>

Here we look to break down an element of Q $\left((-2)^{\frac{1}{3}}\right)$ *into a set of primes with relatively small norms. We need procedures* for Norm, Multiplication, Inversion and Division in the field

The three roots of the equation z^3 = -2 are given below. If z1 is the real root then the two conjugate complex roots are z2 = wz and z3 = w^2z where w:= 1/2*(-1 +i *sqrt(3));

```
> solve(z^3 = -2, z);
```

$$-2^{1/3}, \frac{2^{1/3}}{2} - \frac{I\sqrt{3}\,2^{1/3}}{2}, \frac{2^{1/3}}{2} + \frac{I\sqrt{3}\,2^{1/3}}{2} \qquad (1)$$

```
> solve(w^2 + w + 1, w);
```

$$-\frac{1}{2} + \frac{I\sqrt{3}}{2}, -\frac{1}{2} - \frac{I\sqrt{3}}{2} \qquad (2)$$

Each element of the ring has the form a + bz +cz^2 where a is an integer which we write as [a,b,c]. The norm of an element [a,b,c] is (a + bz + cz^2)(a+bwz+ cw^2)(a+bw^2z+cwz^2). Because z^3 = -2 and w^2+w+1 = 0 this simplifies as norm1 below.

```
> norm1 := proc(a, b, c) global k; k := a^3 - 2·b^3 + 4·c^3 + 6·a·b·c;end;
```

```
norm1 := proc(a, b, c) global k; k := a^3 - 2*b^3 + 4*c^3 + 6*b*a*c end proc      (3)
```

```
> norm1(66, 53, 0);
```

$$-10258 \qquad (4)$$

The following procedures perform multiplication, inversion and division in our field

```
> mult2 := proc(a, b, c, d, e, f) global mul1, mul2, mul3;
  mul1 := a·d - 2·b·f - 2·c·e;
  mul2 := a·e + b·d - 2·c·f;
  mul3 := a·f + b·e + c·d;
  RETURN(mul1, mul2, mul3);end;
```

```
mult2 := proc(a, b, c, d, e, f)                                              (5)
    global mul1, mul2, mul3;
    mul1 := a*d - 2*b*f - 2*c*e;
    mul2 := a*e + b*d - 2*c*f;
    mul3 := a*f + b*e + c*d;
    RETURN(mul1, mul2, mul3)
end proc
```

```
> invert2 := proc(a, b, c) global inv1, inv2, inv3;
  inv1 := (a^2 + 2·b·c) / norm1(a, b, c);
```

$$inv2 := \frac{(-a\cdot b - 2\cdot c^2)}{norm1(a,b,c)};$$

$$inv3 := \frac{(b^2 - a\cdot c)}{norm1(a,b,c)};$$

$RETURN(inv1, inv2, inv3)$;**end**;

```
invert2 := proc(a, b, c)                                              (6)
    global inv1, inv2, inv3;
    inv1 := (a^2 + 2*b*c)/norm1(a, b, c);
    inv2 := (-b*a - 2*c^2)/norm1(a, b, c);
    inv3 := (b^2 - a*c)/norm1(a, b, c);
    RETURN(inv1, inv2, inv3)
end proc
```

```
> divide3 := proc(a, b, c, d, e, f); mult2(a, b, c, invert2(d, e, f) );end;
        divide3 := proc(a, b, c, d, e, f) mult2(a, b, c, invert2(d, e, f) ) end proc   (7)
>
```

Divide3 is a procedure to produce the result of dividing two triples of the form (a,b,c) = a +bz + cz^2. If this division produces an element with integer values we say (a,b,c) is divisible. Next we define our factor base:

```
> U := [1, 1, 0]; A := [0, 1, 0]; B := [-1, 1, 0]; C := [1, 0, 1]; D1 := [1, 1, -1]; E := [1,
    -2, 0]; F := [3, 0, -1];
```

$$U := [1, 1, 0]$$
$$A := [0, 1, 0]$$
$$B := [-1, 1, 0]$$
$$C := [1, 0, 1]$$
$$D1 := [1, 1, -1]$$
$$E := [1, -2, 0]$$
$$F := [3, 0, -1] \quad (8)$$

If A = [0,1,0], B = [-1,-1,0], C = [1,0,1], D = [1,1,-1], E = [1,-2,0], F = [3,0,-1] then we have prime elements with norm = +/- 2, 3, 5, 11, 17, 23

We need procedures to decide on divisibility and then how many times we can divide out the prime element. We also need to find how many times we can divide out by a unit element which we choose as U = [1,1,0]

The 'divisibleby' procedures check if the outcome of the division rule produces

integer values in each of the three positions s[1], s[2], and s[3]. If it does we can perform the 'divideby' procedure to extract all powers of the member of the factor base.

```
> divisiblebyA := proc(a, b, c); s := [mult2(a, b, c, invert2(0, 1, 0))]; if type(s[1], integer)
      and type(s[2], integer) and type(s[3], integer) then true else false;fi; end;
  divisiblebyB := proc(a, b, c); s := [mult2(a, b, c, invert2(-1, 1, 0))]; if type(s[1], integer)
      and type(s[2], integer) and type(s[3], integer) then true else false;fi; end;
  divisiblebyC := proc(a, b, c); s := [mult2(a, b, c, invert2(1, 0, 1))]; if type(s[1], integer)
      and type(s[2], integer) and type(s[3], integer) then true else false;fi; end;
  divisiblebyD := proc(a, b, c); s := [mult2(a, b, c, invert2(1, 1,-1))]; if type(s[1], integer)
      and type(s[2], integer) and type(s[3], integer) then true else false;fi; end;
  divisiblebyE := proc(a, b, c); s := [mult2(a, b, c, invert2(1,-2, 0))]; if type(s[1], integer)
      and type(s[2], integer) and type(s[3], integer) then true else false;fi; end;
  divisiblebyF := proc(a, b, c); s := [mult2(a, b, c, invert2(3, 0,-1))]; if type(s[1], integer)
      and type(s[2], integer) and type(s[3], integer) then true else false;fi; end;
  divisiblebyU := proc(a, b, c); s := [mult2(a, b, c, invert2(1, 1, 0))];if type(s[1], integer)
      and type(s[2], integer) and type(s[3], integer) then true else false;fi; end;


Warning, (in divisiblebyA) `s` is implicitly declared local
Warning, (in divisiblebyB) `s` is implicitly declared local
Warning, (in divisiblebyC) `s` is implicitly declared local
Warning, (in divisiblebyD) `s` is implicitly declared local
Warning, (in divisiblebyE) `s` is implicitly declared local
Warning, (in divisiblebyF) `s` is implicitly declared local
Warning, (in divisiblebyU) `s` is implicitly declared local
divisiblebyA := proc(a, b, c)
    local s;
    s := [mult2(a, b, c, invert2(0, 1, 0))];
    if type(s[1], integer) and type(s[2], integer) and type(s[3], integer) then
        true
    else
        false
    end if
end proc
divisiblebyB := proc(a, b, c)
    local s;
    s := [mult2(a, b, c, invert2(-1, 1, 0))];
    if type(s[1], integer) and type(s[2], integer) and type(s[3], integer) then
        true
    else
```

```
        false
    end if
end proc
divisiblebyC := proc(a, b, c)
    local s;
    s := [mult2(a, b, c, invert2(1, 0, 1))];
    if type(s[1], integer) and type(s[2], integer) and type(s[3], integer) then
        true
    else
        false
    end if
end proc
divisiblebyD := proc(a, b, c)
    local s;
    s := [mult2(a, b, c, invert2(1, 1, -1))];
    if type(s[1], integer) and type(s[2], integer) and type(s[3], integer) then
        true
    else
        false
    end if
end proc
divisiblebyE := proc(a, b, c)
    local s;
    s := [mult2(a, b, c, invert2(1, -2, 0))];
    if type(s[1], integer) and type(s[2], integer) and type(s[3], integer) then
        true
    else
        false
    end if
end proc
divisiblebyF := proc(a, b, c)
    local s;
    s := [mult2(a, b, c, invert2(3, 0, -1))];
    if type(s[1], integer) and type(s[2], integer) and type(s[3], integer) then
        true
    else
        false
    end if
```

```
end proc
divisiblebyU := proc(a, b, c)                                          (9)
    local s;
    s := [mult2(a, b, c, invert2(1, 1, 0))];
    if type(s[1], integer) and type(s[2], integer) and type(s[3], integer) then
        true
    else
        false
    end if
end proc
```

```
> dividebyA := proc(a, b, c) global countA, s; t; oldk1 := a : oldk2 := b : oldk3 := c : countA :=
      0; s := [a, b, c];while divisiblebyA(s[1], s[2], s[3]) = true do countA := countA + 1;
      k1 := s[1]; k2 := s[2]; k3 := s[3]; s := [divide3(k1, k2, k3, 0, 1, 0)];od; s; t := (s[1],
      s[2], s[3]);end;
Warning, (in dividebyA) `oldk1` is implicitly declared local
Warning, (in dividebyA) `oldk2` is implicitly declared local
Warning, (in dividebyA) `oldk3` is implicitly declared local
Warning, (in dividebyA) `k1` is implicitly declared local
Warning, (in dividebyA) `k2` is implicitly declared local
Warning, (in dividebyA) `k3` is implicitly declared local
Warning, (in dividebyA) `t` is implicitly declared local
```

```
dividebyA := proc(a, b, c)                                             (10)
    local oldk1, oldk2, oldk3, k1, k2, k3, t;
    global countA, s;
    t;
    oldk1 := a;
    oldk2 := b;
    oldk3 := c;
    countA := 0;
    s := [a, b, c];
    while divisiblebyA(s[1], s[2], s[3]) = true do
        countA := countA + 1; k1 := s[1]; k2 := s[2]; k3 := s[3]; s := [divide3(k1, k2, k3,
        0, 1, 0)]
    end do;
    s;
    t := s[1], s[2], s[3]
end proc
```

```
> dividebyB := proc(a, b, c) global countB, s; t; oldk1 := a : oldk2 := b : oldk3 := c : countB :=
      0; s := [a, b, c];;while divisiblebyB(s[1], s[2], s[3]) do countB := countB + 1; k1 :=
      s[1]; k2 := s[2]; k3 := s[3]; s := [divide3(k1, k2, k3, -1, 1, 0)];od; s; t := (s[1], s[2],
      s[3]);end;
```

```
Warning, (in divideby B) `oldk1` is implicitly declared local
Warning, (in divideby B) `oldk2` is implicitly declared local
Warning, (in divideby B) `oldk3` is implicitly declared local
Warning, (in divideby B) `k1` is implicitly declared local
Warning, (in divideby B) `k2` is implicitly declared local
Warning, (in divideby B) `k3` is implicitly declared local
Warning, (in divideby B) `t` is implicitly declared local
```

$$dividebyB := \mathbf{proc}(a, b, c) \tag{11}$$

```
    local oldk1, oldk2, oldk3, k1, k2, k3, t;
    global countB, s;
    t;
    oldk1 := a;
    oldk2 := b;
    oldk3 := c;
    countB := 0;
    s := [a, b, c];
    while divisiblebyB(s[1], s[2], s[3]) do
        countB := countB + 1;
        k1 := s[1];
        k2 := s[2];
        k3 := s[3];
        s := [divide3(k1, k2, k3, -1, 1, 0)]
    end do;
    s;
    t := s[1], s[2], s[3]
end proc
```

```
> dividebyC := proc(a, b, c) global countC, s, t; oldk1 := a : oldk2 := b : oldk3 := c : countC :=
      0; s := [a, b, c];;while divisiblebyC(s[1], s[2], s[3]) do countC := countC + 1; k1 :=
      s[1]; k2 := s[2]; k3 := s[3]; s := [divide3(k1, k2, k3, 1, 0, 1)];od; s; t := (s[1], s[2],
      s[3]);end;
Warning, (in divideby C) `oldk1` is implicitly declared local
Warning, (in divideby C) `oldk2` is implicitly declared local
Warning, (in divideby C) `oldk3` is implicitly declared local
Warning, (in divideby C) `k1` is implicitly declared local
Warning, (in divideby C) `k2` is implicitly declared local
Warning, (in divideby C) `k3` is implicitly declared local
```

$$dividebyC := \mathbf{proc}(a, b, c) \tag{12}$$

```
    local oldk1, oldk2, oldk3, k1, k2, k3;
    global countC, s, t;
    oldk1 := a;
    oldk2 := b;
```

```
    oldk3 := c;
    countC := 0;
    s := [a, b, c];
    while divisiblebyC(s[1], s[2], s[3]) do
        countC := countC + 1; k1 := s[1]; k2 := s[2]; k3 := s[3]; s := [divide3(k1, k2, k3,
        1, 0, 1)]
    end do;
    s;
    t := s[1], s[2], s[3]
end proc
```

```
> dividebyD := proc(a, b, c) global countD, s, t; oldk1 := a : oldk2 := b : oldk3 := c :
      countD := 0; s := [a, b, c];;while divisiblebyD(s[1], s[2], s[3]) do countD := countD
       + 1; k1 := s[1]; k2 := s[2]; k3 := s[3]; s := [divide3(k1, k2, k3, 1, 1, -1)];od; s; t :=
      (s[1], s[2], s[3]);end;
Warning, (in dividebyD) `oldk1` is implicitly declared local
Warning, (in dividebyD) `oldk2` is implicitly declared local
Warning, (in dividebyD) `oldk3` is implicitly declared local
Warning, (in dividebyD) `k1` is implicitly declared local
Warning, (in dividebyD) `k2` is implicitly declared local
Warning, (in dividebyD) `k3` is implicitly declared local
```

```
dividebyD := proc(a, b, c)                                                    (13)
    local oldk1, oldk2, oldk3, k1, k2, k3;
    global countD, s, t;
    oldk1 := a;
    oldk2 := b;
    oldk3 := c;
    countD := 0;
    s := [a, b, c];
    while divisiblebyD(s[1], s[2], s[3]) do
        countD := countD + 1;
        k1 := s[1];
        k2 := s[2];
        k3 := s[3];
        s := [divide3(k1, k2, k3, 1, 1, -1)]
    end do;
    s;
    t := s[1], s[2], s[3]
end proc
```

```
> dividebyE := proc(a, b, c) global countE, s, t; oldk1 := a : oldk2 := b : oldk3 := c : countE :=
      0; s := [a, b, c];while divisiblebyE(s[1], s[2], s[3]) do countE := countE + 1; k1 :=
```

```
    s[1]; k2 := s[2]; k3 := s[3]; s := [divide3(k1, k2, k3, 1, -2, 0)];od; s; t := (s[1], s[2],
    s[3]);end;
Warning, (in dividebyE) `oldk1` is implicitly declared local
Warning, (in dividebyE) `oldk2` is implicitly declared local
Warning, (in dividebyE) `oldk3` is implicitly declared local
Warning, (in dividebyE) `k1` is implicitly declared local
Warning, (in dividebyE) `k2` is implicitly declared local
Warning, (in dividebyE) `k3` is implicitly declared local
dividebyE := proc(a, b, c)                                                    (14)
    local oldk1, oldk2, oldk3, k1, k2, k3;
    global countE, s, t;
    oldk1 := a;
    oldk2 := b;
    oldk3 := c;
    countE := 0;
    s := [a, b, c];
    while divisiblebyE(s[1], s[2], s[3]) do
        countE := countE + 1;
        k1 := s[1];
        k2 := s[2];
        k3 := s[3];
        s := [divide3(k1, k2, k3, 1, -2, 0)]
    end do;
    s;
    t := s[1], s[2], s[3]
end proc
> dividebyF := proc(a, b, c) global countF, s, t; oldk1 := a : oldk2 := b : oldk3 := c : countF :=
    0; s := [a, b, c];;while divisiblebyF(s[1], s[2], s[3]) do countF := countF + 1; k1 :=
    s[1]; k2 := s[2]; k3 := s[3]; s := [divide3(k1, k2, k3, 3, 0, -1)];od; s; t := (s[1], s[2],
    s[3]);end;
Warning, (in dividebyF) `oldk1` is implicitly declared local
Warning, (in dividebyF) `oldk2` is implicitly declared local
Warning, (in dividebyF) `oldk3` is implicitly declared local
Warning, (in dividebyF) `k1` is implicitly declared local
Warning, (in dividebyF) `k2` is implicitly declared local
Warning, (in dividebyF) `k3` is implicitly declared local
dividebyF := proc(a, b, c)                                                    (15)
    local oldk1, oldk2, oldk3, k1, k2, k3;
    global countF, s, t;
    oldk1 := a;
    oldk2 := b;
    oldk3 := c;
```

```
    countF := 0;
    s := [a, b, c];
    while divisiblebyF(s[1], s[2], s[3]) do
        countF := countF + 1;
        k1 := s[1];
        k2 := s[2];
        k3 := s[3];
        s := [divide3(k1, k2, k3, 3, 0, -1)]
    end do;
    s;
    t := s[1], s[2], s[3]
end proc
```

```
>
```

```
> dividebyU2 := proc(a, b, c) global countU, s, sign1; sign1 := -1; if norm1(a, b, c) ≠ 1
      and norm1(a, b, c) ≠ -1 then false else oldk1 := a : oldk2 := b : oldk3 := c; s := [a, b,
      c]; if s = [0, 0, 0] then false elif s = [-1, 1, -1] then false elif s = [1, -1, 1] then false
      elif s = [1, 0, 0] then countU := 0; sign1 := 0; print(countU); print(sign1);elif s = [-1,
      0, 0] then countU := 0; sign1 := 1; print(countU); print(sign1); else if s = [-1, 1, -1]
      then false else countU := 1;while not((s[1] = 1 and s[2] = 1 and s[3] = 0) or (s[1] =
      -1 and s[2] = -1 and s[3] = 0)) do countU := countU + 1; k1 := s[1]; k2 := s[2];
      k3 := s[3]; s := [divide3(k1, k2, k3, 1, 1, 0)];od;if k1 = 1 then sign1 := 1 else sign1 := 1
      fi; fi;fi;fi;end;
Warning, (in dividebyU2) `oldk1` is implicitly declared local
Warning, (in dividebyU2) `oldk2` is implicitly declared local
Warning, (in dividebyU2) `oldk3` is implicitly declared local
Warning, (in dividebyU2) `k1` is implicitly declared local
Warning, (in dividebyU2) `k2` is implicitly declared local
Warning, (in dividebyU2) `k3` is implicitly declared local
```

```
dividebyU2 := proc(a, b, c)                                                    (16)
    local oldk1, oldk2, oldk3, k1, k2, k3;
    global countU, s, sign1;
    sign1 := -1;
    if norm1(a, b, c) <> 1 and norm1(a, b, c) <> -1 then
        false
    else
        oldk1 := a;
        oldk2 := b;
        oldk3 := c;
        s := [a, b, c];
        if s = [0, 0, 0] then
            false
```

```
        elif s = [ − 1, 1, − 1 ] then
            false
        elif s = [1, − 1, 1 ] then
            false
        elif s = [1, 0, 0 ] then
            countU := 0; sign1 := 0; print(countU); print(sign1)
        elif s = [ − 1, 0, 0 ] then
            countU := 0; sign1 := 1; print(countU); print(sign1)
        else
            if s = [ − 1, 1, − 1 ] then
                false
            else
                countU := 1;
                while not(s[1] = 1 and s[2] = 1 and s[3] = 0 or s[1] = − 1 and s[2] =
                 − 1 and s[3] = 0) do
                    countU := countU + 1;
                    k1 := s[1];
                    k2 := s[2];
                    k3 := s[3];
                    s := [divide3(k1, k2, k3, 1, 1, 0)]
                end do;
                if k1 = 1 then sign1 := 1 else sign1 := 1 end if
            end if
        end if
    end if
end proc
```

>

The factorisation procedure below performs the divisions by each of the factor bases

```
> factorisation := proc(a, b, c); if a = 0 and b = 0 and c = 0 then false
      else dividebyU2(dividebyA(dividebyB(dividebyC(dividebyD(dividebyE(dividebyF(a, b,
      c)))))))) : if sign1 ≥ 0 then print(a, b, [sign1, countU, countA, countB, countC, countD,
      countE, countF])else print(a, b, `False - does not factor`);false;fi;fi;end;
```

```
factorisation := proc(a, b, c)                                                    (17)
    if a = 0 and b = 0 and c = 0 then
        false
    else
        dividebyU2(dividebyA(dividebyB(dividebyC(dividebyD(dividebyE(dividebyF(a, b,
        c)))))));
```

```
        if 0 <= sign1 then
            print(a, b, [sign1, countU, countA, countB, countC, countD, countE, countF])
        else
            print(a, b, False - does not factor); false
        end if
    end if
end proc
```

>

The next two procedures perform multiplication of triples [a,b,c]; mult1 combines a pair of tripes while mult4 takes a string and uses mult1 to combines them pairwise

>

```
> mult1 := proc(x, y); mult2(x[1], x[2], x[3], y[1], y[2], y[3]) ;end;
```

```
mult1 := proc(x, y) mult2(x[1], x[2], x[3], y[1], y[2], y[3]) end proc
```

(18)

>

>

```
> mult4 := proc( )global L; L := [ ]; for i from 1 to nargs do L := [op(L), args[i]];od;
      if nops(L) = 2 then mult1(op(1, L), op(2, L)) else k := [mult1(op(1, L), op(2, L))];
      L := subsop(1 = NULL, L); L := subsop(1 = NULL, L); L := [k, op(L)]; mult4(op(L));
      fi;end;
```

```
Warning, (in mult4) `i` is implicitly declared local
Warning, (in mult4) `k` is implicitly declared local
mult4 := proc( )
    local i, k;
    global L;
    L := [ ];
    for i to nargs do L := [op(L), args[i]] end do;
    if nops(L) = 2 then
        mult1(op(1, L), op(2, L))
    else
        k := [mult1(op(1, L), op(2, L))];
        L := subsop(1 = NULL, L);
        L := subsop(1 = NULL, L);
        L := [k, op(L)];
        mult4(op(L))
    end if
end proc
```

(19)

```
> mult4(U, U, B, E);
```

1, 5, 3

(20)

>

>

Now we use these procedures to try to factorise N = 9263 = 59*157 = 21^3 + 2. Hence r = 21 and we can work in the field Q $\left((-2)^{\frac{1}{3}} \right)$ whose primes we have studied

```
> N := 21^3 + 2;
```

$$N := 9263 \quad \textbf{(21)}$$

```
> ifactor(N);
```

$$(59)\ (157) \quad \textbf{(22)}$$

In the next step we populate a list B with the values of a and b (small) where the factor base for a + 21b contains only the small primes {2,3,5,7,11,13}

There are 47 pairs (a,b) with a and b between -9 and 9 where both a + 21b and [a,b,0] can be completely factorised using a small factor base. The values of the factors form a 15 column matrix; the first 7 columns are the powers of -1, 2, 3, 5, 7, 11, 13 that factor a + 21b and the last 8 are the powers of -1, U, A, B, C, D, E, F that factorise [a,b,0]. To display the matrix we need to increase the default size to 50x50.

```
> interface(rtablesize = 50)
```

$$50 \quad \textbf{(23)}$$

```
>
```

Row	a	b	a+21b
1	-9	-3	-72
2	-9	0	-9
3	-7	-3	-70
4	-7	1	14
5	-6	-4	-90
6	-6	-2	-48
7	-6	0	-6
8	-4	-4	-88
9	-4	4	80
10	-3	-3	-66
11	-3	-2	-45
12	-3	-1	-24
13	-3	3	60
14	-2	-3	-65
15	-2	-2	-44
16	-2	0	-2
17	-2	2	40
18	-1	-1	-22
19	-1	0	-1
20	0	-4	-84
21	0	-3	-63
22	0	-1	-21
23	0	2	42
24	0	3	63
25	9	9	198
26	0	4	84
27	1	-1	-20

28	1	1	22
29	2	-2	-40
30	2	2	44
31	2	3	65
32	3	-3	-60
33	3	1	24
34	3	3	66
35	4	-4	-80
36	4	4	88
37	6	0	6
38	6	2	48
39	6	4	90
40	7	-1	-14
41	7	3	70
42	9	0	9
43	9	3	72
44	-9	9	180
45	-8	8	160
46	6	6	132
47	8	8	176

```
> R := Matrix(47, 15, [1, 3, 2, 0, 0, 0, 0, 1, 2, 0, 3, 2, 0, 0, 0, 1, 0, 2, 0, 0, 0, 0, 1, 2, 0, 6, 0, 0, 0, 0,
    1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 2, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 2, 1, 0, 0, 0,
    1, 1, 3, 0, 0, 1, 0, 0, 1, 4, 3, 0, 0, 0, 0, 1, 1, 3, 0, 2, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 3, 3, 0, 0, 0,
    0, 1, 3, 0, 0, 0, 1, 0, 1, 1, 6, 0, 0, 0, 0, 0, 0, 4, 0, 1, 0, 0, 0, 0, 0, 6, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1,
    0, 1, 2, 0, 3, 0, 0, 0, 0, 1, 0, 2, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 3, 1, 0, 0, 0, 0, 1, 1, 0, 0, 2, 0,
    0, 0, 0, 2, 1, 1, 0, 0, 0, 1, 1, 0, 4, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 2, 0, 0, 0,
    1, 0, 1, 1, 3, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 3, 0, 1, 0, 0, 0, 1, 0, 3, 1, 0,
    0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 2, 1, 0,
    1, 0, 0, 1, 0, 7, 0, 0, 0, 0, 0, 1, 0, 2, 0, 1, 0, 0, 1, 1, 1, 3, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0,
    0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 4, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 1, 1, 1, 3, 0, 0, 0, 0, 0, 1, 2,
    0, 0, 1, 0, 1, 2, 0, 3, 0, 1, 0, 0, 0, 2, 1, 0, 1, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 1, 2, 0, 1, 0, 0, 0, 1, 0, 0,
    1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 3, 0, 1, 0, 0, 0, 0, 0, 3, 1, 0, 0, 0, 0, 0, 2,
    0, 0, 0, 1, 0, 1, 1, 3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 2, 1, 1, 0, 0, 0, 1, 1,
    0, 4, 0, 0, 0, 0, 0, 3, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 2, 0, 3, 0, 0, 0, 0, 1,
    4, 0, 1, 0, 0, 0, 1, 0, 6, 1, 0, 0, 0, 0, 0, 3, 0, 0, 0, 1, 0, 1, 1, 6, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1,
    1, 3, 3, 0, 0, 0, 0, 0, 4, 1, 0, 0, 0, 0, 1, 1, 3, 0, 2, 0, 0, 0, 0, 1, 2, 1, 0, 0, 0, 1, 1, 3, 0, 0, 1, 0, 0,
    1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 0,
    1, 2, 0, 6, 0, 0, 0, 0, 0, 0, 3, 2, 0, 0, 0, 0, 0, 1, 2, 0, 3, 2, 0, 0, 0, 0, 2, 2, 1, 0, 0, 0, 1, 2, 0, 7, 0, 0, 0,
    0, 0, 5, 0, 1, 0, 0, 0, 1, 0, 9, 1, 0, 0, 0, 0, 0, 0, 2, 1, 0, 0, 1, 0, 1, 2, 3, 3, 0, 0, 0, 0, 0, 4, 0, 0, 0, 1,
    0, 1, 1, 9, 0, 0, 0, 0, 0]);
```

$$R := \begin{bmatrix}
1 & 3 & 2 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 3 & 2 & 0 & 0 & 0 \\
1 & 0 & 2 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 6 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 2 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
1 & 1 & 2 & 1 & 0 & 0 & 0 & 1 & 1 & 3 & 0 & 0 & 1 & 0 & 0 \\
1 & 4 & 3 & 0 & 0 & 0 & 0 & 1 & 1 & 3 & 0 & 2 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 3 & 3 & 0 & 0 & 0 & 0 \\
1 & 3 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 6 & 0 & 0 & 0 & 0 & 0 \\
0 & 4 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 6 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 2 & 0 & 3 & 0 & 0 & 0 & 0 \\
1 & 0 & 2 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 3 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 2 & 0 & 0 & 0 \\
0 & 2 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 4 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
1 & 2 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 3 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\
0 & 3 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 3 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 2 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 7 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 2 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 3 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 2 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 3 & 0 & 0 & 0 & 0 \\
0 & 1 & 2 & 0 & 0 & 1 & 0 & 1 & 2 & 0 & 3 & 0 & 1 & 0 & 0 \\
0 & 2 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 7 & 0 & 0 & 0 & 0 & 0 \\
1 & 2 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 3 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 3 & 1 & 0 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 3 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
1 & 2 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 4 & 0 & 0 & 0 & 0 \\
0 & 3 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 2 & 0 & 3 & 0 & 0 & 0 & 0
\end{bmatrix} \tag{24}$$

We aim to find sets of rows that are linearly dependent modulo 2; one possibility is rows 23, 37, 41, 45

These rows give us a factor (59) of N when we calculate the values of u and v that give a congruence $u^2 = v^2 \bmod N$ and find the gcd of N and u - v