Factorization of a semiprime (pq-number) in polynomial time using precalculated tables

Jesús Sánchez

Independent Researcher, Bilbao, Spain
Email: jesus.sanchez.bilbao@gmail.com
https://www.researchgate.net/profile/Jesus_Sanchez64
ORCID 0000-0002-5631-8195

Copyright © 2021 by author

Abstract

In this paper, it will be shown that it is possible to detect if a number p is prime [1] or not, using the following expression involving the q-polygama function [2] (represented as ψ) and being c whatever positive real number higher than zero:

$$q_2 = \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{pj\omega}\left(\frac{\psi_q(1)+\ln\left(1-e^{-c-j\omega}\right)}{\ln\left(e^{-c-j\omega}\right)} - \frac{e^{-c-j\omega}+e^{-2c-2j\omega}}{1-e^{-c-j\omega}}\right)d\omega \quad (1)$$

If the result is zero, p is a prime number. If it is different from zero, the result will give information about the number of factors that the number p has.

We will check that using this as a basis, we can obtain the following integral:

$$q_3 = \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{pj\omega}\left(\sum_{k=2}^{k=\infty} e^{-c(2+k)}e^{-2kj\omega}\frac{1}{1-e^{-c}e^{-kj\omega}}\right)d\omega \quad (2)$$

And it is possible to obtain the sum of the factors of a semiprime number p [3] with it. Once we have the sum and the product of the factors, it is immediate to obtain the two factors of the semiprime number. The solution of that integral (solving it numerically) is obtained in polynomial time (quadratic).

To do so, the second element of the product inside the integral has had to be calculated before and stored in a table (data base). Once this prework is done, the result is given in polynomial time (linear) independently of p or its size. You can check that this is possible because the second element of the product within the integral does not depend on p:

$$\sum_{k=2}^{k=\infty} e^{-c(2+k)}e^{-2kj\omega}\frac{1}{1-e^{-c}e^{-kj\omega}} \quad (3)$$

Keywords

Primality test, factorization, pq-number, semiprime, biprime, prime, number theory, RSA encryption

1. Introduction

This paper will start in a very similar way than the paper [4]. So, if you have already read that paper you can skip the chapters 1-7.

We will obtain the following integral (being ψ the q-polygamma function):

$$q_2 = \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{pj\omega}\left(\frac{\psi_q(1) + \ln\left(1 - e^{-c-j\omega}\right)}{\ln\left(e^{-c-j\omega}\right)} - \frac{e^{-c-j\omega} + e^{-2c-2j\omega}}{1 - e^{-c-j\omega}}\right)d\omega \quad (1)$$

Being p the number that we want to check if it is a prime or not. And being c whatever real number different form zero. We will check that this "whatever" is theoretical, as the integral will be calculated numerically, so we will use "c" as a "calibration factor" to try to get the most accurate result from the integral with the less computation effort.

If the result of the integral is zero (or it is so small that it can be considered to be zero) the number p is a prime. If not, the result will tell us how many permutations of two divisors, the number p has.

To obtain the expression of the integral (1), the following steps shall be taken. First, we will create a domain with all the composite numbers in the domain included.

Then, we will use the Fourier transform [5] to change from the original domain (time domain) to the frequency domain.

Using the Parseval's theorem [6][7], we can check if a certain number is the above created domain or not. The use of Parseval's theorem will lead to the above integration.

If the number p that we want to check is not in the domain, the result of the integration is zero and the number is a prime. If instead, the result is an integer, this integer will tell us how many permutations of two divisors the number p has. And in consequence, how many divisors, the number has.

The important thing is that making some modifications to the integral that will be commented in chapter 11, we will get to the following expression:

$$q_3 = \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{pj\omega}\left(\sum_{k=2}^{k=\infty} e^{-c(2+k)}e^{-2kj\omega}\frac{1}{1 - e^{-c}e^{-kj\omega}}\right)d\omega \quad (2)$$

If p is a semiprime number (a pq-number), the result of the integral will lead us to the value of one of the factors. This integral. if it is performed completely every time, is computationally very expensive. But as the second part of the product inside of the integral does not depend on p, we can pre-calculate it and then use the integral to factorize different semiprime numbers with a much less expensive computation per number. In fact, we can do it in linear time (after the pre-calculation).

2. Matrix of composite numbers

As commented, if you have already read [4] you can skip chapters 1 to 7 as they will be a repetition of it (with little variants).

It is possible to create a matrix with the only composite numbers in certain domain. We can create it assuring that each element of the matrix is the product of every two integers in the domain. We can perform this operation one by one until having used all the integers. So, the result will be all the composite numbers in a certain domain.

You can see an example in figure 1,

$$\begin{bmatrix} 2\cdot 2 & 2\cdot 3 & 2\cdot 4 & 2\cdot 5 \\ 3\cdot 2 & 3\cdot 3 & 3\cdot 4 & 3\cdot 5 \\ 4\cdot 2 & 4\cdot 3 & 4\cdot 4 & 4\cdot 5 \\ 5\cdot 2 & 5\cdot 3 & 5\cdot 4 & 5\cdot 5 \end{bmatrix}$$

Fig.1

And the result attached in figure 2:

$$\begin{bmatrix} 4 & 6 & 8 & 10 \\ 6 & 9 & 12 & 15 \\ 8 & 12 & 16 & 20 \\ 10 & 15 & 20 & 25 \end{bmatrix}$$

Fig.2

We can see that all the composite numbers between 4 and 10 are included in the matrix: 4,6,8,9,10. And as expected, the prime numbers 2,3,5 and 7 are not. But we can see that the number 14 that is composite is not included. Why? Because it is composed by 7·2 and the 7 is not included in the multiplications. This means, the domain where it is true that all the composite numbers are included (and none of the primes) finishes in 2·5=10 (being 5 the last number we are using in the multiplications).

But we can consider a general matrix with m rows and m columns (meaning the last element is (m+1)). If (m+1) is a finite number, the matrix will include all the composite numbers until 2·(m+1). If instead, we consider the case where this last element (m+1) tends to infinity, we are including all the composite numbers in this matrix, meaning that the natural numbers that will not appear in this hypothetical matrix are all the prime numbers.

$$\begin{bmatrix} 2\cdot 2 & 2\cdot 3 & 2\cdot 4 & 2\cdot 5 & \dots & 2\cdot(m+1) \\ 3\cdot 2 & 3\cdot 3 & 3\cdot 4 & 3\cdot 5 & \dots & 3\cdot(m+1) \\ 4\cdot 2 & 4\cdot 3 & 4\cdot 4 & 4\cdot 5 & \dots & 4\cdot(m+1) \\ 5\cdot 2 & 5\cdot 3 & 5\cdot 4 & 5\cdot 5 & \dots & 5\cdot(m+1) \\ \dots & \dots & \dots & \dots & \dots & \dots \\ (m+1)\cdot 2 & (m+1)\cdot 3 & (m+1)\cdot 4 & (m+1)\cdot 5 & \dots & (m+1)\cdot(m+1) \end{bmatrix}$$

Fig.3

$$\begin{bmatrix} 4 & 6 & 8 & 10 & \dots & 2\cdot(m+1) \\ 6 & 9 & 12 & 15 & \dots & 3\cdot(m+1) \\ 8 & 12 & 16 & 20 & \dots & 4\cdot(m+1) \\ 10 & 15 & 20 & 25 & \dots & 5\cdot(m+1) \\ \dots & \dots & \dots & \dots & \dots & \dots \\ (m+1)\cdot 2 & (m+1)\cdot 3 & (m+1)\cdot 4 & (m+1)\cdot 5 & \dots & (m+1)\cdot(m+1) \end{bmatrix}$$

Fig.4

3. Including the elements of the composite matrix in the discrete time domain

Now, the next movements will be quite straight forward. The first thing we will do, is to introduce these calculated composite numbers in the discrete time domain. We will put unitary delta functions in the positions of the x axis indicated by the matrix of composite numbers (figure 5).

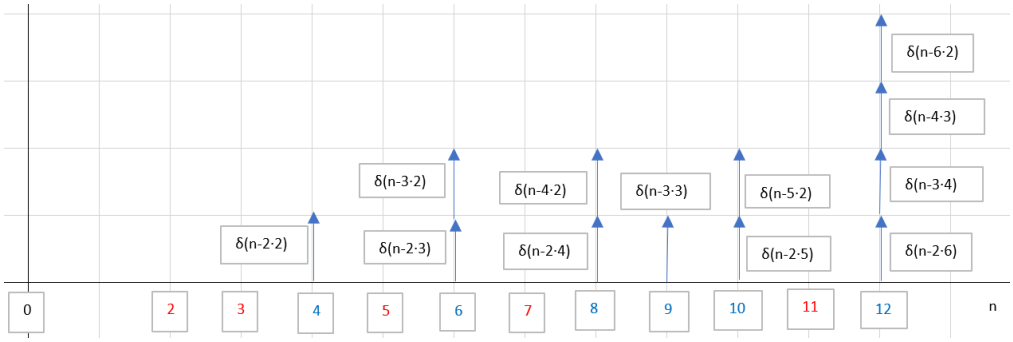


Fig.5

We can see in the above figure, that each Dirac delta is located in a composite number that is obtained as the product of two integers. This is, each Dirac delta represents the factorization of two numbers of each composite number. And the order of the factors counts. This is, δ(n-2·3) is one delta and δ(n-3·2) counts as a second delta in the same position.

Therefore, if we do not have any delta in a certain position (see the red numbers), this means that the number is prime.

If we have only one Dirac Delta in one position, it means that the number is the square of a prime number (see 4=2·2 or 9=3·3 for example). The reason for this, is that if the order counts, the only way to have only one product -and not a second, exchanging the order- is that the two numbers of the product are the same (leading to a square).

If the number of Dirac deltas is odd but higher than one, the number has an integer square root (but this root is not a prime). If the number of deltas is even, it has different permutations of different products, but it does not have an integer square root.

There is also the special case when the number of deltas is two. In general, this would mean that the number is semiprime (pq-number) []. As generally, two deltas represent the product of two prime numbers in the two possible orders. See the number 6 for example (2·3 and 3·2). But you can also check that the number 8 has two deltas but is not a semiprime. We will come back to this later.

4. Creating the functions f(n) and g(n)

We can create a function that is the sum of all these deltas. And this function will have the information of all these composite numbers. This function in the discrete domain is defined like this:

$$f(n) = \sum_{k_2=2}^{k_2=m+1} \sum_{k_1=2}^{k_1=m+1} \delta(n - k_1 \cdot k_2) = \sum_{k_2=2}^{k_2=m+1} (\delta(n - 2 \cdot k_2) + \delta(n - 3 \cdot k_2) + \delta(n - 4 \cdot k_2) + \cdots \delta(n - (m+1) \cdot k_2)) = \\ = \delta(n - 2 \cdot 2) + \delta(n - 2 \cdot 3) + \cdots \delta\big(n - 2 \cdot (m+1)\big) + \\ + \delta(n - 3 \cdot 2) + \delta(n - 3 \cdot 3) \ \ldots \ \delta\big(n - (m+1) \cdot (m+1)\big) \qquad (4)$$

This function f(n) has the information regarding all the composite numbers inside the domain 2·(m+1). But the important information there is what it is not included. All the integers smaller than 2·(m+1) that are not included in this function, are prime numbers.

How do we spot them? If we want to know if the number 5 is prime or not. We have to check if there is a Dirac delta in position 5. This means, we have to check if the following function g(n) is inside the function f(n):

$$g(n) = \delta(n - 5) \qquad (5)$$

In general, if we want to check if whatever number p is prime, g(n) would be:

$$g(n) = \delta(n - p) \qquad (6)$$

We could do this performing the multiplication one by one for all the points in the x axis between the function f(n) and the function g(n), as follows:

$$\sum_{n=-\infty}^{n=\infty} f(n) \cdot g(n) = \sum_{n=4}^{n=(m+1)} f(n) \cdot g(n) \qquad (7)$$

As the function g(n) is zero everywhere except in the point p, if the result of the multiplication point by point between f(n) and g(n) is zero, it means that f(n) does not have a Dirac delta in p. So, in that case (no deltas in p), p would be prime.

If the result is different, it will tell us the value of f(n) at that point. And the value of f(n) at that point is the number of deltas included. This means, the number of all the possible products of two integers that the number p has.

The problem is that to perform this operation in the time domain, does not add any value. Because to know the result, we have to perform all the multiplications between f(n) and g(n) in all the points of the domain, to see if f(n) has a value different from zero in the position where g(n) has the delta.

The only way to perform this operation efficiently is to do it in the frequency domain as we will see now.

5. The frequency domain

What we will do is to convert the functions f(n) an g(n) to the frequency domain. To do so, we use the Fourier transform.

We start with f(n). We know from signal theory that the Fourier transform of a shifted Dirac delta is:

$$\delta(n-a) \rightarrow e^{-ajw} \qquad (8)$$

So, the function f(n), in the frequency domain has the form F(ω):

$$\begin{gathered} F(\omega) = \sum_{k_1=2}^{k_1=m+1} \sum_{k_2=2}^{k_2=m+1} e^{-k_1 \cdot k_2 j\omega} = \\ = e^{-2\cdot 2j\omega} + e^{-2\cdot 3j\omega} + \cdots + e^{-2\cdot(m+1)j\omega} + \\ + e^{-3\cdot 2j\omega} + e^{-3\cdot 3j\omega} + \cdots + e^{-3\cdot(m+1)j\omega} + \\ + \cdots \quad + \cdots \quad + \cdots + \cdots \quad + \\ + e^{-(m+1)\cdot 2j\omega} + e^{-(m+1)\cdot 3j\omega} \ldots + e^{-(m+1)\cdot(m+1)j\omega} \end{gathered} \qquad (9)$$

We can put above sum in the following form:

$$\begin{gathered} = (e^{-2j\omega})^2 + (e^{-2j\omega})^3 + \cdots + (e^{-2j\omega})^{m+1} + \\ + (e^{-3j\omega})^2 + (e^{-3j\omega})^3 + \cdots + (e^{-3j\omega})^{m+1} + \\ + \cdots \quad + \cdots \quad + \cdots + \cdots \quad + \\ + (e^{-(m+1)j\omega})^2 + (e^{-(m+1)j\omega})^3 + \cdots + (e^{-(m+1)j\omega})^{m+1} \end{gathered} \qquad (10)$$

As you can see above, each line is a geometric series sum. So, using the known result of geometric series [8], we can write:

$$\begin{gathered} = e^{-2\cdot 2j\omega} \frac{1 - e^{-2mj\omega}}{1 - e^{-2j\omega}} + \\ + e^{-2\cdot 3j\omega} \frac{1 - e^{-3mj\omega}}{1 - e^{-3j\omega}} + \\ + \cdots + \\ + e^{-2(m+1)j\omega} \frac{1 - e^{-(m+1)mj\omega}}{1 - e^{-(m+1)j\omega}} = \end{gathered}$$

$$= \sum_{k=2}^{k=m+1} e^{-2kj\omega} \frac{1 - e^{-mkj\omega}}{1 - e^{-kj\omega}} \qquad (11)$$

For the function g(n), following the same process:

$$g(n) = \delta(n - p) \qquad (12)$$

The Fourier transform G(ω) of g(n) would be:

$$g(n) = \delta(n - p) \rightarrow G(\omega) = e^{-pj\omega} \qquad (13)$$

6. Parseval's theorem

Now, we get to the point. In the chapter 4, we wanted to perform the following operation:

$$\sum_{n=-\infty}^{n=\infty} f(n) \cdot g(n) = \sum_{n=4}^{n=(m+1)} f(n) \cdot g(n) \qquad (14)$$

If the result is different from zero, it means that p is composite, as it means that:

$$g(n) = \delta(n - p) \qquad (15)$$

g(n) has found delta/s in the same position in the function f(n) defined before, that has only deltas in composite number positions.

This operation in the time domain does not have added value (as you have to perform all multiplications), but fortunately a guy called Marc-Antoine Parseval [6][7] saved our day discovering the following theorem.

The general Parseval's theorem applied to discrete functions (as it is the case) says the following [7 Page 48/51]:

$$\sum_{n=-\infty}^{n=\infty} f(n) \cdot g^*(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} F(\omega) \cdot G^*(\omega)\, d\omega \qquad (16)$$

The asterisk* means the complex conjugate. All the functions we are dealing in this paper with, are real in the time domain so in the left-hand side no conjugation is necessary.

$$g^*(n) = g(n) \qquad (17)$$

For the righthand side, yes, we have complex functions. But as you can check all of them are exponential based. So, the conjugate is just to change the sign of the imaginary part of the exponent. This is immediate from Euler's formula [9].

$$G(\omega) = e^{-pj\omega} \qquad (18)$$

$$G^*(\omega) = e^{pj\omega} \quad (19)$$

So, coming back, this means, we can perform the operation of multiplying point by point the two functions in the time domain, just making the definite integral form -π to π of the multiplication of the two functions (its Fourier Transform) in the frequency domain.

So, the integral in the frequency domain (that represents the multiplication point by point of the functions f(n) and g(n) in the time domain) according Parseval's theorem is the following (and we will call its result as q):

$$q = \sum_{n=-\infty}^{n=\infty} f(n)\cdot g^*(n) = \sum_{n=-\infty}^{n=\infty} f(n)\cdot g(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} F(\omega)\cdot G^*(\omega)\,d\omega$$
$$= \frac{1}{2\pi}\int_{-\pi}^{\pi}\left(\sum_{k=2}^{k=m+1} e^{-2kj\omega}\frac{1-e^{-mkj\omega}}{1-e^{-kj\omega}}\right)\cdot e^{pj\omega}d\omega$$
$$= \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{pj\omega}\left(\sum_{k=2}^{k=m+1} e^{-2kj\omega}\frac{1-e^{-mkj\omega}}{1-e^{-kj\omega}}\right)d\omega \quad (20)$$

So, summing up:

$$q = \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{pj\omega}\left(\sum_{k=2}^{k=m+1} e^{-2kj\omega}\frac{1-e^{-mkj\omega}}{1-e^{-kj\omega}}\right)d\omega \quad (21)$$

We will call q the result of above expression. If q is zero, p is prime, if q is different from zero, p is composite, and the number q has information regarding the number of factors.

For possible calculation issues we may find later, it is to be noted that as, k is independent from ω the sum can be taken outside the integral, as follows:

$$q = \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{pj\omega}\left(\sum_{k=2}^{k=m+1} e^{-2kj\omega}\frac{1-e^{-mkj\omega}}{1-e^{-kj\omega}}\right)d\omega$$
$$= \frac{1}{2\pi}\sum_{k=2}^{k=m+1}\int_{-\pi}^{\pi} e^{pj\omega}e^{-2kj\omega}\frac{1-e^{-mkj\omega}}{1-e^{-kj\omega}}d\omega \quad (22)$$

There are two problems with above expression. The first one is that the domain is limited to (m+1), meaning that not all the composite primes are included in the summation (only the ones equal or below 2·(m+1)).

The second problem is that you cannot separate the summation from the exponential depending on p, so you have to perform the summation completely every time you calculate the integral with different numbers p.

Anyhow, we will continue using this expression to explain the theory and we will handle

these two issues from chapter 8 on.

I have attached in Annex B1 a program in Matlab/Octave that performs that operation and compare the time of computation with the brute force checking.

We will make some examples later.

7. Number of factors of p

We know that if p is composite, the number q (result from equation (22)) is the number of possible combinations of multiplications of two integer numbers that get the value p. And yes, with this number we can get the number of factors that p has.

This is the formula:

$$q = (n_{f1} + 1)(n_{f2} + 1)(n_{f3} + 1) \dots (n_{fz} + 1) - 2 \quad (23)$$

Being nf_1 the number of times the factor f_1 is repeated, nf_2 the number of times f_2 is repeated and so on.

Putting in another way is:

$$q + 2 = (n_{f1} + 1)(n_{f2} + 1)(n_{f3} + 1) \dots (n_{fz} + 1) \quad (24)$$

So, the factors of the obtained number q+2 (much smaller than p) tell us how many factors the number p originally has and how many of them are repeated.

Let's see with an example. We apply equation (22) to the number 20. To do this operation you can use the commented program in Annex B1 in Matlab/Octave. You will see that the result is q=4. So, we can check:

$$q + 2 = 4 + 2 = 6 = (n_{f1} + 1)(n_{f2} + 1)(n_{f3} + 1) \dots (n_{fz} + 1) = 3 \cdot 2 = (2 + 1)(1 + 1) \quad (25)$$

This means, p has to have a factor that is repeated twice and one that appears only once. And we see that it is true:

$$20 = 2 \cdot 2 \cdot 5 \quad (26)$$

The factor 2 is repeated twice and 5 once.

And the q is 4 because, there are four combinations of multiplications of two integers that get the result 20:

2·10, 4·5, 5·4, 10·2

So, everything fits.

Let's check, other ones:

p=102
q=6

$$q + 2 = 8 = 2 \cdot 2 \cdot 2 = (1 + 1)(1 + 1)(1 + 1) \qquad (27)$$

This means, three factors that are not repeated, and in fact 102=2·3·17

Another one:
p=36
q=7

$$q + 2 = 9 = 3 \cdot 3 = (2 + 1)(2 + 1) \qquad (28)$$

This means two factors repeated twice (each). In fact, 36=2·2·3·3

You can see that the formula (17) works.
We can use q to know how many factors a number p has if it is composite (or instead, detect it is a prime if q=0).

One of the problems, that this system has is that sometimes there are different solutions for the number of factors. For example, for q=2, the typical answer is:

$$q + 2 = 4 = 2 \cdot 2 = (1 + 1)(1 + 1) \qquad (29)$$

This means, two different factors not repeated. For example, 21=3·7=7·3.

But it could be:

$$q + 2 = 4 = (3 + 1) \qquad (30)$$

For example, the number 8=2·4=4·2

It has q=2 (two combinations of factors) but 8=2·2·2 This means one factor three times, as seen before.

So, the number q could have some values that have different solutions regarding the number of factors.

8. Extending the expression of F(ω) when (m+1) tends to infinity

To do this, we will repeat the exact same steps as before, but with little changes. These changes will let the summation converge also when (m+1) tends to infinity.

The way to do it is instead of having unitary deltas, to have deltas that are multiplied by

the real factor $e^{-k1\cdot k2\cdot c}$, being c a real number higher than zero, that we can choose freely. The criteria of selection of c will be based in the in the computational efforts that we need to get a valid result in the range of numbers desired. We will give some recommendations later.

So, we will follow the same steps as before. The only difference is that now we are reducing the value of the magnitude of each delta, as k_1 and k_2 approach infinity. This way we will force the last term of the summation to tend to zero when (m+1) tends to infinity. As we will see later, we can use a similar trick to introduce valuable information in the magnitude of the deltas, but this, we will check later.

So, following the same steps as before, but now with this reduction factor in the magnitude of the delta $e^{-k1\cdot k2\cdot c}$, we have:

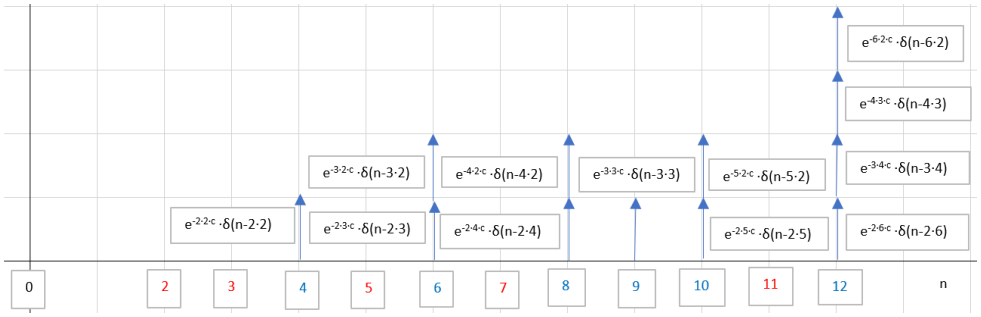


Fig.6

Now, we can get the value of f(n) in this case as:

$$f(n) = \sum_{k_2=2}^{k_2=m+1}\sum_{k_1=2}^{k_1=m+1} e^{-k_1\cdot k_2\cdot c}\delta(n-k_1\cdot k_2) = \sum_{k_2=2}^{k_2=m+1}(e^{-2\cdot k_2\cdot c}\delta(n-2\cdot k_2) + e^{-3\cdot k_2\cdot c}\delta(n-3\cdot k_2) + e^{-4\cdot k_2\cdot c}\delta(n-4\cdot k_2) + \cdots e^{-(m+1)\cdot k_2\cdot c}\delta(n-(m+1)\cdot k_2)) =$$
$$= e^{-2\cdot 2\cdot c}\delta(n-2\cdot 2) + e^{-2\cdot 3\cdot c}\delta(n-2\cdot 3) + \cdots e^{-2\cdot (m+1)\cdot c}\delta\big(n-2\cdot (m+1)\big) +$$
$$+e^{-3\cdot 2\cdot c}\delta(n-3\cdot 2) + e^{-3\cdot 3\cdot c}\delta(n-3\cdot 3)\ \ldots\ e^{-(m+1)\cdot (m+1)\cdot c}\delta\big(n-(m+1)\cdot (m+1)\big) \qquad (31)$$

Now if we make (m+1) tend to infinity we have:

$$f(n) = \sum_{k_2=2}^{k_2=\infty}\sum_{k_1=2}^{k_1=\infty} e^{-k_1\cdot k_2\cdot c}\delta(n-k_1\cdot k_2) = \sum_{k_2=2}^{k_2=m+1}(e^{-2\cdot k_2\cdot c}\delta(n-2\cdot k_2) + e^{-3\cdot k_2\cdot c}\delta(n-3\cdot k_2) + e^{-4\cdot k_2\cdot c}\delta(n-4\cdot k_2) + \cdots 0) =$$
$$= e^{-2\cdot 2\cdot c}\delta(n-2\cdot 2) + e^{-2\cdot 3\cdot c}\delta(n-2\cdot 3) + \cdots 0 +$$
$$+e^{-3\cdot 2\cdot c}\delta(n-3\cdot 2) + e^{-3\cdot 3\cdot c}\delta(n-3\cdot 3)\ \ldots\ 0 \qquad (32)$$

The zeroes in the end of the series appear because the exponential $e^{-k1\cdot k2\cdot c}$ tends to zero as k_1 and k_2 tend to infinity (and we have also defined c as a real number higher than zero). Remind that the previous limit for k_1 and k_2 was (m+1) before, and now (m+1) tends to infinity.

We can calculate the Fourier transform as in chapter 5 like this:

$$F(\omega) = \sum_{k_1=2}^{k_1=m+1} \sum_{k_2=2}^{k_2=m+1} e^{-k_1 \cdot k_2 \cdot c} e^{-k_1 \cdot k_2 j\omega} =$$
$$= e^{-2\cdot 2\cdot c} e^{-2\cdot 2 j\omega} + e^{-2\cdot 3\cdot c} e^{-2\cdot 3 j\omega} + \cdots + 0 +$$
$$+ e^{-3\cdot 2\cdot c} e^{-3\cdot 2 j\omega} + e^{-3\cdot 3\cdot c} e^{-3\cdot 3 j\omega} + \cdots + 0 +$$
$$+ \cdots \quad + \cdots \quad + \cdots + \cdots \quad +$$
$$+0 + 0 \ldots + 0 \qquad (33)$$

And now, we can also simplify it as geometric sums, but with a difference. As the last term tends to zero as k_1 and k_2 tend to infinity, we can eliminate it from the geometric sum expression [8] like this:

$$F(\omega) = e^{-2\cdot 2\cdot c} e^{-2\cdot 2 j\omega} \frac{1}{1 - e^{-2\cdot c} e^{-2 j\omega}} +$$
$$+ e^{-2\cdot 3\cdot c} e^{-2\cdot 3 j\omega} \frac{1}{1 - e^{-3\cdot c} e^{-3 j\omega}} +$$
$$+ e^{-2\cdot 4\cdot c} e^{-2\cdot 4 j\omega} \frac{1}{1 - e^{-4\cdot c} e^{-4 j\omega}} +$$
$$+ \cdots +$$
$$+0 =$$
$$= \sum_{k=2}^{k=\infty} e^{-2\cdot k\cdot c} e^{-2 k j\omega} \frac{1}{1 - e^{-k\cdot c} e^{-k j\omega}} = \sum_{k=2}^{k=\infty} \frac{(e^{-2\cdot c - 2 j\omega})^k}{1 - (e^{-c - j\omega})^k} \qquad (34)$$

Regarding the function g(n), it is exactly the same as in chapter 5. So, we will follow the same process:

$$g(n) = \delta(n - p) \qquad (35)$$

The Fourier transform G(ω) of g(n) would be:

$$g(n) = \delta(n - p) \rightarrow G(\omega) = e^{-p j\omega} \qquad (36)$$

9. Relation of F(ω) with the q-polygamma function

If we want to get rid of the summation in equation (22), there is a way. It exists a function called q-polygama function $\psi_k(z)$ [2] which definition is:

$$\psi_q(z) = -\ln(1 - q) + \ln(q) \sum_{n=0}^{n=\infty} \frac{q^{n+z}}{1 - q^{n+z}} \qquad (37)$$

For z=2, we have:

$$\psi_q(2) = -\ln(1-q) + \ln(q)\sum_{n=0}^{n=\infty}\frac{q^{n+2}}{1-q^{n+2}} \qquad (38)$$

If we substitute by k=n+2:

$$\psi_q(2) = -\ln(1-q) + \ln(q)\sum_{k=2}^{k=\infty}\frac{q^{k}}{1-q^{k}} \qquad (39)$$

For the geometric sum form $q^k/(1-q^k)$ that is inside the summation we can separate the first element and leave it as $q^k + (q^{2k}/(1-q^k))$, starting the geometric form sum in the second element:

$$\begin{aligned}\psi_q(2) &= -\ln(1-q) + \ln(q)\sum_{k=2}^{k=\infty}\left(q^k + \frac{q^{2k}}{1-q^k}\right) = \\ &= -\ln(1-q) + \ln(q)\sum_{k=2}^{k=\infty}q^k + \ln(q)\sum_{k=2}^{k=\infty}\frac{q^{2k}}{1-q^k} = \\ &= -\ln(1-q) + \ln(q)\frac{q^2}{1-q} + \ln(q)\sum_{k=2}^{k=\infty}\frac{q^{2k}}{1-q^k} \qquad (40)\end{aligned}$$

This is:

$$\psi_q(2) = -\ln(1-q) + \ln(q)\frac{q^2}{1-q} + \ln(q)\sum_{k=2}^{k=\infty}\frac{q^{2k}}{1-q^k} \qquad (41)$$

Now isolating the summation, we have:

$$\sum_{k=2}^{k=\infty}\frac{q^{2k}}{1-q^k} = \frac{1}{\ln(q)}(\psi_q(2) + \ln(1-q)) - \frac{q^2}{1-q} \qquad (42)$$

Being $\Gamma_q(z)$ the q-gamma function [10].

One of the properties of q-polygamma function $\psi_k(z)$ is that is part of the solution of the following summation [2]:

$$\sum_{k=1}^{k=\infty}\frac{q^{k}}{1-q^k} = \frac{\psi_q(1) + \ln(1-q)}{\ln(q)} \qquad (43)$$

We can see that this summation is very similar to the one in (34) but there are two main differences.

The first one is that element in the numerator (first element of each series) is the one squared instead of the one that starts as the power of one.

But we can solve that doing:

$$\begin{aligned}&\sum_{k=2}^{k=\infty}\frac{(e^{-2\cdot c-2j\omega})^k}{1-(e^{-c-j\omega})^k} = \sum_{k=2}^{k=\infty}\left(\frac{(e^{-c-j\omega})^k}{1-(e^{-c-j\omega})^k} - e^{-c-j\omega}\right) \\ &= \sum_{k=2}^{k=\infty}\frac{(e^{-c-j\omega})^k}{1-(e^{-c-j\omega})^k} - \sum_{k=2}^{k=\infty}e^{-c-j\omega} = \sum_{k=2}^{k=\infty}\frac{(e^{-c-j\omega})^k}{1-(e^{-c-j\omega})^k} - \frac{e^{-2c-2j\omega}}{1-e^{-c-j\omega}} \qquad (44)\end{aligned}$$

The second issue is that the summation starts in k=2 instead of in k=1. This can be solved in a similar way -starting in k=1 and subtracting the first element):

$$\sum_{k=2}^{k=\infty}\frac{(e^{-2\cdot c-2j\omega})^k}{1-(e^{-c-j\omega})^k}=\sum_{k=2}^{k=\infty}\frac{(e^{-c-j\omega})^k}{1-(e^{-c-j\omega})^k}-\frac{e^{-2c-2j\omega}}{1-e^{-c-j\omega}}=$$
$$\sum_{k=1}^{k=\infty}\frac{(e^{-c-j\omega})^k}{1-(e^{-c-j\omega})^k}-\frac{(e^{-c-j\omega})^1}{1-(e^{-c-j\omega})^1}-\frac{e^{-2c-2j\omega}}{1-e^{-c-j\omega}}=$$

$$\sum_{k=1}^{k=\infty}\frac{(e^{-c-j\omega})^k}{1-(e^{-c-j\omega})^k}-\frac{e^{-c-j\omega}+e^{-2c-2j\omega}}{1-e^{-c-j\omega}} \quad (45)$$

Now yes:

$$\sum_{k=2}^{k=\infty}\frac{(e^{-2\cdot c-2j\omega})^k}{1-(e^{-c-j\omega})^k}=\sum_{k=1}^{k=\infty}\frac{(e^{-c-j\omega})^k}{1-(e^{-c-j\omega})^k}-\frac{e^{-c-j\omega}+e^{-2c-2j\omega}}{1-e^{-c-j\omega}} \quad (46)$$

The first element of the second right hand above has the form of the left-hand side of the commented equation (34):

$$\sum_{k=1}^{k=\infty}\frac{q^k}{1-q^k}=\frac{\psi_q(1)+\ln(1-q)}{\ln(q)} \quad (47)$$

With:

$$q=e^{-c-j\omega} \quad (48)$$

So, coming up to equation (34), we have:

$$F(\omega)=\sum_{k=2}^{k=\infty}\frac{(e^{-2\cdot c-2j\omega})^k}{1-(e^{-c-j\omega})^k}$$
$$=\frac{\psi_q(1)+\ln(1-e^{-c-j\omega})}{\ln(e^{-c-j\omega})}-\frac{e^{-c-j\omega}+e^{-2c-2j\omega}}{1-e^{-c-j\omega}} \quad (49)$$

This means, we can write the equation (34) not using summations but just with functions:

$$F(\omega)=\frac{\psi_q(1)+\ln(1-e^{-c-j\omega})}{\ln(e^{-c-j\omega})}-\frac{e^{-c-j\omega}+e^{-2c-2j\omega}}{1-e^{-c-j\omega}} \quad (50)$$

10. Using Parseval's theorem when (m+1) tends to infinity

Now, as in chapter 6 we want to calculate the following expression:

$$\sum_{n=-\infty}^{n=\infty}f(n)\cdot g(n)=\sum_{n=4}^{n=(m+1)(m+1)}f(n)\cdot g(n) \quad (51)$$

If the result is different from zero, it means that p is composite, as it means that:

$$g(n)=\delta(n-p) \quad (52)$$

g(n) has found another delta/s in the same position for the function f(n) defined before, that has only deltas in composite number positions.

We can perform this operation using the Parseval theorem as we made in chapter 6.

$$\sum_{n=-\infty}^{n=\infty} f(n)\cdot g^*(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} F(\omega)\cdot G^*(\omega)\,d\omega \qquad (53)$$

Being he asterisk* means the complex conjugate. As in point 5, the functions in the left hand-side are real so:

$$g^*(n) = g(n) \qquad (54)$$

For the righthand side, yes, we have complex functions. But as all of them are exponential based, the conjugate is just to change the sign of the imaginary part of the exponent. This is immediate from Euler's formula [9].

$$G(\omega) = e^{-pj\omega} \qquad (55)$$

$$G^*(\omega) = e^{pj\omega} \qquad (56)$$

So, coming back, we can operate now the integral but using the new F(ω) and G(ω) calculated in chapters 8 and 9.

$$\begin{aligned} q_2 = \sum_{n=-\infty}^{n=\infty} f(n)\cdot g^*(n) = \sum_{n=-\infty}^{n=\infty} f(n)\cdot g(n) &= \frac{1}{2\pi}\int_{-\pi}^{\pi} F(\omega)\cdot G^*(\omega)\,d\omega \\ &= \frac{1}{2\pi}\int_{-\pi}^{\pi}\left(\sum_{k=2}^{k=\infty}\frac{(e^{-2\cdot c-2j\omega})^k}{1-(e^{-c-j\omega})^k}\right)\cdot e^{pj\omega}d\omega \\ &= \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{pj\omega}\left(\sum_{k=2}^{k=\infty}\frac{(e^{-2\cdot c-2j\omega})^k}{1-(e^{-c-j\omega})^k}\right)d\omega \\ &= \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{pj\omega}\left(\frac{\psi_q(1)+\ln(1-e^{-c-j\omega})}{\ln(e^{-c-j\omega})}\right. \\ &\left. -\frac{e^{-c-j\omega}+e^{-2c-2j\omega}}{1-e^{-c-j\omega}}\right)d\omega \qquad (57) \end{aligned}$$

So, summing up:

$$q_2 = \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{pj\omega}\left(\sum_{k=2}^{k=\infty}\frac{(e^{-2\cdot c-2j\omega})^k}{1-(e^{-c-j\omega})^k}\right)d\omega \qquad (58)$$

$$q_2 = \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{pj\omega}\left(\frac{\psi_q(1)+\ln(1-e^{-c-j\omega})}{\ln(e^{-c-j\omega})} - \frac{e^{-c-j\omega}+e^{-2c-2j\omega}}{1-e^{-c-j\omega}}\right)d\omega \qquad (59)$$

As you can see in equation (59) we have gotten rid of the summation. And it is the formula, that we will use whenever possible. But we will keep an eye in equation (58) also, as it is the only one that can be used in some cases (even if the computation time is higher due to the summation). For example, the program Mathematica does include the q-

polygamma function in its library, but Matlab/Octave does not. And also, we will see later that, for example, to calculate the factors of a number we cannot use a simple equation as (59) and there is no other way as using a similar one to (58) -one that includes the summation-.

We will call q_2 the result of above expression. If q_2 is zero, p is prime, if q_2 is different from zero, q_2 has information regarding the number of factors.

But now, this information is not as immediate as in chapter 7. Take into account that all the deltas in (n-p) have been multiplied by e^{-k1k2c} which in fact is the same as e^{-pc} in this case. The reason for this, is that the integral has found that in (n-p) we have deltas of the type $(n-k_1k_2)$, so this means that $p=k_1k_2$. Only when there is no $k_1k_2=p$, the result of the integral is zero, meaning p is prime.

So, to revert the multiplication by e^{-pc} that has been done in the process, we have to multiply by the inverse (e^{pc}) to get the number q equivalent we commented in chapter 7.

$$q = e^{pc} q_2 \quad (60)$$

And now, yes, q is exactly the same as in chapter 7. Meaning that all the study we have done regarding the relation of number of factors with q in point 7 is the same.

For the calculation of (60) you can use the program in Annex A1 in Mathematica that uses the expression (59) for the calculation of q_2 -it uses the q-polygmma function-. Or if you use Matlab/Octave you can use the program in Annex B2 (that uses the expression (58) for the calculation of q_2). Anyhow, both programs should give the same results.

Once you have calculated the value of q (the number of combinations of 2-factor products), the equation (23) is the same, this means:

$$q = (n_{f1} + 1)(n_{f2} + 1)(n_{f3} + 1) \dots (n_{fz} + 1) - 2 \quad (61)$$

So,

$$q = e^{pc} q_2 = (n_{f1} + 1)(n_{f2} + 1)(n_{f3} + 1) \dots (n_{fz} + 1) - 2 \quad (62)$$

I invite you to make some testing with the programs A1 and/or B2 to check results and even compare with brute-force testing.

Anyhow, remember that the pre-calculation is repeated every time you open the program because it is not stored. The important thing is that if you stored the result of the pre-calculation, it would not be needed to be repeated anytime, making the real time of calculation very small as you can check with the programs.

It is important to remark that the pre-calculation precision should be decided carefully (this means the number of steps n or n4 of the integral -the higher the better- and the lower

in absolute value of c or c2 -the lower the better but different from zero-).

Once you have decided the precision and you perform the pre-calculation -even taking days if you want, or various computers-, you do not have to repeat it anymore. And the time of calculation for all the numbers in the range of validity for that precision will be very small -meaning linear with the size of the number, not exponential-.

11. Sum of the exponential of the factors of the number p

In the chapter 2 we have put all the Dirac deltas as unitary Dirac deltas. And in the chapter 8 we have used an exponential that includes the product of the factors (but this does not add information, as we know that the product is already p). But we can introduce more information, if instead of putting unitary Dirac deltas of the product of the factors, we introduce the deltas multiplied by a number that represents the sum of the value of the factors. This means, in the position of each delta will be adding up deltas which magnitude is related to the sum of the factors. When we recover the information of the magnitude of delta in one position, we will have not only the product (that we know is p) but also the sum of the factors. We will see that for the special case of semiprime numbers, having the information of the product and the sum of the two factors, it is immediate to obtain these two factors using a simple equation.

We can check an example (see figure 7). We can multiply the delta by an exponential of the sum of the two factors. As before (in the chapter 8), we will also multiply by the factor -c to get a convergent sum in the end (being c a non-zero positive real number). We will see from experience that a value of c in the range of the square root of p is sufficient.

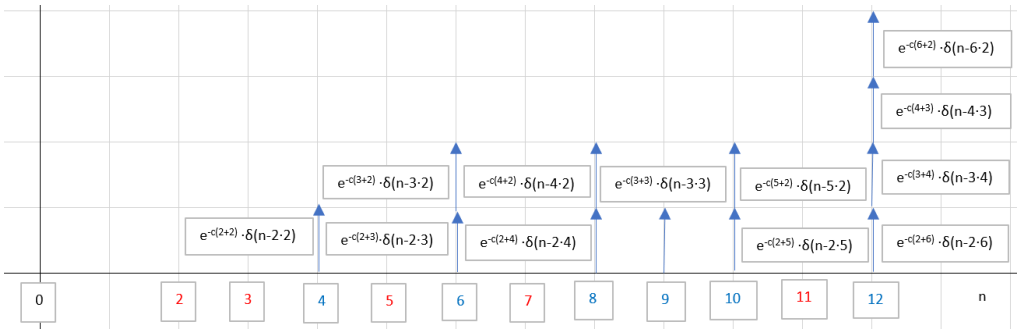


Fig.7

So, in this case:

$$f(n)=\sum_{k_2=2}^{k_2=m+1}\sum_{k_1=2}^{k_1=m+1} e^{-c(k_1+k_2)}\delta(n-k_1\cdot k_2)=\sum_{k_2=2}^{k_2=m+1}(e^{-c(2+k_2)}\delta(n-2\cdot k_2)$$
$$+e^{-c(3+k_2)}\delta(n-3\cdot k_2)+e^{-(4+k_2)}\delta(n-4\cdot k_2)$$
$$+\cdots e^{-c((m+1)+k_2)}\delta(n-(m+1)\cdot k_2))=$$
$$=e^{-c(2+2)}\delta(n-2\cdot 2)+e^{-c(2+3)}\delta(n-2\cdot 3)+\cdots e^{-c(2+(m+1))}\delta\big(n-2\cdot(m+1)\big)+$$
$$+e^{-c(3+2)}\delta(n-3\cdot 2)$$
$$+e^{-c(3+3))}\delta(n-3\cdot 3)\ \ldots\ e^{-c((m+1)+(m+1))}\delta\big(n-(m+1)$$
$$\cdot(m+1)\big)\qquad(63)$$

Now if we make (m+1) tend to infinity we have:

$$f(n)=\sum_{k_2=2}^{k_2=\infty}\sum_{k_1=2}^{k_1=\infty} e^{-c(k_1+k_2)}\delta(n-k_1\cdot k_2)=\sum_{k_2=2}^{k_2=m+1}(e^{-c(2+k_2)}\delta(n-2\cdot k_2)$$
$$+e^{-c(3+k_2)}\delta(n-3\cdot k_2)+e^{-c(4+k_2)}\delta(n-4\cdot k_2)+\cdots 0)=$$
$$=e^{-c(2+2)}\delta(n-2\cdot 2)+e^{-c(2+3)}\delta(n-2\cdot 3)+\cdots 0+$$
$$+e^{-c(3+2)}\delta(n-3\cdot 2)\ +e^{-c(3+3)}\delta(n-3\cdot 3)\ \ldots\ 0\qquad(64)$$

The zeroes in the end of the series appear because the exponential $e^{-c(k1+k2)}$ tends to zero as k_1 and k_2 tend to infinity (and we have also defined c as a real number higher than zero). Remind that the previous limit for k_1 and k_2 was (m+1) before, and now (m+1) tends to infinity.

We can calculate the Fourier transform as in chapter 5 like this:

$$F(\omega)=\sum_{k_1=2}^{k_1=m+1}\sum_{k_2=2}^{k_2=m+1} e^{-c(k_1+k_2)}e^{-k_1\cdot k_2 j\omega}=$$
$$=e^{-c(2+2)}e^{-2\cdot 2j\omega}+e^{-c(2+3)}e^{-2\cdot 3j\omega}+\cdots+0+$$
$$+e^{-c(3+2)}e^{-3\cdot 2j\omega}+e^{-c(3+3)}e^{-3\cdot 3j\omega}+\cdots+0+$$
$$+\cdots\qquad+\cdots\qquad+\cdots+\cdots\qquad+$$
$$+0+0\ldots+0\qquad(65)$$

And now, we can also simplify it as geometric sums:

$$F(\omega)=e^{-c(2+2)}e^{-2\cdot 2j\omega}\frac{1}{1-e^{-c}e^{-2j\omega}}+$$
$$+e^{-c(2+3)}e^{-2\cdot 3j\omega}\frac{1}{1-e^{-c}e^{-3j\omega}}+$$
$$+e^{-c(2+4)}e^{-2\cdot 4j\omega}\frac{1}{1-e^{-c}e^{-4j\omega}}+$$
$$+\cdots+$$
$$+0=$$
$$=\sum_{k=2}^{k=\infty}e^{-c(2+k)}e^{-2kj\omega}\frac{1}{1-e^{-c}e^{-kj\omega}}=\sum_{k=2}^{k=\infty}\frac{e^{-2c}(e^{-c-2j\omega})^k}{1-(e^{-c-j\omega})^k}\qquad(66)$$

Regarding the function g(n), it is exactly the same as in previous chapters. So, we will

follow the same process:

$$g(n) = \delta(n-p) \qquad (67)$$

The Fourier transform G(ω) of g(n) would be:

$$g(n) = \delta(n-p) \rightarrow G(\omega) = e^{-pj\omega} \qquad (68)$$

So now, we can apply the Parseval's theorem [6] [7] again (this time to (66) and (68)) and see the result:

$$q_3 = \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{pj\omega}\left(\sum_{k=2}^{k=\infty} e^{-c(2+k)}e^{-2kj\omega}\frac{1}{1-e^{-c}e^{-kj\omega}}\right)d\omega \qquad (69)$$

We call q_3 this value that has this information regarding the factors. We will see how to use it in the next chapter.

One important thing to remark is that the summation inside the integral does not depend on p. this means I can pre-calculate that summation -with the precision I decide. And then I can apply the integral to the different numbers p I need -not needing to make the summation again-. This means using a high computation time once to make the summation. But then, the computation time is much lower for every p I want to check.

When I say the precision I decide, there are two factors I can play with. The parameter c, the lower the better -but always higher than zero-. And the second parameter is the number of elements in the summation I want to use (we will call it n or n4 in the programs). The higher this number, the most precise the summation is.

12. Discovering the factors of a semiprime (pq-number)

There are two cases where with all this information, it is immediate to calculate the factors of a number. One of the cases is trivial. When using equations (58,59,60) and obtaining q=1, it means that p is a perfect square, so you can just make the square root to get it.

But there is another special case. When q=2, it means that the number is composite but only by two factors (it is a semiprime or pq-number). This case is used for example in RSA encryption [11].

So, for this case when p is a semiprime number, we can calculate the two factors of p the following way. We will call a and b these unknown factors of p (that it is semiprime in this case):

-First you calculate the q (58,59,60). And if q= 2, this means that the number is a semiprime (aka pq- number).
-Then you calculate the q_3 according (69). In the case of a semiprime number q_3 will be 2 times the exponential $e^{-c(k1+k2)}$.
-As the delta is located in the position of the factors of p, k_1 and k_2 are, for this case, a and b, the factors of p. So q_3 is in fact two times $e^{-c(a+b)}$.
-We will call d the sum of the two factors (a+b). And we will calculate it (d) according to the following equation:

$$d = (a+b) = -\frac{1}{c} Ln\left(\frac{q_3}{2}\right) \quad (70)$$

Being c the calibration factor we have chosen before, a positive real number different from zero.

If you want to know how we have arrived to equation (70). It is the following way:

$$q_3 = 2e^{-c(a+b)} = 2e^{-c\cdot d} \quad (71)$$

So, dividing by 2:

$$\frac{q_3}{2} = e^{-c\cdot d} \quad (72)$$

Taking natural logarithm:

$$Ln\left(\frac{q_3}{2}\right) = -c \cdot d \quad (73)$$

Isolating d:

$$d = (a+b) = -\frac{1}{c} Ln\left(\frac{q_3}{2}\right) \quad (74)$$

-And, then we solve the following equation system (being a and b the unknown factors of p) and d is the known number calculated before (the sum of the two factors:

$$a \cdot b = p \quad (75)$$
$$a + b = d \quad (76)$$

You have the solution:

$$b = \frac{d + \sqrt{d^2 - 4p}}{2} \quad (77)$$

$$a = \frac{p}{b} = d - b = \frac{d - \sqrt{d^2 - 4p}}{2} \quad (78)$$

So, you can get, immediately the factors a and b. This, of course is tricky. As for the calculation of q_3 you have used exponentials, summations and numerical integrals. So, the complete process, as commented, is at this stage is inefficient.

-So, summing up the process to calculate the two factors of a semiprime number are:

-Calculate q_3:

$$q_3 = \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{pj\omega} \left(\sum_{k=2}^{k=\infty} e^{-c(2+k)} e^{-2kj\omega} \frac{1}{1 - e^{-c} e^{-kj\omega}} \right) d\omega \quad (79)$$

-Calculate d:

$$d = (a+b) = -\frac{1}{c} Ln\left(\frac{q_3}{2}\right) \quad (80)$$

-Then, it is immediate to calculate b and a:

$$b = \frac{d + \sqrt{d^2 - 4p}}{2} \quad (81)$$

$$a = d - b = \frac{d - \sqrt{d^2 - 4p}}{2} \qquad (82)$$

In the Annex A2 you can find a Mathematica program that performs this calculation. An in the Annex B3 a Matlab/Octave program that performs it.

Please, take into account that in the program, some "shortcuts" are used but having in the end the same equations. For example, for the calculation d instead of dividing by 2 the result q (the integral which is divided by 2π), the integral is divided directly by 4π in the first place.

Also, take into account that in the program of Matlab/Octave in Annex B3 to perform an exact calculation you need to put the variable multic=50. If you put multic=30, you will perform an approximate calculation but having in the end the exact result (see next chapter, to see why this is really possible).

13. Calculating an approximation of the integral instead of the exact value

In general, we have seen that the brute force checking is quicker than calculating using the integral. This applies both to check if a number is prime or to calculate the two factors of a semiprime number.

But we have two aces in the sleeve:

-The first one, as we have commented, is that we can make a pre-calculation of the components of the integral not depending on the prime or semiprime number we are checking. This means, we can use a number of computers during certain time (even months) to make this pre-calculation. And then, to use just some minutes for whatever number we decide (in a certain predefined range) to check it or factorize it.

-But we have a second trick. This one, we can use only to factorize semiprimes, not to check if a number is prime or not. To know if a number is prime the result of the integral must be zero or a very small number. To check this, we need a very precise result to avoid errors. But to make the factorization **we do not need a precise result** of the integral. As the result of the integral is the exact sum of the factors, what we can do is to obtain a result near the exact sum and to use the equations (77)(78) to start obtaining possible candidates of the factors. Then we can check if they are correct candidates, just multiplying them. If they are not correct, we go to another possible sum.

We can see it with one example. Imagine we want to factorize the number 221. We can use the integral to obtain a perfect sum of the factors. In this case, we will obtain 30. And using equations (77)(78), we will see immediately that the factors are 13 and 17.

But we can another thing. We can calculate the integral very roughly (to have a quicker result) and for example we obtain the number 28 instead of 30. If we consider that the real sum is in the range of 28±3 (we will see how to calculate this ±3). We can apply the equations (77)(78) considering the sums from 25 to 31 until we get the values of the factors that are correct.

As commented, we can know which factors are correct, just performing the multiplication. That is the reason we can get a not perfect result of the integral and start iterating on the surroundings. In the case of trying to know if a number is prime or not, there is no way to make an after checking -apart from brute force of course-. This is the reason the integral has to be exact in the case of primality checking, but not needed in the case of factorization of a semiprime number.

You can check this with the Mathematica program in Annex A3. The surroundings of the exact result are considered in the domain d±4*Ln(d). This works as far as I have checked. If we have to go to other values as d±√(d), probably to make this approximation would not be useful. Anyhow, more checking is needed to check this and if possible with big numbers.

If you use Matlab/Octave, you can use the program in Annex B3. You can use "multic=30" and the result of the integral will not be correct (just an approximation). Then the program looks in the domain of d±4*Ln(d) for the exact solution and shows it.

14. Testing the factorization of prime numbers

We will use a computer OMEN by HP Laptop 17-an0xx Intel Core i7-7700HQ CPU 2.80 GHz 32.0 GB GeForce GTX 1070 8.0GB Windows 10 Home 20H2.

Wolfram Mathematica Version 12.1 and Matlab R2020b.

I will make all the examples using Mathematica 12.1, program in Annex A3..

When using Mathematica to calculate the examples (the computer is working in parallel with other programs) the CPU is between 25-30% 3.52 GHz, the memory 13/31.9 GB and the GPU 1-6% approx.

-Semiprime 15:

```
Prework done in 1.64063 s
Factorization of semiprime number 15. using 450 steps
Time of calculation of the approximation (once prework is done):0. s
Integral result = 0.122013 - 1.86502×10^-16 i
Value of d = 8.14731
Approximate sum of factors 8
Approximate First factor is 3
Approximate Second factor is 5
Time for exact calculation in the neighbouring of the approximate solution(+-Log[d]) 0. s
If first factor=1, no solution has been found
Exact First factor is 1
Exact Second factor is 15
```

-Semiprime 39:

```
Prework done in 10.9844 s
Factorization of semiprime number 39. using 1170 steps
Time of calculation of the approximation (once prework is done):0.015625 s
Integral result = 0.0712275 - 2.82991×10^-15 i
Value of d = 16.4985
Approximate sum of factors 16
Approximate First factor is 3
Approximate Second factor is 13
Time for exact calculation in the neighbouring of the approximate solution(+-Log[d]) 0. s
If first factor=1, no solution has been found
Exact First factor is 3
Exact Second factor is 13
```

-Semiprime 69:

```
Prework done in 34.0938 s
Factorization of semiprime number 69. using 2070 steps
Time of calculation of the approximation (once prework is done):0. s
Integral result = 0.0372466 + 5.62957×10^-15 i
Value of d = 27.3304
Approximate sum of factors 27
Approximate First factor is 3
Approximate Second factor is 24
Time for exact calculation in the neighbouring of the approximate solution(+-Log[d]) 0. s
If first factor=1, no solution has been found
Exact First factor is 3
Exact Second factor is 23
```

-Semiprime 115:

```
Prework done in 89.75 s
Factorization of semiprime number 115. using 3450 steps
Time of calculation of the approximation (once prework is done):0.015625 s
Integral result = 0.0665933 + 2.04449×10^-15 i
Value of d = 29.0524
Approximate sum of factors 29
Approximate First factor is 5
Approximate Second factor is 24
Time for exact calculation in the neighbouring of the approximate solution(+-Log[d]) 0. s
If first factor=1, no solution has been found
Exact First factor is 5
Exact Second factor is 23
```

-Semiprime 253:

```
Prework done in 449.891 s
Factorization of semiprime number 253. using 7590 steps
Time of calculation of the approximation (once prework is done):0.046875 s
Integral result = 0.110617 + 5.40748×10^-14 i
Value of d = 35.0199
Approximate sum of factors 35
Approximate First factor is 10
Approximate Second factor is 25
Time for exact calculation in the neighbouring of the approximate solution(+-Log[d]) 0. s
If first factor=1, no solution has been found
Exact First factor is 11
Exact Second factor is 23
```

-Semiprime 187:

```
Prework done in 251.344 s
Factorization of semiprime number 187. using 5610 steps
Time of calculation of the approximation (once prework is done):0.015625 s
Integral result = 0.12188 + 1.09436×10^-14 i
Value of d = 28.7815
Approximate sum of factors 29
Approximate First factor is 10
Approximate Second factor is 19
Time for exact calculation in the neighbouring of the approximate solution(+-Log[d]) 0. s
If first factor=1, no solution has been found
Exact First factor is 11
Exact Second factor is 17
```

-Semiprime 323:

```
Prework done in 738.438 s
Factorization of semiprime number 323. using 9690 steps
Time of calculation of the approximation (once prework is done):0.046875 s
Integral result = 0.12748 - 1.22974×10^-14 i
Value of d = 37.019
Approximate sum of factors 37
Approximate First factor is 14
Approximate Second factor is 23
Time for exact calculation in the neighbouring of the approximate solution(+-Log[d]) 0. s
If first factor=1, no solution has been found
Exact First factor is 17
Exact Second factor is 19
```

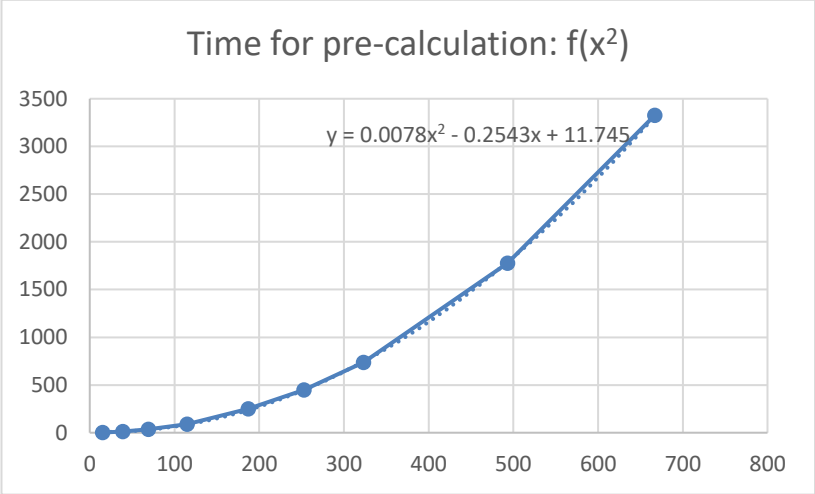
-Semiprime 493:

```
Prework done in 1776.52 s
Factorization of semiprime number 493. using 14790 steps
Time of calculation of the approximation (once prework is done):0.0625 s
Integral result = 0.118364 - 3.49143×10^-14 i
Value of d = 47.3823
Approximate sum of factors 47
Approximate First factor is 16
Approximate Second factor is 31
Time for exact calculation in the neighbouring of the approximate solution(+-Log[d]) 0. s
If first factor=1, no solution has been found
Exact First factor is 17
Exact Second factor is 29
```

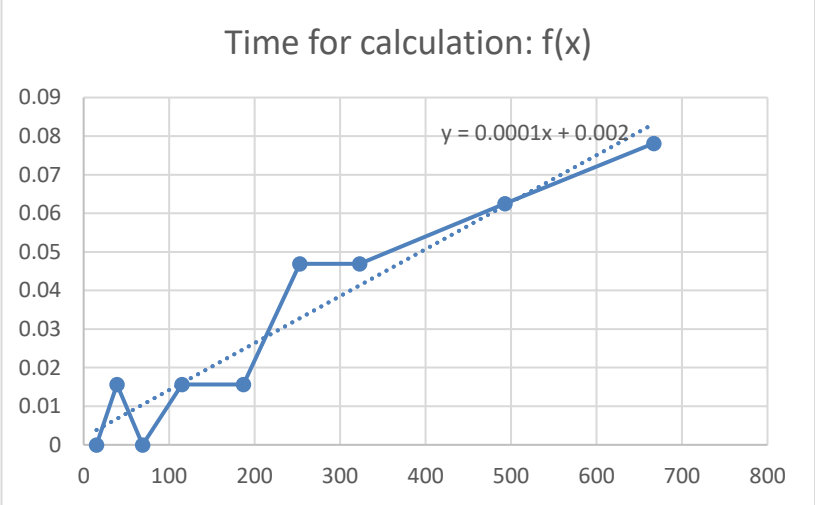
-Semiprime 667:

```
Prework done in 3326.47 s
Factorization of semiprime number 667. using 20010 steps
Time of calculation of the approximation (once prework is done):0.078125 s
Integral result = 0.125825 - 2.24083×10^-13 i
Value of d = 53.5346
Approximate sum of factors 54
Approximate First factor is 19
Approximate Second factor is 35
Time for exact calculation in the neighbouring of the approximate solution(+-Log[d]) 0. s
If first factor=1, no solution has been found
Exact First factor is 23
Exact Second factor is 29
```

We can see that the time for pre-calculation depends on x^2:



But also, that the time for factorization of a specific semiprime -once the pre-calculation has been done is linear depending on x-.



Anyhow, this is not exactly true, as the calculations have done using a fixed number of decimals (precision of the calculations). As we increase the numbers, the precision of the calculation should be increased. Leading very probably to an exponential or a very high grade polynomial result.

Anyhow, considering these numbers true, we can see that for a typical 2^{256} number we would need:

$$y = 0.0078 \cdot (2^{256})^2 - 0.2543 \cdot 2^{256} + 11.745 = 1.0458 \cdot 10^{152} s$$

This is $3.314 \cdot 10^{144}$ years for the pre-calculation.

Once this pre-calculation has been done, we would need:

$$y = 0.0001 \cdot 2^{256} + 0.002 = 1.15792 \cdot 10^{73} s$$

This is $3.66923 \cdot 10^{65}$ years.

Of course, these times would be shorter for lower numbers, better computers (that uses powerful GPUs for the calculation) and many computers in parallel. Anyhow, we are very far for trying to get practical numbers.

It is also to be noted, that when I have performed similar tests with Matlab, the results of computation time also improve (probably the program in Annex B3 is more optimized than the programs in A2 and A3). Anyhow, there are other issues with Matlab, as not having the q-polygamma function integrated, for example. It is not the intention of this paper to compare both applications. Both are incredibly good for the use.

15. Conclusions

In this paper, it has been shown a way of checking if a number is prime or not, using numerical integration in the complex plane. One of the advantages of the way shown here, compared to previous papers [4], is that here we can make a pre-calculation of the part of the integral that does not depend on the number to be checked. This way, using only once a lot of computation time to create the necessary tables, we can make primality test for different numbers in very short time.

The same concept has been applied to factorize semiprime numbers (pq-numbers). We have shown a way of make this factorization in polynomial time using a numeric integral. And we have taken into account two aspects:

- It is necessary a pre-calculation for the parts of the integral not depending in the semiprime number (polynomial quadratic time). Once this pre-calculation is done, we can check the semiprime number in a computation time depending linearly in the number.
- We can use a second “trick”. Instead of calculating the integral very precisely, we can obtain an approximation. And then, to work with the numbers near the approximation until we get the correct answer. This reduces the time of calculation drastically.

Bilbao, 7th April 2021 (viXra-v1).

16. Acknowledgements

To Guillermo, Adso, my family and friends.

17. References

[1] https://en.wikipedia.org/wiki/Prime_number

[2] https://mathworld.wolfram.com/q-PolygammaFunction.html

[3] https://en.wikipedia.org/wiki/Semiprime

[4]https://www.researchgate.net/publication/329890939_How_to_Check_If_a_Number_Is_Prime_Using_a_Finite_Definite_Integral

[5] https://en.wikipedia.org/wiki/Fourier_transform

[6]https://en.wikipedia.org/wiki/Parseval%27s_theorem#:~:text=In%20mathematics%2C%20Parseval's%20theorem%20usually,the%20square%20of%20its%20transform.

[7] http://www.astro.rug.nl/~vdhulst/SignalProcessing/Hoorcolleges/college03.pdf

[8] https://en.wikipedia.org/wiki/Geometric_series

[9] https://en.wikipedia.org/wiki/Euler%27s_formula

[10] https://en.wikipedia.org/wiki/Gamma_function

[11] https://en.wikipedia.org/wiki/RSA_(cryptosystem)

A1. Annex A1. Mathematica program to calculate integral (59)

With this program of Mathematica. you can calculate the integral (59) using the q-polygamma function:

```
p2=Input["Input a number"];
p2=N[p2];

(* the lower is c2, the more exact is the result but with more computation time *)

c2=1/p2;

pi2=N[Pi];
Remove[func];
Remove[func2];

Remove[y];

Remove[y2];

func[p_,c_,w_]:=Exp[p*(c+I*w)]*((QPolyGamma[2,Exp[-c-I*w]]+Log[1-Exp[(-c-I*w)]])/Log[Exp[-c-I*w]]-
Exp[(-c-I*w)]^2/(1-Exp[(-c-I*w)]));
func2[c_,w_]:=((QPolyGamma[2,Exp[-c-I*w]]+Log[1-Exp[(-c-I*w)]])/Log[Exp[-c-I*w]]-Exp[(-c-I*w)]^2/(1-
Exp[(-c-I*w)]));
(*In func 3 I remove the exp (p*c) because it is constant and I re-introduce it in the final expression *)

func3[p_,c_,w_]:=Exp[p*(I*w)];
(* Integration using Mathematica function *)

tiempo=Timing[NIntegrate[func[p2,c2,w],{w,-pi2,pi2}]/(2*pi2)];
Print["Number ",p2," The result of the integration (using the Mathematica internal numerical integration function
and the q-polygamma function) is q=",Last[tiempo]," Time ",First[tiempo]," s"];

(* Integration using a "manual" loop but with prework for the parts of the integrand that are not depending on p2
*)

(* the higher n, the more exact os the integration with calculated with "manual loop" nut with    more computation
time*)

n=Round[p2^3];


Array[y,0,n];
y[-1]=0;

(* I could put liminf=0 and then multiply by two -taking the real part, the integral is symmetric- but I leave it in -
pi2, to receive the information also of the imaginary part that should be always zero (or very close to zero) if the
result is ok*)

liminf=-pi2;
limsup=pi2;
n2=(limsup-liminf)/n;
n3=liminf;
tiempo=Timing[For[i=0,i<n+1,i++,

      y[i]=func2[c2,n3];
      n3=n3+n2;]];
Print[""];
Print["Prework done in ",First[tiempo]," s"];

Array[y2,0,n];
y2[-1]=0;
```

```
n3=liminf;
tiempo=Timing[For[i=0,i<n+1,i++,

      y2[i]=y2[i-1]+func3[p2,c2,n3]*y[i];
      n3=n3+n2;]];

solman=y2[n]*n2*Exp[p2*c2]/(2.*pi2);
Print["Numeric Integration for the number ",p2," using a loop of ",n," steps, once the prework has been done."];
Print["Result q=",solman," Time for the numeric integration (using a manual loop and q-polygamma function) ",
First[tiempo]," s, once the prework has been done."];
```

A2. Annex A2. Mathematica program to find the factors of a semi-prime number (using exact integral (69))

With this program of Mathematica. you can find the two factors of a semiprime number using integral (69).

The first step is a pre-calculation that could be stored in a database and not done again (it does not depend in the number to check, just in the precision we need -the order of magnitude of the numbers we will need to factorize-).

The second step is the calculation that depends in the number and you can see that the computation time is very low compared to the previous one. Probably higher than brute checking for low numbers, but to be checked if better in very big numbers.

In Annex A3, you will see even a better way to perform the calculation.

```
SetOptions[EvaluationNotebook[],CellContext->Notebook];

k2=0.;
pi2=N[Pi];



p2=Input["Input a semiprime number"];
p2=N[p2];

c2=-1/Sqrt[p2];


f[c_,w_,n_]:=Exp[c*(n+2)]*Exp[2*n*I*w]/(1-Exp[c]*Exp[n*I*w]);



func2[c_,w_]:=((QPolyGamma[2,Exp[c+I*w]]+Log[1-Exp[(c+I*w)]])/Log[Exp[c+I*w]]-Exp[(c+I*w)]^2/(1-
Exp[(c+I*w)]));


func3[p_,k_]=Exp[-p*k];

func4[p_,w_]=Exp[-p*(I*w)];

n4=Round[p2*50];
n=Round[p2*50];

Array[y,0,n];


(* I could put liminf=0 and then multiply by 2 (integral symmetric). I do not do it to keep imaginary part as a
checking -should be close to zero if everything ok- *)
liminf=-pi2;
limsup=pi2;
w2=(limsup-liminf)/n;
w3=liminf;
tiempo=Timing[For[i2=0,i2<n+1,i2++,
```

```
        y4=0.;
        For[i=2,i<n4+1,i++,y4=N[y4+f[c2,w3,i]]];
        y[i2]=y4;

        w3=w3+w2;

        ]];

(*End of i2 loop*)

Print["Prework done in ",First[tiempo]," s"];


w3=liminf;

y5=0.;

tiempo=Timing[For[i3=0,i3<n+1,i3++,

          y5=N[y5+func4[p2,w3]*y[i3]];
          w3=w3+w2; ];];



Print["Factorization of semiprime number ",p2," using ",n," steps"];
Print["Time of calculation (once prework is done):",First[tiempo]," s"];

d=y5*w2*(func3[p2,k2]/(4*pi2));

Print["Integral result = ",d];

d= Log[d]/c2;



Print["Value of d = ",d];
d=Round[d];
Print["Sum of factors ",d];
b=(d+Sqrt[d^2-4*p2])/2;
a=d-b;
Print["First factor is ",Round[a]];
Print["Second factor is ",Round[b]];
```

A3. Annex A3. Mathematica program to find the factors of a semiprime number (using an approximation of the integral (69))

With this program of Mathematica. you can find the two factors of a semiprime number using integral (69). It is similar to the one of A2, but with one difference.

In A2 we try to get the most precise possible result of the integral (69) to get an exact sum of the factors. And subsequently to obtain immediately after, the value of the factors.

Here, we will have a different approach. Instead of trying to obtain the perfect result of the integral (increasing the computation), we will get an approximate result (with less computation). Then, knowing that the real result should be near it, we calculate the possible factors near the approximate solution until we find the correct ones. This approach is much quicker than in A2. But it is still perfectly valid, as we can check when the factors are ok (no possibility of incorrect results, in the worst case -but it has not happened in all the tests- no result at all, but not incorrect). In case of no result, we should increase the precision of the integral or increase the range of possible solutions to be checked near the approximate result.

Also, you will see that after the calculation of the first number, you will be asked to input a second number. For this second number, the pre-calculation will not be repeated, and you will see the difference in time needed (only post-calculation linear time is needed, not the quadratic for the precalculation).

```
SetOptions[EvaluationNotebook[],CellContext->Notebook];

k2=0.;
pi2=N[Pi];



p2=Input["Input a semiprime number"];
p2=N[p2];

c2=-1/Sqrt[p2];


f[c_,w_,n_]:=Exp[c*(n+2)]*Exp[2*n*I*w]/(1-Exp[c]*Exp[n*I*w]);



func2[c_,w_]:=((QPolyGamma[2,Exp[c+I*w]]+Log[1-Exp[(c+I*w)]])/Log[Exp[c+I*w]]-Exp[(c+I*w)]^2/(1-
Exp[(c+I*w)]));


func3[p_,k_]=Exp[-p*k];

func4[p_,w_]=Exp[-p*(I*w)];

(* As we look for approxiamte solution, the number of steps will only be 10 times the value of the number. For
the precalculation (can be done only one time not knowing the number to factorize) goes to the square *)
(*For the final calculation, once we know the number to factorize and we have done the precalculation the com-
putation is linear with the number *)
(* ñiradacapovt *)

n4=Round[p2*30];
n=Round[p2*30];

Array[y,0,n];


(* I could put liminf=0 and then multiply by 2 (integral symmetric). I do not do it to keep imaginary part as a
checking -should be close to zero if everything ok- *)
liminf=-pi2;
limsup=pi2;
w2=(limsup-liminf)/n;
w3=liminf;
tiempo=Timing[For[i2=0,i2<n+1,i2++,



    y4=0.;
    For[i=2,i<n4+1,i++,y4=N[y4+f[c2,w3,i]]];
    y[i2]=y4;

    w3=w3+w2;

    ]];

(*End of i2 loop*)

Print["Prework done in ",First[tiempo]," s"];


w3=liminf;

y5=0.;

tiempo=Timing[For[i3=0,i3<n+1,i3++,

    y5=N[y5+func4[p2,w3]*y[i3]];
    w3=w3+w2; ]];



Print["Factorization of semiprime number ",p2," using ",n," steps"];
Print["Time of calculation of the approximation (once prework is done):",First[tiempo]," s"];
```

```
d=y5*w2*(func3[p2,k2]/(4*pi2));

Print["Integral result = ",d];

d=Abs[d];
d= Log[d]/c2;



Print["Value of d = ",d];
d=Round[Abs[d]];
Print["Approximate sum of factors ",d];
b=(d+Sqrt[d^2-4*p2])/2;
a=d-b;
Print["Approximate First factor is ",Round[a]];
Print["Approximate Second factor is ",Round[b]];
Print "";

d2=Round[d-4*Log[d]-1];

a=1;

tiempo=Timing[For[i6=1,i6<(8*Log[d]+2),i6++,

      b=Round[((d2+i6)+Sqrt[(d2+i6)^2-4*p2])/2];



      If[Round[b*((d2+i6)-b)]==Round[p2],a=(d2+i6)-b,a=a]]];
Print["Time for exact calculation in the neighbouring of the approximate solution(+-Log[d]) ",First[tiempo]," s"]
Print["If first factor=1, no solution has been found"];
Print["Exact First factor is ",Round[a]];
Print["Exact Second factor is ",Round[p2/a]];


Print[""];


p2=Input["Input a semiprime number of the same order or lower than the previous one. This will be calculated with the prework already done, so you can see the difference"];



w3=liminf;

y5=0.;

tiempo=Timing[For[i3=0,i3<n+1,i3++,

      y5=N[y5+func4[p2,w3]*y[i3]];
      w3=w3+w2; ]];



Print["Factorization of semiprime number ",p2," using ",n," steps"];
Print["Time of calculation of the approximation (once prework is done):",First[tiempo]," s"];

d=y5*w2*(func3[p2,k2]/(4*pi2));

Print["Integral result = ",d];

d=Abs[d];
d= Log[d]/c2;



Print["Value of d = ",d];
d=Round[Abs[d]];
Print["Approximate sum of factors ",d];
b=(d+Sqrt[d^2-4*p2])/2;
a=d-b;
Print["Approximate First factor is ",Round[a]];
Print["Approximate Second factor is ",Round[b]];
```

```
Print "";

d2=Round[d-4*Log[d]-1];

a=1;

tiempo=Timing[For[i6=1,i6<(8*Log[d]+2),i6++,

      b=Round[((d2+i6)+Sqrt[(d2+i6)^2-4*p2])/2];


      If[Round[b*((d2+i6)-b)]==Round[p2],a=(d2+i6)-b,a=a]]];
Print["Time for exact calculation in the neighbouring of the approximate solution(+-Log[d]) ",First[tiempo]," s"]
Print["If first factor=1, no solution has been found"];
Print["Exact First factor is ",Round[a]];
Print["Exact Second factor is ",Round[p2/a]];
```

B1. Annex B1. Matlab/Octave program for expression (21)

With this program for Matlab/Octave you can check if a number is prime or not (and the number of factors that it has) using the expression (58) and compare the time of computation when using brute-force checking. You can check that brute-force checking is better. That is the reason the expression (58) is not sufficient, and we have to look for an improved way of calculation (from chapter 8 to 10 to get to expression (58)).

```
B=0;
p=input('Input a number : ');
%using numerical integration
t1=cputime;
m=fix((p+1)/2);
for k=2:m+1;
    fun=@(w)          exp(p.*1j.*w).*exp(-2.*k.*1j.*w).*(1-exp(-m.*k.*1j.*w))./(1-exp(-
k.*1j.*w));
    a=integral(fun,-pi,pi,'AbsTol',0.24);
    b=b+a;
end;
b=b/(2*pi);
disp('-------------------------------');
disp('Using numerical integration');
disp (num2str(b));
c=round(abs(b));
if c==0;

    disp([num2str(p),' is a prime number']);
else
    disp([num2str(p),' is a composite number and has ',num2strI,' permutations of 2
factors']);
end

t1=cputime-t1;
disp([num2str(t1),' seconds']);

%Brute force checking
t2=cputime;
disp('-------------------------------');
disp(['Using brute-force checking']);
if rem(p,2)==0;
    disp([num2str(p),' is a composite number and 2 is one of the factors']);
else;
    k=3;
    c=sqrt(p);
    while k<=c;
      if mod (p,k)==0;
       disp([num2str(p),' is a composite number and ',num2str(k),' is one of the fac-
tors']);
       k=c+10;
      else
          k=k+2;
      end
    end
    if k~=(c+10)
        disp([num2str(p),' is a prime number']);
    end
end
t2=cputime-t2;
disp([num2str(t2),' seconds'])
disp('-------------------------------');
```

```
disp(['Numerical integration is ',num2str(t1/t2), ' times slower']);
```

B2. Annex B2. Matlab/Octave program for expression (58)

With this program for Matlab/Octave you can check if a number is prime or not (and the number of factors that it has) using the expression (58). The difference with the expression (21) used in Annex B1 is that here, as the infinite sum has the last element=0, you can make a pre-calculation of the sum valid for all the numbers (with a certain precision). And once the pre-calculation is finished, the calculation for a specific number is much lower in computation time.

Apart from the input number, I have included more examples to confirm this is issue (you can check the real time of calculation once the pre-calculation has been done and it is not needed to be repeated anymore). If this pre-calculation were included in a data base, it would not need to be repeated anymore.

The higher n, n4 and the lower c (in absolute value but different from zero) the more accurate the pre-calculation will bed (it will be valid for higher numbers).

```
format long;

pi2=pi;

k2=0;

w5=1.5;

p3=input('Input a natural number: ');

reference=107;

p3=abs(round(p3));

if p3>reference

   reference=p3;

end


c2=-1/reference;


f= @(c,w,n) exp(c*2*n)*exp(2*n*w*1i)/(1-exp(n*c)*exp(n*w*1i));


func3=@(p,k) exp(-p*k);
func4=@(p,w) exp(-p*w*1i);

n4=10*reference;
n=100*reference;

% array y from 0 to n
y=zeros(1,(n+1));

liminf=-pi2;
limsup=pi2;
w2=(limsup-liminf)/n;
w3=liminf;
tiempo=cputime;

disp('I start the prework');

for i2=1:(n+1)

   y4=0;

   for i1=2:n4

      y4=y4+f(c2,w3,i1);

   end

      y(i2)=y4;
      w3=w3+w2;
```

```
end
tiempo2=cputime;
disp('I have done the prework');
disp(['Time of prework ',num2str(tiempo2-tiempo),' secs']);


disp('I start the calculations');


%First number to check

p2=7;
tiempo=cputime;
w3=liminf;

y5=0;

for i3=1:(n+1)

   y5=y5+func4(p2,w3)*y(i3);
   w3=w3+w2;

end


tiempo2=cputime;
d=y5*w2*(func3(p2,c2))/(2*pi2);

disp(d);
disp(['Number ',num2str(p2),' has ',num2str(round(abs(d))), ' combinations of 2 factor
products']);
disp('Time of calculation (secs)');
disp(tiempo2-tiempo);

%Second number to check
p2=9;

tiempo=cputime;
w3=liminf;

y5=0;

for i3=1:(n+1)

   y5=y5+func4(p2,w3)*y(i3);
   w3=w3+w2;

end


tiempo2=cputime;
d=y5*w2*(func3(p2,c2))/(2*pi2);

disp(d);
disp(['Number ',num2str(p2),' has ',num2str(round(abs(d))), ' combinations of 2 factor
products']);
disp('Time of calculation (secs)');
disp(tiempo2-tiempo);


%Third number to check
p2=20;

tiempo=cputime;
w3=liminf;

y5=0;

for i3=1:(n+1)

   y5=y5+func4(p2,w3)*y(i3);
   w3=w3+w2;

end


tiempo2=cputime;
d=y5*w2*(func3(p2,c2))/(2*pi2);

disp(d);
disp(['Number ',num2str(p2),' has ',num2str(round(abs(d))), ' combinations of 2 factor
products']);
disp('Time of calculation (secs)');
disp(tiempo2-tiempo);
```

```
%Fourth number to check the input number
p2=p3;

tiempo=cputime;
w3=liminf;

y5=0;

for i3=1:(n+1)

    y5=y5+func4(p2,w3)*y(i3);
    w3=w3+w2;

end


tiempo2=cputime;
d=y5*w2*(func3(p2,c2))/(2*pi2);

disp(d);
disp(['Number ',num2str(p2),' has ',num2str(round(abs(d))), ' combinations of 2 factor
products']);
disp('Time of calculation (secs)');
disp(tiempo2-tiempo);
```

B3. Annex B3. Matlab/Octave program for factorization of a semi-prime number (69)

You can find here attached a program in Matlab/Octave to calculate the expression (69) and to find the factors of a semiprime number. With the variable “multic” you can modify the precision. For multic=50, the precision is good, and you will get the result directly form the integral.

For multic=30, the precision is worse but with less computation time. Instead of obtaining the correct result directly, the program will test numbers in the neighborhood of the result until it finds the correct result. With a total time much lower,

For both methods, there is a pre-calculation that once done with the precision considered, it will be valid for all the numbers that can be checked with that precision, not needing to repeat the pre-calculation again (if this was stored in a data base).

```
format long;

pi2=pi;

k2=0;

p2=input('Introduce a semiprime number: ');

p2=abs(round(p2));
reference=p2;

c2=-1/sqrt(reference);


f= @(c,w,n) exp(c*(n+2))*exp(2*n*w*1i)/(1-exp(c)*exp(n*w*1i));


func3=@(p,k) exp(-p*k);
func4=@(p,w) exp(-p*w*1i);

% If you want exact calculation put multic=50
% If you want an approximate result and then fine tune
% in the neighbouring, multic=30 is sufficient

multic=50;

n4=multic*reference;
n=multic*reference;

% array y de 0 a n
y=zeros(1,(n+1));

liminf=-pi2;
limsup=pi2;
w2=(limsup-liminf)/n;
```

```
w3=liminf;
disp('I start the prework');
tiempo=cputime;
for i2=1:(n+1)

   y4=0;

   for i1=2:n4

      y4=y4+f(c2,w3,i1);

   end

      y(i2)=y4;
      w3=w3+w2;

end
tiempo2=cputime;
disp(['Time of prework ',num2str(tiempo2-tiempo),' secs']);

%i5=fix((w5-liminf)*n/(limsup-liminf))+1;

%disp(y(i5));

%salto los siguientes prints que no tengo func2


%First number
tiempo=cputime;

w3=liminf;

y5=0;

for i3=1:(n+1)

   y5=y5+func4(p2,w3)*y(i3);
   w3=w3+w2;

end


tiempo2=cputime;
d=log(y5*w2*(func3(p2,k2)/(4*pi2)))/c2;
disp(d);
d=round(abs(d));
disp('The approximate sum of factors is');
disp(d);

b=(d+sqrt(d^2-4*p2))/2;

a=d-b;
disp(['If ',num2str(p2),' has 2 factors, approximately these are']);
disp(round(a));
disp(round(b));
disp('Time of calculation (secs)');
disp(tiempo2-tiempo);

tiempo=cputime;

m=round(log(reference));
d=d-4*m;

for i3=1:(8*m)

b=(d+sqrt(d^2-4*p2))/2;

b=round(b);
a=d-b;

if (abs(a)*abs(b))==p2

   disp('The exact sum is ');
disp(d);


disp(['If ',num2str(p2),' has 2 factors, these exactly are']);
disp(round(a));
disp(round(b));


end
d=d+1;

end
tiempo2=cputime;
```

```
disp('Time of calculation (secs)');
disp(tiempo2-tiempo);
```