An algorithm for whether the $\sigma_{i,j}$'s span $\mathrm{Int}(o_L, o_L)$

Shashidhara Balla, Dragoș Crișan, Jingjie Yang

Summer 2021

1 Introduction

Let $\mathbb{Q}_p \subseteq L \subsetneq \mathbb{C}_p$ be a field of finite degree d over $\mathbb{Q}_p$, o_L the ring of integers of L, $\pi \in o_L$ a fixed prime element, and $q := |o_L/\pi_L o_L|$ the dimension of the residue field.

For an o_L-submodule S of $L[Y]$ and an integer n, let $S_n = \{f \in S : \deg(f) < n\}$.

Recall that the polynomials $P_m(Y)$ are defined by

$$\exp(Y \cdot \log_{\mathrm{LT}}(Z)) = \sum_{i=0}^{\infty} P_m(Y) Z^m.$$

We will choose the coordinate Z such that $\log_{\mathrm{LT}}(Z) = \sum_{k=0}^{\infty} \pi^{-k} Z^{q^k}$.

Define the upper triangular matrix $(\sigma_{i,j})_{i,j\geq 0}$ with entries in $L[Y]$ by

$$P_j(Ys) = \sum_{i=0}^{j} \sigma_{i,j}(Y) P_i(s).$$

By Lemma 9.9 in bounded26, we know that $\sigma_{i,j}(Y) \in \mathrm{Int}(o_L, o_L)$ and that $\deg(\sigma_{i,j}(Y)) \leq j$. The question is whether the o_L-linear span of $\{\sigma_{i,j}(Y) : 0 \leq i \leq j\}$ equals $\mathrm{Int}(o_L, o_L)$. In this write-up we develop an algorithm to check whether $\big(\mathrm{Int}(o_L, o_L)\big)_n$ is contained in the o_L-linear span of $\{\sigma_{i,j}(Y) : 0 \leq i \leq j < N\}$ for some fixed N, where for convenience we require $q - 1 \mid N$.

2 Theory

2.1 Reduction to $\tau_{i,j}^{(a)}$

To ease notation, for a fixed $a \in \{0, 1, \ldots, q-2\}$, we denote $\underline{i} = a + (q-1)i$.

By Proposition 10.8(2) in bounded24, there exists upper triangular matrices $\tau_{i,j}^{(a)}(Y)$ such that

$$\sigma_{\underline{i},\underline{j}}(Y) = Y^a \cdot \tau_{i,j}^{(a)}(Y^{q-1}). \tag{1}$$

Definition 2.1. For a polynomial $P(x)$, we denote by $\gamma_n(P)$ the coefficient of x^n in P.

Definition 2.2. Let M be the o_L-linear span of $\{\sigma_{i,j}(Y) : 0 \leq i \leq j\}$. For a fixed a, let $M^{(a)}$ be the o_L-linear span of $\left\{\sigma_{\underline{i},\underline{j}}(Y) : 0 \leq i \leq j\right\}$. Let $S^{(a)}$ be the o_L-linear span of $\left\{\tau_{i,j}^{(a)}(Y) : 0 \leq i \leq j\right\}$.

Lemma 2.3. *Let $(f_b^{(a)})_{b\geq 0}$ be a regular basis for $S^{(a)}$, such that each $f_b^{(a)}$ has degree b. Then, $M = \mathrm{Int}(o_L, o_L)$ if and only if for all $a \in \{0, 1, \ldots q-2\}$ and $b \geq 0$, we have*

$$\nu_\pi(\gamma_b(f_b^{(a)})) = -w_q(a + b(q-1)).$$

Proof. For a fixed $a \in \{0, 1, \ldots q-2\}$, by (1), we have $\gamma_s(\sigma_{i,j}(Y)) = 0$ if $s \not\equiv j \pmod{q-1}$. So, by definition, $M = \bigoplus_{a=0}^{q-2} M^{(a)}$.

We write $S^{(a)}(Y^{q-1}) = \{f(Y^{q-1}) : f \in S^{(a)}\}$. Equation (1) shows that

$$M^{(a)} = Y^a \cdot N^{(a)}(Y^{q-1}).$$

Having chosen a regular basis $(f_b^{(a)})_{b\geq 0}$, these give regular bases $\left(f_b^{(a)}(Y^{q-1})\right)_{b\geq 0}$ for $S^{(a)}(Y^{q-1})$.

So, we get regular bases $\left(Y^a f_b^{(a)}(Y^{q-1})\right)_{b\geq 0}$ for $M^{(a)}$ and thus a regular basis $\{Y^a f_b^{(a)}(Y^{q-1}) : a \in \{0, 1, \ldots q-2\}, b \geq 0\}$ for M.

Then, $M = \mathrm{Int}(o_L, o_L)$ is equivalent to $\nu_\pi(\gamma_{a+b(q-1)}(Y^a f_b^{(a)}(Y^{q-1}))) = -w_q(a + b(q-1))$, which is equivalent to $\nu_\pi(\gamma_b(f_b^{(a)})) = -w_q(a + b(q-1))$. □

Let $n = a + b(q-1)$, where a, b are integers, with $a \in \{0, 1, \ldots q-2\}$. The proof above shows that a polynomial of degree n with π-valuation of leading term equal to $w_q(n)$ exists in M_N if and only a polynomial of degree b with the same valuation of leading term exists in $S^{(a)}_{N/(q-1)}$. So, the strategy will be to compute regular bases for $S^{(a)}_{N/(q-1)}$.

2.2 A formula for $\tau_{i,j}^{(a)}$

One advantage of this approach is that the matrices $\tau_{i,j}^{(a)}(Y)$ can be computed quickly. Recall Definition 10.1 of bounded24 (where we merely change notation, calling m by a instead):

Definition 2.4. For each $j \geq i \geq 0$, we define

$$Q_a(i,j) := \left\{ \mathbf{k} \in \mathbb{N}^\infty : \sum_{\ell=0}^{\infty} k_\ell = \underline{i}, \sum_{\ell=1}^{\infty} k_\ell \left(\frac{q^\ell - 1}{q-1}\right) = j - i \right\};$$

$$r_{i,j}^{(a)} := \sum_{\mathbf{k} \in Q_a(i,j)} \binom{\underline{i}}{k_0; k_1; \ldots} \cdot \pi^{-\sum_{\ell=1}^{\infty} \ell \cdot k_\ell}.$$

Moreover, define the following upper diagonal matrix of coefficients, which doesn't depend on a.

Definition 2.5. Let

$$D_{i,j} = i! \gamma_i P_j(Y).$$

From Proposition 1.20 in outline9, we obtain the following recursion formula, valid for $i \geq 1$:

$$D_{i,j} = \sum_{r \geq 0} \pi^{-r} D_{i-1, j-q^r},$$

with the initial conditions being $D_{0,j} = \delta_{0,j}$.

Now, by Remark 10.4 in bounded24 it follows that $r_{i,j}^{(a)} = D_{\underline{i},\underline{j}}$. We define:

Definition 2.6.

$$\mathcal{D}_Y := \mathrm{diag}(1, Y, Y^2, \ldots)$$

Then, Lemma 10.11 in bounded24 gives $\tau^{(a)} = (r^{(a)})^{-1} \cdot \mathcal{D}_Y \cdot r^{(a)}$. This gives a fast algorithm to compute the matrices $\tau^{(a)}$, as the recurrence relation for D allows us to compute $r^{(a)}$ easily.

2.3 Gaussian elimination over a (discrete) valuation ring

Let R be a (discrete) valuation ring and let A be an $n \times m$ matrix with entries in R. We define notions of elementary row operations and row echelon form over R, similarly to the definitions over a field.

Definition 2.7. Given a matrix A as above define the elementary row operations:

1. Swap two rows
2. Multiply an entire row by a unit in R
3. Add an R-multiple of a row to another row

Lemma 2.8. *Performing elementary row operations on a matrix preserves its R-row span.*

Proof. For each elementary row operation on A, we define an $n \times n$ matrix B with entries in R such that the result of applying the elementary row operation on A is BA. Observe that in each case, B is invertible, so BA has the same R-row span as A. $\square$

Now, we use elementary row operations to put A in row echelon form. The algorithm is somewhat similar to the classical Gaussian elimination over a field. We have some rows, at the top of the matrix, which are already good (i.e. they are part of the final row echelon form). All the other rows will be bad. At the beginning, no row is good. We will only operate on bad rows, so we call an entry bad if it is on an bad row.

For each (column) k from 1 to m:

1. If all bad entries in column k are 0, completely ignore the column.
2. If there are non-zero bad entries in column k, find the element of minimum valuation among them. Let it be on row i.
3. Swap row i and the first bad row i_0.
4. Subtract the correct multiple of row i_0 from all the other bad rows, such that all but the i_0-th bad entries in column k are 0 (this is possible, as the entry on line i_0 has minimum valuation among the other bad elements on column k).
5. Mark row i_0 as good.

3 Implementation

We focus on the totally ramified extension $L = \mathbb{Q}_p(p^{1/d})$ and the unramified extension of degree d, where we take the prime p, the degree d, and the cutoff N as input parameters.

Fix $a \in \{0, 1, \ldots, q-2\}$. Firstly, we compute the matrices $(\tau^{(a)})_{0 \leq i \leq j < N/(q-1)}$ following the method discussed in Section 2.2. Then, for $0 \leq s < N/(q-1)$, we will appeal to the following result to inductively compute a basis $(g_b^{(a),s})_{0 \leq b \leq s}$ for the o_L-span of $\{\tau_{i,j}^{(a)} : 0 \leq i \leq j \leq s\}$ where each $g_b^{(a),s}$ is of degree b.

Proposition 3.1. *Fix $s \geq 0$, and let $(g_b^{(a),s-1})_{0 \leq b \leq s-1}$ be a basis for the o_L-span of $\{\tau_{i,j}^{(a)} : 0 \leq i \leq j \leq s-1\}$ such that each $g_b^{(a),s-1}$ is of degree b.*

Set B to be the $(2j'+1)\times(j'+1)$ matrix

$$\begin{array}{cccccc} Y^s & Y^{s-1} & & 1 & \\ \left(\begin{matrix} 0 \\ \vdots \\ 0 \\ * \\ \vdots \\ * \end{matrix}\right. & \begin{matrix} \star \\ \\ \\ \cdots \\ \\ \cdots \end{matrix} & \begin{matrix} \cdots \\ \ddots \\ \\ \cdots \\ \\ \cdots \end{matrix} & \left.\begin{matrix} * \\ \vdots \\ \star \\ * \\ \vdots \\ * \end{matrix}\right) & \begin{matrix} g_{s-1}^{(a),s-1} \\ \\ g_0^{(a),s-1} \\ \tau_{0,s}^{(a)} \\ \\ \tau_{s,s}^{(a)} \end{matrix} \end{array}$$

with coefficients in L, where the $\star$'s are non-zero.

Perform Gaussian elimination on B over the discrete valuation ring o_L, as explained in Subsection 2.3, to obtain a matrix B' in row echelon form with $s+1$ non-zero rows. For $0\le b\le s$, let

$$g_b^{(a),s}(Y)=\sum_{j=0}^{s}B'_{s-b,j}Y^{s-j}.$$

Then $(g_b^{(a),s})_{0\le b\le s}$ forms a basis for the o_L-span of $\{\tau_{i,j}^{(a)}:0\le i\le j\le s\}$ with each $g_b^{(a),s}$ having degree b. Furthermore

$$\nu_\pi(\gamma_b(g_b^{(a),s}))\le\nu_\pi(\gamma_b(g_b^{(a),s-1}))$$

for $0\le b\le s-1$.

Proof. Firstly, as $\sigma_{\underline{s},\underline{s}}=Y^{\underline{s}}$ by Lemma 9.9 of bounded24, we have $\tau_{s,s}^{(a)}=Y^s$ by Equation 1. Then the leftmost column of B is not entirely zero, so by the procedure $B'_{0,0}\neq 0$ and thus $\deg g_s^{(a),s}=s$. Note that the rows encoding the $g_*^{(a),s-1}$'s have a zero entry in the leftmost column, so they are left intact when we operate on this column.

Now let $0<j\le s$, and consider the operations on column j after having finished with columns to the left. Assume that the new $g_*^{(a),s}$ satisfy $\deg g_b^{(a),s}=b$ for every $s\ge b>s-j$ and that the entries of the rows encoding the old $g_b^{(a),s-1}$ are left intact for every $0\le b\le s-j$. Then the row encoding $g_{s-j}^{(a),s-1}$ still has a non-zero entry in the jth column, so the minimum valuation of non-zero bad entries in this column is at most $\nu_\pi(\gamma_{s-j}(g_{s-j}^{(a),s-1}))$. Also, by the inductive hypothesis the first bad row is j, so when reading off this row we have $\deg g_{s-j}^{(a),s}=s-j$ with $\nu_\pi(\gamma_{s-j}(g_{s-j}^{(a),s}))\le\nu_\pi(\gamma_{s-j}(g_{s-j}^{(a),s-1}))$. Moreover the rows encoding $g_{b^{(a),s-1}}$ have a zero entry in the jth column, so again we leave these rows intact when operating on this column.

Finally, for degree reasons the $g_*(a),s$'s are clearly linearly independent; they furthermore span the o_L-module generated by $\{\tau_{i,j}^{(a)}:0\le i\le j\le s\}$ by Lemma 2.8 and by induction. □

For b fixed, we see that $\nu_\pi(\gamma_b(g_b^{(a),s}))$ is non-increasing in s for all $s\ge b$. Moreover, as $g_b^{(a),s}\in S^{(a)}$ can be written as an o_L-linear combination of the $f_i^{(a)}$'s and each $f_i^{(a)}$ is of degree i, we must have $g_b^{(a),s}=\sum_{0\le i\le b}\lambda_i f_i^{(a)}$ for some $\lambda_i\in o_L$, and thus $\nu_\pi(\gamma_b(g_b^{(a),s}))\ge\nu_\pi(\gamma_b(f_b^{(a)}))$. These observations make the following quantity meaningful.

Definition 3.2. For $n=a+b(q-1)$, let $s_0(n)$ be the minimal $s\ge b$ such that $(g_b^{(a),s})_{0\le b\le s}$ satisfies $\nu_\pi(\gamma_b(g_b^{(a),s}))=-w_q(n)$, if such s exists; otherwise set $s_0(n)=\infty$.

Then once we reach $s_0(n)$ in the computations, we can stop and conclude that the equality $\nu_\pi(\gamma_b(f_b^{(a)}))=-w_q(a+b(q-1))$ in Lemma 2.3 holds for this $n=a+b(q-1)$.

We may also make a small optimisation: at any stage s, if $s \geq s_0(a + b(q-1))$ for all $0 \leq b < d$ then we can just drop the last d columns when carrying out Gaussian elimination: the calculations of the leading terms of $(g_b^{(a),s+1})_{d \leq b \leq s+1}$ are not affected, and we already know that each of $(g_b^{(a),s+1})_{0 \leq b < d}$ has the desired π-valuation of the leading term.

4 Results

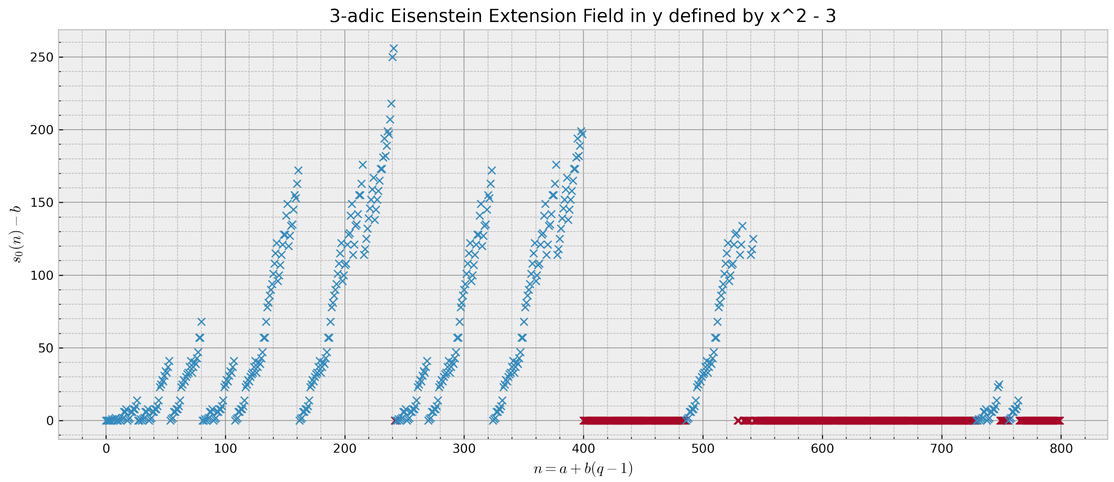

Figure 1: $s_0(n)$ in the quadratic ramified extension $\mathbb{Q}_3(\sqrt{3})$ for $n < 800$. Red points are the n's for which $s_0(n) \geq 800$.

For reference, the computations in Figure 1 took

- 227.04 seconds for D;
- 616.45 seconds for $\tau^{(0)}$ and 616.43 seconds for $\tau^{(1)}$;
- 0.20 seconds for $s = 50$, 1.89 seconds for $s = 100$, 6.15 seconds for $s = 150$, 12.09 seconds for $s = 200$, etc. for $a = 0$, and slightly less for $a = 1$.

We see that $s_0(n) - b$ seems to depend on the p-adic digits of n; we only managed to prove a special case of this pattern, which we will discuss below. Nonetheless, the data do suggest that $s_0(n)$ is finite for every n and hence that $\mathrm{Int}(o_L, o_L)$ is spanned by the $\sigma_{i,j}$'s as an o_L-module.

A similar pattern emerges for larger p and unramified extensions: see Figures 2 and 3 below.

More data and plots can be found on our GitHub repository.

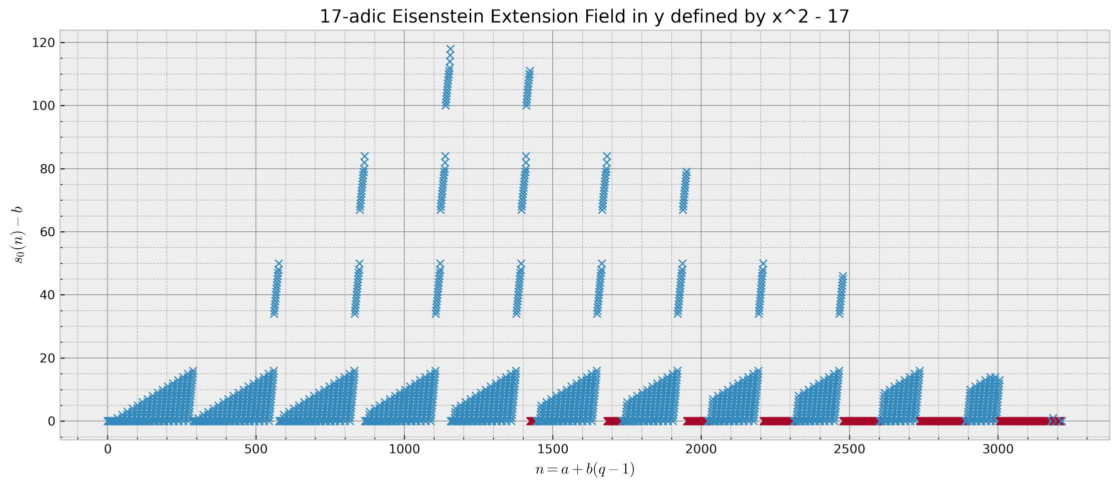

Figure 2: $s_0(n)$ in the quadratic ramified extension $\mathbb{Q}_{17}(\sqrt{17})$ for $n < 3216$. Red points are the n's for which $s_0(n) \geq 3216$.

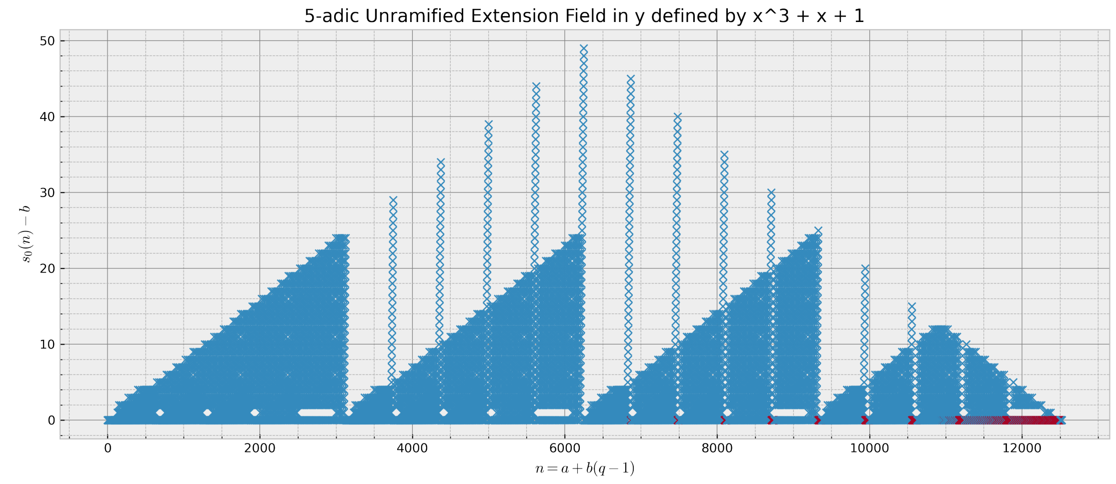

Figure 3: $s_0(n)$ in the cubic unramified extension of $\mathbb{Q}_5$ for $n < 12524$. Red points are the n's for which $s_0(n) \geq 12524$.

5 $s_q(n) < p$

TODO(Dragos)

A Sage-9.4 Code

```
extension = "3,2,100,ram"  # Choose the extension to compute with
precision = 1000           # Choose the precision that Sage will use

parse = extension.split(',')
p = int(parse[0])  # Prime to calculate with
d = int(parse[1])  # Degree to calculate with
N = int(parse[2])  # Cutoff; must be divisible by q-1
ram = parse[3]


# Python imports
from time import process_time
import matplotlib.pyplot as plt
import numpy as np

# Definitions
from sage.rings.padics.padic_generic import ResidueLiftingMap
from sage.rings.padics.padic_generic import ResidueReductionMap
import sage.rings.padics.padic_extension_generic

power = p^d - 1
t_poly = ""

if ram == "ram":
    t_poly = f"x^{d}-{p}"
else:
    # generate poly for unramified case
    Fp = GF(p)
    Fp_t.<t> = PolynomialRing(Fp)
    unity_poly = t^(power) - 1
    factored = unity_poly.factor()
    factored_str = str(factored)
    start = factored_str.find("^"+str(d))
    last_brac_pos = factored_str.find(")",start)
    first_brac_pos = len(factored_str) \
                    - factored_str[::-1].find("(",len(factored_str)-start)
    t_poly = factored_str[first_brac_pos:last_brac_pos].replace('t','x')


# Define the polynomial to adjoin a root from
Q_p = Qp(p,precision)
R_Qp.<x> = PolynomialRing(Q_p)
f_poly = R_Qp(t_poly)

# Define the p-adic field, its ring of integers and its residue field
# These dummy objects are a workaround to force the precision wanted
dummy1.<y> = Zp(p).ext(f_poly)
dummy2.<y> = Qp(p).ext(f_poly)

o_L.<y> = dummy1.change(prec=precision)
L.<y> = dummy2.change(prec=precision)
k_L = L.residue_field()
```

```
print(L)

# Find the generator of the unique maximal ideal in o_L.
Pi = o_L.uniformizer()

# Find f, e and q
f = k_L.degree()              # The degree of the residual field extension
e = L.degree()/k_L.degree()   # The ramification index
q = p^f

# Do linear algebra over the ring of polynomials L[X]
# in one variable X with coefficients in the field L:
L_X.<X> = L[]
L_Y.<Y> = L[]

v = L.valuation()

# The subroutine Dmatrix calculates the following sparse matrix of coefficients.
# Let D[k,n] be equal to k! times the coefficient of Y^k in the polynomial P_n(Y).
# I compute this using the useful and easy recursion formula
#       D[k,n] = \sum_{r \geq 0} \pi^{-r} D[k-1,n-q^r]
# that can be derived from Laurent's Prop 1.20 of "outline9".
# The algorithm is as follows: first make a zero matrix with S rows and columns
# (roughly, S is (q-1)*Size), then quickly populate it one row at a time,
# using the recursion formula.
def Dmatrix(S):
    D = matrix(L, S,S)
    D[0,0] = 1
    for k in range(1,S):
        for n in range(k,S):
            r = 0
            while n >= q^r:
                D[k,n] = D[k,n] + D[k-1,n-q^r]/Pi^r  # the actual recursion
                r = r+1
    return D


# \Tau^{(m)} in Definition 10.10 of "bounded21":
def TauMatrix(Size, m, D=None):
    if D is None:
        D = Dmatrix((q - 1) * (Size + 1))
    R = matrix(L, Size,Size, lambda x,y: D[m + (q-1)*x, m + (q-1)*y])

    # Define a diagonal matrix:
    Diag = matrix(L_X, Size,Size, lambda x,y: kronecker_delta(x,y) * X^x)

    # Compute the inverse of R:
    S = R.inverse()

    # Compute the matrix Tau using Lemma 10.11 in "bounded21":
    Tau = S * Diag * R

    return Tau
```

```
def underscore(m, i):
    return m + i*(q-1)

def w_q(n):
    return (n - sum(n.digits(base=q))) / (q-1)

def compute_s(N, filename=None):
    assert N%(q-1) == 0

    t_start = process_time()
    D = Dmatrix(N)
    t_end = process_time()
    print(f"D matrix: {t_end-t_start : .2f} sec")

    s0_s = [-1 for _ in range(N)]

    for a in range(q-1):
        t_start = process_time()
        Tau_a = TauMatrix(N//(q-1), a, D)
        t_end = process_time()
        print(f"a={a}, Tau matrix: {t_end-t_start : .2f} sec")

        B_old = Matrix(0,0)
        d = 0
        for s in range(N // (q-1)):
            t_start = process_time()

            # 1. Use the non-zero rows from previous calculations
            # 2. Add a 0 column to its left
            # 3. Add rows corresponding to entries from the j_th column of Tau_a
            B = Matrix(L, 2*s-d+1, s-d+1)
            B[:s-d, 1:] = B_old
            for i in [0 .. s]:
                coeffs = Tau_a[i, s].list()
                B[s-d+i, B.ncols()-len(coeffs)+d:] = vector(L, reversed(coeffs[d:]))

            # Perform Gaussian elimination
            i0 = 0
            ks = []
            for k in range(B.ncols()):
                valuation_row_pairs = [
                    (v(B[i,k]), i) for i in range(i0, B.nrows()) if B[i,k] != 0]

                # If we have a zero column, move on to the next one
                if not valuation_row_pairs:
                    continue
                minv, i_minv = min(valuation_row_pairs)
                ks.append(k)

                # Swap the row of minimum valuation with the first bad row
                B[i0, :], B[i_minv, :] = B[i_minv, :], B[i0, :]

                # Divide the top row by a unit in o_L
                u = B[i0, k] / Pi^int(e * v(B[i0, k]))
                B[i0, :] /= u

                # Cleave through the other rows
                for i in range(B.nrows()):
                    if i != i0 and v(B[i, k]) >= v(B[i0, k]):
                        B[i, :] -= B[i, k]/B[i0, k] * B[i0, :]

                i0 += 1

            d_is_updated = False
            for b in [d .. s]:
                n = a + b*(q-1)
                if v(B[s-b, s-b]) * e == -w_q(n):
                    if s0_s[n] == -1:
                        s0_s[n] = s
                else:
                    if not d_is_updated:
                        d = b
                        d_is_updated = True
            B_old = B[:s-d+1, :s-d+1]

            t_end = process_time()
            print(f"a={a}, s={s}: {t_end-t_start : .2f} sec", end='\r')
            if filename is not None:
                with open(filename, 'w') as f:
                    f.write("n,s0\n")
                    for n, s0 in enumerate(s0_s):
                        f.write(f"{n},{s0}\n")
        print()

    plt.style.use('bmh')
    fig = plt.figure(figsize=(15,6), dpi=300)
    for n, s0 in enumerate(s0_s):
        if s0 != -1:
            b = n // (q-1)
            plt.plot(n, s0-b, 'x', c='C0')
        else:
            plt.plot(n, 0, 'x', c='C1')
    plt.xlabel(r"$n = a + b(q-1)$")
    plt.ylabel("$s_0(n) - b$")
    plt.title(str(L))
    plt.minorticks_on()
    plt.grid(which='both')
    plt.grid(which='major', linestyle='-', c='grey')

    return s0_s, fig


s0_s = compute_s(N);
```