется библиотека time.

Искать кривую, порядок точки которой можно определить за 10 минут полным перебором я решил перебором параметров. Для начала я прогнал несколько циклов, в который параметры кривой изменялись от -5 до 5 с шагом 2 и фиксированной начальной точкой, чтобы посмотреть, какие вообще точки получаются при сложениях.

В постановке задачи не совсем понятно, требуется ли, чтобы порядок любой точки на кривой находился за 10 минут, или же достаточно и одной. Я не уверен, что бывают кривые, где можно вычислить порядок любой точки, так что, для начала, решаю задачу для хотя бы одной.

В одном из моих циклов совершенно случайно порядок был найден:

```
a: 5,b: 2
(4, sqrt(86))
(10/19, 128/361*sqrt(38))
```

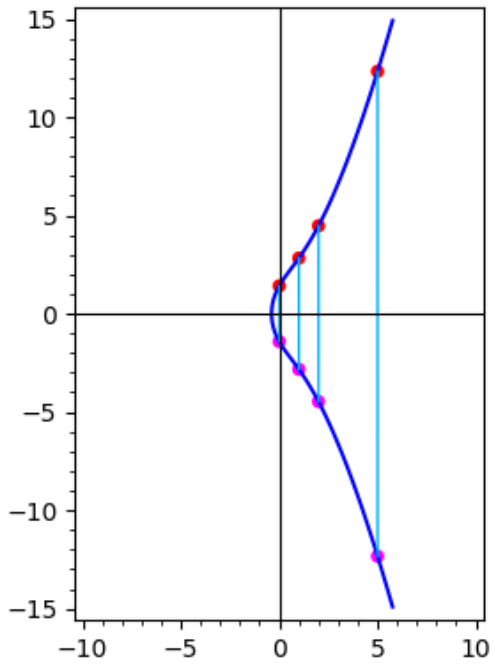



```
starting calculation
-0.360848086968593 0.385711098546611
49.5526754689648 -349.177254085506
-0.144610287136884 1.12868261900656
12.3108692185330 -43.9244683400387
0.277433672650858 1.84621837235744
5.31920877754951 -13.3827366930643
0.917406733344048 2.71277632466737
2.75766822830474 -6.06297572453714
calculation finished
    order: 418
    time passed: 55.08897876739502 sec
a: 5,b: 4
(4, 2*sqrt(22))
(30/77, 1171/5929*sqrt(154))
```

После дополнительных экспериментов с этой кривой, я обнаружил, что на ней порядок не вычисляется за 10 минут, например, для точек с x = 3 и 4, для точек с x = 0, 1, 2, 5 же, порядок вычислить можно.

Для наглядности я изобразил на графике линии, соединяющие начальные и предпоследние точки. Видно, что они довольно вертикальные.

```
curve: x^3 - y^2 + 5*x + 2

_____________________________
start point: (0.0,1.4142135623730951)
rank: 325, time spent: 17.38436269760132
fin point: (-0.00106586597305085, -1.41232810243365)
_____________________________
start point: (1.0,2.8284271247461903)
rank: 651, time spent: 35.56583547592163
fin point: (1.00213429212286, -2.83144627572441)
_____________________________
start point: (2.0,4.47213595499958)
rank: 536, time spent: 201.72330164909363
fin point: (1.99812402996831, -4.46857131737210)
_____________________________
start point: (5.0,12.328828005937952)
rank: 418, time spent: 49.16419529914856
fin point: (5.00248649423668, -12.3368963801319)
```

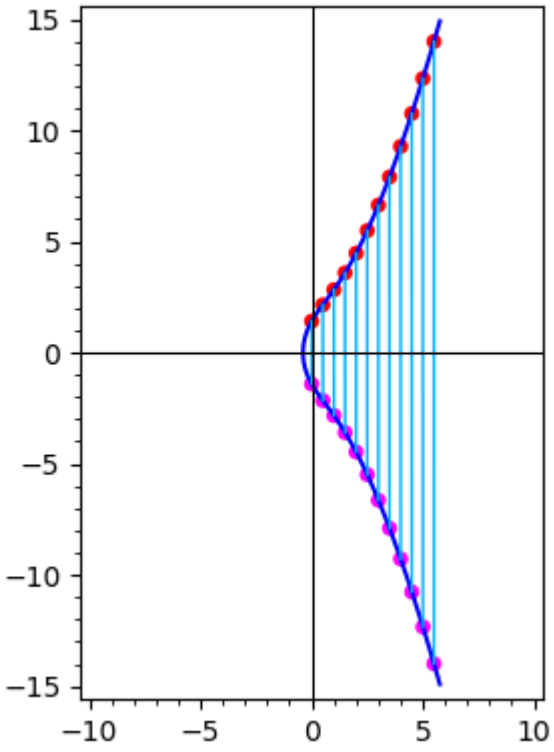


О критерии окончания.

Как видно из распечатки выше, значения предпоследних точек не равны в точности отражениям начальных точек. Я использовал проверку на равенство с точностью 0.01. Очевидно, если задавать более высокую точность, продолжительность вычислений сильно возрастет.

Интересно, что прежде я давал на вход только целые числа, потому программа проводила расчет в рациональных дробях. Из-за этого эффективность вычислений сильно падала. Сейчас же я, чтобы проверить точки типа 0.5, 1.5 и т.д. поделил координату x начальной точки на 2.0, отчего расчёты перешли в вещественное поле, и программа значительно ускорилась. Результат анализа той же кривой в вещественном поле:

```
curve: x^3 - y^2 + 5*x + 2

_____________________________
start point: (0.0,1.4142135623730951)
rank: 325, time spent: 0.27794432640075684
fin point: (-0.00106606584127178, -1.41232774864857)

_____________________________
start point: (0.5,2.1505813167606567)
rank: 336, time spent: 0.21098065376281738
fin point: (0.504421816787452, -2.15649133115283)

_____________________________
start point: (1.0,2.8284271247461903)
rank: 651, time spent: 0.3544602394104004
fin point: (1.00217807573548, -2.83150822866879)

_____________________________
start point: (1.5,3.588175023601831)
rank: 492, time spent: 0.2768843173980713
fin point: (1.50392394664276, -3.59460369503238)

_____________________________
start point: (2.0,4.47213595499958)
rank: 536, time spent: 0.3251974582672119
fin point: (1.99801332671800, -4.46836102269659)

_____________________________
start point: (2.5,5.488624600025037)
rank: 560, time spent: 0.2935044765472412
fin point: (2.50013085073816, -5.48890770859852)

_____________________________
start point: (3.0,6.6332495807108)
rank: 6334, time spent: 3.0657782554626465
fin point: (3.00134537486415, -6.63649522422190)

_____________________________
start point: (3.5,7.897784499465657)
rank: 2117, time spent: 1.0396451950073242
fin point: (3.50187523882001, -7.90274147790423)

_____________________________
start point: (4.0,9.273618495495704)
rank: 5363, time spent: 2.6335132122039795
fin point: (4.00238205829342, -9.28042724386713)

_____________________________
start point: (4.5,10.752906583803284)
rank: 2490, time spent: 1.2686707973480225
fin point: (4.50274420242051, -10.7612978942624)

_____________________________
start point: (5.0,12.328828005937952)
rank: 418, time spent: 0.21002578735351562
fin point: (5.00262518347566, -12.3373464760726)

_____________________________
start point: (5.5,13.995535002278405)
rank: 9749, time spent: 4.7740607261657715
fin point: (5.49891418016428, -13.9916848258813)
```


15
10
5
0
−5
−10
−15
−10
−5
0
5
10

Эллиптические кривые над конечным полем

Сложение точек выполняется так:

```
def pt_sum_fin(pt1,pt2,a,b,p): #обе точки должн лежать на эллиптической кривой
    if(pt1 == (0,0)):
        return pt2
    if(pt2 == (0,0)):
        return pt1
    if(abs(pt1[0] - pt2[0]) == 0 and abs(pt1[1] - pt2[1]) != 0):
        return (0,0)
    if(pt1 != pt2):
        m = ((pt1[1] - pt2[1]) / (pt1[0] - pt2[0])) % p
    else:
        m = ((3*(pt2[0])**2 + a)/(2*pt2[1])) % p
    x = ( m*m - pt1[0] - pt2[0] ) % p
    y = (- pt1[1] + m*(pt1[0] - x) ) % p
    return (x,y)
```

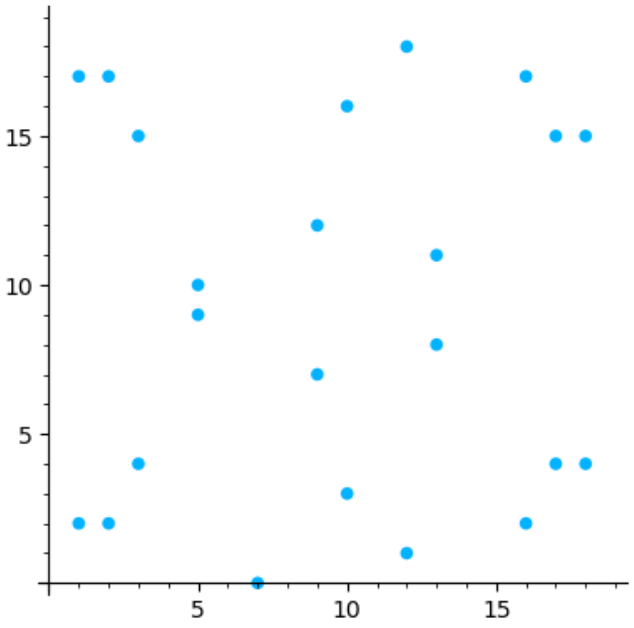
Функция, генерирующая все точки криво над конечным полем характеристики p. Генерировать точки будем наивным алгоритмом.

```
#эллиптическая кривая над конечным полем
def func (x, a, b):
    return (x**3 + a*x + b)

def finiteCurve(a,b,p,silent = false):
    if(4*a**3 + 27*b**2)%p == 0 :
        return ()

    v1 = [(k,f) for k in range (p)
                for f in range (p)
                    if (f*f) % p == func(k,a,b) % p]

    plt = point((0,0), rgbcolor=hue(0), pointsize=1)
    for i in v1:
        plt += point(i, rgbcolor=hue(0.55), pointsize=30)
    if(not silent):
        show(plt,aspect_ratio = 1, xmin = 0,xmax=p,ymin=0,ymax=p)
    return v1
```

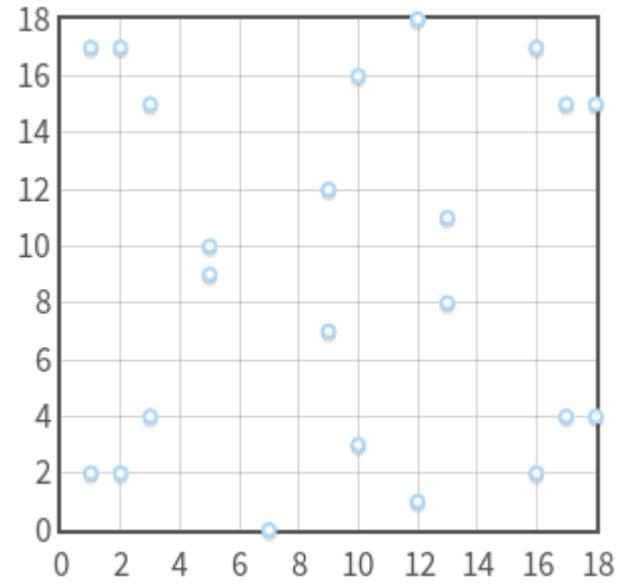
Пример работы этой функции (параметры из примера в статье «Доступно о криптографии на эллиптических кривых»)

```
a = -7
b = 10
p = 19
```

```
pts = finiteCurve(a,b,p)
```



Та же кривая в примере из статьи на Хабре:



Видно, что алгоритм работает правильно.

Убедимся, что последовательно сложение точки с собой порождает циклический процесс:

```
print("point: {}".format(ptstart))
print("______")
ptcur = ptstart
print("0*P =\t (0, 0)")
for k in range (100):
    print("{}*P =\t {}".format(k + 1,ptcur))
    ptcur = pt_sum_fin(ptstart,ptcur,a,b,p)
```

```
point: (1, 2)
______
0*P =    (0, 0)
1*P =    (1, 2)
2*P =    (18, 15)
3*P =    (9, 12)
4*P =    (7, 0)
5*P =    (9, 7)
6*P =    (18, 4)
7*P =    (1, 17)
8*P =    (0, 0)
9*P =    (1, 2)
10*P =   (18, 15)
11*P =   (9, 12)
12*P =   (7, 0)
13*P =   (9, 7)
14*P =   (18, 4)
15*P =   (1, 17)
15*P =   (0, 0)
```

Реализуем отыскание порядка точки кривой над конечным полем:

```python
def orderPt(ptf,a,b,p,trace = 0):
    ptsum = ptf
    it = 0
    while (ptsum != (0,0) or it == 0):
        it += 1
        ptsum = pt_sum_fin(ptf,ptsum,a,b,p)
        if(trace and it % trace == 0):
            print("step: {} point: {}".format(it,ptsum))
    return it + 1
```

И найдем с ее помощью порядок всех точек построенной кривой:

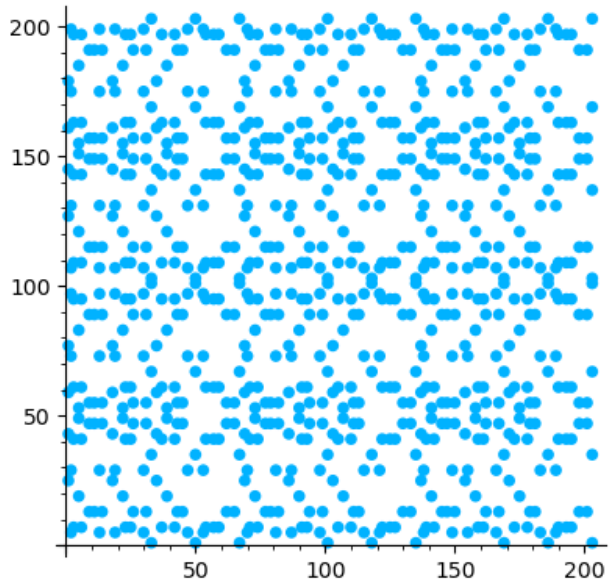
```python
i = 0
for pt99 in pts:
    try:
        ord = orderPt(pt99,a,b,p)
        print("{}  \torder: {}".format(pt99, ord))
    except ZeroDivisionError:
        print("{}  \tunable to calculate order".format(pt99))
```

```
(1, 2)          order: 339
(1, 17)         order: 54
(2, 2)          order: 345
(2, 17)         order: 88
(3, 4)          order: 54
(3, 15)         order: 337
(5, 9)          order: 161
(5, 10)         order: 168
(7, 0)          unable to calculate order
(9, 7)          order: 338
(9, 12)         order: 182
(10, 3)         order: 115
(10, 16)        order: 5
(12, 1)         order: 121
(12, 18)        order: 179
(13, 8)         order: 361
(13, 11)        order: 40
(16, 2)         order: 40
(16, 17)        order: 359
(17, 4)         order: 346
(17, 15)        order: 69
(18, 4)         order: 324
(18, 15)        order: 361
```

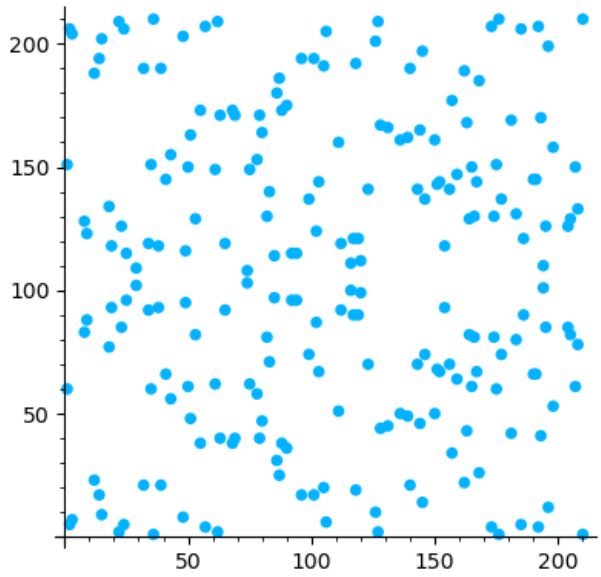
Видим, что в одной из точек порядок найти не удалось. Сама ошибка, которая может произойти, выбрасывается функцией %, которая в определенных случаях при рациональных операндах вычисляет функцию inverse_mod(a, b), а эта функция выдает ошибку ZeroDivisionError, если a – делитель b. Физически же, можно заметить, что точка (7,0), порядок которой вычислить не удалось, не имеет симметричной пары, так что неудивительно, что порядок не считается.

Также я заметил, что если число p – не простое, то не удается найти порядок у очень большого количества точек.

К тому же, если число p – четное, то нанесенные на плоскость точки образуют часто повторяющуюся структуру:



С простым p этого не происходит:



Полагаю, чем больше делителей у p, тем более структура точек циклична.

В цикличной же структуре, видимо, порядок точки по какой-то причине часто найти нельзя даже если симметричная точка есть. Возможно сложение запирается в каком-то подцикле без нуля и не попадает в нужную точку, может что-то еще. Этот вопрос я исследовать не стал.

Кривая, порядок каждой точки которой находится полным перебором

Одно из интерпретаций поставленной задачи – найти такую кривую, у которой порядок каждой точки можно найти. Так как мы видели, что даже кривые над полем простой характеристики имеют точки, порядок которых найти не удается, это может быть непростая задача.

Впрочем, выполнив незамысловатый алгоритм перебора, я быстро нашел такую.

Алгоритм:

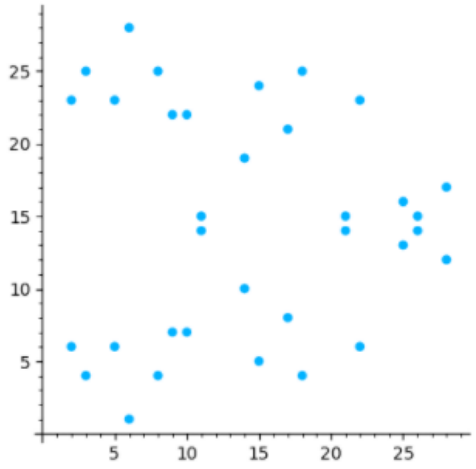
```
def searchIdeal():
    for a in range(-10,10, 3):
        for b in range(-10,10, 3):
            for p in range(31,15,-2):
                print("a = {} b = {} p = {}".format(a,b,p))
                ptsc = finiteCurve(a,b,p,silent = true)
                print("points total: {}".format(len(ptsc)))
                if(len(ptsc) == 0):
                    continue
                found = 0
                err = 0
                i = 0
                for pt99 in ptsc:
                    if(i ==0):
                        i+= 1
                        continue
                    try:
                        ord = orderPt(pt99,a,b,p)
                        found += 1
                    except ZeroDivisionError:
                        err += 1
                print("\tpoints found\t {}".format(found))
                print("\terror in\t {}".format(err))
                if(err == 0):
                    return(a,b,p)
```

Найденная кривая:

```
searchIdeal()
```

```
a = -10 b = -10 p = 31
points total: 31
        points found     29
        error in         1
a = -10 b = -10 p = 29
points total: 34
        points found     33
        error in         0

(-10, -10, 29)
```

Проверка:




```
(2, 23)         order: 35
(3, 4)          order: 35
(3, 25)         order: 35
(5, 6)          order: 7
(5, 23)         order: 7
(6, 1)          order: 35
(6, 28)         order: 35
(8, 4)          order: 35
(8, 25)         order: 35
(9, 7)          order: 35
(9, 22)         order: 35
(10, 7)         order: 35
(10, 22)        order: 35
(11, 14)        order: 35
(11, 15)        order: 35
(14, 10)        order: 35
(14, 19)        order: 35
(15, 5)         order: 35
(15, 24)        order: 35
(17, 8)         order: 5
(17, 21)        order: 5
(18, 4)         order: 35
(18, 25)        order: 35
(21, 14)        order: 35
(21, 15)        order: 35
(22, 6)         order: 35
(22, 23)        order: 35
(25, 13)        order: 5
(25, 16)        order: 5
(26, 14)        order: 7
(26, 15)        order: 7
(28, 12)        order: 7
(28, 17)        order: 7
```

Также я нашел несколько таких кривых над полями с большим p, Перебирая p по простым числам, а не по нечетным. Например:

```
searchIdealP(myPrimes)
```

```
a = -10 b = -10 p = 997
points total: 990
        points found     989
        error in         0

(-10, -10, 997)
```

Кривая, порядок одной из точек которой находится за 10 минут

Полагаю, это и есть правильная интерпретация поставленной задачи. Так как порядок точки ищется несравненно быстрее отыскания всех точек кривой наивным алгоритмом, нужно:
1. Найти какую-нибудь точку кривой с простым p.
2. Попробовать рассчитать ее порядок, замерив время.
3. Если получится << 10 мин, увеличить p.

В качестве a и b я брал случайные небольшие числа, всю сложность вычислений перекладывая на большое p.

Для начала я написал функцию поиска больших простых чисел:

```
def highPrime(l):
    for i in range(l, 0, -1):
        if(is_prime(i)):
            return(i)
```

```
highPrime(120000000)
```

```
119999987
```

И функцию отыскания единственной точки кривой наивным алгоритмом:

```
def finiteCurvePt(a,b,p):
    if(4*a**3 + 27*b**2)%p == 0 :
        return ()

    for k in range (p):
        for f in range (p):
            if (f*f) % p == func(k,a,b) % p:
                return (k,f)
```

Однако и эта функция оказалась слишком медленной

Несмотря на это она примерно за 20 минут нашла одну точку кривой с таким p:

```
p = 99999989
```

(a и b я не записал, а затем изменил, но точку новой кривой искать не стал, потому что 20 минут)

Вот эта точка:

```
pte
```

```
(2, 21571378)
```

Ее порядок нашелся за 7 минут.

```
try:
    ts = time.time()
    ord = orderPt(pt,a,b,p)
    tf = time.time()
    print("{}  \torder: {}".format(pt, ord))
    print(" found in {} seconds".format(tf-ts))
except ZeroDivisionError:
    print("{}  \tunable to calculate order".format(pt))
```

```
(2, 21571378)   order: 24995904
 found in 427.37215995788574
```

Порядок нашелся за 427 секунд, что есть примерно 7 минут

Так как столь медленный алгоритм отыскания точки – не дело, я решил реализовать более эффективный его аналог, работающий так:

1. Выбрать случайное x.
2. Вычислить $f = x^3 + ax + b$.
3. Вычислить символ Лежандра f по p.
4. Если f - квадратичный невычет, перейти к пункту 1.
5. Вычислить y - квадратный корень из f по модулю p (например, с помощью алгоритма Тонелли-Шенкса).

(https://intuit.ru/studies/courses/13837/1234/lecture/31193?page=3)

Нужные алгоритмы я взял отсюда:

(https://codereview.stackexchange.com/questions/43210/tonelli-shanks-algorithm-implementation-of-prime-modular-square-root)

Получилась такая функция, которая ищет точку кривой со схожими параметрами почти мгновенно:

```
def finiteCurvePtFast(a,b,p):
    if(4*a**3 + 27*b**2)%p == 0 :
        return ()

    x = 0
    fin = 0
    while not fin:
        x+=1
        f = func(x,a,b)
        lgn = legendre_symbol(f,p)
        if(lgn < 0):
            continue
        return (x,prime_mod_sqrt(f,p)[0])
```

В качестве быстрого теста, я нашел одну точку случайной кривой с тем же p, что и в прошлый раз и попытался посчитать ее порядок.

```
newPt = finiteCurvePtFast(12, 34, 99999989)
print(newPt)
```

```
(1, 8643338)
```

```
try:
    ts = time.time()
    ord = orderPt(pt,a,b,p,trace = 1000000)
    tf = time.time()
    print("{}  \torder: {}".format(pt, ord))
    print(" found in {} seconds".format(tf-ts))
except ZeroDivisionError:
    print("{}  \tunable to calculate order".format(pt))
```

```
step: 1000000 point: (1853808, 43388433)
step: 2000000 point: (80678179, 35239511)
step: 3000000 point: (29217713, 40061258)
step: 4000000 point: (78526004, 3071301)
step: 5000000 point: (64679438, 69731602)
step: 6000000 point: (22453338, 21121757)
step: 7000000 point: (75020076, 39827742)
step: 8000000 point: (90117766, 59628260)
step: 9000000 point: (42665367, 51139036)
step: 10000000 point: (37720730, 51769192)
step: 11000000 point: (71996685, 3222107)
step: 12000000 point: (82575887, 80456610)
step: 13000000 point: (6732967, 62472323)
step: 14000000 point: (59115527, 25480328)
step: 15000000 point: (95545359, 97924136)
step: 16000000 point: (42786581, 67231469)
step: 17000000 point: (72656008, 89322400)
step: 18000000 point: (26684969, 66015249)
step: 19000000 point: (76878617, 73261357)
step: 20000000 point: (71327591, 42774109)
step: 21000000 point: (82934126, 49002587)
step: 22000000 point: (54512434, 78870847)
step: 23000000 point: (77708591, 48891158)
step: 24000000 point: (37738019, 24830824)
step: 25000000 point: (53222083, 34220805)
step: 26000000 point: (79790244, 90029804)
step: 27000000 point: (34913443, 24753781)
step: 28000000 point: (74768753, 71693253)
step: 29000000 point: (83482111, 78339566)
step: 30000000 point: (33726511, 21804904)
step: 31000000 point: (43422856, 45273770)
step: 32000000 point: (97066837, 23030692)
step: 33000000 point: (89429247, 6471187)
(2, 21571378)   order: 33332436
 found in 553.9865818023682 seconds
```

На удивление, порядок нашелся за 553 секунды, а это 9,2 минуты, что приемлемо близко к 10 минутам. Если бы число сильно отличалось, я бы выполнял поиск циклом, который замерял время отыскания порядка, и, если оно отличалось от 10 минут больше, чем на минуту, брала p на 1/10 порядка больше или меньше текущей. Но раз мне так повезло с первого раза, не буду тратить на это время.

Итак, параметры найденной кривой:

a = 12
b = 34
p = 99999989

Алгоритмы для облегчения / ускорения решения задачи

Я знаю, что существуют алгоритмы взлом задачи дискретного логарифмирования, т.е. отыскания для точек Q и P такого k, что Q = k*P.

В частности, это алгоритмы Baby-step, giant-step и ρ Полларда.

Полагаю, их можно применить для отыскания k для точки, порядок которой надо найти и ее симметричной пары, которая может быть очень быстро найдена с помощью функции finiteCurvePtFast. Затем прибавить к найденному k единицу и получить порядок точки P.

Характеристики вычислителя

Система компьютерной алгебры SageMath 9.2, процессор AMD FX(tm)-6300 Six-Core Processor 3.50 GHz, память 16ГБ, 64-разрядная система.

3. Вывод

В ходе выполнения этой лабораторной работы, я получил опыт исследования функций над конечными полями, в частности, эллиптических кривых.

Вообще с каждым днем выполнения работы мне открывались какие-то новые интересные аспекты данной темы. Конечно, я мог бы получше подготовиться и заранее прочитать где-нибудь, что для того чтобы порядок точек уверенно находился, p должно быть простым, но, по-моему, гораздо больше пользы и опыта принесло то, что я сам это выяснил в ходе анализа кривых.

В итоге, правда, решению самой задачи, поставленной в лабораторной работе у меня уделено очень мало внимания, да и с ним тоже можно всякие эксперименты поделать, графики зависимости времени отыскания от p построить, посмотреть, что будет при больших a и b, но я и так уже на два дня опаздываю, так что не буду этого делать. Вообще никогда бы не подумал, что мне будет интересно работать с эллиптическими кривыми.

Еще я получил полезный опыт, когда минут десять думал, как рассчитывать большие простые числа, уже собирался решето Эратосфена писать, когда понял, что на каждом шаге можно просто проверять встроенной в sage функцией is_prime. Действительно, порой можно очень сильно усложнить себе жизнь с ничего.

А, ну и я этого нигде не упоминал, но во всех случаях параметры кривой – целые, точки – целые, так что рассматриваемые кривые именно над Z_p.