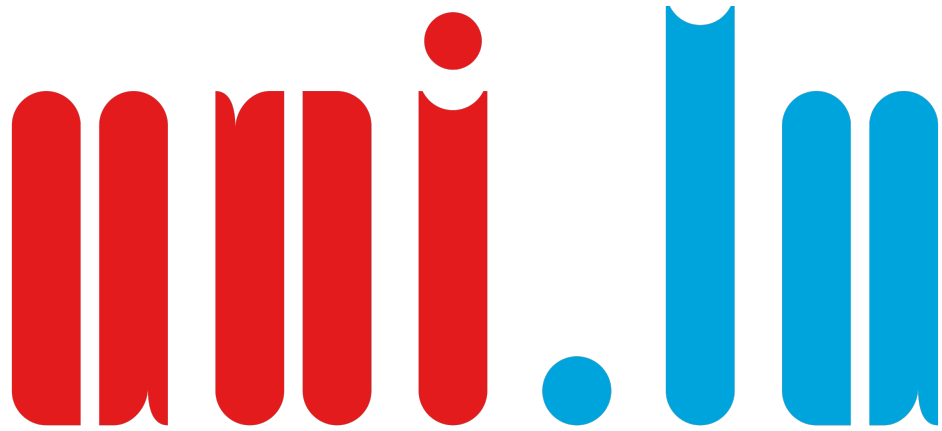
Discrete Fourier Transform Matrix

Husson Lola
Reichert Noa
Sachsen Tom

Semestre 3



UNIVERSITÉ DU LUXEMBOURG

Contents

Click on the section you wish to access

1 Introduction

Our project for the class of "Mathématiques expérimentales" was all about the Discrete Fourier Transform Matrix, also known as the DFT Matrix.

Before we could start working on it, we had to learn a few things about the matrix in question, how to compute it and what are its properties. Later, we will write programs to calculate this DFT matrix and a certain mysterious number d_n investigating their properties and seeing some interesting facts.
At the end we create beautiful images based on these DFT matrices.

2 Euler's totient function

We start by defining Euler's totient function, which will later help us to determine the dimension of the DFT matrix:

2.1 Definition of the function

The map

$$\begin{aligned}\theta : \mathbb{N}^* &\longrightarrow \mathbb{N} \\ n &\longmapsto \#(\mathbb{Z}/n\mathbb{Z}) = \#\{0 \leq k \leq n \mid \gcd(k,n) = 1\}\end{aligned}$$

is called Euler's totient function.

We computed a few examples to get familiar with the function:

$$\begin{aligned}\theta(5) &= \#\{0 \leq k \leq 5 \mid \gcd(k,5) = 1\} &&= \#\{1,2,3,4\} = 4 \\ \theta(7) &= \#\{0 \leq k \leq 7 \mid \gcd(k,7) = 1\} &&= \#\{1,2,3,4,5,6\} = 6 \\ \theta(11) &= \#\{0 \leq k \leq 11 \mid \gcd(k,11) = 1\} &&= \#\{1,2,3,4,5,6,7,8,9,10\} = 10 \\ \theta(6) &= \#\{0 \leq k \leq 6 \mid \gcd(k,6) = 1\} &&= \#\{1,5\} = 2\end{aligned}$$

Let n be a prime number. We observe that $\theta(n) = n - 1$

Now we introduce two more properties of the Euler's totient function to help us calculate the image of much larger numbers:

2.2 Proposition of the function

Let θ be the Euler's totient function.

(i) If $m, n \geq 1$ are coprime, then $\theta(m \cdot n) = \theta(m) \cdot \theta(n)$

(ii) If p is prime and $r \geq 1$, then $\theta(p^r) = p^r - p^{r-1}$

For example:

$$\begin{aligned}\theta(50) &= \theta(2 \cdot 5^2) = \theta(2) \cdot \theta(5^2) = 1 \cdot (5^2 - 5^1) = 25 - 5 = 20 \\ \theta(180) &= \theta(2^2 \cdot 5 \cdot 3^2) = \theta(2^2) \cdot \theta(3^2) \cdot \theta(5) = (2^2 - 2^1) \cdot 4 \cdot (3^2 - 3^1) = 2 \cdot 4 \cdot 6 = 48\end{aligned}$$

3 Discrete Fourier Transform Matrix

Before defining the DFT matrix, we have to define one more thing:

3.1 Definition of w_n

We define for $n \geq 1$:

$$w_n := \exp\left(\frac{2\pi i}{n}\right) = \cos\left(\frac{2\pi}{n}\right) + i\sin\left(\frac{2\pi}{n}\right)$$

We now come to the main part of this entire project, what actually is the DFT matrix and how do we compute it?

3.2 Definition of the DFT matrix

We define the DFT (Discrete Fourier Transform) matrix of order n to be:

$$D_n := ({w_n}^{ij})_{i,j}$$

for i & j running through $(\mathbb{Z}/n\mathbb{Z})^*$ and $\{0, ..., \theta(n) - 1\}$, respectively.

$K^* := \{x \in K \mid x \text{ invertible}\}$

Examples of DFT matrices:

If $n = 5$, put $w := w_5$. Then

$$D_5 = \begin{pmatrix} w^0 & w^1 & w^2 & w^3 \\ w^0 & w^2 & w^4 & w^6 \\ w^0 & w^3 & w^6 & w^9 \\ w^0 & w^4 & w^8 & w^{12} \end{pmatrix} = \begin{pmatrix} 1 & w & w^2 & w^3 \\ 1 & w^2 & w^4 & w \\ 1 & w^3 & w & w^4 \\ 1 & w^4 & w^3 & w^2 \end{pmatrix}$$

If $n = 12$, put $w := w_{12}$. Then

$$D_{12} = \begin{pmatrix} w^0 & w^1 & w^2 & w^3 \\ w^0 & w^5 & w^{10} & w^{15} \\ w^0 & w^7 & w^{14} & w^{21} \\ w^0 & w^{11} & w^{22} & w^{33} \end{pmatrix} = \begin{pmatrix} 1 & w & w^2 & w^3 \\ 1 & w^2 & w^{10} & w^3 \\ 1 & w^7 & w^2 & w^9 \\ 1 & w^{11} & w^{10} & w^9 \end{pmatrix}$$

Now that we know how to compute the DFT matrix, we define a few properties and functions that will help us find what we're looking for in this DFT matrix.

3.3 Proposition of the DFT matrix

For every $n \geq 1$, D_n is square and invertible.

Proof

As $\#(\mathbb{Z}/n\mathbb{Z}) = \#\{0, ..., \theta(n) - 1\}$, D_n is a square matrix.

To check the invertibility of the DFT matrix, we will use the Vandermonde Matrix. Let $(z_0, ..., z_{k-1}) \in \mathbb{C}$ and consider the Vandermonde Matrix:

$$V = \begin{pmatrix} 1 & z_0 & {z_0}^2 & \dots & {z_0}^{k-1} \\ 1 & z_1 & {z_1}^2 & \dots & {z_1}^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & z_{k-1} & {z_{k-1}}^2 & \dots & {z_{k-1}}^{k-1} \end{pmatrix}$$

Then $\det(V) = \prod_{i \leq j} (z_j - z_i) \neq 0$ for $z_i \neq z_j$

Let us now consider the DFT matrix with $n = 5$:

$$D_5 = \begin{pmatrix} 1 & w & w^2 & w^3 \\ 1 & w^2 & w^4 & w^6 \\ 1 & w^3 & w^6 & w^9 \\ 1 & w^4 & w^8 & w^{12} \end{pmatrix} = \begin{pmatrix} 1 & w & (w)^2 & (w)^3 \\ 1 & w^2 & (w^2)^2 & (w^2)^3 \\ 1 & w^3 & (w^3)^2 & (w^3)^3 \\ 1 & w^4 & (w^4)^2 & (w^4)^3 \end{pmatrix}$$

We see that the D_5 has exactly the form of the Vandermonde Matrix for $z_0 = w$, $z_1 = w^2$, $z_2 = w^3$ and $z_3 = w^4$ and

$$\det(D_5) = \prod_{1 \leq i < j \leq 4} (w^j - w^i) \neq 0 \text{ as } w^i \neq w^j$$

We can compute every DFT matrix with $n \geq 1$ in a form of a Vandermonde Matrix.

So $\det(D_n) = \prod_{i<j} (w^j - w^i) \neq 0$ as $w^i \neq w^j$

As $\det(D_n) \neq 0$, we proved that D_n is invertible. □

4 Infinity norm of the inverse DFT matrix

For now, we still need to define one function in relation to matrices that will helps us determine what we're looking for.

4.1 Definition of the infinity form

Let $A = (a_{ij}) \in M_{m,n}(\mathbb{C})$. The infinity norm of A is defined as the following:

$$||A||_\infty := \max_{i=1,\dots,n} \left(\sum_{j=1}^{n} |a_{ij}| \right)$$

Example:

$$\left\| \begin{pmatrix} 1 & 2 \\ -1 & 4 \end{pmatrix} \right\|_\infty = 5 \quad \text{and} \quad \|D_{12}\|_\infty = 4$$

We now come to the main subject of our project. We will now define a number, that is being computed using infinity norm and the inverse DFT matrix. Number in question hasn't been named yet, so in this document we will just keep calling it "d_n" for simplification. This number is what we will analyze and see if we can spot patterns for certain DFT matrices.

4.2 Definition of d_n

We define for $n \geq 1$:

$$d_n := ||{D_n}^{-1}||_\infty$$

In non-mathematical terms, d_n is the infinity norm of the inverse of the DFT matrix for a certain $n \geq 1$.

5 Programing and Exploring

5.1 Programing D_n & d_n

After we had defined all this and got a little bit familiar with the DFT matrix and our d_n number, we got given the following tasks:

1. Write a program which outputs D_n & d_n for given $n \geq 1$
2. Create a plot depicting the points (n, d_n) for $n \leq 200$

Here are our results of the tasks:

1. For the begin we wrote the following program which computed D_n & d_n for us:

```
def totient(n):
    uniqueprimefact = [i for i in divisors(n) if is_prime(i)]
    for i in uniqueprimefact:
        n *= (1 - 1 / i)
    return n

def I_n(n):
    I1 = [x for x in Set(srange(n)) if gcd(x,n) == 1]
    return I1

def DFT(n):
    A = zero_matrix(CC,euler_phi(n))
    w = cos((2*pi)/n)+I*sin((2*pi)/n)
    for i in srange(euler_phi(n)):
        for j in srange(euler_phi(n)):
            I1 = I_n(n)
            k = I1[i]
            c = k*j
            A[i,j] = w**c
    return A

def dnumber(n):
    A = DFT(n)
    d_n = A.inverse().norm(infinity)
    return d_n
```

Figure 1: Program for D_n & d_n

We wrote a program to get the value of the Euler's totient function but we took the existing euler_phi function so that the computer can work better.

Example

```
DFT(3)
show(DFT(3))
```

$$\begin{pmatrix} 1.00000000000000 & -0.500000000000000 + 0.866025403784439i \\ 1.00000000000000 & -0.500000000000000 - 0.866025403784439i \end{pmatrix}$$

```
dnumber(3)
```

```
1.1547005383792517
```

Figure 2: Matrix D_3 & Value of d_3

For more efficiency in our programs we imported numpy which does not take so many decimal places in the calculations:

```
import numpy as np
from numpy import linalg as LA

def I_n(n):
    I1 = [x for x in Set(srange(n)) if gcd(x,n) == 1]
    return I1

def DFT_numpy(n):
    s = (euler_phi(n),euler_phi(n))
    x = np.zeros(s)
    A = 1j*np.asmatrix(x)
    w = np.cos((2*np.pi)/n)+1j*np.sin((2*np.pi)/n)
    for i in srange(euler_phi(n)):
        for j in srange(euler_phi(n)):
            I1 = I_n(n)
            k = I1[i]
            c = k*j
            A[i,j] = w**c
    return A

def dnumber_numpy(n):
    A = DFT_numpy(n)
    B = np.linalg.inv(A)
    d_n = LA.norm(B, np.inf)
    return d_n
```

Figure 3: Program for D_n & d_n with numpy

2. We then created the following plot depicting the point (n, d_n) for $n \leq 200$:

```
points = [[n,dnumber_numpy(n)] for n in srange(1,200)]
show(list_plot(points))
```

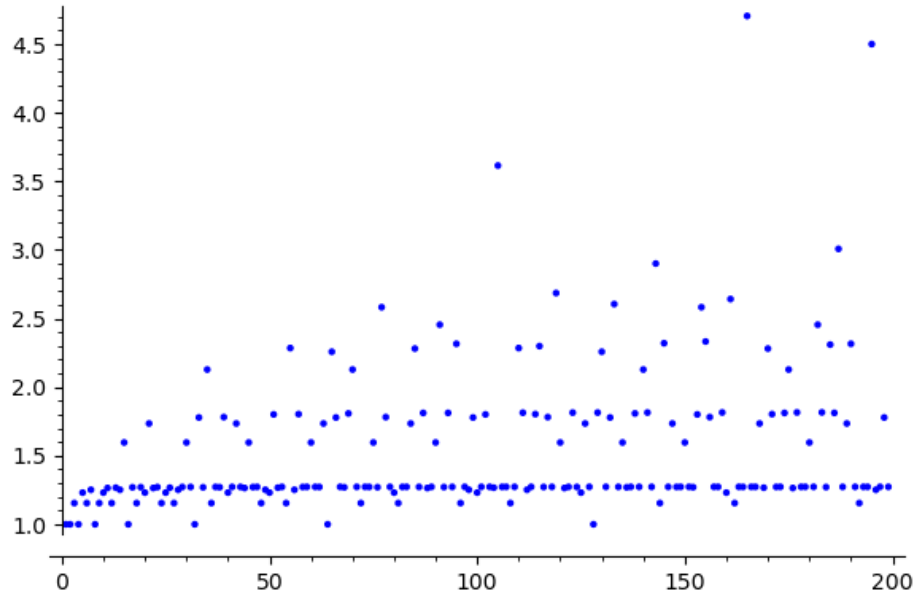


Figure 4: Plot (n, d_n)

5.2 Properties of d_n

To study some properties of d_n, we got given the following tasks:

1. Take some $n \geq 2$ & compute d_n, then take p | n prime & compute d_{pn}, d_{p^2n},d_{p^3n}. Repeat it and note your observations.

2. Compute d_p for primes p ≤ 500 & depict them. Can you guess the actual value of $\lim_{p\to\infty} d_p$?

3. Take some $n \geq 2$ & odd & compute d_n, then compute d_{2n}. Repeat it and note your observations.

4. Take some $n \geq 2$, then d_n only depends on the odd prime factors of n.

Here are our results of the tasks:

1. We choose $n = 6$ and $p = 3$. p is a prime number and $p \mid n$ works.

```
dnumber_numpy(6)
1.1547005383792512

dnumber_numpy(18)
1.1547005383792515

dnumber_numpy(54)
1.1547005383792517

dnumber_numpy(162)
1.1547005383798006
```

Figure 5: Value of d_6, d_{18}, d_{54} & d_{162}

We see that they are all the same. Before making our assumption, we tried the same for $n = 10$ and $p = 5$:

```
dnumber_numpy(10)
1.2310734148701012

dnumber_numpy(50)
1.2310734148701032

dnumber_numpy(250)
1.2310734148717926

dnumber_numpy(1250)
1.2310734148701012
```

Figure 6: Value of d_{10}, d_{50}, d_{250} & d_{1250}

Again, we obtain the same results, so we made the following assumption:
For every $n \geq 2$, for every $p \mid n$ prime, we have $d_n = d_{pn} = d_{p^2n} = d_{p^3n}$

2. To compute the limit of d_p we started by looking in our plot what d_p tends to.

```
points = [[n,dnumber_numpy(n)] for n in srange(1,300) if n in Primes()]
show(list_plot(points))
```

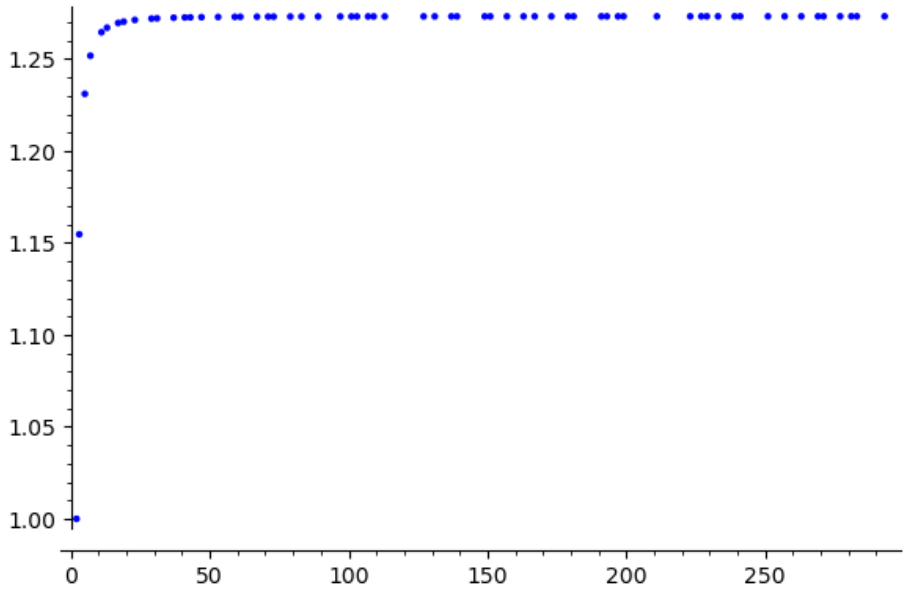


Figure 7: Plot (n, d_n) & n prime

With the plot we can see that d_p tends to $\approx 1,275$.
To get a further approximation, we computed $d_{499} = 1.273235339136858$
With the help of Wolfram Alpha, we found that:

$$\lim_{p \to +\infty} d_p = \frac{4}{\pi}$$

3. Lastly computing d_n and d_{2n} for $n \geq 2$ & odd, we tried for $n \in \{7, 13, 17\}$:

$$\begin{aligned} d_7 = 1.251796076438521 &\approx d_{14} = 1.2517960764385205 \\ d_{13} = 1.267037069922848 &\approx d_{26} = 1.2670370699228455 \\ d_{17} = 1.269613959677841 &\approx d_{34} = 1.2696139596778349 \end{aligned}$$

So we made the assumption that for every $n \geq 2$ & odd, we have $d_n = d_{2n}$

4. In other words, if m is the product of all odd prime factor of n, then $d_n = d_m$.

Example

$$\begin{aligned} d_{50} &= d_{2 \cdot 5^2} = d_{5^2} = d_5 \\ d_{90} &= d_{2 \cdot 3^2 \cdot 5} = d_{3^2 \cdot 5} = d_{3 \cdot 5} = d_{15} \end{aligned}$$

With this theorem we can now compute the d_n of higher numbers easier and faster, by decomposing n into prime numbers and keeping only the odd prime factors of n.
Like for example let's compute:

$$d_{14348907} = d_{3^{15}} = d_3 = 1.1547005383792517$$
$$d_{5184} = d_{2^6.3^4} = d_3 = 1.1547005383792517$$
$$d_{893025} = d_{3^6.5^2.7^2} = d_{3.5.7} = d_{105} = 3.6154773875055364$$

5.3 Plots of distinct odd prime numbers

Let us now create a plot depicting (n, d_n) for $n \leq 500$ being the product of distinct odd primes.

```
def distinct_prime(n):
    if n % 2 == 0:
        return False
    F = factor(n)
    for i in srange(len(F)):
        if F[i][1] != 1:
            return False
    return True
```

Figure 8: Program to check n distinct prime

The function begins with a check to see if the input n is even (n % 2 == 0). If n is even, the function returns **False**, indicating that n is not a distinct prime number. Next, the code iterates over the prime factors stored in F. F is a list of tuples, where each tuple represents a prime factor and its exponent. The loop checks if any prime factor has an exponent other than 1 ($F[i][1] != 1$). If such a case is found, the function returns **False**, indicating that n is not a distinct prime number. If none of the prime factors have an exponent other than 1, the function returns **True**, indicating that n is a distinct prime number.

Now that we defined this function, we can write the program that considers the plot of (n, d_n) for $n \leq 500$ product of distinct odd primes. When we run the program, we get the following plot:

```
points = [[n,dnumber_numpy(n)] for n in srange(1,500)  if distinct_prime(n) == True]
show(list_plot(points))
```

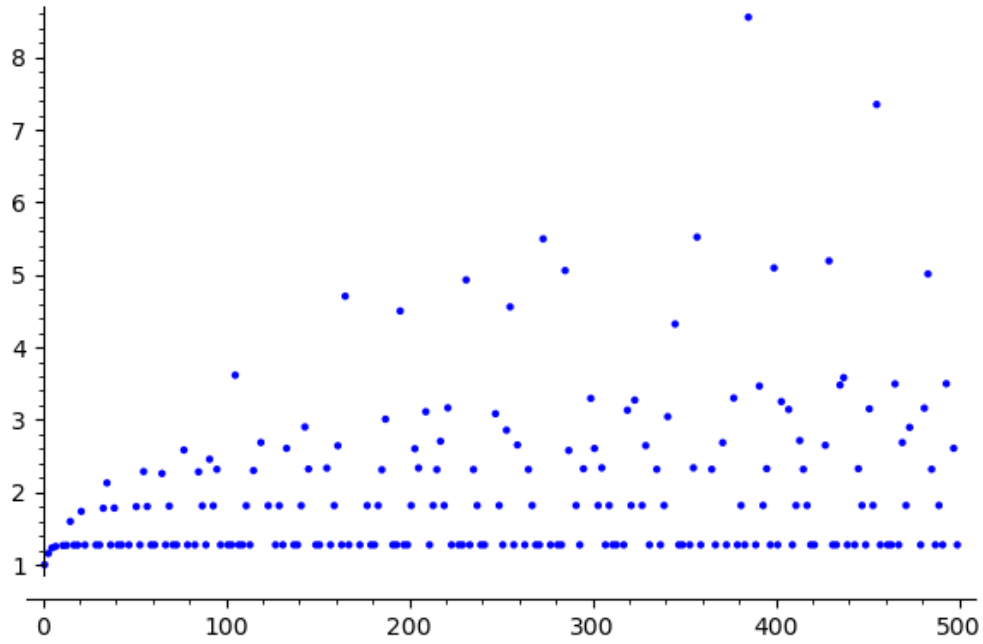


Figure 9: Plot (n, d_n) & n distinct prime

Looking at the plot, we notice that some horizontal lines appear on the graph. For three such lines, let's find a criterion for a point (n, d_n) to lie on them. We observe the behaviour of these three lines to find their asymptotic behaviour and find a general conjecture.
After observations we notice that for $q \geq 3$ prime, there is k_q such that

$$\lim_{p \to +\infty \ prime} d_{p \cdot q} = k_q \in \mathbb{R}$$

We also notice that the first horizontal asymptote is $\frac{4}{\pi}$ as its the line containing only prime numbers. We deduce it with the theorem that claims

$$\lim_{p \to +\infty \ prime} d_p = \frac{4}{\pi}$$

To get a better view of the three first lines, we create a plot containing these lines and their horizontal asymptotes.

First, we compute the three first lines. The first one is defined as d_n for n prime. The second line and third line are defined respectively as $d_{n\cdot 3}$ and $d_{n\cdot 5}$ & n still prime.

```
line_1 = [[n,dnumber_numpy(n)] for n in srange(1,250) if n in Primes()]
line_2 = [[3*n,dnumber_numpy(3*n)] for n in srange(5,85) if n in Primes()]
line_3 = [[5*n,dnumber_numpy(5*n)] for n in srange(7,50) if n in Primes()]
```

Figure 10: First three lines

Then, we compute the asymptote for each line. For the first line we claimed that it's equal to $\frac{4}{\pi}$. And for the two other asymptotes we take the maximum value it takes on each line and we define it as their horizontal asymptotes.

```
horizontal_1 = [[n,4/pi] for n in srange(1,250)]

horizontal2 = [dnumber_numpy(3*n) for n in srange(5,85) if n in Primes()]
h2 = max(horizontal2)
horizontal_2 = [[n,h2] for n in srange(1,250)]

horizontal3 = [dnumber_numpy(5*n) for n in srange(7,50) if n in Primes()]
h3 = max(horizontal3)
horizontal_3 = [[n,h3] for n in srange(1,250)]
```

Figure 11: Horizontal asymptotes

Now we only have to put everything together.
To make the lines more apparent, we coloured each one with their horizontal asymptotes with different colors.

```
show(list_plot(horizontal_1, color = 'blue', alpha = 0.1) +
     list_plot(horizontal_2, color = 'red', alpha = 0.1) +
     list_plot(horizontal_3, color = 'green', alpha = 0.1) +
     list_plot(line_1 , color = 'blue') +
     list_plot(line_2 , color = 'red') +
     list_plot(line_3 , color = 'green'))
```

Figure 12: Final program of the plot

After running the program, we get the following plot:

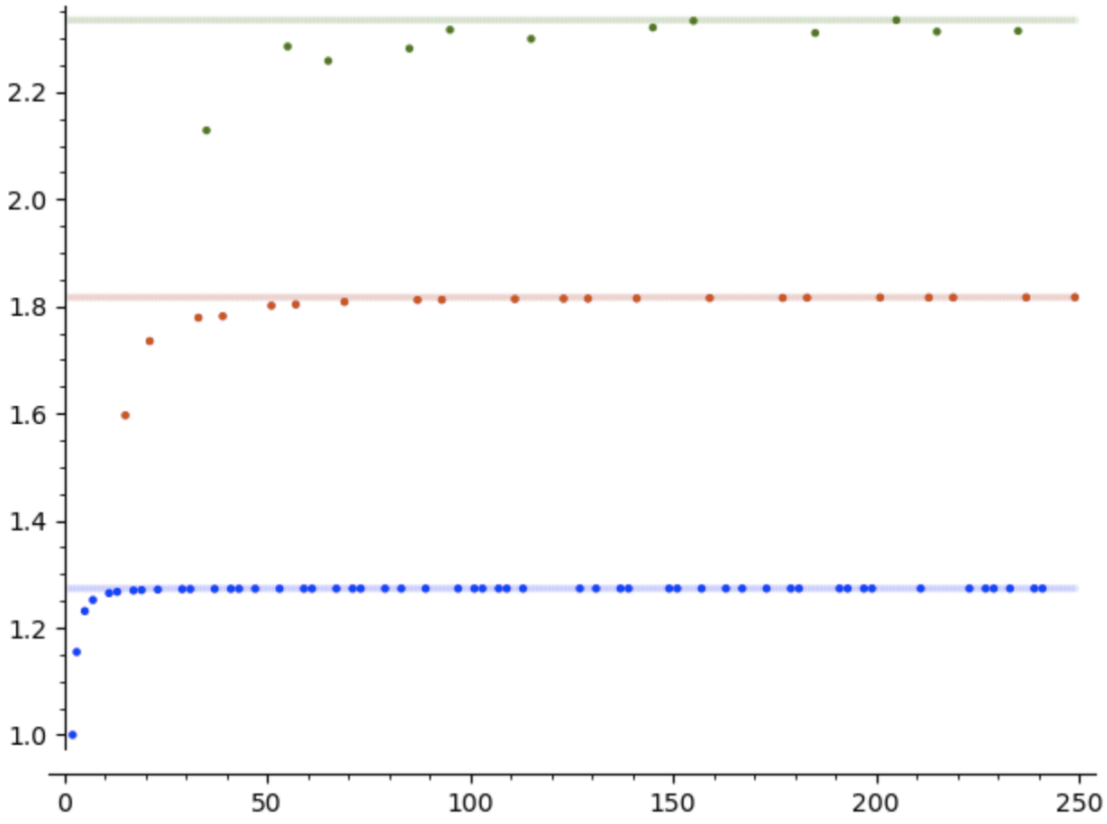


Figure 13: Plot of the three first 'lines'

6 Maximal and Minimal rows in D_n^{-1}

We will now concentrate on finding which rows are maximal and minimal in D_n^{-1}.

6.1 Maximal rows

We first ask ourselves which rows of D_n^{-1} are maximal. In other words, we want to find all rows $i \in \{0, ..., \phi(n) - 1\}$ for which we have

$$\left| \left(\sum_{j \in (\mathbb{Z}/n\mathbb{Z})^*} |e_{ij}| \right) - d_n \right| < 10^{-6} \qquad (*)$$

where $D_n^{-1} = (e_{ij})$ with i and j running through $\{0, ..., \phi(n) - 1\}$ and $(\mathbb{Z}/n\mathbb{Z})^*$, respectively.
Here we actually want to find all rows for which $(*)$ is equal to 0, but since we're working on a computer program it will never be exactly 0. So we have to put this condition to make it work. 10^{-6} is so small small enough to be consider as almost 0, so it's enough to take outputs underneath this number.

For the program, we define the function as follows:

1. First, we define the function as 'dnumber_numpy' of n.
2. We store the result of the DFT matrix for size n in the matrix A.
3. This line computes the inverse of the DFT matrix A using NumPy's **np.linalg.inv** function. The result is stored in the matrix B.
4. We use the **LA.norm** function to compute the matrix norm of B. The parameter **np.inf** indicates that the function should compute the spectral norm, which is the maximum singular value of the matrix. In other words, it calculates the maximum absolute column sum of the matrix B. The result is assigned to the variable d_n.
5. Finally we can return d_n.

```
def dnumber_numpy_max(n):
    A = DFT_numpy(n)
    B = np.linalg.inv(A)
    d_n = LA.norm(B, np.inf)
    return d_n
```

Figure 14: Maximal dnumber_numpy

6.2 Minimal rows

We now have to do the same but now for the minimal row.

$$D_n : \begin{cases} i \in (\mathbb{Z}/n\mathbb{Z})^* \\ j \in \{0, ..., \phi(n) - 1\} \end{cases}$$

$$D_n^{-1} : \begin{cases} j \in \{0, ..., \phi(n) - 1\} \\ i \in (\mathbb{Z}/n\mathbb{Z})^* \end{cases}$$

For this program, we define the function almost the same as the previous one, expect for the variable d_n. The parameter changes from **np.inf** to **-np.inf**, which corresponds to the minimum absolute row sum.

```
def dnumber_numpy_min(n):
    A = DFT_numpy(n)
    B = np.linalg.inv(A)
    d_n = LA.norm(B,-np.inf)
    return d_n
```

Figure 15: Minimal dnumber_numpy

6.3 Difference between maximum & minimum rows

Now we will focus on the difference between the maximal row of D_n^{-1} and the minimal row of D_n^{-1} and how this difference change with n. First, we write the program who computes this difference:

```
def difference_dnumber(n):
    dn_max = dnumber_numpy(n)
    dn_min = dnumber_numpy_min(n)
    difference = dn_max - dn_min
    return difference
```

Figure 16: Difference between maximum & minimal row

The program is quite simple since we only need to use our functions from before, who compute the maximum and minimum number from the matrix and compute the difference of these.

Example

```
difference_dnumber(32)
```

```
1.3322676295501878e-15
```

```
difference_dnumber(296)
```

```
1.9984014443252818e-14
```

Figure 17: Differences

Those examples show that the difference between the maximal row and minimal row of D_n^{-1} is 0.
Let us now compute a plot of the difference to better see how the difference between the maximal row of D_n^{-1} and the minimal row of D_n^{-1} change with n.

```
sizes = list(range(2, 200))
differences = [dnumber_numpy_max(n) - dnumber_numpy_min(n) for n in sizes]

points = [(sizes[i],differences[i]) for i in range(len(sizes))]
list_plot(points, plotjoined = False, color = 'blue', marker ='o')
```

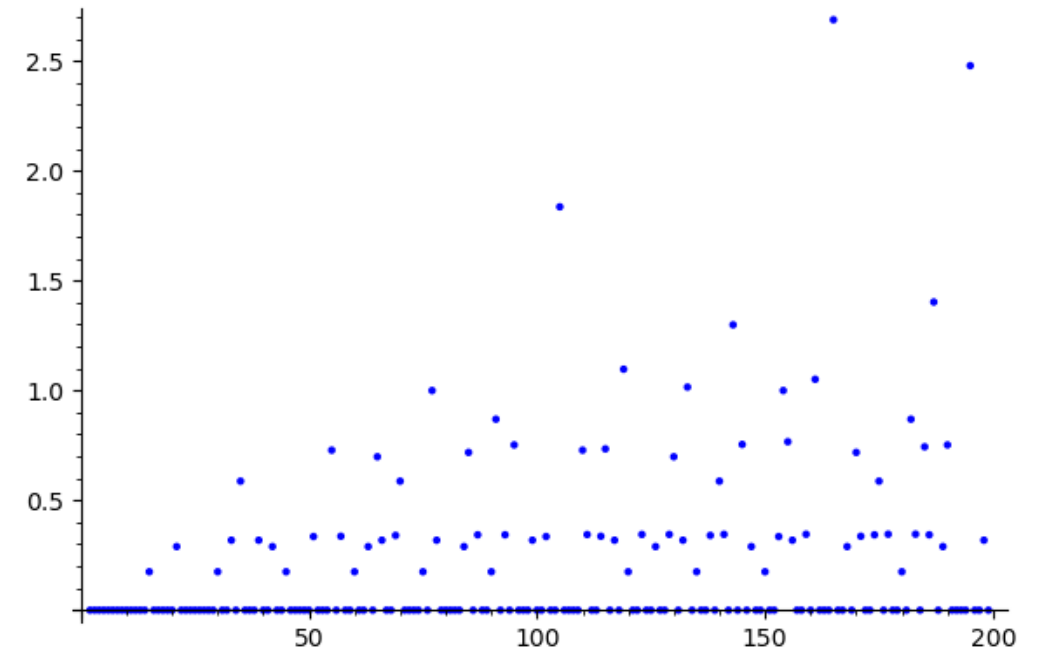


Figure 18: Plot of the difference between minimal & maximal rows

We can see that most of these differences are equal to 0, but only for distinct odd prime numbers, there exist a difference ≥ 0

Let us show the plot (n, difference_dnumber(n)) for n distinct odd prime numbers.

```
line_1 = [[n,difference_dnumber(n)] for n in srange(1,200) if n in Primes()]
line_2 = [[3*n,difference_dnumber(3*n)] for n in srange(5,66) if n in Primes()]
line_3 = [[5*n,difference_dnumber(5*n)] for n in srange(7,40) if n in Primes()]

horizontal_1 = [[n,0] for n in srange(1,200)]

horizontal2 = [difference_dnumber(3*n) for n in srange(5,66) if n in Primes()]
h2 = max(horizontal2)
horizontal_2 = [[n,h2] for n in srange(1,200)]

horizontal3 = [difference_dnumber(5*n) for n in srange(7,40) if n in Primes()]
h3 = max(horizontal3)
horizontal_3 = [[n,h3] for n in srange(1,200)]

show(list_plot(horizontal_1, color = 'blue', alpha = 0.1) +
     list_plot(horizontal_2, color = 'red', alpha = 0.1) +
     list_plot(horizontal_3, color = 'green', alpha = 0.1) +
     list_plot(line_1 , color = 'blue') +
     list_plot(line_2 , color = 'red') +
     list_plot(line_3 , color = 'green'))
```

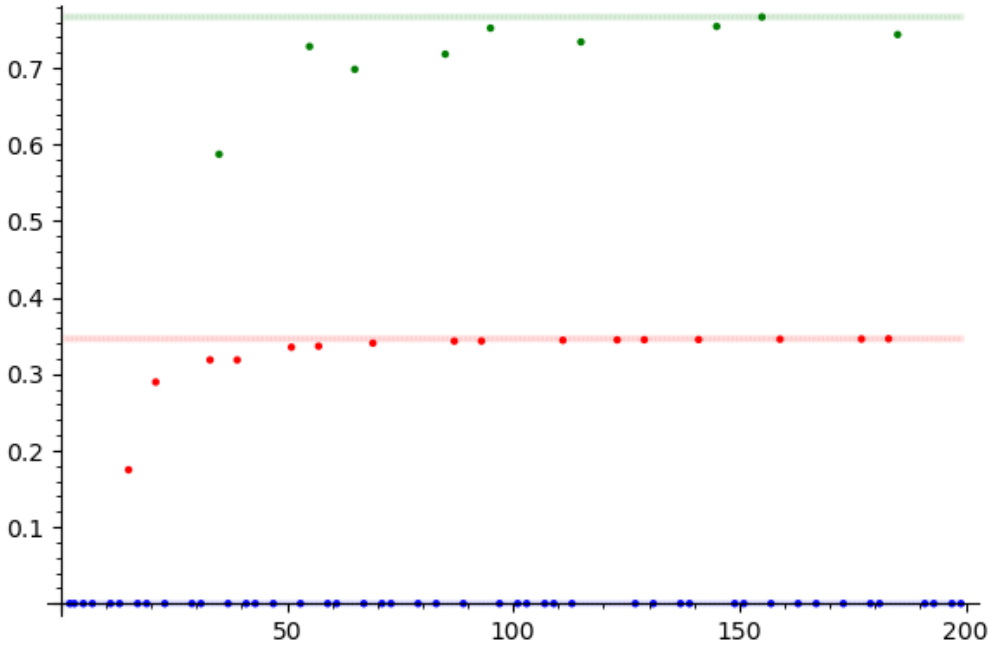


Figure 19: Plot of the lines of difference_dnumber(n)

7 Matrix Plots

When we take the absolute value of D_n^{-1}, we can create some beautiful images with the program **matrix_plot** of SageMath.
Let us first write a program to get the absolute value of D_n^{-1}:

```
def abs_DFT_numpy(n):
    A = DFT_numpy(n)
    B = np.linalg.inv(A)
    C = abs(B)
    return C
```

Figure 20: Absolute value of the inverse DFT matrix

With the help of the program **matrix_plot** we can write a function to create the images:

```
def matrix_plots(N):
    A = abs_DFT_numpy(N)
    P = matrix_plot(A, ticks = [[], []], cmap = 'GnBu')
    return P
```

Figure 21: Program matrix_plot

```
matrix_plots(205)
```

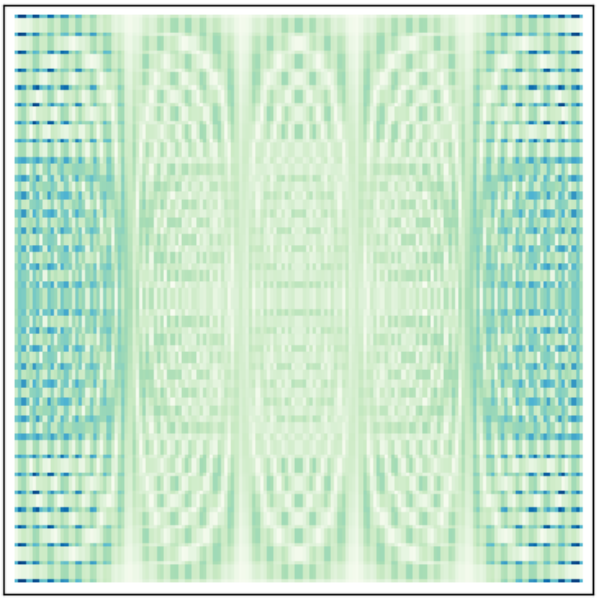


Figure 22: Image D_{205}^{-1}

The color of the images can of course be exchanged.

We can observe some interesting patterns for a certain n in these images:

Let us write a program to create many images to produce a video with these images where we can observe the patterns.

```
for i in srange(5,100):
    if i in Primes():
        A = abs_DFT_numpy(5*i)
        R = matrix_plot(A, ticks = [[], []], cmap = 'RdYlBu')
        R.save(f'{5*i}.pdf')
```

Figure 23: Creating images

Let p be a prime number. This program create the images for $5p$.

Here you can find the link for the video:

https://www.youtube.com/watch?v=8w_scM6KU_w

We can observe that the images are divided into horizontal and vertical segments. Let p be prime bumber and q such that $p < q$, then the image of $D_{p \cdot q}^{-1}$ has $p - 1$ horizontal segments and p vertical segments.

References

[1] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-lwe cryptography. 2013. https://eprint.iacr.org/2013/293.

[2] Shai Halevi and Victor Shoup. Bootstrapping for helib. Cryptology ePrint Archive, Paper 2014/873, 2014. https://eprint.iacr.org/2014/873.

[3] Shai Halevi and Victor Shoup. Design and implementation of helib: a homomorphic encryption library. 2020. https://eprint.iacr.org/2020/1481.

[1] [2] [3]