

Universitat de Lleida
Escola Politècnica Superior
Màster en Ciències Aplicades a l'Enginyeria

Master Thesis

**Kumar's algorithm for solving
Toeplitz systems of equations over
finite fields**

Author:

Núria Busom i Figueres

Advisers:

Josep M^a Miret Biosca

Francesc Sebé Feixas

July 2011

Contents

Contents	iii
Preface	1
1 Introduction	3
1.1 State of the art	3
1.2 Applications involving Toeplitz systems	4
1.3 Purpose	4
2 Number Theoretic Transform	7
2.1 Radix-2 DIT Cooley-Tukey's Algorithm	10
2.2 General Cooley-Tukey's Algorithm	13
2.3 Rader's algorithm	15
2.4 Shoup's circular convolution	17
3 Kumar's algorithm	21
3.1 First step of Kumar's algorithm	22
3.2 Second step of Kumar's algorithm	23
3.3 Third step of Kumar's algorithm	23
4 Adapting Kumar's algorithm	25
4.1 Working over field extensions	29
5 Outcomes and Future work	31
5.1 Outcomes	31
5.2 Further improvements	33
A Polynomial multiplication and convolutions	35
A.1 Polynomial multiplication and linear convolution	35
A.2 Linear and circular convolution	37
Bibliography	39

Preface

In [Kum85], Kumar presented an algorithm that solves a Toeplitz system of n linear equations with n unknowns defined over the real field in time $O(n \log^2 n)$. In this work, we have studied how to use Kumar's algorithm when the system of equations is defined over a finite field. This task is not straightforward due to some difficulties that arise from the computation of Fast Fourier Transforms over finite fields.

During our work, the following tasks have been performed:

- (i) Study Kumar's algorithm.
- (ii) Learn the basis of finite fields theory.
- (iii) Study the Discrete Fourier Transform over real numbers and its equivalent over finite fields, *i.e.*, the Number Theoretic Transform (NTT).
- (iv) Study how the NTT can be computed in quasi-linear time.
- (v) Analyse how to adapt Kumar's algorithm so as to work over finite fields.

Chapter 1

Introduction

In this chapter, the concept of Toeplitz matrix is defined and some methods to solve equation systems with such a coefficients matrix are referenced. This kind of linear equation systems appear in many different areas, as we will show in section [1.2](#).

1.1 State of the art

Solving an $n \times n$ linear system of equations employing the Gauss algorithm has a complexity $O(n^3)$. Fortunately, when the coefficients matrix is a Toeplitz matrix, there exist faster algorithms.

A Toeplitz matrix T of order n , whose elements belong to a given field \mathbb{K} , is a squared matrix in which each descending diagonal from left to right is constant. That is:

$$T = \begin{pmatrix} t_0 & t_{-1} & t_{-2} & \cdots & t_{-n+1} \\ t_1 & t_0 & t_{-1} & \cdots & t_{-n+2} \\ t_2 & t_1 & t_0 & \cdots & t_{-n+3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_{n-1} & t_{n-2} & t_{n-3} & \cdots & t_0 \end{pmatrix}.$$

The solution of a linear system of equations entailing a Toeplitz matrix can be found in quadratic time using iterative methods, like Levinson's algorithm [[Lev47](#)], or computing the inverse of the matrix.

Levinson recursion is a procedure to recursively calculate the solution to a Toeplitz system which runs in $O(n^2)$. It was first proposed by Norman Levinson in 1947 and it was later improved by Durbin [[Dur60](#)] in 1960 and subsequently by Trench [[Tre64](#)] and Zohar [[Zoh74](#)].

Faster algorithms running in quasi-linear time do exist. Rajendra Kumar [Kum85] proposed an algorithm for solving systems of linear equations involving a Toeplitz matrix over the real field, which has a running time $O(n \log^2 n)$. This algorithm is a fast implementation of Trench's algorithm [Tre64].

1.2 Applications involving Toeplitz systems

Toeplitz systems appear in many areas, some of them are exposed below:

- In cryptography, a good elliptic curve has prime (or semi-prime) cardinality (number of points) [BMSS06]. A semi-prime number is a big prime number multiplied by a small cofactor. Once a good elliptic curve is known, by means of isogeny computations, we can obtain other curves with the same cardinality. Some fast methods for isogeny computation involve Toeplitz matrices.
- In signal processing, discrete convolutions are widely used, for instance in digital filters [FS74], [SA96]. Convolutions can be considered a Toeplitz Matrix operation where each row is a shifted copy of the convolution kernel.
- In digital image processing, a restoration images process is performed using Toeplitz systems [Kim08]. Such processing has many applications, such as in satellite, military reconnaissance missions, medical, forensic science and astronomical imaging. It is also applicable in the restoration of poor-quality family portraits.

1.3 Purpose

The purpose of this Master Thesis is to study Kumar's algorithm and see how it can be used to work over finite fields. The resulting adaptation has been implemented in Sage [Sage].

Sage is a free open-source mathematics software system licensed under the GPL. It is a compilation of original Python, C, C++, and Cython code, and existing free mathematics-related software (Magma, Maple, Mathematica, Matlab, Maxima,...).

This document is divided into five chapters, a preface and an appendix. In the preface, the work carried out during the development of this project is enumerated. In the first chapter, the definition of a Toeplitz matrix is given,

as well as some references to fast methods to solve Toeplitz systems of equations and some real applications. The second chapter, is a background on the Discrete Fourier and Number Theoretic Transforms and some fast algorithms to solve them. In the third one, Kumar's algorithm is summarized and in the fourth one, our modifications and contributions to it are detailed. In the last chapter, some numerical results are shown and possible improvements to this work are enumerated. Finally, in appendix, the relation between polynomial multiplication and linear and circular convolutions is explained.

Chapter 2

Number Theoretic Transform

Kumar's algorithm makes use of the Fast Fourier Transform (FFT) to speed up its computations. In this chapter, we review its basis.

Discrete Fourier Transforms are computed in $O(n \log n)$ using FFT algorithms. Such algorithms depend on the factorization of the input sequence length. In this chapter, some of them are detailed and exemplified focusing on their applicability to solve number theoretic transforms. After this study we conclude that any NTT can be computed in quasi linear time. We wish to stress that FFT algorithms are found described over the real and the complex fields, so that, an extra work to see its applicability over finite fields has been necessary.

The **Discrete Fourier Transform** (DFT) transforms a discrete function into another represented in the frequency domain. The input to the DFT is a finite sequence of **real or complex** samples.

The DFT is widely employed in signal processing and related fields to analyse the frequencies contained in a sampled signal, to solve partial differential equations, and to perform other operations such as convolutions or multiplication of large integers.

A given sequence of N complex numbers x_0, x_1, \dots, x_{N-1} is transformed into a sequence of N complex numbers X_0, X_1, \dots, X_{N-1} by the DFT according to the formula:

$$X_j = \sum_{k=0}^{N-1} x_k e^{-\frac{2\pi i}{N} jk}, \quad j = 0, 1, \dots, N-1, \quad (2.1)$$

where i is the imaginary unit and $e^{\frac{2\pi i}{N}}$ is a primitive N -th root of unity.

Definition 1 z is an N -th root of unity, for positive integers N , if

$$z^N = 1.$$

Definition 2 An N -th root of unity z is primitive if

$$z^k \neq 1, \quad \forall k \in \{1, 2, \dots, N-1\}.$$

The Inverse discrete Fourier transform (IDFT) is given by:

$$x_j = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N} jk}, \quad j = 0, 1, \dots, N-1. \quad (2.2)$$

The **Number Theoretic Transform** (NTT) is the equivalent to DFT, when operating over **finite fields**.

Given a prime field \mathbb{F}_p and a sequence x_0, x_1, \dots, x_{N-1} of N elements of \mathbb{F}_p , in order to consider the corresponding transformed sequence we need that $N \mid (p-1)$, that is $p = \zeta N + 1$ for some positive integer ζ . The equivalent to $e^{\frac{2\pi i}{N}}$ in DFT is now ω^ζ (a positive N -th root of the unity) such that ω is a primitive ζN -root of the unity. Therefore, ζ is the lowest positive integer such that $\omega^{\zeta N} \equiv 1 \pmod{p}$.

Let $x = (x_0, x_1, \dots, x_{N-1})$ be the input sequence of field elements and $X = (X_0, X_1, \dots, X_{N-1})$ be the NTT of x . The sequence $X = NTT(x)$ is defined as:

$$X_j \equiv \sum_{k=0}^{N-1} x_k (\omega^\zeta)^{jk} \pmod{p}, \quad j = 0, 1, \dots, N-1. \quad (2.3)$$

Similarly, the Inverse Number Theoretic Transform $x = NTT^{-1}(X)$ is given by expression:

$$x_j \equiv N^{-1} \sum_{k=0}^{N-1} X_k (\omega^{-\zeta})^{jk} \pmod{p}, \quad j = 0, 1, \dots, N-1, \quad (2.4)$$

where N^{-1} and $\omega^{-\zeta}$ are the modular multiplicative inverses of N and ω^ζ , respectively.

Example 1 Computation of the NTT of the five elements sequence

$$x = \boxed{4 \mid 1 \mid 7 \mid 9 \mid 8}$$

over the finite field \mathbb{F}_{11} :

The length of the array is $N=5$. Therefore, $\zeta=2$, since:

$$\zeta = \frac{p-1}{N} = 2$$

and $\omega = 2$ is a primitive 10-th root of the unity. Indeed

$$\omega = 2, \omega^2 = 4, \omega^3 = 8, \omega^4 = 5, \omega^5 = 10, \omega^6 = 9, \omega^7 = 7, \omega^8 = 3, \omega^9 = 6, \omega^{10} = 1.$$

Now, we can compute the NTT with $\omega^\zeta = 2^2 = 4$ as:

$$\begin{aligned} X_0 &= 4 \cdot (\omega^2)^{0 \cdot 0} + 1 \cdot (\omega^2)^{1 \cdot 0} + 7 \cdot (\omega^2)^{2 \cdot 0} + 9 \cdot (\omega^2)^{3 \cdot 0} + 8 \cdot (\omega^2)^{4 \cdot 0} = 29 \\ &\equiv 7 \pmod{11} \end{aligned}$$

$$\begin{aligned} X_1 &= 4 \cdot (\omega^2)^{0 \cdot 1} + 1 \cdot (\omega^2)^{1 \cdot 1} + 7 \cdot (\omega^2)^{2 \cdot 1} + 9 \cdot (\omega^2)^{3 \cdot 1} + 8 \cdot (\omega^2)^{4 \cdot 1} = 148 \\ &\equiv 5 \pmod{11} \end{aligned}$$

$$\begin{aligned} X_2 &= 4 \cdot (\omega^2)^{0 \cdot 2} + 1 \cdot (\omega^2)^{1 \cdot 2} + 7 \cdot (\omega^2)^{2 \cdot 2} + 9 \cdot (\omega^2)^{3 \cdot 2} + 8 \cdot (\omega^2)^{4 \cdot 2} = 996 \\ &\equiv 6 \pmod{11} \end{aligned}$$

$$\begin{aligned} X_3 &= 4 \cdot (\omega^2)^{0 \cdot 3} + 1 \cdot (\omega^2)^{1 \cdot 3} + 7 \cdot (\omega^2)^{2 \cdot 3} + 9 \cdot (\omega^2)^{3 \cdot 3} + 8 \cdot (\omega^2)^{4 \cdot 3} = 7720 \\ &\equiv 9 \pmod{11} \end{aligned}$$

$$\begin{aligned} X_4 &= 4 \cdot (\omega^2)^{0 \cdot 4} + 1 \cdot (\omega^2)^{1 \cdot 4} + 7 \cdot (\omega^2)^{2 \cdot 4} + 9 \cdot (\omega^2)^{3 \cdot 4} + 8 \cdot (\omega^2)^{4 \cdot 4} = 64332 \\ &\equiv 4 \pmod{11} \end{aligned}$$

$$X = \boxed{7 \mid 5 \mid 6 \mid 9 \mid 4}$$

A Fast Fourier Transform (FFT) is a computationally efficient algorithm that computes the DFT and its inverse in time $O(n \log n)$. There exist many different FFT algorithms that apply to different cases. For instance, Radix-2 DIT Cooley-Tukey's Algorithm (for sequences with a power of two length), General Cooley-Tukey's Algorithm (for sequences whose length is a composite number) and Rader's algorithm (for prime length sequences). In the following sections, they will be explained focusing on their applicability to NTT.

After studying them, we have concluded that a fast computation of NTT (of any length) can be achieved in a recursive manner that employs a combination of the previously cited algorithms. These three algorithms have been compiled in procedure 2.0.1. Figure 2.1 shows this in a graphical manner.

Procedure 2.0.1 Fast Fourier Transform

Input: x_0, \dots, x_{N-1} a size N input sequence of elements of \mathbb{F}_p and ω^ζ an order N element of \mathbb{F}_p .

Output: X_0, \dots, X_{N-1} the NTT of x_0, \dots, x_{N-1}

if *length* is a power of two **then**

return Radix-2-DIT($x_0, \dots, x_{N-1}, N, \omega^\zeta$)

else if *length* is prime **then**

return Rader($x_0, \dots, x_{N-1}, N, \omega^\zeta$)

else

return General-Cooley-Tukey($x_0, \dots, x_{N-1}, N, \omega^\zeta$)

end if

return X_0, \dots, X_{N-1} .

2.1 Radix-2 DIT Cooley-Tukey's Algorithm

The radix-2 decimation-in-time (DIT) is a particular case of FFT, which can be applied when the input sequence length is a power of two. It is easy to see that this (divide-and-conquer) recursive algorithm runs in $O(n \log n)$.

This method reduces the computation of one NTT of size N to the computation of two interleaved NTTs of size $N/2$. This algorithm requires N to be a power of two. Radix-2 DIT recursively computes the NTTs (equation 2.3) of the even-indexed inputs and the odd-indexed inputs and then melds those results to produce the overall NTT of the entire sequence.

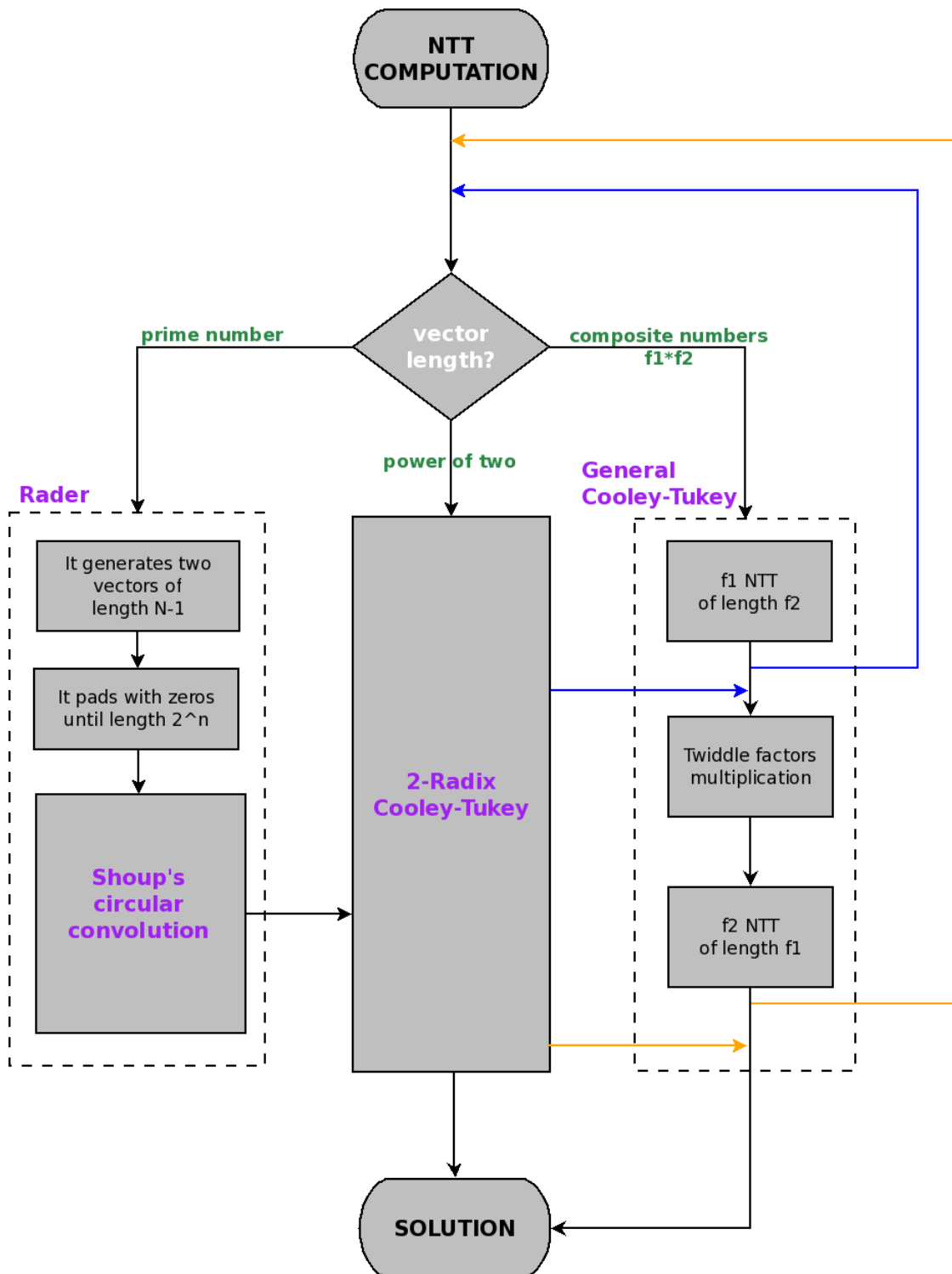


Figure 2.1: FFT computation Scheme

Procedure 2.1.1 Radix-2 DIT Cooley-Tukey's

Input: x_0, \dots, x_{N-1} a size N sequence of elements of \mathbb{F}_p with $p = \zeta N + 1$ and N a power of two, ω^ζ an order N element of \mathbb{F}_p and $s=1$.

Output: X_0, \dots, X_{N-1} the NTT of x_0, \dots, x_{N-1}

if $N=1$ **then**

$X_0 \leftarrow x_0$ //trivial size-1 NTT case

else

$X_0, \dots, X_{N/2-1} \leftarrow \text{Radix-2 DIT Cooley-Tukey's}(x, N/2, (\omega^\zeta)^2, 2s)$

$X_{N/2}, \dots, X_{N-1} \leftarrow \text{Radix-2 DIT Cooley-Tukey's}(x + s, N/2, (\omega^\zeta)^2, 2s)$

for $k = 0$ to $N/2 - 1$ **do**

$aux \leftarrow X_k$

$X_k \leftarrow aux + (\omega^\zeta)^k X_{k+N/2}$

$X_{k+N/2} \leftarrow aux - (\omega^\zeta)^k X_{k+N/2}$

end for

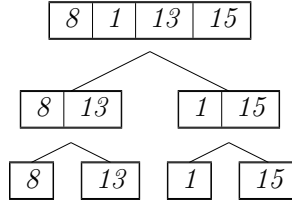
end if

Example 2 Computation of the NTT of the size 4 input sequence

8	1	13	15
---	---	----	----

over the finite field \mathbb{F}_{17} with $\omega = 3$, $\zeta = 4$, $\omega^\zeta = 13$ employing Radix-2 DIT algorithm.

The algorithm is recursively called with the even and odd-indexed inputs.



$$\begin{array}{ll}
 8 + ((\omega^\zeta)^2)^0 \cdot 13 = 4 \pmod{17} & \searrow \\
 8 - ((\omega^\zeta)^2)^0 \cdot 13 = 12 \pmod{17} & \nearrow \\
 1 + ((\omega^\zeta)^2)^0 \cdot 15 = 16 \pmod{17} & \searrow \\
 1 - ((\omega^\zeta)^2)^0 \cdot 15 = 3 \pmod{17} & \nearrow
 \end{array}
 \begin{array}{l}
 \boxed{4 \mid 12} \\
 \boxed{16 \mid 3}
 \end{array}
 \begin{array}{ll}
 4 + (\omega^\zeta)^0 \cdot 16 = 3 \pmod{17} & \\
 12 + (\omega^\zeta)^1 \cdot 3 = 0 \pmod{17} & \\
 4 - (\omega^\zeta)^0 \cdot 16 = 5 \pmod{17} & \\
 12 - (\omega^\zeta)^1 \cdot 3 = 7 \pmod{17} &
 \end{array}
 \underbrace{\hspace{10em}}
 \begin{array}{l}
 X: \boxed{3 \mid 0 \mid 5 \mid 7}
 \end{array}$$

2.2 General Cooley-Tukey's Algorithm

The general Cooley-Tukey's Algorithm [DV90] reduces the computation of an NTT over an array $x = x_0, \dots, x_{N-1}$ of size $N = N_1 N_2$ to the computation of N_1 NTT's of size N_2 plus the computation of N_2 NTT's of size N_1 . Given ω^ζ , a primitive N -root of unity, this is done in the following four steps:

- (i) Compute N_1 NTT's of size N_2 with $(\omega^\zeta)^{N_1}$, a primitive N_2 -root of unity: Let y_0, \dots, y_{N_1-1} be size N_2 arrays, with $y_i[j] = x[j \cdot N_1 + i]$, $1 \leq i \leq N_1$, $1 \leq j \leq N_2$. Compute the NTT of each y_i and store the result in vectors Y_i , $1 \leq i \leq N_1$.
- (ii) Apply the "twiddle factors" (roots of unity) over the result of the previous step: Given vectors Y_i , compute $Y'_i[j] = Y_i[j] \cdot (\omega^\zeta)^{ij}$, $1 \leq i \leq N_1$, $1 \leq j \leq N_2$.
- (iii) Compute N_2 NTT's of size N_1 with $(\omega^\zeta)^{N_2}$ as a primitive N_1 -root of unity: Given vectors Y'_i , let w_0, \dots, w_{N_2-1} be size N_1 arrays, with $w_j[i] = Y'_i[j]$. Compute the NTT of each w_i and store the result in vectors W_i , $1 \leq i \leq N_1$, $1 \leq j \leq N_2$.
- (iv) Compose the final result Z as $Z[k] = W_{k \pmod{N_2}}[k \div N_2]$, $0 \leq k \leq N-1$.

In our implementation, we have taken N_1 as the largest prime factor of N . This permits that, in the case that $N = 2^k N'$, applying this algorithm recursively, at the end we will have to compute the NTT of a vector whose length is 2^k which can be done efficiently using Radix-2 DIT algorithm. Moreover, this facilitates our computations by avoiding some recursive calls to Shoup's circular convolution. Later we will see that Shoup's circular convolution is hard to compute when extended fields are involved.

Example 3 NTT computation of the size 12 sequence

$$x = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 1 & 4 & 11 & 3 & 1 & 7 & 9 & 8 & 2 & 10 & 6 & 1 \\ \hline \end{array}$$

over the finite field \mathbb{F}_{13} with $\omega = 2$, $\zeta = 1$, $\omega^\zeta = 2$ using the General Cooley-Tukey's algorithm

- (i) Let the length of the array factorize as $12 = 2^2 \cdot 3$. We assign $N_1 = 3$ and $N_2 = 4$

(ii) Perform 3 NTT of size 4. As the length is a power of two, this can be done with the Radix-2 DIT algorithm

$$\begin{aligned}
 y_0 &= \boxed{1 \mid 3 \mid 9 \mid 10} \xrightarrow{FFT} Y_0 = \boxed{10 \mid 1 \mid 10 \mid 9} \\
 y_1 &= \boxed{4 \mid 1 \mid 8 \mid 6} \xrightarrow{FFT} Y_1 = \boxed{6 \mid 8 \mid 5 \mid 10} \\
 y_2 &= \boxed{11 \mid 7 \mid 2 \mid 1} \xrightarrow{FFT} Y_2 = \boxed{8 \mid 5 \mid 5 \mid 0}
 \end{aligned}$$

(iii) Multiply the resulting vectors by the twiddle factors

$$\begin{array}{lll}
 Y_0 = \boxed{10 \mid 1 \mid 10 \mid 9} & Y_1 = \boxed{6 \mid 8 \mid 5 \mid 10} & Y_2 = \boxed{8 \mid 5 \mid 5 \mid 0} \\
 10 \cdot (\omega^\zeta)^{0 \cdot 0} = 10 & 6 \cdot (\omega^\zeta)^{1 \cdot 0} = 6 & 8 \cdot (\omega^\zeta)^{2 \cdot 0} = 8 \\
 1 \cdot (\omega^\zeta)^{0 \cdot 1} = 1 & 8 \cdot (\omega^\zeta)^{1 \cdot 1} = 3 & 5 \cdot (\omega^\zeta)^{2 \cdot 1} = 7 \\
 10 \cdot (\omega^\zeta)^{0 \cdot 2} = 10 & 5 \cdot (\omega^\zeta)^{1 \cdot 2} = 7 & 5 \cdot (\omega^\zeta)^{2 \cdot 2} = 2 \\
 9 \cdot (\omega^\zeta)^{0 \cdot 3} = 9 & 10 \cdot (\omega^\zeta)^{1 \cdot 3} = 2 & 0 \cdot (\omega^\zeta)^{2 \cdot 3} = 0 \\
 Y'_0 = \boxed{10 \mid 1 \mid 10 \mid 9} & Y'_1 = \boxed{6 \mid 3 \mid 7 \mid 2} & Y'_2 = \boxed{8 \mid 7 \mid 2 \mid 0}
 \end{array}$$

(iv) Compute 4 NTT of size 3. As 3 is a prime number, this can be done through the Rader's algorithm

$$\begin{aligned}
 w_0 &= \boxed{10 \mid 6 \mid 8} \xrightarrow{FFT} W_0 = \boxed{11 \mid 9 \mid 10} \\
 w_1 &= \boxed{1 \mid 3 \mid 7} \xrightarrow{FFT} W_1 = \boxed{11 \mid 8 \mid 10} \\
 w_2 &= \boxed{10 \mid 7 \mid 2} \xrightarrow{FFT} W_2 = \boxed{6 \mid 10 \mid 1} \\
 w_3 &= \boxed{9 \mid 2 \mid 0} \xrightarrow{FFT} W_3 = \boxed{11 \mid 2 \mid 1}
 \end{aligned}$$

(v) Compose the final result

$$Z = \boxed{11 \mid 11 \mid 6 \mid 11 \mid 9 \mid 8 \mid 10 \mid 2 \mid 10 \mid 10 \mid 1 \mid 1}$$

2.3 Rader's algorithm

When the length of our input vector is prime, Cooley-Tukey recursion can not be applied. In this case, Rader's algorithm [Rad68] re-expresses the NTT of a sequence of prime length N as a cyclic convolution of two length $N - 1$ sequences.

Remember that the NNT of a sequence x_0, \dots, x_{N-1} with elements over \mathbb{F}_p is defined as:

$$X_j \equiv \sum_{k=0}^{N-1} x_k (\omega^\zeta)^{jk} \pmod{p}, \quad j = 0, 1, \dots, N - 1.$$

When N is a prime number, the sequence of indices $k = 1, \dots, N - 1$ forms a multiplicative group modulo N , which is cyclic. Then, there exists a primitive root of unity ω , so that $\forall k, 1 \leq k \leq N - 1$ there exists $r \in \mathbb{N}$ such that $k = \omega^r \pmod{N}$. Similarly, we can define the generator of indices $j = 1, \dots, N - 1$ as $j = \omega^{-s} \pmod{N}$.

Substituting k and j , we obtain a new definition of the NTT:

$$X_0 = \sum_{k=0}^{N-1} x_k, \quad (2.5)$$

$$X_{\omega^{-s}} = x_0 + \sum_{r=0}^{N-2} x_{\omega^r} (\omega^{-\zeta})^{\omega^{r-s}}, \quad s = 0, 1, \dots, N - 2, \quad (2.6)$$

where the second addend of equation 2.6 is the same as the circular convolution of two sequences of size $N - 1$ (as it can be seen in A), obtaining:

$$X_{\omega^{-s}} = x_0 + \underbrace{(x_{\omega^r} \otimes (\omega^{-\zeta})^{\omega^{-r}})}_{\text{circular convolution}}, \quad s = 0, 1, \dots, N - 2. \quad (2.7)$$

In order to efficiently convolute them using the Shoup's circular convolution (section 2.4), the length must be a power of two so that it can call the Radix-2 DIT Cooley-Tukey's algorithm (section 2.1). To this end, we first append zeros to the end of these arrays to double their length (now it is a linear convolution) and then we add some more zeros until their length is a power of two. For more details about convolutions see appendix A.

Example 4 Computation of the NTT of the length 5 sequence

$$x = \boxed{1 \mid 8 \mid 5 \mid 10 \mid 7}$$

over the finite field \mathbb{F}_{11} with $\omega = 2$, $\zeta = 2$, $\omega^\zeta = 4$

(i) Compute two new arrays a, b of length $N - 1$

$$\left. \begin{array}{l} \omega^0 = 1 \rightarrow a_0 = x_1 = 8 \\ \omega^1 = 2 \rightarrow a_1 = x_2 = 5 \\ \omega^2 = 4 \rightarrow a_2 = x_4 = 7 \\ \omega^3 = 3 \rightarrow a_3 = x_3 = 10 \end{array} \right\} \rightarrow a = \boxed{8 \mid 5 \mid 7 \mid 10}$$

$$\left. \begin{array}{l} (w^{-\zeta})^{\omega^0} = 4 \rightarrow b_0 = 4 \\ (w^{-\zeta})^{\omega^{-1}} = 9 \rightarrow b_1 = 9 \\ (w^{-\zeta})^{\omega^{-2}} = 3 \rightarrow b_2 = 3 \\ (w^{-\zeta})^{\omega^{-3}} = 5 \rightarrow b_3 = 5 \end{array} \right\} \rightarrow b = \boxed{4 \mid 9 \mid 3 \mid 5}$$

(ii) Perform the linear convolution $a \otimes b$.

$$\left. \begin{array}{l} a = \boxed{8 \mid 5 \mid 7 \mid 10 \mid 0 \mid 0 \mid 0 \mid 0} \\ b = \boxed{4 \mid 9 \mid 3 \mid 5 \mid 0 \mid 0 \mid 0 \mid 0} \end{array} \right\} \xrightarrow{\otimes} \boxed{10 \mid 4 \mid 9 \mid 4 \mid 4 \mid 10 \mid 6 \mid 0}$$

(iii) Transform the linear convolution into the circular one.

$$\boxed{10 \mid 4 \mid 9 \mid 4 \mid 4 \mid 10 \mid 6 \mid 0} \xrightarrow{\text{linear to circular conv}} cc = \boxed{3 \mid 3 \mid 4 \mid 4}$$

(iv) Join these results as formula 2.7:

$$\begin{aligned} X_0 &= \sum_{k=0}^{N-1} x_k = 1 + 8 + 5 + 10 + 7 = 31 \pmod{11} = 9 \\ X_1 &= x_0 + cc_0 = 1 + 3 = 4 \\ X_2 &= x_0 + cc_3 = 1 + 4 = 5 \\ X_3 &= x_0 + cc_1 = 1 + 3 = 4 \\ X_4 &= x_0 + cc_2 = 1 + 4 = 5 \end{aligned}$$

$$X = \boxed{9 \mid 4 \mid 5 \mid 4 \mid 5}$$

2.4 Shoup's circular convolution

Victor Shoup [Sho96] implemented a fast $O(n \log n)$ FFT-based algorithm to efficiently multiply and divide polynomials of high degree (more than 30). Since the multiplication of two polynomials can be viewed as the convolution of two signals whose samples are the polynomials coefficients, we will employ Shoup's proposal to this end. The description we give assumes the input signal length is a power of two. In this way, the very efficient Radix-2 DIT NTT algorithm can be employed.

The algorithm to circularly convolute two signals g, h with coefficients in \mathbb{F}_p whose amount of samples is 2^M is performed in 4 steps:

- (i) Choose a set of "NTT-primes" q_1, \dots, q_ℓ such that:
 - $2^M \mid q_i - 1$, for $1 \leq i \leq \ell$.
 - The product $P = \prod_{i=1}^{\ell} q_i$ is larger than $2^M p^2$.
- (ii) Reduce all the coefficients modulo each of the "NTT-primes" q_1, \dots, q_ℓ , obtaining signals g_i, h_i with coefficients in \mathbb{F}_{q_i} , for $1 \leq i \leq \ell$.
- (iii) Compute $u_i = g_i \cdot h_i$ using the FFT-based **circular convolution**, which consists of 3 steps:
 - Compute the NTT of each g_i, h_i , for $1 \leq i \leq \ell$,

$$G_i = NTT(g_i), H_i = NTT(h_i), \text{ for } 1 \leq i \leq \ell.$$
 - Compute $U_i = G_i \cdot H_i$, for $1 \leq i \leq \ell$, multiplying samples one by one.
 - Compute the inverse NTT (INTT) of each U_i ,

$$U_i = INTT(u_i), \text{ for } 1 \leq i \leq \ell.$$
- (iv) Apply the Chinese Remainder Theorem (CRT) to obtain the final solution s in \mathbb{F}_p .

In our implementation, this algorithm is always called to convolute the sequences resulting from Rader's reduction. Such sequences are previously padded with zeros until its length is a power of two. This permits NTT and INTT to be performed employing Radix-2 DIT Cooley-Tukey's algorithm 2.1.

It is also worth mentioning that, since the coefficients of the input sequence are reduced modulo a prime, we need to provide a new ω for each "NTT-prime".

Example 5 Computation of the circular convolution of length 4 inputs

$$a = \boxed{54} \boxed{123} \boxed{2} \boxed{23} \quad b = \boxed{82} \boxed{37} \boxed{69} \boxed{36}$$

over the finite field \mathbb{F}_{127} :

(i) Compute the “NTT-primes”:

$$\text{NTT-primes} = \boxed{17} \boxed{97} \boxed{113} \boxed{193}$$

since:

$$\begin{array}{l} 2^4 \mid 16 \\ 2^4 \mid 96 \\ 2^4 \mid 112 \\ 2^4 \mid 192 \end{array} \quad \text{and} \quad \prod_{i=1}^4 q_i = 17 \cdot 97 \cdot 113 \cdot 193 = 35963041 > 258064 = 2^4 \cdot 127^2.$$

(ii) Reduce each array modulo each one of the “NTT-primes”:

$$\begin{array}{ll} a_{17} = \boxed{3} \boxed{4} \boxed{2} \boxed{6} & b_{17} = \boxed{14} \boxed{3} \boxed{1} \boxed{2} \\ a_{97} = \boxed{54} \boxed{26} \boxed{2} \boxed{23} & b_{97} = \boxed{82} \boxed{37} \boxed{69} \boxed{36} \\ a_{113} = \boxed{54} \boxed{10} \boxed{2} \boxed{23} & b_{113} = \boxed{82} \boxed{37} \boxed{69} \boxed{36} \\ a_{193} = \boxed{54} \boxed{123} \boxed{2} \boxed{23} & b_{193} = \boxed{82} \boxed{37} \boxed{69} \boxed{36} \end{array}$$

(iii) The circular convolution is computed in three steps: the computation of the NTT, multiplication of the coefficients one by one and the computation of the INTT:

$$\begin{array}{ll} a_{17} \xrightarrow{\text{NTT}} \boxed{15} \boxed{9} \boxed{12} \boxed{10} & \searrow \\ b_{17} \xrightarrow{\text{NTT}} \boxed{3} \boxed{9} \boxed{10} \boxed{0} & \nearrow \end{array} \rightarrow \boxed{11} \boxed{13} \boxed{1} \boxed{0} \xrightarrow{\text{INTT}} \boxed{2} \boxed{7} \boxed{4} \boxed{15}$$

$$\begin{array}{ll} a_{97} \xrightarrow{\text{NTT}} \boxed{8} \boxed{21} \boxed{7} \boxed{83} & \searrow \\ b_{97} \xrightarrow{\text{NTT}} \boxed{30} \boxed{35} \boxed{78} \boxed{88} & \nearrow \end{array} \rightarrow \boxed{46} \boxed{56} \boxed{61} \boxed{29} \xrightarrow{\text{INTT}} \boxed{48} \boxed{66} \boxed{54} \boxed{72}$$

$$\begin{array}{ll} a_{113} \xrightarrow{\text{NTT}} \boxed{89} \boxed{21} \boxed{23} \boxed{83} & \searrow \\ b_{113} \xrightarrow{\text{NTT}} \boxed{111} \boxed{111} \boxed{78} \boxed{28} & \nearrow \end{array} \rightarrow \boxed{48} \boxed{71} \boxed{99} \boxed{64} \xrightarrow{\text{INTT}} \boxed{14} \boxed{70} \boxed{3} \boxed{74}$$

$$\begin{array}{ll} a_{193} \xrightarrow{\text{NTT}} \boxed{9} \boxed{58} \boxed{103} \boxed{46} & \searrow \\ b_{193} \xrightarrow{\text{NTT}} \boxed{31} \boxed{125} \boxed{78} \boxed{94} & \nearrow \end{array} \rightarrow \boxed{86} \boxed{109} \boxed{121} \boxed{78} \xrightarrow{\text{INTT}} \boxed{2} \boxed{40} \boxed{5} \boxed{39}$$

(iv) And finally, these four results are combined with the Chinese Remainder Theorem in order to obtain the final solution modulo 127, which is:

66	27	125	72
----	----	-----	----

Chapter 3

Kumar's algorithm

In this chapter, Kumar's algorithm to solve Toeplitz systems over the real field is described. This algorithm is composed of three differentiated steps. However, just the first one is widely detailed since this is the only one whose adaptation to work over finite fields is not straightforward.

Kumar [Kum85] proposed an algorithm, which is a modification of the Trench algorithm [Tre64], to solve a Toeplitz system of equations $Tx = y$ over real numbers, where T is a Toeplitz matrix of order $n + 1$, x is the $n + 1$ -size vector of unknowns and y is a known vector of size $n + 1$. Instead of beginning with the inversion of a lower-size matrix and repetitively compute inverse matrices of higher dimension, Kumar reverses the process which mainly consists of three following steps:

- (i) Embed the Toeplitz matrix T into a Circulant matrix C (a circulant matrix is a Toeplitz one which is a cyclic right shift matrix) and compute the inverse of C .
- (ii) Compute the inverse of the Toeplitz matrix T , T^{-1} , from the first row and column of C^{-1} .
- (iii) Solve the Toeplitz system in terms of the first row and column of T^{-1} .

3.1 First step of Kumar's algorithm

Computation of the first row and column of the inverse circulant matrix performing Fourier transforms.

- (i) Embed the Toeplitz matrix T of order $n + 1$

$$T = \begin{pmatrix} t_0 & t_{-1} & \dots & t_{-n} \\ t_1 & t_0 & \dots & t_{-n+1} \\ t_2 & t_1 & \dots & t_{-n+2} \\ \vdots & \vdots & \ddots & \vdots \\ t_n & t_{n-1} & \dots & t_0 \end{pmatrix}$$

into a Circulant matrix C of order $2n + 1$,

$$C = \begin{pmatrix} t_0 & t_{-1} & \dots & t_{-n} & t_n & t_{n-1} & \dots & t_1 \\ t_1 & t_0 & \dots & t_{-n+1} & t_{-n} & t_n & \dots & t_2 \\ t_2 & t_1 & \dots & t_{-n+2} & t_{-n+1} & t_{-n} & \dots & t_3 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ t_n & t_{n-1} & \dots & t_0 & t_{2n} & t_{2n-1} & \dots & t_{n+1} \\ t_{n+1} & t_n & \dots & t_1 & t_0 & t_{2n} & \dots & t_{n+2} \\ t_{n+2} & t_{n+1} & \dots & t_2 & t_1 & t_0 & \dots & t_{n+3} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ t_{2n} & t_{2n-1} & \dots & t_{n+2} & t_{n+1} & t_n & \dots & t_0 \end{pmatrix}.$$

Note that a circulant matrix is uniquely represented by its first row. We will denote the first row of C by $C_{(1)}$.

- (ii) Let us assume C is non-singular (with non-zero determinant), then its inverse matrix C^{-1} does exist (it is also circulant). Given the first row of C , the first row of C^{-1} can be computed as:
- (a) Compute the DFT of the sequence $C_{(1)}$.
 - (b) Compute the sequence $C'_{(1)} = 1/C_{(1)}$.
 - (c) Compute $C^{-1}_{(1)}$, the IDFT of the sequence $C'_{(1)}$.

The computational cost of step **i** is $O(n)$, while the cost of step **ii** is dominated by the computation of DFT transforms. This motivates the need to employ efficient FFT algorithms to compute such transformations.

3.2 Second step of Kumar's algorithm

Computation of the first row and column of the inverse Toeplitz matrix from the first row and column of the inverse circulant matrix.

- (i) Let $r_{\bar{n},0}, r_{\bar{n},1}, \dots, r_{\bar{n},\bar{n}}$ and $c_{0,\bar{n}}, c_{1,\bar{n}}, \dots, c_{\bar{n},\bar{n}}$, $\bar{n} = 2n$ represent the first row and the first column elements of C^{-1} obtained from Step 1.
- (ii) Define polynomials:

$$r_{\bar{n}}(x) = \sum_{i=0}^{\bar{n}} r_{\bar{n},i} x^i, \quad c_{\bar{n}}(x) = \sum_{i=0}^{\bar{n}} c_{\bar{n},i} x^i, \quad r_{\bar{n},0}, c_{\bar{n},0} \neq 0.$$

Furthermore, define also polynomials:

$$\hat{r}_{\bar{n}}(x) = \sum_{i=0}^{\bar{n}} r_{\bar{n},i} x^{\bar{n}-i}, \quad \hat{c}_{\bar{n}}(x) = \sum_{i=0}^{\bar{n}} c_{\bar{n},i} x^{\bar{n}-i}.$$

- (iii) The corresponding polynomials $r_n(x), c_n(x)$ for the desired matrix T^{-1} are obtained by the following iterative procedure:

$$r_{\bar{n}-i-1}(x) = \text{remainder in the polynomial division } \frac{r_{\bar{n}-i}(x)}{\hat{c}_{\bar{n}-i}(x)},$$

$$c_{\bar{n}-i-1}(x) = \text{remainder in the polynomial division } \frac{c_{\bar{n}-i}(x)}{\hat{r}_{\bar{n}-i}(x)},$$

for $i = 0, 1, \dots, (\bar{n}-n-1)$, where $r_n(x)$ and $c_n(x)$ represent respectively the first row and the first column of T^{-1} .

3.3 Third step of Kumar's algorithm

Obtaining the solution x of the Toeplitz system of equations from the first row of the inverse Toeplitz matrix.

- (iv) Define the following $(2n+1)$ -dimensional vectors:

$$\begin{aligned} h &= [c_n, \dots, c_1, r_0, r_1, \dots, r_n], \\ \bar{r} &= [0, \dots, 0, \bar{r}_1, \bar{r}_2, \dots, \bar{r}_n], \quad \bar{r}_i = r_0^{-1} r_i, \quad i = 1, \dots, n, \\ \bar{c} &= [0, \dots, 0, \bar{c}_n, \bar{c}_{n-1}, \dots, \bar{c}_1], \quad \bar{c}_i = c_0^{-1} c_i, \quad i = 1, \dots, n, \\ \bar{y} &= [0, \dots, 0, y_0, y_1, \dots, y_n]. \end{aligned}$$

(v) With \otimes denoting the cyclic convolution, compute the following:

$$\begin{aligned}\hat{u} &= \bar{y} \otimes h, \\ \hat{v} &= \bar{y} \otimes \bar{r}, \\ \hat{w} &= \bar{y} \otimes \bar{c}.\end{aligned}$$

(vi) Define vectors:

$$\begin{aligned}\tilde{u} &= [\hat{u}_0, \dots, \hat{u}_n], \\ \tilde{v} &= [0, \dots, 0, \hat{v}_0, \dots, \hat{v}_{n-1}], \\ \tilde{w} &= [0, \dots, 0, \hat{w}_0, \dots, \hat{w}_{n-1}],\end{aligned}$$

\tilde{v} and \tilde{w} are $(2n + 1)$ -dimensional vectors obtained by padding \hat{v} and \hat{w} with zeros.

(vii) Evaluate the circular correlations $\bar{\zeta} = \bar{c} \otimes \tilde{v}$, $\bar{\eta} = \bar{r} \otimes \tilde{w}$ and define:

$$\begin{aligned}\hat{\zeta} &= [\bar{\zeta}_n, \bar{\zeta}_{n-1}, \dots, \bar{\zeta}_0], \\ \hat{\eta} &= [\bar{\eta}_n, \bar{\eta}_{n-1}, \dots, \bar{\eta}_0].\end{aligned}$$

(viii) Finally, compute the solution x to the system of equations as follows:

$$x = \tilde{u} + (\hat{\zeta} - \hat{\eta})r_0.$$

Chapter 4

Adapting Kumar's algorithm

In the previous chapter, we have seen that the inverse of a circulant matrix C can be computed by means of DFT transforms implemented using fast FFT algorithms. Adapting such procedure to work over finite fields is not straightforward, since NTT transforms include some limitations that did not appear on DFT.

Let T be a Toeplitz matrix of order $n + 1$, C the circulant matrix of order $2n + 1$ generated from T and \mathbb{F}_p the finite field over which are defined T and C .

The computation of the inverse of C , can be performed in the following steps:

- (i) Take the first row and the first column of the Toeplitz matrix:

$$T = \begin{pmatrix} t_0 & t_{-1} & \dots & t_{-n} \\ t_1 & t_0 & \dots & t_{-n+1} \\ t_2 & t_1 & \dots & t_{-n+2} \\ \vdots & \vdots & \ddots & \vdots \\ t_n & t_{n-1} & \dots & t_0 \end{pmatrix}$$

and **generate the first row of C** :

$$C_{(1)} = (t_0 \ t_{-1} \ \dots \ t_{-n} \ t_n \ t_{n-1} \ \dots \ t_1).$$

So as to perform a NTT, one element of order the input sequence length is required. It may happen that such element does not exist in \mathbb{F}_p . So as to overcome this drawback, different options are available:

- (a) Embed zeros between the first row and the first column of T until the resulting vector length permits to find an element in \mathbb{F}_p whose

order is the new length of $C_{(1)}$.

$$C_{(1)} = (t_0 \ \dots \ t_{-n} \ t_n \ \underbrace{0 \ \dots \ 0}_{\text{extra zeros}} \ t_{n-1} \ \dots \ t_1).$$

- (b) Extend field \mathbb{F}_p until the resulting extended field contains elements with the required order.
- (c) Employ a combination of (a) and (b)

All the previous decisions lead to a correct solution.

If no extension is considered, the following steps are exactly the same as in Kumar's paper, but changing the DFT by NTT.

- (ii) **Compute the NTT of $C_{(1)}$, $C'_{(1)}$.**

NTT is computed with the algorithm shown in procedure 2.0.1 using the four algorithms detailed in sections 2.1-2.4.

- (iii) **Compute the sequence $C''_{(1)} = 1/C'_{(1)}$.**

This is performed by computing the multiplicative inverse of each element of sequence $C'_{(1)}$.

- (iv) **Compute $C_{(1)}^{-1}$, the INTT of the sequence $C''_{(1)}$,** which is done using again the algorithm shown in procedure 2.0.1, but employing INTT instead of NTT. The output is the first row of the inverse matrix C^{-1} .

Example 6 Solving the following Toeplitz system of equations defined over \mathbb{F}_{11} using the adaptation of Kumar's algorithm over finite fields.

$$\underbrace{\begin{pmatrix} 1 & 2 & 3 & 5 \\ 4 & 1 & 2 & 3 \\ 6 & 4 & 1 & 2 \\ 9 & 6 & 4 & 1 \end{pmatrix}}_T \begin{pmatrix} x \\ y \\ z \\ t \end{pmatrix} = \begin{pmatrix} 3 \\ 9 \\ 10 \\ 8 \end{pmatrix}$$

- (i) Take the first row and the first column of T and generate the first row of C . We need to add some zeros until the new vector length allows having an element of this latter order in \mathbb{F}_{11} :

$$C_{(1)} = \boxed{1 \mid 2 \mid 3 \mid 5 \mid 0 \mid 0 \mid 0 \mid 9 \mid 6 \mid 4}$$

(ii) Compute the NTT of $C_{(1)}$:

$$C'_{(1)} = \boxed{8 \mid 8 \mid 4 \mid 7 \mid 6 \mid 1 \mid 10 \mid 4 \mid 10 \mid 7}$$

(iii) Compute the modular inverses of the previous sequence:

$$C''_{(1)} = \boxed{7 \mid 7 \mid 3 \mid 8 \mid 2 \mid 1 \mid 10 \mid 3 \mid 10 \mid 8}$$

(iv) Compute the INTT of $C''_{(1)}$:

$$C^{-1}_{(1)} = \boxed{7 \mid 3 \mid 10 \mid 4 \mid 8 \mid 6 \mid 2 \mid 10 \mid 10 \mid 2}$$

(v) Generate C^{-1} from its first row:

$$C^{-1} = \begin{pmatrix} 7 & 3 & 10 & 4 & 8 & 6 & 2 & 10 & 10 & 2 \\ 2 & 7 & 3 & 10 & 4 & 8 & 6 & 2 & 10 & 10 \\ 10 & 2 & 7 & 3 & 10 & 4 & 8 & 6 & 2 & 10 \\ 10 & 10 & 2 & 7 & 3 & 10 & 4 & 8 & 6 & 2 \\ 2 & 10 & 10 & 2 & 7 & 3 & 10 & 4 & 8 & 6 \\ 6 & 2 & 10 & 10 & 2 & 7 & 3 & 10 & 4 & 8 \\ 8 & 6 & 2 & 10 & 10 & 2 & 7 & 3 & 10 & 4 \\ 4 & 8 & 6 & 2 & 10 & 10 & 2 & 7 & 3 & 10 \\ 10 & 4 & 8 & 6 & 2 & 10 & 10 & 2 & 7 & 3 \\ 3 & 10 & 4 & 8 & 6 & 2 & 10 & 10 & 2 & 7 \end{pmatrix}$$

(vi) Define polynomials $r_{\bar{n}}(x)$, $c_{\bar{n}}(x)$, $\hat{r}_{\bar{n}}(x)$ and $\hat{c}_{\bar{n}}(x)$:

$$r_{\bar{n}}(x) = 2x^9 + 10x^8 + 10x^7 + 2x^6 + 6x^5 + 8x^4 + 4x^3 + 10x^2 + 3x + 7$$

$$c_{\bar{n}}(x) = 3x^9 + 10x^8 + 4x^7 + 8x^6 + 6x^5 + 2x^4 + 10x^3 + 10x^2 + 2x + 7$$

$$\hat{r}_{\bar{n}}(x) = 7x^9 + 3x^8 + 10x^7 + 4x^6 + 8x^5 + 6x^4 + 2x^3 + 10x^2 + 10x + 2$$

$$\hat{c}_{\bar{n}}(x) = 7x^9 + 2x^8 + 10x^7 + 10x^6 + 2x^5 + 6x^4 + 8x^3 + 4x^2 + 10x + 3$$

(vii) Apply the iterative method until get polynomials of the same degree as the initial matrix T :

$$r_{\bar{n}-1}(x) = 4x^7 + 7x^6 + 7x^5 + 8x^3 + x^2 + 8x + 3$$

$$c_{\bar{n}-1}(x) = 4x^8 + 6x^7 + x^5 + x^4 + 6x^3 + x^2 + 4x + 3$$

$$\hat{r}_{\bar{n}-1}(x) = 3x^8 + 8x^7 + x^6 + 8x^5 + 7x^3 + 7x^2 + 4x$$

$$\hat{c}_{\bar{n}-1}(x) = 3x^8 + 4x^7 + x^6 + 6x^5 + x^4 + x^3 + 6x + 4$$

$$r_{\bar{n}-2}(x) = 4x^7 + 7x^6 + 7x^5 + 8x^3 + x^2 + 8x + 3$$

$$c_{\bar{n}-2}(x) = 10x^7 + 6x^6 + 5x^5 + x^4 + 4x^3 + 10x^2 + 6x + 3$$

$$\hat{r}_{\bar{n}-2}(x) = 3x^7 + 8x^6 + x^5 + 8x^4 + 7x^2 + 7x + 4$$

$$\hat{c}_{\bar{n}-2}(x) = 3x^7 + 6x^6 + 10x^5 + 4x^4 + x^3 + 5x^2 + 6x + 10$$

$$r_{\bar{n}-3}(x) = 10x^6 + x^5 + 2x^4 + 3x^3 + 9x^2 + 8$$

$$c_{\bar{n}-3}(x) = 5x^6 + 9x^5 + 4x^3 + 5x^2 + x + 8$$

$$\hat{r}_{\bar{n}-3}(x) = 8x^6 + 9x^4 + 3x^3 + 2x^2 + x + 10$$

$$\hat{c}_{\bar{n}-3}(x) = 8x^6 + x^5 + 5x^4 + 4x^3 + 9x + 5$$

$$r_{\bar{n}-4}(x) = 8x^5 + 4x^4 + 9x^3 + 9x^2 + 8x + 10$$

$$c_{\bar{n}-4}(x) = 9x^5 + 4x^4 + 9x^3 + x^2 + 10x + 10$$

$$\hat{r}_{\bar{n}-4}(x) = 10x^5 + 8x^4 + 9x^3 + 9x^2 + 4x + 8$$

$$\hat{c}_{\bar{n}-4}(x) = 10x^5 + 10x^4 + x^3 + 9x^2 + 4x + 9$$

$$r_{\bar{n}-5}(x) = 7x^4 + 6x^3 + 4x^2 + 7x + 5$$

$$c_{\bar{n}-5}(x) = 10x^4 + 2x^3 + 5x^2 + 2x + 5$$

$$\hat{r}_{\bar{n}-5}(x) = 5x^4 + 7x^3 + 4x^2 + 6x + 7$$

$$\hat{c}_{\bar{n}-5}(x) = 5x^4 + 2x^3 + 5x^2 + 2x + 10$$

$$r_{\bar{n}-6}(x) = x^3 + 8x^2 + 2x + 2 = r_n(x)$$

$$c_{\bar{n}-6}(x) = 10x^3 + 8x^2 + x + 2 = c_n(x)$$

The first row and column of T^{-1} are $[2, 2, 8, 1]$ and $[2, 1, 8, 10]$, respectively.

(viii) Define the $(2n + 1)$ -dimensional vectors:

$$h = [10, 8, 1, 2, 2, 8, 1],$$

$$\hat{r} = [0, 0, 0, 0, 1, 4, 6],$$

$$\hat{c} = [0, 0, 0, 0, 5, 4, 6],$$

$$\hat{y} = [0, 0, 0, 3, 9, 10, 8].$$

(ix) Perform the cyclic convolutions:

$$\hat{u} = [2, 6, 3, 7, 5, 10, 3],$$

$$\hat{v} = [9, 9, 8, 0, 7, 0, 0],$$

$$\hat{w} = [1, 5, 7, 0, 7, 0, 1].$$

(x) Define vectors:

$$\tilde{u} = [2, 6, 3, 7],$$

$$\tilde{v} = [0, 0, 0, 0, 9, 9, 8],$$

$$\tilde{w} = [0, 0, 0, 0, 1, 5, 7].$$

(xi) Evaluate the circular correlations and define $\hat{\zeta}$ and $\hat{\eta}$:

$$\begin{aligned}\bar{\zeta} &= [8, 2, 10, 0, 0, 7, 0], \\ \bar{\eta} &= [8, 1, 6, 0, 0, 7, 0], \\ \hat{\zeta} &= [0, 10, 2, 8], \\ \hat{\eta} &= [0, 6, 1, 8].\end{aligned}$$

(xii) Finally, compute the solution x to the system of equations:

$$x = \tilde{u} + (\hat{\zeta} - \hat{\eta})r_0 = [2, 3, 5, 7].$$

4.1 Working over field extensions

When performing NTT's over an extension of the base field \mathbb{F}_p , we have to take into account that Shoup's method to compute a circular convolution (section 2.4) was designed to work over prime fields and it can not be directly applied over extended fields. So as to overcome this limitation, we have designed an alternative procedure.

The convolution of two sequences a and b is performed by splitting each element of the extended field into d elements of the base field, where d is the degree of the extension, obtaining for each sequence of elements of \mathbb{F}_{p^d} d sequences of elements of \mathbb{F}_p . Next, we perform one convolution for each pair of sequences $a_i, b_i, i \in [0, d - 1]$. See step (ii) in the following example.

Example 7 Computation of the NTT of the length 3 sequence

$$x = \boxed{3} \boxed{2} \boxed{1}$$

of elements of \mathbb{F}_5 with $\omega = 2$ and $\omega^\zeta = 2\alpha + 1$, α is a generator of $\mathbb{F}_{5^2}^*$, using Rader's algorithm:

(i) Compute two new sequences a, b of length $N - 1 = 2$

$$\left. \begin{aligned}\omega^0 = 1 &\rightarrow a_0 = x_1 = 2 \\ \omega^1 = 2 &\rightarrow a_1 = x_2 = 1\end{aligned} \right\} \longrightarrow a = \boxed{2} \boxed{1}$$

$$\left. \begin{aligned}(\omega^\zeta)^{\omega^0} = 2\alpha + 1 &\rightarrow b_0 = 2\alpha + 1 \\ (\omega^\zeta)^{\omega^{-1}} = 3\alpha + 3 &\rightarrow b_1 = 3\alpha + 3\end{aligned} \right\} \longrightarrow b = \boxed{2\alpha + 1} \boxed{3\alpha + 3}$$

- (ii) To perform the circular convolution $a \circledast b$, we split the sequence b into two new sequences, one with the elements of degree 0 and the other with degree 1. Each one is convoluted with sequence a and afterwards, both solutions are added.

$$\begin{array}{r}
 b = \boxed{2\alpha + 1} \mid \boxed{3\alpha + 3} \mid \boxed{0} \mid \boxed{0} \\
 \nearrow b_0 = \boxed{1} \mid \boxed{3} \mid \boxed{0} \mid \boxed{0} \\
 \searrow b_1 = \boxed{2} \mid \boxed{3} \mid \boxed{0} \mid \boxed{0} \\
 \\
 \left. \begin{array}{l} a = \boxed{2} \mid \boxed{1} \mid \boxed{0} \mid \boxed{0} \\ b_0 = \boxed{1} \mid \boxed{3} \mid \boxed{0} \mid \boxed{0} \end{array} \right\} \xrightarrow{\circledast} \boxed{2} \mid \boxed{2} \mid \boxed{3} \mid \boxed{0} \searrow \\
 \left. \begin{array}{l} a = \boxed{2} \mid \boxed{1} \mid \boxed{0} \mid \boxed{0} \\ b_1 = \boxed{2} \mid \boxed{3} \mid \boxed{0} \mid \boxed{0} \end{array} \right\} \xrightarrow{\circledast} \boxed{4} \mid \boxed{3} \mid \boxed{3} \mid \boxed{0} \nearrow \\
 \boxed{4\alpha + 2} \mid \boxed{3\alpha + 2} \mid \boxed{3\alpha + 3} \mid \boxed{0}
 \end{array}$$

If both a and b belonged to an extended field, we would have divided both sequences into sequences of elements of the basis field.

- (iii) Transform the linear convolution into a circular one.

$$\boxed{4\alpha + 2} \mid \boxed{3\alpha + 2} \mid \boxed{3\alpha + 3} \mid \boxed{0} \xrightarrow{\text{linear to circular conv}} cc = \boxed{2\alpha} \mid \boxed{3\alpha + 2}$$

- (iv) Join these results as indicated in formula 2.7:

$$\begin{aligned}
 X_0 &= \sum_{k=0}^{N-1} x_k = 3 + 2 + 1 = 6 = 1 \pmod{5} \\
 X_1 &= x_0 + cc_0 = 3 + 2\alpha = 2\alpha + 3 \\
 X_2 &= x_0 + cc_3 = 3 + 3\alpha + 2 = 3\alpha
 \end{aligned}$$

$$X = \boxed{1} \mid \boxed{2\alpha + 3} \mid \boxed{3\alpha}$$

Chapter 5

Outcomes and Future work

Some examples with different matrices and field sizes have been carried out and the results are shown in this chapter. Possible improvements to this project are detailed in section 5.2.

5.1 Outcomes

Table 5.1 shows NTT computation time results of input sequences of size 10, 20, 50, 100, 150, 300 and 625 defined over different fields \mathbb{F}_p of size 15, 57, 110, 164, 296, 1093, 4849 and 7153 bits. It can be easily seen in table 5.1 and in figure 5.1 that computation time grows with the size of the matrix and/or the group.

Bit length	Size of circulant matrix						
	10×10	20×20	50×50	100×100	150×150	300×300	625×625
15	0.066	0.086	0.550	0.918	2.294	4.499	14.724
57	0.182	0.226	1.429	2.254	5.627	9.807	30.253
110	0.297	0.375	2.383	3.644	9.236	15.890	47.682
164	0.417	0.529	3.419	5.181	12.987	21.788	67.153
296	0.691	0.873	5.472	8.480	21.439	35.349	107.666
1093	2.165	2.724	17.285	26.716	66.850	111.931	334.479
4849	12.272	15.761	82.272	130.547	306.360	514.092	1598.823
7153	22.779	28.334	132.97	209.967	486.719	819.150	2502.071

Table 5.1: Computation time of the inverse circulant matrix in \mathbb{F}_p

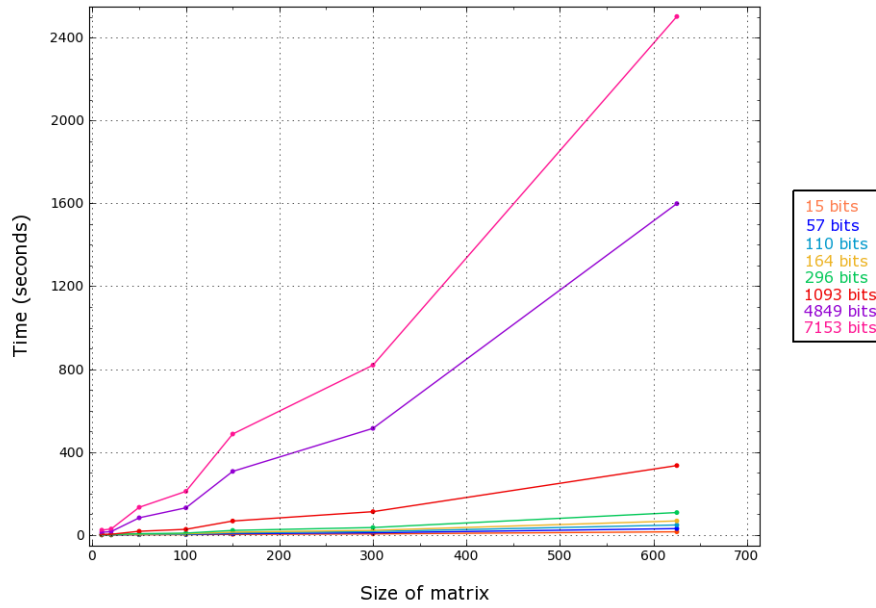


Figure 5.1: Computation time of the inverse circulant matrix in \mathbb{F}_p

Figure 5.2 shows NTT computation time results of input sequences up to size 230 defined over different fields \mathbb{F}_{p^2} of size 10, 15, 20 and 25 bits. From p of 20 to 25 bits there is a little increase of time and for higher numbers, the time is extremely high.

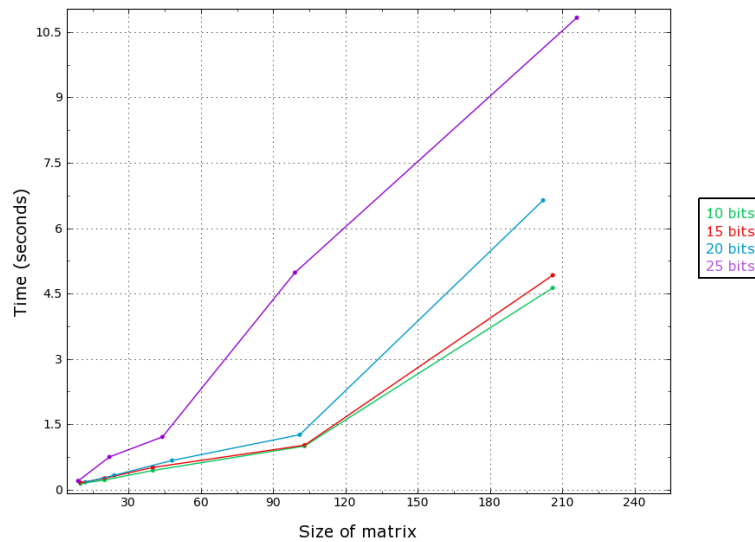


Figure 5.2: Computation time of the inverse circulant matrix in \mathbb{F}_{p^2}

5.2 Further improvements

As future work, some improvements may be implemented and afterwards incorporated into the current project are:

- (i) Compute “NTT-primes” just once.

Shoup’s circular convolution is called inside a loop from Rader’s algorithm. Although the “NTT-primes” are always the same (because they only depend on the field and the length of the input sequence), they are computed as many times as the Shoup’s circular convolution is called.

- (ii) Improve the algorithm that decides whether to add zeros at the end of the sequence or to consider an extension of the field when the necessary condition to perform the NTT is not fulfilled. This is the length of the input sequence must divide the field size minus one.

Study if there exists a condition that simplifies this decision.

- (iii) Implement the second and third steps of Kumar’s algorithm and study if some further contribution is possible.

Appendix A

Polynomial multiplication and convolutions

Polynomial multiplication can be re-expressed as the linear convolution of two signals whose samples correspond to the input polynomials coefficients. In this appendix we show the relation between polynomial multiplication, linear convolutions and circular convolutions.

A.1 Polynomial multiplication and linear convolution

Let $p(x)$ and $q(x)$ be two polynomials of degree m and n respectively with coefficients p_i and q_i in field \mathbb{K} ,

$$p(x) = p_0 + p_1x + p_2x^2 + \dots + p_mx^m. \quad q(x) = q_0 + q_1x + q_2x^2 + \dots + q_nx^n.$$

They can be expressed as summations:

$$p(x) = \sum_{i=0}^m p_i \cdot x^i, \quad q(x) = \sum_{i=0}^n q_i \cdot x^i.$$

Therefore, their **product** is the polynomial of degree $m + n$ given by

$$p(x)q(x) = p_0q_0 + (p_0q_1 + p_1q_0)x + (p_0q_2 + p_1q_1 + p_2q_0)x^2 + \dots \\ + (p_0q_{m+n} + p_1q_{m+n-1} + \dots + p_{m+n}q_0)x^{m+n},$$

which can be expressed as:

$$p(x)q(x) = \sum_{k=0}^{m+n} \left(\sum_{i=0}^k p_i \cdot q_{k-i} \right) \cdot x^k = r(x).$$

Now, we have that the k -th coefficient is:

$$r_k = \sum_{i=0}^k p_i \cdot q_{k-i}. \quad (\text{A.1})$$

Given two sequences p (of m elements) and q (of n elements) of a field \mathbb{K} , their **linear convolution** is given by:

$$s = p * q, \quad s[k] = \sum_{i=0}^k p[i] \cdot q[k-i], \quad (\text{A.2})$$

where $s[k]$ is a $(m+n-1)$ -length sequence.

We can express a polynomial of degree t as a sequence, taking each k -th coefficient as the k -th element in the sequence, $0 \leq k \leq t$.

If we express polynomials $p(x)$ and $q(x)$ in sequence form, it is easy to see that expressions A.1 and A.2 represent the same, so we can obtain the multiplication of two polynomials expressing them as sequences and computing their linear convolution. Let's see it with a numerical example:

Example 8 Given $p(x) = 3x^2 + 2x + 4$ and $q(x) = x^3 + 5x^2 + 7$, their product is given by:

$$p(x)q(x) = (3 \cdot 1)x^5 + (3 \cdot 5 + 2 \cdot 1)x^4 + (2 \cdot 5 + 4 \cdot 1)x^3 + (3 \cdot 7 + 4 \cdot 5)x^2 + (2 \cdot 7)x + 4 \cdot 7$$

Now, we can express each polynomial as a sequence:

$$p = \boxed{4} \boxed{2} \boxed{3} \quad q = \boxed{7} \boxed{0} \boxed{5} \boxed{1}$$

and perform the linear convolution. Initially, only one element of each sequence are in contact. In each step, the second sequence is moved one position to the right and the matching boxes are multiplied.

$$\begin{array}{cccc}
 & & \boxed{4} & \boxed{2} & \boxed{3} \\
 \boxed{1} & \boxed{5} & \boxed{0} & \boxed{7} & & \\
 \hline
 & & \boxed{4} & \boxed{2} & \boxed{3} \\
 \boxed{1} & \boxed{5} & \boxed{0} & \boxed{7} & & \\
 \hline
 & & \boxed{4} & \boxed{2} & \boxed{3} \\
 \boxed{1} & \boxed{5} & \boxed{0} & \boxed{7} & & \\
 \hline
 & \boxed{4} & \boxed{2} & \boxed{3} & & \\
 \boxed{1} & \boxed{5} & \boxed{0} & \boxed{7} & & \\
 \hline
 & \boxed{4} & \boxed{2} & \boxed{3} & & \\
 \boxed{1} & \boxed{5} & \boxed{0} & \boxed{7} & & \\
 \hline
 & & \boxed{4} & \boxed{2} & \boxed{3} & \\
 \boxed{1} & \boxed{5} & \boxed{0} & \boxed{7} & & \\
 \hline
 & & & \boxed{4} & \boxed{2} & \boxed{3} \\
 \boxed{1} & \boxed{5} & \boxed{0} & \boxed{7} & & \\
 \hline
 & & & & \boxed{4} & \boxed{2} & \boxed{3} \\
 \boxed{1} & \boxed{5} & \boxed{0} & \boxed{7} & & & \\
 \hline
 & & & & & \boxed{4} & \boxed{2} & \boxed{3} \\
 \boxed{1} & \boxed{5} & \boxed{0} & \boxed{7} & & & & \\
 \hline
 \end{array}$$

A.2 Linear and circular convolution

Given two sequences p (of m elements) and q (of n elements), their linear convolution is a $(m + n - 1)$ -length sequence given by [A.3](#).

If we take $N = \max(m, n)$, the **circular convolution** is a N -length sequence given by:

$$s = p \otimes q, \quad s[k] = \sum_{i=0}^k p[i] \cdot q[k - i], \quad (\text{A.3})$$

Notice that the length of a circular convolution of two sequences is equal to the length of the longest one while the length of the linear one is the summation of both sequences lengths.

Let's see it with an example:

Example 9 Given two sequences

$$p = \boxed{4 \mid 2 \mid 3 \mid 0} \quad q = \boxed{7 \mid 0 \mid 5 \mid 1}$$

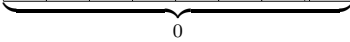
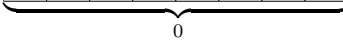
we are going to compute their circular convolution. As before, initially only one element of each sequence are in contact. In contrast, the elements that were left alone in linear convolution are now wrapped around, resulting in a shorter and different result.

<table style="border-collapse: collapse; margin: auto;"> <tr><td style="border: 1px solid black; padding: 2px 5px;">4</td><td style="border: 1px solid black; padding: 2px 5px;">2</td><td style="border: 1px solid black; padding: 2px 5px;">3</td><td style="border: 1px solid black; padding: 2px 5px;">0</td></tr> <tr><td style="border: 1px solid black; padding: 2px 5px;">7</td><td style="border: 1px solid black; padding: 2px 5px;">1</td><td style="border: 1px solid black; padding: 2px 5px;">5</td><td style="border: 1px solid black; padding: 2px 5px;">0</td></tr> </table>	4	2	3	0	7	1	5	0	<table style="border-collapse: collapse; margin: auto;"> <tr><td style="border: 1px solid black; padding: 2px 5px;">4</td><td style="border: 1px solid black; padding: 2px 5px;">2</td><td style="border: 1px solid black; padding: 2px 5px;">3</td><td style="border: 1px solid black; padding: 2px 5px;">0</td></tr> <tr><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">7</td><td style="border: 1px solid black; padding: 2px 5px;">1</td><td style="border: 1px solid black; padding: 2px 5px;">5</td></tr> </table>	4	2	3	0	0	7	1	5	<table style="border-collapse: collapse; margin: auto;"> <tr><td style="border: 1px solid black; padding: 2px 5px;">4</td><td style="border: 1px solid black; padding: 2px 5px;">2</td><td style="border: 1px solid black; padding: 2px 5px;">3</td><td style="border: 1px solid black; padding: 2px 5px;">0</td></tr> <tr><td style="border: 1px solid black; padding: 2px 5px;">5</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">7</td><td style="border: 1px solid black; padding: 2px 5px;">1</td></tr> </table>	4	2	3	0	5	0	7	1	<table style="border-collapse: collapse; margin: auto;"> <tr><td style="border: 1px solid black; padding: 2px 5px;">4</td><td style="border: 1px solid black; padding: 2px 5px;">2</td><td style="border: 1px solid black; padding: 2px 5px;">3</td><td style="border: 1px solid black; padding: 2px 5px;">0</td></tr> <tr><td style="border: 1px solid black; padding: 2px 5px;">1</td><td style="border: 1px solid black; padding: 2px 5px;">5</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">7</td></tr> </table>	4	2	3	0	1	5	0	7
4	2	3	0																																
7	1	5	0																																
4	2	3	0																																
0	7	1	5																																
4	2	3	0																																
5	0	7	1																																
4	2	3	0																																
1	5	0	7																																
$4 \cdot 7 + 2 \cdot 1 + 3 \cdot 5 + 0 \cdot 0 = 45$	$4 \cdot 0 + 2 \cdot 7 + 3 \cdot 1 + 0 \cdot 5 = 17$	$4 \cdot 5 + 2 \cdot 0 + 3 \cdot 7 + 0 \cdot 1 = 41$	$4 \cdot 1 + 2 \cdot 5 + 3 \cdot 0 + 0 \cdot 7 = 14$																																

In this way, the result is not equivalent to the linear convolution, because the appropriate elements do not match. However, if both lengths are doubled adding zeros, then the result in both convolutions is exactly the same.

$$p = \boxed{4 \mid 2 \mid 3 \mid 0 \mid 0 \mid 0 \mid 0 \mid 0} \quad q = \boxed{7 \mid 0 \mid 5 \mid 1 \mid 0 \mid 0 \mid 0 \mid 0}$$

<table style="border-collapse: collapse; margin: auto;"> <tr><td style="border: 1px solid black; padding: 2px 5px;">4</td><td style="border: 1px solid black; padding: 2px 5px;">2</td><td style="border: 1px solid black; padding: 2px 5px;">3</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td></tr> <tr><td style="border: 1px solid black; padding: 2px 5px;">7</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">1</td><td style="border: 1px solid black; padding: 2px 5px;">5</td><td style="border: 1px solid black; padding: 2px 5px;">0</td></tr> </table>	4	2	3	0	0	0	0	0	7	0	0	0	0	1	5	0	<table style="border-collapse: collapse; margin: auto;"> <tr><td style="border: 1px solid black; padding: 2px 5px;">4</td><td style="border: 1px solid black; padding: 2px 5px;">2</td><td style="border: 1px solid black; padding: 2px 5px;">3</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td></tr> <tr><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">7</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">1</td><td style="border: 1px solid black; padding: 2px 5px;">5</td></tr> </table>	4	2	3	0	0	0	0	0	0	7	0	0	0	0	1	5	<table style="border-collapse: collapse; margin: auto;"> <tr><td style="border: 1px solid black; padding: 2px 5px;">4</td><td style="border: 1px solid black; padding: 2px 5px;">2</td><td style="border: 1px solid black; padding: 2px 5px;">3</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td></tr> <tr><td style="border: 1px solid black; padding: 2px 5px;">5</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">7</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">1</td></tr> </table>	4	2	3	0	0	0	0	0	5	0	7	0	0	0	0	1
4	2	3	0	0	0	0	0																																											
7	0	0	0	0	1	5	0																																											
4	2	3	0	0	0	0	0																																											
0	7	0	0	0	0	1	5																																											
4	2	3	0	0	0	0	0																																											
5	0	7	0	0	0	0	1																																											
$4 \cdot 7$	$2 \cdot 7$	$4 \cdot 5 + 3 \cdot 7$																																																
<table style="border-collapse: collapse; margin: auto;"> <tr><td style="border: 1px solid black; padding: 2px 5px;">4</td><td style="border: 1px solid black; padding: 2px 5px;">2</td><td style="border: 1px solid black; padding: 2px 5px;">3</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td></tr> <tr><td style="border: 1px solid black; padding: 2px 5px;">1</td><td style="border: 1px solid black; padding: 2px 5px;">5</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">7</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td></tr> </table>	4	2	3	0	0	0	0	0	1	5	0	7	0	0	0	0	<table style="border-collapse: collapse; margin: auto;"> <tr><td style="border: 1px solid black; padding: 2px 5px;">4</td><td style="border: 1px solid black; padding: 2px 5px;">2</td><td style="border: 1px solid black; padding: 2px 5px;">3</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td></tr> <tr><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">1</td><td style="border: 1px solid black; padding: 2px 5px;">5</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">7</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td></tr> </table>	4	2	3	0	0	0	0	0	0	1	5	0	7	0	0	0	<table style="border-collapse: collapse; margin: auto;"> <tr><td style="border: 1px solid black; padding: 2px 5px;">4</td><td style="border: 1px solid black; padding: 2px 5px;">2</td><td style="border: 1px solid black; padding: 2px 5px;">3</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td></tr> <tr><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">1</td><td style="border: 1px solid black; padding: 2px 5px;">5</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">7</td><td style="border: 1px solid black; padding: 2px 5px;">0</td><td style="border: 1px solid black; padding: 2px 5px;">0</td></tr> </table>	4	2	3	0	0	0	0	0	0	0	1	5	0	7	0	0
4	2	3	0	0	0	0	0																																											
1	5	0	7	0	0	0	0																																											
4	2	3	0	0	0	0	0																																											
0	1	5	0	7	0	0	0																																											
4	2	3	0	0	0	0	0																																											
0	0	1	5	0	7	0	0																																											
$4 \cdot 1 + 2 \cdot 5$	$2 \cdot 1 + 3 \cdot 5$	$3 \cdot 1$																																																

4	2	3	0	0	0	0	0	0
0	0	0	1	5	0	7	0	0
								
4	2	3	0	0	0	0	0	0
0	0	0	0	1	5	0	7	0
								

To sum up,

- The convolution of two sequences is like the multiplication of two polynomials in coefficient form where the coefficients of the second polynomial are rotated and matching coefficients are added.
- If the coefficient representations of two m -degree polynomials are extended by padding the representation with m zeros in the higher-order terms, then the circular convolution is equivalent to the linear one.

Bibliography

- [BMSS06] Bostan, A., Morain, F., Salvy, B., Schost, É., *Fast algorithms for computing isogenies between elliptic curves*. Mathematics of Computation. 2006.
- [CLW67] Cooley, J.W., Lewis, P., Welch, P., *Historical notes on the fast Fourier transform*. Proceedings of the IEEE, vol. 55, pp. 1675–1677. 1967.
- [CT65] Cooley, J.W., Tukey, J.W., *An algorithm for the machine calculation of complex Fourier series*. Mathematics of Computation 19, pp. 297–301. 1965.
- [DV90] Duhamel, P., Vetterli, M., *Fast Fourier transforms: a tutorial review and a state of the art*. Signal processing 19, 259-299. 1990.
- [Dur60] Durbin, J., *The fitting of time series models*. Rev. Inst. Int. Stat., v. 28, pp. 233-243. 1960.
- [FS74] Farden, D.C., Scharf, L.L., *Statistical Design of Nonrecursive digital filters*. IEEE transactions on acustics, speech and signal processing, vol. 22. 1974.
- [Kim08] Kimitei, S. Algorithms for Toeplitz matrices with applications to image deblurring. Master Thesis in the College of Arts and Sciences, Georgia. 2008.
- [Kum85] Kumar, R., *A Fast algorithm for Solving a Toeplitz System of Equations*. IEEE. 1985.
- [Lev47] Levinson, N., *The Wiener RMS error criterion in filter design and prediction*. J. Math. Phys., v. 25, pp. 261-278. 1947.
- [Rad68] Rader, C.M., Discrete Fourier transforms when the number of data samples is prime. Proceedings of the IEEE, vol. 56, pp.1107-1108. 1968.

- [Sho96] Shoup, V., *A new polynomial factorization algorithm and its implementation*. Journal of Symbolic Computation. 1996.
- [Sage] Stein, W. and Joyner, D., *SAGE: System for Algebra and Geometry Experimentation*. <http://www.sagemath.org/>. 2005.
- [SA96] Sullivan, J.L., Adams, J.W., *An algorithm for solving the Toeplitz systems of equations in FIR digital filter design problems*. IEEE transactions on signal processing, vol. 44. 1996.
- [Tre64] Trench, W.F., *An algorithm for the inversion of finite Toeplitz matrices*. J. SIAM, vol. 12, pp. 515-522. 1964.
- [Zoh74] Zohar, S., *The solution of a Toeplitz set of linear equations*. J. Ass. Comput. Mach., vol. 21, pp. 272-276. 1974.