

Rank of $\Gamma_0^{ab}(\mathfrak{p})$ for some prime ideals \mathfrak{p} of $\mathbb{Z}[i]$

Marius Beceanu

May 20, 2003

Abstract

This paper serves as a computational confirmation of the Taniyama-Shimura conjecture for $\mathbb{Z}[i]$, in the sense that no counterexample occurs among the computed values. Its theoretical part contains no original result, being just a compilation of the sources listed in the bibliography and information received from Professor Peter Sarnak. The computational part, on the other hand, gives some previously uncomputed values for the group rank mentioned in the title of the paper, and attempts to formulate some possible bounds for these values. Indeed, it seems that $r(\Gamma_{\mathfrak{p}}^{ab}) = \mathcal{O}(\log N(\mathfrak{p}))$ and it is also likely that $r(\Gamma_{\mathfrak{p}}^{ab}) = \Omega(\log N(\mathfrak{p}))$ infinitely often.

1 Introduction

Consider the group $PSO_4(f, \mathbb{Z}) = \{A \in M_4(\mathbb{Z}) \mid f(Au, Au) = f(u, u)\} / \{\pm I_4\}$, where $f(u, u) = u_1^2 + u_2^2 - u_3u_4$. The subgroup $PSO_4^+(f, \mathbb{Z})$ of elements of determinant 1 is generated by the (image of the) elements

$$u_1 = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad u_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \quad \text{and } u_3 = \begin{pmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -2 & 0 & 1 & 1 \end{pmatrix}.$$

There exists an isomorphism ϕ between $PSO_4^+(f, \mathbb{Z})$ and $PGL_2(\mathbb{Z}[i])$ given by

$$\phi(u_1) = \begin{pmatrix} i & 0 \\ 0 & 1 \end{pmatrix}, \quad \phi(u_2) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \phi(u_3) = \begin{pmatrix} 1 & 0 \\ 1 & -1 \end{pmatrix}.$$

We can identify the hyperbolic 3-space \mathbb{H}^3 with the set $\{v \in \mathbb{P}^3(\mathbb{R}) \mid f(v, v) > 0\}$, where f is the quadratic form defined above. Then, by means of the isomorphism ϕ , we obtain an action of $PGL_2(\mathbb{Z}[i])$ on \mathbb{H}^3 , and, in particular, of $PSL_2(\mathbb{Z}[i])$ (which is a subgroup of the former), also on \mathbb{H}^3 .

For an ideal \mathfrak{p} of $\mathbb{Z}[i]$, define $\Gamma_{\mathfrak{p}} = \{A \in SL_2(\mathbb{Z}[i]) \mid A \equiv \begin{pmatrix} * & * \\ 0 & * \end{pmatrix} \pmod{\mathfrak{p}}\}$. We are interested in the modular forms of $\Gamma_{\mathfrak{p}}$, which are the harmonic 1-forms $\omega \in \Omega^1(\mathbb{H}^3, \mathbb{C})$ invariant under $\Gamma_{\mathfrak{p}}$, i. e. $\gamma^*(\omega) = \omega$, $\forall \gamma \in \Gamma_{\mathfrak{p}}$. Each

such form corresponds to a 1-form ω_0 on the fundamental domain of $\Gamma_{\mathfrak{p}}$ (defined as $\mathbb{H}^3/\Gamma_{\mathfrak{p}}$) such that $\omega(z) = \omega_0([z])$. But we know, by a suitable version of Hodge's theorem, that every cohomology class in $\mathbb{H}^3/\Gamma_{\mathfrak{p}}$ has a unique harmonic representative. Thus what we want to compute is the dimension of $H^1(\mathbb{H}^3/\Gamma_{\mathfrak{p}})$, which is furthermore equal to the rank of the first homology group.

Proposition 1.1 *The rank of the first homology group $H_1(\mathbb{H}^3/\Gamma_{\mathfrak{p}})$ is equal to the rank of the abelianized group $\Gamma_{\mathfrak{p}}^{ab}$.*

Consider the group $N_{\mathfrak{p}} = \{A \in SL_2(\mathbb{Z}[i]) \mid A \equiv \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \pmod{\mathfrak{p}}\}$. This is a normal subgroup of $SL_2(\mathbb{Z}[i])$; the quotient group is isomorphic to $SL_2(\mathbb{Z}[i]/\mathfrak{p})$.

We say that the prime ideal $\mathfrak{p} \in \mathbb{Z}[i]$ has degree 1 if it is not invariant under conjugation, i.e. if $N(\mathfrak{p}) = p$ is a $4k + 1$ -prime. In the sequel, let me assume this is the case. Then we have $SL_2(\mathbb{Z}[i]/\mathfrak{p}) \cong SL_2(\mathbb{F}_p)$.

Proposition 1.2 *For $H \triangleleft G$, G acts on H on the left by $x.h = xhx^{-1}$. This induces an action of $(G/H)/Z(G/H) = G/(Z(G)H)$ on H^{ab} given by $[x].[h] = [xhx^{-1}]$. Both operations have the property that $x.(h_1h_2) = (x.h_1)(x.h_2)$.*

In our case, this gives an action of $PSL_2(\mathbb{F}_p)$ on $N_{\mathfrak{p}}^{ab}$. Because of the last property $(x.(h_1h_2) = (x.h_1)(x.h_2))$, $N_{\mathfrak{p}}^{ab}$ can be described as a $PSL_2(\mathbb{F}_p)$ -module. Then, the tensor product $N_{\mathfrak{p}}^{ab} \otimes_{\mathbb{Z}} \mathbb{C}$ can be seen as a $PSL_2(\mathbb{F}_p)$ -module as well. This tensor product is also a \mathbb{C} -vector space of finite dimension, since $N_{\mathfrak{p}}^{ab}$ is finitely generated.

The $PSL_2(\mathbb{F}_p)$ -module structure of this vector space gives us a matrix representation of the group $PSL_2(\mathbb{F}_p)$, that can be in turn expressed as a direct sum of irreducible representations. With respect to that we have

Proposition 1.3 (also in [1], p. 523) *The multiplicity of the irreducible representation of degree p of $PSL_2(\mathbb{F}_p)$ in $N_{\mathfrak{p}}^{ab} \otimes \mathbb{C}$ is equal to the rank of $\Gamma_{\mathfrak{p}}^{ab}$.*

Furthermore, this multiplicity can be computed using

Proposition 1.4 (see [1], p. 532, for proof) *Consider the linear system*

$$(1.1) \quad u_i - u_{-i} = 0$$

$$(1.2) \quad u_i + u_{Wi} = 0$$

$$(1.3) \quad u_i + u_{Si} + u_{S^2i} = 0$$

$$(1.4) \quad u_i + u_{Yi} + u_{Y^2i} = 0$$

where i runs over the $p + 1$ elements of $\mathbb{P}^1(\mathbb{F}_p)$ and W , S , and Y are the transformations

$$Wi = \frac{1}{i}, \quad Si = 1 - \frac{1}{i}, \quad \text{and} \quad Yi = -\rho + \frac{1}{i},$$

ρ being the solution to $\rho^2 \equiv (-1) \pmod{p}$.

The system can be solved by numerical computations, which is what I did in this paper for $p < 45000$.

This identity has a geometrical interpretation. Namely, we can construct a 3-complex that is homeomorphic to $H_1(\mathbb{H}^3/\Gamma_{\mathfrak{p}})$ and whose homology group $H_1(M)$ is isomorphic to a group described by the relations (1.1)-(1.4).

2 Theoretical results

The Taniyama-Shimura conjecture, among other things, states that every elliptic curve over $\mathbb{Z}[i]$ is modular, which means that there exists a modular form (of the cohomological type described in the previous section) with the same L -function, and, in particular, with the same conductor. In order for this to be true, the dimension of the space of modular forms of conductor \mathfrak{p} has to be greater than or equal to the number of distinct (non-isogeneous) elliptic curves of the same conductor. Thus, using the ranks of $\Gamma_{\mathfrak{p}}^{ab}$ computed below and a list of elliptic curves over $\mathbb{Z}[i]$ provided by Andrew Booker, I was able to perform a partial verification of the conjecture, in the sense that for every given elliptic curve I found that the dimension of the space of modular forms with the same conductor was at least 1. Furthermore, in the one situation when *two* distinct elliptic curves had the same conductor, the dimension of the vector space of forms was at least 2. These statements are illustrated in Table 2, where each elliptic curve is given in the form $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$.

3 Numerical experiments

By following the method outlined above, I computed the rank of $\Gamma_0^{ab}(\mathfrak{p})$ for $p < 45000$. In Table 3 are listed those values of p for which the rank is non-zero, as well as the corresponding Betti numbers.

Figure 3 plots the rank of $\Gamma_{\mathfrak{p}}^{ab}$ for $p < 45000$. It seems plausible that $r(\Gamma_{\mathfrak{p}}^{ab}) = \mathcal{O}(\log p)$, with a bounding constant of around 0.75.

Figure 3 shows the density of primes p for which $r(\Gamma_{\mathfrak{p}}^{ab})$ is 0 among all $4k + 1$ -primes smaller than a given number. It seems possible that this density could go to 0, but the computations have not yet provided sufficient evidence to be certain about it. It presently looks like the density tends to a limit close to 0.1.

| $N(\mathfrak{p})$ | Elliptic curve(s) | $r(\Gamma_{\mathfrak{p}}^{ab})$ |
|-------------------|--|---------------------------------|
| 233 | $0, 1 + i, i, i, 0$ | 1 |
| 257 | $1, i, i, 0, 0$ | 1 |
| 277 | $i, -1 + i, 1 + i, 1 - 2i, 0$ | 1 |
| 509 | $1 + i, -1, i, 1 - i, 0$ | 1 |
| 757 | $1 + i, -1 + i, i, 1 - 3i, i$ | 2 |
| 853 | $1 + i, -1 - i, 1, -1, 0$ | 2 |
| 1049 | $0, -1 - i, i, 0, 0$ | 1 |
| 1153 | $1, 1 - i, 1, -1 - i, -1$ | 1 |
| 1373 | $1, 1, i, 0, 0$ | 1 |
| 1493 | $1 + i, 0, i, 1, 0$ | 1 |
| 1777 | $1, 1 - i, i, 1 - 2i, -i$ | 3 |
| 1997 | $i, 1, i, 1 + i, 0$ | 1 |
| 2053 | $1 + i, -1 + i, 1, -1 - 2i, 0$ | 1 |
| 2521 | $1, -1, i, -i, 0$ | 1 |
| 3109 | $1 + i, -1 - i, i, 1 - i, 0$ | 1 |
| 3361 | $i, i, 0, -i, 0$ | 1 |
| 3637 | $1 + i, 1 + i, 1, 0, 0$ | 2 |
| 4013 | $i, i, i, 1 - i, 0$ | 1 |
| 4177 | $1, -1, 0, -i, 0$ | 1 |
| 4289 | $i, 1 - i, i, -1 - i, -1$ | 1 |
| 4597 | $1, 0, i, -1 - 2i, -i$ | 1 |
| 4621 | $1 + i, -i, i, 0, 0$ | 1 |
| 4721 | $i, -1, 0, -i, 0$ | 1 |
| 5021 | $i, 1, i, 3 + 4i, -3 + i$ | 1 |
| 5237 | $1, i, i, -i, 0$ | 1 |
| 5309 | $1, 0, 1 + i, 1 - 2i, -1 - i$ $1 + i, 1 - i, i, 1, 0$ | 6 |

Table 1: Computational support for the Taniyama-Shimura conjecture in the case of $\mathbb{Q}[i]$

| $N(\mathfrak{p})$ | $r(\Gamma_{\mathfrak{p}}^{ab})$ | $N(\mathfrak{p})$ | $r(\Gamma_{\mathfrak{p}}^{ab})$ | $N(\mathfrak{p})$ | $r(\Gamma_{\mathfrak{p}}^{ab})$ | $N(\mathfrak{p})$ | $r(\Gamma_{\mathfrak{p}}^{ab})$ | $N(\mathfrak{p})$ | $r(\Gamma_{\mathfrak{p}}^{ab})$ |
|-------------------|---------------------------------|-------------------|---------------------------------|-------------------|---------------------------------|-------------------|---------------------------------|-------------------|---------------------------------|
| 137 | 1 | 233 | 1 | 257 | 1 | 277 | 1 | 433 | 2 |
| 509 | 1 | 569 | 1 | 709 | 2 | 733 | 1 | 757 | 2 |
| 853 | 2 | 941 | 3 | 953 | 2 | 977 | 1 | 1009 | 1 |
| 1013 | 1 | 1021 | 1 | 1049 | 1 | 1153 | 1 | 1277 | 1 |
| 1321 | 2 | 1373 | 1 | 1489 | 1 | 1493 | 1 | 1549 | 2 |
| 1753 | 1 | 1777 | 3 | 1901 | 2 | 1973 | 2 | 1997 | 1 |
| 2053 | 1 | 2081 | 1 | 2377 | 1 | 2441 | 1 | 2521 | 1 |
| 2609 | 1 | 2657 | 2 | 2729 | 1 | 2753 | 2 | 2917 | 1 |
| 3109 | 1 | 3313 | 2 | 3361 | 1 | 3469 | 2 | 3529 | 2 |
| 3637 | 2 | 3877 | 2 | 3929 | 1 | 4013 | 1 | 4177 | 1 |
| 4289 | 1 | 4421 | 1 | 4517 | 5 | 4597 | 1 | 4621 | 1 |
| 4721 | 1 | 5021 | 1 | 5113 | 3 | 5237 | 1 | 5309 | 6 |
| 5741 | 1 | 5749 | 1 | 5801 | 1 | 5849 | 2 | 5857 | 2 |
| 6029 | 1 | 6361 | 1 | 6689 | 2 | 6701 | 1 | 6781 | 1 |
| 6793 | 1 | 6857 | 1 | 6949 | 1 | 7001 | 1 | 7069 | 1 |
| 7121 | 1 | 7577 | 2 | 7793 | 1 | 7937 | 1 | 8081 | 2 |
| 8297 | 1 | 8377 | 1 | 8461 | 1 | 8513 | 1 | 8537 | 1 |
| 8753 | 1 | 8893 | 4 | 9041 | 1 | 9349 | 2 | 9413 | 1 |
| 9629 | 2 | 10357 | 1 | 10369 | 1 | 10477 | 1 | 10657 | 1 |
| 10729 | 1 | 10861 | 1 | 10937 | 1 | 11437 | 2 | 11701 | 1 |
| 11953 | 1 | 11969 | 3 | 12113 | 1 | 12253 | 1 | 12269 | 2 |
| 12373 | 1 | 12553 | 1 | 12941 | 2 | 12953 | 2 | 13093 | 2 |
| 13381 | 1 | 13457 | 1 | 13477 | 2 | 13633 | 1 | 15161 | 1 |
| 15497 | 1 | 15569 | 1 | 15629 | 1 | 15749 | 1 | 15761 | 2 |
| 16097 | 1 | 16349 | 1 | 16649 | 1 | 16673 | 1 | 16921 | 2 |
| 17021 | 4 | 17033 | 2 | 17209 | 1 | 17921 | 1 | 18289 | 1 |
| 18553 | 1 | 18701 | 1 | 18757 | 2 | 18869 | 1 | 18913 | 1 |
| 19213 | 1 | 19237 | 2 | 19417 | 1 | 19841 | 1 | 19937 | 2 |
| 19997 | 1 | 20129 | 2 | 20297 | 1 | 20549 | 1 | 20773 | 7 |
| 21001 | 1 | 21221 | 3 | 21821 | 2 | 21893 | 2 | 21977 | 1 |
| 22013 | 1 | 22073 | 1 | 22157 | 2 | 22229 | 1 | 22433 | 1 |
| 22441 | 3 | 22721 | 1 | 22777 | 2 | 22861 | 2 | 23209 | 3 |
| 23293 | 4 | 23357 | 2 | 23629 | 1 | 23977 | 2 | 23993 | 1 |
| 24121 | 3 | 24133 | 1 | 24329 | 1 | 24421 | 1 | 24469 | 1 |
| 24953 | 2 | 25037 | 2 | 25457 | 1 | 25541 | 1 | 25589 | 1 |
| 25601 | 2 | 25997 | 2 | 26113 | 1 | 26153 | 1 | 26513 | 1 |
| 26557 | 2 | 26561 | 2 | 27073 | 1 | 27541 | 2 | 27941 | 2 |
| 27961 | 3 | 27997 | 2 | 28201 | 1 | 28393 | 4 | 28813 | 1 |
| 29077 | 1 | 29209 | 1 | 29429 | 1 | 29473 | 1 | 29633 | 1 |
| 29873 | 1 | 30137 | 1 | 30253 | 1 | 30269 | 1 | 30313 | 1 |
| 30469 | 2 | 31477 | 1 | 31729 | 1 | 31769 | 1 | 32173 | 2 |
| 32933 | 2 | 33029 | 2 | 33349 | 2 | 33377 | 1 | 33413 | 1 |
| 34061 | 1 | 34129 | 1 | 34369 | 1 | 34421 | 1 | 34429 | 1 |
| 34457 | 1 | 34501 | 1 | 34841 | 1 | 35149 | 2 | 35281 | 1 |
| 35521 | 1 | 35573 | 1 | 35677 | 1 | 36013 | 2 | 36161 | 1 |
| 36497 | 1 | 36721 | 1 | 36857 | 1 | 36901 | 1 | 37097 | 2 |
| 37273 | 1 | 37649 | 2 | 38201 | 1 | 38333 | 2 | 38461 | 1 |
| 38557 | 1 | 38629 | 2 | 38917 | 1 | 38977 | 1 | 39041 | 1 |
| 39541 | 1 | 39709 | 1 | 39829 | 1 | 40529 | 1 | 41213 | 1 |
| 41281 | 1 | 41593 | 1 | 42349 | 1 | 42461 | 1 | 42641 | 3 |
| 43177 | 1 | 43189 | 1 | 43237 | 1 | 43313 | 2 | 43721 | 1 |
| 43969 | 1 | 44021 | 2 | 44201 | 1 | 44389 | 2 | 44549 | 1 |

Table 2: Non-zero ranks of $\Gamma_{\mathfrak{p}}^{ab}$ and corresponding values of $N(\mathfrak{p})$, for $N(\mathfrak{p}) < 45000$

| $r(\Gamma_{\mathfrak{p}}^{ab})$ | Number of primes |
|---------------------------------|------------------|
| 7 | 1 |
| 6 | 1 |
| 5 | 1 |
| 4 | 4 |
| 3 | 10 |
| 2 | 66 |
| 1 | 177 |
| 0 | 2060 |

Table 3: Distribution of ranks for $\Gamma_{\mathfrak{p}}^{ab}$, when $p = 4k + 1$, $p < 45000$

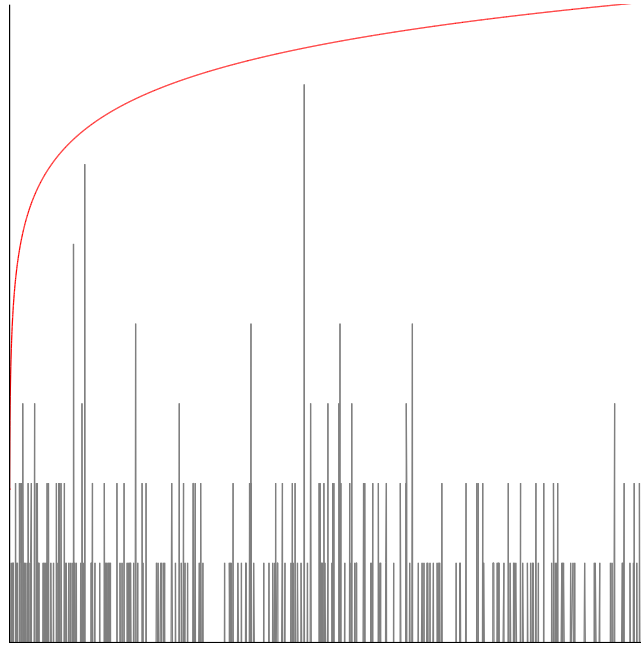


Figure 1: The rank of $\Gamma_0^{ab}(\mathfrak{p})$ for $p < 45000$.

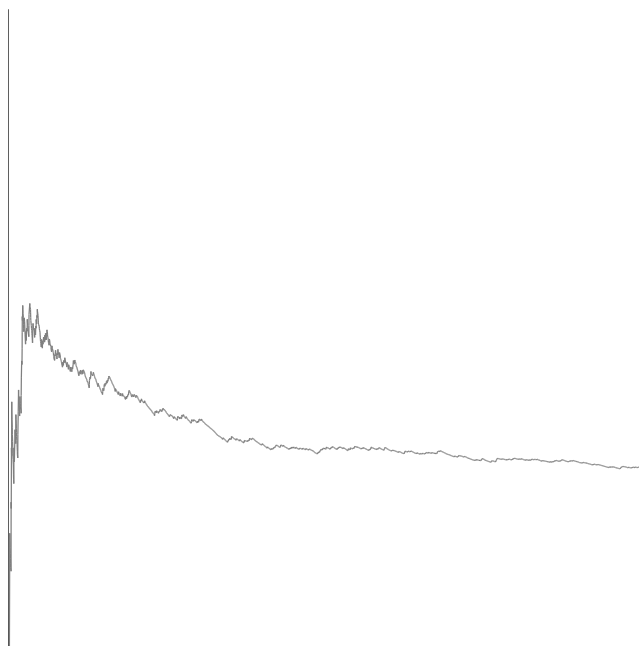


Figure 2: Density of primes p at which the rank of $\Gamma_0^{ab}(\mathfrak{p})$ is non-zero, for $p < 45000$.

References

- [1] F. Grunewald, H. Helling, J. Mennicke, *SL₂ over Complex Quadratic Number Fields. I*, Algebra i logika, 5(1978), pp. 512-580.

Appendix

This is the program that I wrote in order to evaluate the rank of $\Gamma_0^{ab}(\mathfrak{p})$. It takes advantage of the sparsity of the matrix under consideration to achieve improved (but not asymptotically better) running times and memory usage. Unfortunately, since the computation is done modulo some large primes, it only provides an upper bound on the rank of $\Gamma_{\mathfrak{p}}^{ab}$. The answer is precise when it is 0. Otherwise, the chance of error is less than 10^{-15} or even lower. From numerical evidence, I can tell that the running time is $\mathcal{O}(p^3)$ and the memory needed is $\mathcal{O}(p^2)$ for each rank computation. I do not know any algorithm for which these values are significantly lower.

```

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <math.h>
#include <string.h>

#define smallN 10
#define bigN 20
#define MEM 100000

static char memory[MEM*sizeof(int)];
static int pos=0;
/* returns size chars */
void *my_malloc(int size) {
    void *result=memory+pos;
    pos+=size;
    if (pos>MEM*sizeof(int)) {
        fprintf(stderr,
            "Memory overflow\n");
        exit(1);
    }
    return result;
}

void my_free(int new_pos) {
    fprintf(stderr,
        "Maximum memory allocated: %d\n",
        pos);
    printf(
        "Maximum memory allocated: %d\n",
        pos);
    pos=new_pos;
    return;
}

void my_heap_size() {
    fprintf(stderr,
        "Current heap size: %d\n",
        pos);
    return;
}

void *my_calloc(int nelem,
    size_t size) {
    void *result=memory+pos;
    pos+=nelem*size;
    if (pos>MEM*sizeof(int)) {
        fprintf(stderr,
            "Memory overflow\n");
        exit(1);
    }
    memset(result, 0, nelem*size);
    return result;
}

typedef struct Line_tag {
    int *values;
    int *non_0_values;
    int length;
    int non_0_length;
    int actual_length;
} *Line;

typedef struct Matrix_tag {
    int lines_no;
    int max_column;
    Line *lines;
} *Matrix;

/* because of the customization of memory
allocation procedures, the following
two procedures are almost unnecessary
*/
void delete_Line(Line l) {
    //if (l!=NULL)
    // free(l->values);
    //free(l);
    return;
}

void delete_Matrix(Matrix *M) {
    //int i;
    //for (i=0; i<(*M)->lines_no; i++)
    // delete_Line((*M)->lines[i]);
    //free((*M)->lines);
    //free(*M);
    my_free(0);
    *M=NULL;
    return;
}

/* b^e (mod p) in a time of O(log e) */
int raise(int b, int e, int p) {
    int power=b, result=1;
    while (e) {
        if (e&1)
            result=(result*power)%p;
        power=(power*power)%p;
        e>>=1;
    }
    return result;
}

/* finds out all 4k+1 primes smaller
than n */
int *sieve(int n) {
    char *array;
    int *result;
    int i, j, sqrt=sqrt(n*1.0);
    int count=0;
    assert(array=
        calloc((unsigned)n, sizeof(char)));
    for (i=2; i<=sqrt; i++)
        if (array[i]==0)
            for (j=i*i; j<n; j+=i)
                array[j]=1;
    for (i=2, count=0; i<n; i++)
        if ((array[i]==0)&&((i&3)==1))
            count++;
    else
        array[i]=1;
    assert(result=
        malloc((count+2)*sizeof(int)));
    result[0]=count;
}

```

```

    result[count+1]=0;
    for (i=2, j=1; i<n; i++)
        if (array[i]==0) {
            result[j]=i;
            j++;
        }
    free(array);
    return result;
}

/* finds root^2=-1 (mod p) in a
sufficiently small time */
int better_root(int p) {
    static int prime_list[]=
        {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
    int i, p4=p>>2, result;
    for (i=0; i<10; i++) {
        result=raise(prime_list[i], p4, p);
        if (((result*result)%p)!=1)
            return result;
    }
    printf("Error in finding the root.\n");
    exit(-1);
    return 0;
}

int *find_inverses(int p) {
    int *result, *power;
    int i, j, pow;
    assert(result=
        calloc((unsigned)p+1, sizeof(int)));
    result[0]=p;
    result[1]=1;
    assert(power=
        malloc((p-1)*sizeof(int)));
    for (i=1; i<p; i++) {
        pow=1; j=0;
        do {
            power[j]=pow;
            j++;
            pow=(pow*i)%p;
        } while (pow!=1);
        if (j==(p-1)) {
            for (j=1; j<(p-1); j++)
                result[power[j]]=power[p-1-j];
            free(power);
            return result;
        }
    }
    free(power);
    return result;
}

Line new_node(int limit) {
    Line result;
    assert(result=
        my_malloc(sizeof(struct Line_tag)));
    assert(result->values=
        my_calloc(8, sizeof(int)));
    assert(result->non_0_values=
        my_calloc(9, sizeof(int)));
    result->length=limit;

    result->non_0_length=0;
    result->actual_length=8;
    return result;
}

Line add_value(int a, int val_a,
int limit, Line l) {
    int i;
    if ((a<2)||(a>=limit))
        return l;
    if (l==NULL) {
        l=new_node(limit);
        l->values[0]=val_a;
        l->non_0_values[0]=a;
        l->non_0_values[1]=-1;
        l->non_0_length++;
        return l;
    }
    if (l->non_0_values[1]==-1) {
        if (a==l->non_0_values[0])
            return l;
        l->values[1]=val_a;
        l->non_0_values[1]=a;
        l->non_0_values[2]=-1;
        l->non_0_length++;
    }
    else if (l->non_0_values[2]==-1) {
        if (a==l->non_0_values[1])
            return l;
        l->values[2]=val_a;
        l->non_0_values[2]=a;
        l->non_0_values[3]=-1;
        l->non_0_length++;
    }
    else assert(0);
    return l;
}

void sort2(int *a, int *b) {
    static int c;
    if (*a<*b)
        return;
    c=*a; *a=*b; *b=c;
    return;
}

void sort3(int *a, int *b, int *c) {
    static int d;
    if (*a<*b) {
        if (*b<*c)
            return;
        if (*c<*a) {
            d=*a; *a=*c; *c=*b; *b=d;
            return;
        }
    }
    d=*b; *b=*c; *c=d;
    return;
}
if (*c<*b) {
    d=*a; *a=*c; *c=d;
    return;
}
}

```

```

    if (*c>*a) {
        d=*a; *a=*b; *b=d;
        return;
    }
    d=*a; *a=*b; *b=*c; *c=d;
    return;
}

Line make_line2(int a, int val_a,
               int b, int val_b,
               int limit) {
    sort2(&b, &a);
    return add_value(a, val_a, limit,
                    add_value(b, val_b, limit, NULL));
}

Line make_line3(int a, int val_a,
               int b, int val_b,
               int c, int val_c,
               int limit) {
    sort3(&c, &b, &a);
    return add_value(a, val_a, limit,
                    add_value(b, val_b, limit,
                    add_value(c, val_c, limit, NULL)));
}

static int op_no=0;

Line subtract2(Line l1, Line l2,
               int m) {
    static int temp_non_0[bigN],
              temp_val[bigN];
    int i1, i2, j, a, b, c;
    int temp;
    i1=0; i2=0; j=0;
    a=l1->values[0];
    assert(a);
    b=l2->values[0];
    assert(b);
    assert(l1->non_0_values[0]==
           l2->non_0_values[0]);
    c=l1->non_0_values[0];

    while ((l1->non_0_values[i1]!=-1)&&
           (l2->non_0_values[i2]!=-1)) {
        if (l1->non_0_values[i1]<
            l2->non_0_values[i2]) {
            temp_non_0[j]=
                l1->non_0_values[i1];
            temp_val[j]=
                (b*l1->values[i1])%m;
            i1++; j++; op_no++;
        }
        else if (l1->non_0_values[i1]>
                 l2->non_0_values[i2]) {
            temp_non_0[j]=
                l2->non_0_values[i2];
            temp_val[j]=
                ((-a)*l2->values[i2])%m;
            i2++; j++; op_no++;
        }
        else {
            temp=(l1->values[i1]*b-
                 l2->values[i2]*a)%m;
            if (temp!=0) {
                temp_non_0[j]=
                    l1->non_0_values[i1];
                temp_val[j]=temp;
                j++;
            }
            i1++; i2++; op_no++;
        }
    }
    while (l1->non_0_values[i1]!=-1) {
        temp_non_0[j]=l1->non_0_values[i1];
        temp_val[j]=(b*l1->values[i1])%m;
        i1++; j++; op_no++;
    }
    while (l2->non_0_values[i2]!=-1) {
        temp_non_0[j]=l2->non_0_values[i2];
        temp_val[j]=((-a)*l2->values[i2])%m;
        i2++; j++; op_no++;
    }
    if (j>l1->actual_length) {
        do
            l1->actual_length<<=2;
        while (j>l1->actual_length);
        l1->values=my_calloc(
            (unsigned)l1->actual_length,
            sizeof(int));
        l1->non_0_values=my_calloc(
            (unsigned)l1->actual_length+1,
            sizeof(int));
    }
    if (j==0)
        return NULL;
    l1->non_0_values[j]=-1;
    l1->non_0_length=j;
    for (j=0; j<l1->non_0_length; j++) {
        l1->non_0_values[j]=temp_non_0[j];
        l1->values[j]=temp_val[j];
    }
    return l1;
}

typedef struct Heap_tag {
    Line *heap;
    int heap_size;
    int (*comp)(Line a, Line b);
} *Heap;

int comp(Line a, Line b) {
    if ((a==NULL)|| (b==NULL)) {
        if ((a==NULL)&&(b==NULL))
            return 0;
        if (a==NULL)
            return 1;
        return -1;
    }
    assert(a->non_0_length);
    assert(b->non_0_length);
    if (a->non_0_values[0]!=
        b->non_0_values[0])
        return a->non_0_values[0]-

```

```

        b->non_0_values[0];
    return a->non_0_length-
        b->non_0_length;
}

void Heapify(Heap h, int heap_pos) {
    Line ch1, ch2, p, min;
    if ((heap_pos<<1)<=h->heap_size)
        ch1=h->heap[heap_pos<<1];
    else
        ch1=NULL;
    if ((heap_pos<<1)|1)<=h->heap_size)
        ch2=h->heap[(heap_pos<<1)|1];
    else
        ch2=NULL;
    p=h->heap[heap_pos];
    min=h->comp(ch1, ch2)<0? ch1: ch2;
    if (h->comp(p, min)>0) {
        h->heap[heap_pos]=min;
        if (min==ch1) {
            h->heap[heap_pos<<1]=p;
            Heapify(h, heap_pos<<1);
        }
        else {
            h->heap[(heap_pos<<1)|1]=p;
            Heapify(h, (heap_pos<<1)|1);
        }
    }
    return;
}

Heap Heap_init(
    Line *array, int heap_size,
    int (*comp)(Line a, Line b)) {
    int i, j;
    Line ch1, ch2, p, min;
    Heap result;
    assert(result=
        my_malloc(sizeof(struct Heap_tag)));
    result->heap=array-1;
    result->heap_size=heap_size;
    result->comp=comp;
    for (i=heap_size>>1; i>1; i--)
        Heapify(result, i);
    return result;
}

int rank_Matrix2(Matrix M, int m) {
    int i, j, k, result=0;
    Line l;
    Heap h=Heap_init(
        M->lines, M->lines_no, comp);
    for (i=2; i<M->max_column; i++) {
        l=NULL;
        while
            ((M->lines[0]!=NULL)&&
             (M->lines[0]->non_0_values[0]==i)) {
            if (l==NULL) {
                l=M->lines[0];
                M->lines[0]=NULL;
                Heapify(h, 1);
            }
            else {
                M->lines[0]=
                    subtract2(M->lines[0], l, m);
                Heapify(h, 1);
            }
        }
        if (l==NULL)
            result++;
        else
            delete_Line(l);
    }
    return result;
}

Matrix init_Matrix(int p) {
    Matrix result;
    char *aux;
    int root;
    int *inverse;
    int i, j;
    int a, b, c;
    inverse=find_inverses(p);
    assert(result=
        my_malloc(sizeof(struct Matrix_tag)));
    result->max_column=p-1;
    if ((p%3)==2)
        result->lines_no=((p/3)<<1)+p-2;
    else
        result->lines_no=((p/3)<<1)+p;
    assert(result->lines=
        my_calloc((unsigned)result->lines_no,
                 sizeof(Line)));
    i=0;
    assert(aux=
        my_calloc((unsigned)p+1,
                 sizeof(char)));
    for (j=2; j<=(p>>1); j++, i++)
        result->lines[i]=
            make_line2(j, 1, p-j, -1, p-1);
    for (j=2; j<(p-1); j++)
        if (aux[j]==0) {
            a=j; b=inverse[j];
            result->lines[i]=
                make_line2(a, 1, b, 1, p-1);
            i++;
            aux[a]=1; aux[b]=1;
        }
    for (j=2; j<(p-1); j++)
        if (aux[j]<=1) {
            a=j; b=p+1-inverse[j];
            c=p-inverse[j-1];
            result->lines[i]=
                make_line3(a, 1, b, 1, c, 1, p-1);
            i++;
            aux[a]=2; aux[b]=2; aux[c]=2;
        }
    root=better_root(p);
    for (j=2; j<(p-1); j++)
        if (aux[j]<=2) {
            a=j; b=(p-root+inverse[j])%p;
            c=inverse[(j+root)%p];
            result->lines[i]=

```

```

        make_line3(a, 1, b, 1, c, 1, p-1);
        i++;
        aux[a]=3; aux[b]=3; aux[c]=3;
    }
    if (i!=result->lines_no) {
        printf("%d %d\n",
            i, result->lines_no);
        exit(0);
    }
    //print_Matrix(result);
    //free(aux);
    free(inverse);
    return result;
}

void print_Matrix(Matrix M) {
    int i, j;
    for (i=0; i<M->lines_no; i++) {
        if (M->lines[i]==NULL)
            printf("void line");
        else
            for (j=0; j<M->lines[i]->length; j++)
                if (M->lines[i]->values[j]!=0)
                    printf("%d:%d ",
                        j, M->lines[i]->values[j]);
    }
    printf("\n");
    return;
}

int main() {
    static int primes2[]={
        {6577, 7517, 7573, 7561, 7541};
    int *primes, i, j, r0, r, count;
    Matrix M;
    primes=sieve(bigN);
    count=primes[0];
    op_no=0;
    for (i=1; primes[i]<smallN; i++);
    for (; i<=count; i++) {
        M=init_Matrix(primes[i]);
        r0=M->max_column;
        for (j=0; (r0>0)&&(j<5);
            j++, M=init_Matrix(primes[i])) {
            r=rank_Matrix2(M, primes2[j]);
            printf("r<=%d; op_no=%d\n",
                r, op_no);
            if (r<r0)
                r0=r;
            delete_Matrix(&M);
            op_no=0;
        }
        printf("\n");
        printf("p=%d rank is at most %d\n",
            primes[i], r0);
        fprintf(stderr,
            "p=%d rank is at most %d\n",
            primes[i], r0);
    }
    free(primes);
    return 0;
}

```