sage-euler-product

Release 0.0.3

the sage-euler-product authors

Nov 25, 2025

CONTENTS:

1 Documentation **3**
- 1.1 Tutorial Euler Product . . . 3
- 1.2 Fast multi-precision computation of some Euler products . . . 6
- 1.3 Installation EULER_PRODUCT . . . 23
- 1.4 API Documentation . . . 26

2 Indices and tables **43**

Python Module Index **45**

Index **47**

sage-euler-product is a Python **SageMath** package implementing Eulerian Product for Number Theory

CHAPTER

ONE

DOCUMENTATION

Release
0.0.3

Date
Nov 25, 2025

AUTHORS:

- Olivier Ramaré : initial version CNRS, Institut de Mathématique de Marseille, Aix-Marseille Université
- Dominique Benielli Aix-Marseille Université, Laboratoire Informatique et des Systèmes Integration as SageMath package. Cellule de développement Institut Archimède

1.1 Tutorial Euler Product

1.1.1 Introduction and principles

The main object of this software is to compute in a very fast manner Euler products with rational local factors over primes in some collections of arithmetic progressions modulo some fixed modulus q. The computations are done with interval arithmetic, the results are certified and given in the form `(lower_bound, upper_bound)`. An Euler product is a product over all the (positive integer) primes, possibly in some subsequence. Here are two examples $E_1 = \prod_{p\equiv 3,5[7]}(1-1/p^2)$, where the product is taken over every prime p congruent to 3 or 5 modulo 7, and $E_2 = \prod_{p\equiv 1[8]}\left(1-\frac{4}{p}\right)\left(\frac{p+1}{p-1}\right)^2$, the so-called Shanks's constant, where the product is taken over every prime congruent to 1 modulo 8.

During the computations, Euler products over rational functions such as E_2 are inferred from simpler Euler products of the shape

$$\prod_{p\in\mathcal{A} \mod q}(1-1/p^s),$$

by following the 2021 paper of Ettahri, Surel and Ramaré, where

- $\mathcal{A}$ is some subset of $G = (\mathbb{Z}/q\mathbb{Z})^\times$. The subset $\mathcal{A}$ has to be the union of "lattice invariant classes", as described below.
- q is a positive integer, the so-called "modulus". We have $q = 7$ for E_1.
- s is a real parameter that is strictly positif and *in this example* strictly larger than 1. A typical choice is $s = 2$. Technically, it should be an exact type, like `2` or `21/10`, or an element of a `RealIntervalField(...)` with enough precision. Since this precision is given in binary digits, using 10 times the number of decimals asked for the final result is a safe choice. Notice that one may have to import `RealNumber` from `sage.all` and that `RealIntervalField(1000)(2.1)` is maybe not what one would expect: `2.1` is understood as a float with 53 binary digits, then extended by adding enough binary digits 0, the result being somewhat different in decimal expansion.

See also

The mathematical proof of this software is taken from the paper ** Fast multi-precision computation of some Euler products ** by Salma Ettahri, Olivier Ramaré and Léon Surel, published in 2021, in volume 90 of *Mathematics of Computations*, pages 2247 to 2265.

In case $q = 1$, the notion of lattice invariant classes is trivial. Let us start by describing this case.

1.1.2 Euler Product over every primes

```
from euler_product.lattice_invariant_euler_products import get_euler_products
get_euler_products(1, 21/10 , 1-x^2, 1+x^3, 103, 20, verbose = 0, with_laTeX = 0, digits_
↪offset = 10)
```

This computes the Euler product $\prod_{p\geq 2} \frac{1-1/p^{2s}}{1+1/p^{3s}}$ where $s = 2.1$ with potentially 103 correct digits and by computing directly the Euler product for all the primes less than $P = 20$. This value of P is 300 by default. The level of comments `verbose` can be set to 0, 1 or 2. The additional parameter `with_laTeX` is either equal to 1 or not equal to 1, with an obvious meaning. If the output does not have enough correct digits, the user is asked to increase the value 103 to 110 for instance. We decided not to automate this behaviour.

On the effect of the choice of s, notice that the two calls

```
get_euler_products(1, 1 , 1-x^4, 1, 103)
get_euler_products(1, 2 , 1-x^2, 1, 103)
```

give the same answer, which is readily seen to be an approximation of $1/\zeta(4)$, where ζ is the Riemann-zeta function. Recall that we have $\zeta(4) = \pi^4/90$, a fact that we may use to check our code.

1.1.3 Lattice Invariant Classes modulo q

- Definition of Lattice Invariant Classes

We subdivide the multiplicative group $G = (\mathbb{Z}/q\mathbb{Z})^\times$ in classes called *Lattice Invariant Classes*. Two points are in the same class if and only if they generate the same subgroup modulo q.

When $q = 15$ these classes are obtained by

```
LatticeInvariant(15)[1]
(frozenset({1}), frozenset({4}), frozenset({11}), frozenset({14}), frozenset({8, 2}),␣
↪frozenset({13, 7}))
```

```
from euler_product.utils_euler_product import LatticeInvariant
LatticeInvariant(15)
((frozenset({1}), frozenset({1, 4}), frozenset({1, 11}), frozenset({1, 14}), frozenset(
↪{8, 1, 2, 4}), frozenset({1, 4, 13, 7})),
 (frozenset({1}), frozenset({4}), frozenset({11}), frozenset({14}), frozenset({8, 2}),␣
↪frozenset({13, 7})))
```

The output is a couple whose first element is the tuple of the monogenic subgroups of $G = (\mathbb{Z}/q\mathbb{Z})^\times$ and whose second element is the tuple of lattice invariant classes.

- Low level tools

```
from euler_product.utils_euler_product import ComponentStructure
mystructure = ComponentStructure(3)
```

This class proposes several quantities. It is used by the high level function `get_vs` and `get_euler_products`, so the user does not have to worry about it. However the quantities computed may have interest.

- `mystructure.q`: the modulus q.
- `mystructure.phi_q`: the value of the Euler phi-function at q.
- `mystructure.the_exponent`: the exponent of the group $G = (\mathbb{Z}/q\mathbb{Z})^\times$.
- `mystructure.invertibles`: the tuple of invertibles in $(\mathbb{Z}/q\mathbb{Z})$, i.e. an enumeration of $G = (\mathbb{Z}/q\mathbb{Z})^\times$.
- `mystructure.the_SG_tuple`: the tuple of the subgroups of $G = (\mathbb{Z}/q\mathbb{Z})^\times$ that are generated by a single element. Such subgroups are also called *monogenic* subgroups.
- `mystructure.the_Class_tuple`: the tuple of the lattice invariant classes.
- `mystructure.nb_class`: the number of lattice invariant classes.
- `mystructure.character_group`: the character group of $G = (\mathbb{Z}/q\mathbb{Z})^\times$.
- `mystructure.invariant_characters`: for each monogenic subgroup in `mystructure.the_SG_tuple`, the list of (the indices of) the characters that has this subgroup in its kernel. The order of `mystructure.invariant_characters` is the same as the one in `mystructure.the_SG_tuple`.
- Some methods are also available.

1.1.4 Euler Product over primes in arithmetic progression

We start with the three data:

- A modulus `q`≥ 1.
- A rational fraction given in the form $F(x)/H(x)$ where $F(x)$ and $H(x)$ are two polynomials with real coefficients and such that $F(0) = H(0) = 1$.
- A parameter `s`.
- A wanted precision `nb_decimals`, given as a number of decimal digits.

We have access to the lattice invariant classes, as per the preceding paragraph. For each of these classes $(\mathcal{A})$, we compute

$$\prod_{p\in\mathcal{A}} \frac{F(1/p^s)}{H(1/p^s)}.$$

There is a condition for this product to converge absolutely: on writing $F(x) - H(x) = x^\Delta T(x)$ for a $\Delta \geq 1$ and a polynomial $T(x)$, we need that $\Delta s > 1$. We assume this condition to hold.

```
from euler_product.lattice_invariant_euler_products import get_euler_products
get_euler_products(q, s, F(x) , H(x), nb_decimals, big_p = 300, verbose = 0, with_laTeX␣
↪= 0, digits_offset = 10)
```

answers a couple whose first component is the tuple of the lattice invariant classes $(\mathcal{A})$, and second component is the tuple of the values $\prod_{p\in\mathcal{A}} \frac{F(1/p^s)}{H(1/p^s)}$, for example

```
from euler_product.lattice_invariant_euler_products import get_euler_products
result = get_euler_products(5, 1, 1-x^2 , 1+x^3, 100, 300, 0)

result[0][0]
frozenset({1})
result[1][0](0.
```

(continues on next page)

(continued from previous page)

```
↪9884028950453419692925625250954713121182210521345380891771586345550561301333511982564965807673436742857698
↪ 0.
↪9884028950453419692925625250954713121182210521345380891771586345550561301333511982564965807673437490090
↪
```

which means that

$$0.9884028950453419692925625250954713121182210521345380891771586345550561301333511982564965807673436742857698$$
$$\leq \prod_{p\equiv 1[5]} \frac{1-1/p^2}{1+1/p^3}$$
$$\leq 0.9884028950453419692925625250954713121182210521345380891771586345550561301333511982564965807673437490090$$

With `verbose = 1` or `verbose = 2`, the results are more explicitly written.

To compute the specific quantities $\prod_{p\in\mathcal{A}}(1-1/p^s)^{-1}$ where the rational fraction is thus fixed, we have a shortcut:

```
from euler_product.lattice_invariant_euler_products import get_vs
get_vs(q, s, nb_decimals = 100, big_p = 100, verbose = 2, with_laTeX = 0, digits_offset␣
↪= 10)
```

The output is similar to the one of `get_euler_products`, with the same effect of the available parameters. However, there is the additional possible value `verbose = -1`. In that case the output takes the shape

```
[big_p, phi_q, r, nb_invariant_class, big_m, time_end - time_start, difference]
```

which is rather explicit. The parameter `big_m` is introduced in the reference paper and `r` is the number of values of m, as per Eq. (5) of the reference paper, that are being used. The timing is given in seconds, and `difference` is an approximation of the number of correct decimals given.

- Auxiliaries

We finally provide two auxiliary functions.

```
from euler_product.lattice_invariant_euler_products import table_performance
table_performance(min_q, max_q, nb_decimals = 100, big_p = 300)
```

This gives some timing info for `get_vs(q, 2, nb_decimals, big_p, -1)`. The output has a LaTeX format of an array, the columns being `q`, `phi_q`, `nb_prime_factors_phi_q`, `r`, `nb_invariant_class`, `big_m` and finally `time_end - time_start` in seconds / 10. The meanings are the same as in `get_vs`.

```
from euler_product.lattice_invariant_euler_products import get_vs_checker
get_vs_checker(q, s, borne = 10000):
```

This is a simple sanity check. The output `get_vs` displays the Euler products computed by `get_vs`, except that these products are only approximated by the truncated Euler product up to `borne`.

1.2 Fast multi-precision computation of some Euler products

Authors

Salma Ettahri, Olivier Ramaré, Léon Surel

Note

This file is a web version of the paper *Fast multi-precision computation of some Euler products* by the same authors and which appeared in Math. Comp. **90** (2021), no. 331, pages 2247–2265. The modifications concerns the numbering and the elements concerning the produced code.

1.2.1 1. Introduction

In formula (16) of *[16]*, D. Shanks obtained the following closed expression to compute the Landau-Ramanujan constant:

$$\prod_{p\equiv 3[4]} \frac{1}{1-1/p^s} = \prod_{k\geq 0} \left(\frac{\zeta(2^k s)(1-2^{-2^k s})}{L(2^k s, \chi_{1,4})} \right)^{1/2^{k+1}} \tag{1.1}$$

where $s > 1$ and $\chi_{1,4}$ is the (only) non-principal Dirichlet character modulo 4. Since both $\zeta(2^k s)$ and $L(2^k s, \chi_{1,4})$ are $1+\mathcal{O}(1/2^{s2^k})$, we only need to compute $\mathcal{O}(\log D)$ values of L-functions (including the Riemann ζ-function) to obtain D decimal digits. In this paper, we generalize this process in several directions, but a main feature of our work is that it applies only to Euler products over primes belonging to some special subsets of $G = (\mathbb{Z}/q\mathbb{Z})^\times$ that we define below. We obtain closed formulas involving only values of L-functions of Dirichlet characters for rational Euler products over primes in these special sets and deduce fast ways to compute a more restricted class of such products. Let us first introduce the players.

Definition 1.

Two elements g_1 and g_2 of the abelian group G are said to be lattice-invariant if and only if they generate the same group. This defines an equivalence relation.

We denote the set of lattice invariant classes by $G^\sharp$ and the set of cyclic subgroups of G by $\mathscr{G}$. The map between $\mathscr{G}$ and $G^\sharp$ which, to a subgroup, associates the subset of its generators, is one-to-one.

The cardinality of $G^\sharp$ can be swiftly inferred from Theorem 3 of *[18]* or from Theorem 1 of *[19]*, both by L. Tóth. When $\mathcal{A}$ is a subset of $G = (\mathbb{Z}/q\mathbb{Z})^\times$, we define $\langle \mathcal{A} \rangle$ to be the (multiplicative) subgroup generated by $\mathcal{A}$.

For any Dirichlet character χ modulo q and any parameter $P \geq 2$, we define

$$L_P(s,\chi) = \prod_{p\geq P} (1-\chi(p)/p^s)^{-1} \tag{1.2}$$

and correspondingly $\zeta_P(s) = \prod_{p\geq P}(1-1/p^s)^{-1}$.

Given K a subgroup of $G = (\mathbb{Z}/q\mathbb{Z})^\times$, we denote by $K^\perp$ the subgroup of Dirichlet characters modulo q that take the value 1 on K. When s is a real number, the number $\prod_{\chi\in K^\perp} L_P(s,\chi)$ is indeed a positive real number because, when χ belongs to $K^\perp$, so does $\overline{\chi}$.

Here is the central theorem of this paper.

Theorem 2. Let q be some modulus and $\mathcal{A}$ be a lattice-invariant class of $G = (\mathbb{Z}/q\mathbb{Z})^\times$. Let $F, H \in \mathbb{R}[X]$ be two polynomials satisfying $F(0) = H(0) = 1$ and let $\Delta \geq 1$ be an integer such that $(F(X) - H(X))/X^\Delta \in \mathbb{R}[X]$. Let $\beta \geq 2$ be an upper bound for the maximum modulus of the inverses of the roots of F and of H. Let $\sigma_1, \sigma_2, \cdots, \sigma_{\deg F}$ be the roots of F (a multiple root appears as many times as its multiplicity), and similarly, let $\rho_1, \rho_2, \cdots, \rho_{\deg H}$ be the roots of H. For any non-negative integer d, we set

$$s_{H/F}(d) = \sum_{1\leq i\leq \deg H} \rho_i^{-d} - \sum_{1\leq j\leq \deg F} \sigma_j^{-d}. \tag{1.3}$$

Let P and $s > 1/\Delta$ be two real parameters such that $P^s \geq 2\beta$. We define, for any cyclic subgroup K of G and any positive integer m,

$$C_{\mathscr{A}}(K, m, F/H) = \sum_{t|m} \mu(t) s_{H/F}(m/t) \sum_{\substack{L \in \mathscr{G}, \\ L^{[t]} = \langle \mathscr{A} \rangle, \\ K \subset L}} \frac{\mu(|L|/|K|)}{|G/K|} \tag{1.4}$$

where $L^{[t]} = \{x^t, x \in L\}$ and $\langle \mathscr{A} \rangle$ is the subgroup generated by $\mathscr{A}$. We have

$$\prod_{\substack{p \geq P, \\ p + q\mathbb{Z} \in \mathcal{A}}} \frac{F(1/p^s)}{H(1/p^s)} = \prod_{m \geq \Delta} \prod_{K \in \mathscr{G}} \Big(\prod_{\chi \in K^\perp} L_P(ms, \chi) \Big)^{C_{\mathscr{A}}(K,m,F/H)/m}. \tag{1.5}$$

For any positive real-valued parameter M, the following bound holds true:

$$\begin{aligned} \pm \log \prod_{m \geq M+1} \prod_{K \in \mathscr{G}} \Big(\prod_{\chi \in K^\perp} L_P(ms, \chi) \Big)^{\frac{C_{\mathscr{A}}(K,m,F/H)}{m}} \\ \leq 4(\deg F + \deg H)|\mathscr{G}|^2 (s + P) \Big(\frac{\beta}{P^s} \Big)^{M+1}. \end{aligned} \tag{1.6}$$

In the case $H/F = 1 - X$, the relevant identity is proved in *18* and is the heart of this paper. Our result applies in particular to $\mathcal{A} = \{1\}$ and to $\mathcal{A} = \{-1\}$. When $q = 4$ and $\mathcal{A} = \{-1\}$, we readily find that only $t = 1$ matters in (1.4), that $C_{\{-1\}}(\{1\}, 2^k, 1/(1 - X)) = -1/2$ and that $C_{\{-1\}}(\{\pm 1\}, 2^k, 1/(1 - X)) = 1$. On recalling Lemma 16, this results in (1.1).

Remark 3. Lemma *21* ensures that we may select

$$\beta = \max\Big(2, \sum_{1 \leq k \leq \deg F} |a_k|, \sum_{1 \leq k \leq \deg H} |b_k|\Big)$$

when $F(X) = 1 + a_1 X + \ldots + a_\delta X^\delta$ and $H(X) = 1 + b_1 X + \ldots + b_{\delta'} X^{\delta'}$. Notice that our assumptions imply that $b_i = a_i$ when $i < \Delta$.

Remark 4. The numbers $s_{H/F}(n)$ may be computed via the Girard-Newton relations recalled in Lemma *19*.

Remark 5. We prove in Lemma *22* that, when K and $\mathscr{A}$ are fixed, the quantity

$$\sum_{\substack{L \in \mathscr{G}, \\ L^{[t]} = \langle \mathscr{A} \rangle, \\ K \subset L}} \mu(|L|/|K|)$$

depends only on $\gcd(t, \varphi(q))$.

Remark 6. We have $C_{\mathscr{A}}(K, m, F/H) = -C_{\mathscr{A}}(K, m, H/F)$, a property we shall use to simplify the typography.

Remark 7. There is some redundancy in our formula as a same character χ may appear in several sets $K^\perp$ (for instance, the principal character appears in all of them). Disentangling these contributions leads to a slightly more complicated formula. We first have to introduce, for any cyclic subgroup S , the subset $S^{\perp\circ} \subset S^\perp$ constituted of those elements that do not belong to any $T^\perp$, for $T \subsetneq S$. It can be readily checked that any $K^\perp$ is the union of $S^{\perp\circ}$ where S ranges the subgroups that are included in K. We then define

$$C^\circ_{\mathscr{A}}(S, m, F/H) = \sum_{t|m} \mu(t) s_{H/F}(m/t) \sum_{\substack{L \in \mathscr{G}, \\ L^{[t]} = \langle \mathscr{A} \rangle, \\ S \subset L}} \frac{\varphi(|L|/|S|)}{|G/S|}. \tag{1.7}$$

Formula (1.5) becomes:

$$\prod_{\substack{p\geq P,\\ p+q\mathbb{Z}\in\mathcal{A}}} \frac{F(1/p^s)}{H(1/p^s)} = \prod_{m\geq\Delta}\prod_{S\in\mathscr{G}}\Big(\prod_{\chi\in S^{\perp\circ}} L_P(ms,\chi)\Big)^{C^\circ_{\mathscr{A}}(S,m,F/H)/m}$$

and the bound (1.6) holds to estimate the tail of this product, as we only shuffled terms with a fixed index m.

Super fast evaluations

Corollary 8. For every positive integer m , the constant $C_{\mathscr{A}}(K,m,1-X)$ vanishes when one prime factor of m is coprime with $\varphi(q)$. As a consequence and under the hypotheses of Theorem 2 with $\Delta=1$, the products

$$\prod_{\substack{p\geq P,\\ p+q\mathbb{Z}\in\mathcal{A}}} \left(1-\frac{1}{p^s}\right)$$

may be computed by $\mathcal{O}((\log D)^r)$ computations of L -functions to get D-decimal digits, where r is the number of prime factors of $\varphi(q)$. The implied constant in the $\mathcal{O}$-symbol may depend on q .

This leads to very fast computations, and we were for instance able to produce 100 (resp.,1000, resp.,5000) digits of these products when $q=3$ in a third of a second (resp.,12 seconds, resp.,35 minutes with $P=400$) on a usual desktop computer. See the implementation notes at the end of this paper. Notice however that the number of L-values required is not the only determinant: when q increases, the dependence in q matters as the character group increases in size, and when the required precision increases, each computation of an L-value may take a long time. We do not address the issue of these computations here. We present some timing data at the end of this paper.

Proof of Corollary 8. Lemma 16 tells us that $C_{\mathscr{A}}(K,m,1-X)$ vanishes when one prime factor of m is coprime with $\varphi(q)$. Let us decompose $\varphi(q)$ in prime factors: $\varphi(q)=p_1^{\alpha_1}\cdots p_r^{\alpha_r}$. Any integer $m\leq M$ such that all its prime factors divide q, can be written as $m=p_1^{\beta_1}\cdots p_r^{\beta_r}$ with $\beta_i\leq(\log M)/\log p_i$ for $i\leq r$. In particular, there are at most $((\log M)/\log 2)^r$ such integers. By (1.6), the contribution of the integers $m>M$ to the Euler product to be computed is $1+\mathcal{O}((\beta/P^s)^M)$, which is $1+\mathcal{O}(2^{-M})$ by the assumption $P^s\geq 2\beta$. We want this error term to be $1+\mathcal{O}(10^{-D})$ to get about $D+\mathcal{O}(1)$ decimal digits. This is ensured by $M\log 2\geq D\log 10$, i.e. it is enough to take $M=4D$.

In order to extend this property to other Euler products, many of the coefficients $C_{\mathscr{A}}(K,m,F/H)$ should vanish when m varies. This is however not likely to happen, except when F/H is a product/quotient of cyclotomic polynomials. Indeed the coefficients $s_{H/F}(m)$ satisfy a linear recurrence (of degree at most $\max(\deg F,\deg H)$) and as such are expected to grow exponentially fast if they are not roots of unity. When for instance the coefficients of the recurrence belong to some number field, this is proved by Evertse in *[3]* and independently by van der Poorten and Schlickewei in *[20]*. This is the case where we may expect cancellations to happen. Since the sum defining $C_{\mathscr{A}}(K,m,F/H)$ is of the form $\sum_{t|m}\mu(t)r_0(t)s_{H/F}(m/t)$ for some function $r_0(t)$ that remains bounded (it takes only a finite set of values), it is dominated by the term $t=1$ when m is large enough; no cancellation due to the Möbius factor can be expected either. We are then left with the case of cyclotomic polynomials, but they can be easily dealt with using Corollary 8; indeed, if we denote by Φ_n the n-th cyclotomic polynomial, the identity $\prod_{d|n}\Phi_d(X)=X^n-1$ gets inverted to $\Phi_n(X)=\prod_{d|n}(X^d-1)^{\mu(n/d)}$.

A Sage script

The material of this paper has been used to write the Sage script *using Python 3*.

The function `get_euler_products(q, s, F, H, nbdecimals)` gives all these Euler products. The polynomials F and H are to be given as polynomial expressions with the variable x. The special function `get_vs(q, s, nbdecimals)` gives all the Euler products of Corollary 8.

Some historical pointers

D. Shanks in *[14]* (resp. *[15]*, resp. *[17]*) has already been able to compute an Euler product over primes congruent to 1 modulo 4 (resp. to 1 modulo 8 in both instances), by using an identity (Lemma of section 2 for *[14]*, Equation (5) in *[15]* and the Lemma of section 3 in *[17]*) that is a precursor of our Lemma 19.

In these three examples, the author has only been able to compute the first five digits, and this is due to three facts: the lack of an interval arithmetic package at that time, the relative weakness of the computers and the absence of a proper study of the error term. We thus complement these results by giving the first hundred decimals.

Complementary to the published papers, three influent preprints on how to compute Euler products with high accuracy have been floating on the web: *[5]* a memo started in 1990 in its 1996 version by Ph. Flajolet and I. Vardi, *[1]* by H. Cohen and *[7]* by X. Gourdon and P. Sebah. Comparing the desired constant with zeta-values is the overarching idea. The set of zeta-values is extended to L-values of (some) quadratic characters in the three, in some way or another, and to the values of Dedekind zeta-function in *[1]*. No complete error term analysis is presented, sometimes because the series used are simple enough to make this analysis rather easy. These three sources also deal with constants that are sums over primes and a similar extension of our work is possible, but kept for later. It should be noticed that Equation (20) from *[5]* is in fact the formula given as Equation (16) in *[16]* for the Landau-Ramanujan constant.

On the methodology

We decided to prove Theorem *2* directly, by giving the formula and shuffling terms. This gives a short and self-contained proof. However, we did not come up with the coefficients $C_{\mathscr{A}}(K, m, F/H)$ by some lucky strike! There is a path leading from abelian field theory to our expression that is much closer to D. Shanks's approach. We say more on this subject in section 4.

Application to some constants

This paper has been inspired by the wish to compute with high numerical precision two constants that appear in the paper *[6]* by É. Fouvry, C. Levesque and M. Waldschmidt. In the notation of that paper, they are

$$\alpha_0^{(3)} = \frac{1}{3^{1/4}\sqrt{2}} \prod_{p\equiv 2[3]} \left(1 - \frac{1}{p^2}\right)^{-1/2}$$

and

$$\beta_0 = \frac{3^{1/4}\sqrt{\pi}}{2^{5/4}} \frac{\log(2+\sqrt{3})^{1/4}}{\Gamma(1/4)} \prod_{p\equiv 5,7,11[12]} \left(1 - \frac{1}{p^2}\right)^{-1/2}. \tag{1.8}$$

Both occur in number theory as densities. The number of integers n of the shape $n = x^2 - xy + y^2$, where x and y are integers (these are the so-called Loeschian numbers, see the sequence A003136 entry in *[12]*) is asymptotically approximated by

$$N(x) = \alpha_0^{(3)} \frac{x(1+o(1))}{\sqrt{\log x}}. \tag{1.9}$$

This motivates our interest in the first constant. The second one arises in counting the number of Loeschian numbers that are also sums of two squares (see sequence A301430 entry of *[12]*), namely we have

$$N'(x) = \beta_0 \frac{x(1+o(1))}{(\log x)^{3/4}}.$$

From the sequence A301429 entry in *[12]*, we know that $\alpha_0^{(3)} = 0.638909\ldots$ but we would like to know (many!) more digits. Similarly it is known that $\beta_0 = 0.30231614235\ldots$.

Corollary 9. We have

$$\begin{aligned}\alpha_0^{(3)} = 0.&63890\,94054\,45343\,88225\,49426\,74928\,24509\,37549\,75508\,02912\\ &33454\,21692\,36570\,80763\,10027\,64965\,82468\,97179\,11252\,86643\cdots\end{aligned}$$

and

$$\beta_0 = 0.30231\,61423\,57065\,63794\,77699\,00480\,19971\,56024\,12795\,18936\\ 96454\,58867\,84128\,88654\,48752\,41051\,08994\,87467\,81397\,92727\cdots$$

This follows from Theorem 2 with the choices $q = 3$ and $\mathcal{A} = \{2\}$ for $\alpha_0^{(3)}$, and $q = 12$ and $\mathcal{A} = \{5, 7, 11\}$ for β_0. The other parameters are uniformly selected as $F(X) = 1 - X^2$, $H(X) = 1$, $\Delta = 2$, $\beta = 2$ and $s = 1$.

Corollary 10 (Shanks' Constant). We have

$$\prod_{p\equiv 1[8]} \left(1 - \frac{4}{p}\right)\left(\frac{p+1}{p-1}\right)^2 = 0.95694\,53478\,51601\,18343\,69670\,57273\,89182\,87531\\ 74977\,29139\,14789\,05432\,60424\,60170\,16444\,88885\\ 94814\,40512\,03907\,95084\cdots$$

As a consequence Shanks' constant satisfies

$$I = \frac{\pi^2}{16\log(1+\sqrt{2})} \prod_{p\equiv 1[8]} \left(1 - \frac{4}{p}\right)\left(\frac{p+1}{p-1}\right)^2\\ = 0.66974\,09699\,37071\,22053\,89224\,31571\,76440\,66883\,70157\,43648\\ 24185\,73298\,52284\,52467\,99956\,45714\,72731\,50621\,02143\,59373\cdots$$

We deduce this Corollary from Theorem 2 by selecting the parameters $q = 8$, $\mathcal{A} = \{1\}$, $F(X) = 1-2X-7X^2-4X^3$, $H(X) = 1-2X+X^2$, $s = 1$, $\Delta = 2$ and $\beta = 4$. As explained in *[15]*, the number of primes $\leq X$ of the form m^4+1 is conjectured to be asymptotically equal to $I \cdot X^{1/4}/\log X$. The name "Shanks' constant" comes from Chapter 2, page 90 of *[4]*.

When using the script that we introduce below, this value is obtained by multiplying by $\frac{\pi^2}{16\log(1+\sqrt{2})}$ the value obtained with the call

`get_euler_products(8, 1, 1-2*x-7*x^2-4*x^3, 1-2*x+x^2, 110, 50, 2, 1)`.

A note is required here: the script evaluates loosely the required working precision in order to get say 100 correct digits at the end. The results are however presented with the precision obtained, and if we had been asking initially for 100 decimal digits, the script would issue only 94 of them. We could have implemented a mechanism that increases the precision until the result satisfies the request, but we have preferred to let the users increase the precision by themselves. When asking for 110 decimal digits, the script is able to compute 106 of them. We can get a thousand decimals for this constant in about 2 minutes on a usual desktop computer (by asking for 1010 decimal digits), see the implementation notes at the end of this paper.

Corollary 11 (Lal's Constant). We have

$$\prod_{p\equiv 1[8]} \frac{p(p-8)}{(p-4)^2} = 0.88307\,10047\,43946\,67141\,78342\,99003\,10853\,46768\\ 88834\,88097\,34707\,19295\,15939\,52119\,46990\,65659\\ 68857\,99383\,28603\,79164\cdots$$

As a consequence Lal's constant satisfies

$$\lambda = \frac{\pi^4}{2^7\log^2(1+\sqrt{2})} \prod_{p\equiv 1[8]} \left(\frac{p+1}{p-1}\right)^4\left(1 - \frac{8}{p}\right)$$

$$= \frac{\pi^4}{2^7\log^2(1+\sqrt{2})} \prod_{p\equiv 1[8]} \left(1 - \frac{4}{p}\right)^2\left(\frac{p+1}{p-1}\right)^4 \prod_{p\equiv 1[8]} \frac{p(p-8)}{(p-4)^2} = 0.79220\,82381\,67541\,66877\,54555\,66579\,02410\,11289\,32250\,98\\ 11172\,27973\,45256\,95141\,54944\,12490\,66029\,53883\,98027\,52$$

We deduce the first value given in this Corollary by using Theorem :ref`2<2>` with the parameters $q = 8$, $\mathcal{A} = \{1\}$, $F(X) = 1 - 8X$, $H(X) = 1 - 8X + 16X^2$, $s = 1$, $\Delta = 2$ and $\beta = 8$. The value of Lal's constant λ is then deduced by combining the value obtained in Corollary *10* together with this one. This splitting of the computation in two introduces smaller polynomials and this leads to a lesser running time. As explained in *[17]*, the number of primes $\leq X$ of the form $(m+1)^2 + 1$ and such that $(m-1)^2 + 1$ is also prime, is conjectured to be asymptotic to $\lambda \cdot X^{1/2}/(\log X)^2$. The name "Lal's Constant" comes from the papers *[8]* and *[17]*. When using the script that we introduce below, the first value is obtained with the call

`get_euler_products(8, 1, 1-8*x, 1-8*x+16*x^2, 110, 50, 2, 1).`

If this call requires about 2 seconds on a usual desktop computer, this time increases to 4 minutes when we ask for a thousand digits. We did not try to get 5000 digits as we did for the products of Corollary *8*.

We close this section by mentioning another series of challenging constants. In *[10]*, P. Moree computes inter alia the series of constants A_χ defined six lines after Lemma 3, page 452, by

$$A_\chi = \prod_{p \geq 2} \left(1 + \frac{(\chi(p) - 1)p}{(p^2 - \chi(p))(p-1)}\right), \tag{1.10}$$

where χ is a Dirichlet character. Our theory applies only when χ is real valued.

A closed formula for primitive roots

Let us recall that a *primitive root* n modulo q is an integer such that the class of n generates $G = (\mathbb{Z}/q\mathbb{Z})^\times$. It is a classical result that such an element exists if and only if q is equal to 2 or 4, or is equal to a prime power of an odd prime or to twice such a prime power.

Corollary 12. Let $\mathscr{A}_0$ be the subset of $G = (\mathbb{Z}/q\mathbb{Z})^\times$ consisting of all the multiplicative generators of G. Assume q is such that such an $\mathscr{A}_0$ is not empty. For any real parameter $P \geq 2$ and $s > 1$, we have

$$\zeta_P(s; q, \mathscr{A}_0) = \prod_{m | q^\infty} \prod_{S \in \mathscr{G}} \left(\prod_{\chi \in K^{\perp \circ}} L_P(ms, \chi) \right)^{e(m,q,S)},$$

where $m|q^\infty$ means that all the prime factors of m divide q and where $e(m, q, S) = \frac{|S|\varphi(q/|S|)}{m\varphi(q)}$.

Proof. Indeed, since $\mathscr{A}_0$ generates G, the only index t in (1.7) is $t = 1$. Hence, only $L = G$ is possible.

Thanks

The authors thank M. Waldschmidt for having drawn their attention to this question, P. Moree and É. Fouvry for helpful discussions on how to improve this paper and X. Gourdon for exchanges concerning some earlier computations. The referees are also to be warmly thanked for their very careful reading and for ideas on how to improve both the presentation and the corresponding script.

1.2.2 2. Proof of Theorem 2 when $F/H = 1/(1-X)$

We follow the notation introduced in (1.4). Since here $F/H = 1/(1-X)$, this leads us to consider, for any cyclic subgroup $K \in \mathscr{G}$, any class $\mathscr{A}$ in $G^\sharp$ and any positive integer m, the coefficient

$$C_{\mathscr{A}}(K, m, 1 - X) = \sum_{t | m} \mu(t) \sum_{\substack{L \in \mathscr{G}, \\ L^{[t]} = \langle \mathscr{A} \rangle}} \frac{\mu(|L|/|K|)}{|G/K|} \tag{1.11}$$

where $L^{[t]} = \{x^t, x \in L\}$. Notice that it is also a cyclic subgroup of G. Let us first note a simple property.

Lemma 13. In a finite cyclic group L, the map that associates to a subgroup of L its cardinality is a one-to-one map between the set of divisors of $|L|$ and the set of its subgroups. Furthermore, any subgroup of a cyclic group is cyclic.

Proof. We can assume that $L = (\mathbb{Z}/\ell\mathbb{Z}, +)$. For each $d|\ell$, the unique subgroup of order d is $\{(\ell/d)n, 0 \le n \le d-1\}$.

Here is the fundamental property satisfied by these coefficients.

Proposition 14. For any positive integer ℓ, any prime p and any lattice-invariant class $\mathscr{A}$, we have

$$\sum_{hm=\ell} \sum_{\substack{K\in\mathscr{G},\\ \chi\in K^\perp}} \chi(p^h) C_{\mathscr{A}}(K, m, 1-X) = 1_{p\in\mathscr{A}}.$$

Proof. Let S be the left-hand side sum to be evaluated. Let B be the subgroup generated by p. By using the orthogonality of characters, we readily obtain

$$S = \sum_{hm=\ell} \sum_{\substack{K\in\mathscr{G},\\ B^{[h]}\subset K}} |G/K| C_{\mathscr{A}}(K, m, 1-X).$$

Next, we introduce the expression given in (1.11), shuffle the summations and get

$$S = \sum_{hm=\ell} \sum_{t|m} \mu(t) \sum_{\substack{L\in\mathscr{G},\\ L^{[t]}=\langle\mathscr{A}\rangle}} \sum_{\substack{K\in\mathscr{G},\\ B^{[h]}\subset K}} \mu(|L|/|K|).$$

By Lemma *13* and the Möbius function characteristic property, the last summation vanishes when $B^{[h]} \neq L$ and takes the value 1 otherwise. Since $(B^{[h]})^{[t]} = B^{[ht]}$, this gives us

$$S = \sum_{hm=\ell} \sum_{\substack{t|m,\\ B^{[ht]}=\langle A\rangle}} \mu(t).$$

We continue in a more classical way:

$$S = \sum_{\substack{ath=\ell,\\ B^{[ht]}=\langle A\rangle}} \mu(t) = \sum_{\substack{ab=\ell,\\ B^{[b]}=\langle A\rangle}} \sum_{t|b} \mu(t) = 1_{B=\langle A\rangle},$$

concluding the proof .

Corollary 15. For any prime p, any positive real number s and any lattice-invariant class $\mathscr{A}$, we have

$$\prod_{m\ge1} \prod_{K\in\mathscr{G}} \Big(\prod_{\chi\in K^\perp} (1-\chi(p)p^{-ms}) \Big)^{-C_{\mathscr{A}}(K,m,1-X)/m} = \begin{cases} (1-p^{-s})^{-1} & \text{when } p\in\langle\mathscr{A}\rangle, \\ 1 & \text{otherwise.} \end{cases}$$

Proof. We first check that, for any positive integer m and any subgroup K, we have

$$\exp \sum_{\chi\in K^\perp} \sum_{h\ge1} \frac{\chi(p^h)}{hp^{mhs}} = \prod_{\chi\in K^\perp} \Big(1 - \frac{\chi(p)}{p^{ms}}\Big)^{-1}.$$

Since s is a positive real number, the right-hand side is also positive, and so can be raised to some rational power, say c. The sum inside the exponential is also a real number and the equation $\exp x = y$ leads obviously to $\exp(cx) = y^c$. The right-hand side of our lemma may thus be written $\exp S(p)$ where

$$S(p) = \sum_{m\ge1} \sum_{K\in\mathscr{G}} \sum_{\chi\in K^\perp} \sum_{h\ge1} \frac{\chi(p^h) C_{\mathscr{A}}(K, m, 1-X)}{mhp^{mhs}}.$$

We set $\ell = mh$ and appeal to Proposition *14* to infer that

$$S(p) = \sum_{\ell\ge1} \frac{1}{\ell p^{\ell s}} 1_{p\in\mathscr{A}},$$

from which our corollary follows readily.

Lemma 16. If m has a prime factor that does not divide $\varphi(q)$, we have $C_{\mathscr{A}}(K, m, 1-X) = 0$.

Proof. When $F/H = 1 - X$, we have $s_{H/F}(m) = -1$ uniformly in m. If $m = m_1 p^a$ for some m_1 prime to p and p prime to the order $\varphi(q)$ of G, any divisor t of m factors in $t_1 p^b$ where $t_1 | m_1$ and $b \leq a$. The Möbius coefficient reduces these choices to $b = a$ or to $b = a - 1$ and since we have $L^{[t]} = L^{[t_1]}$, both are possible. If we denote the contribution of $p^a t_1$ to $C_{\mathscr{A}}(K, m, 1-X)$ by S_1 say, the contribution or $p^{a-1}t_1$ is $-S_1$, and on pairing them we get zero.

Lemma 17. Let $f > 1$ be a real parameter. We have

$$\left|\log \zeta_P(f)\right| \leq \frac{1 + P/(f-1)}{P^f}.$$

Proof. We use

$$\log \zeta_P(f) = -\sum_{p \geq P} \sum_{k \geq 1} \frac{1}{k p^{kf}}$$

hence, by using a comparison to an integral, we find that

$$\left|\log \zeta_P(f)\right| \leq \sum_{n \geq P} \frac{1}{n^f} \leq \frac{1}{P^f} + \int_P^\infty \frac{dt}{t^f} = \left(\frac{f-1}{P} + 1\right) \frac{1}{(f-1)P^{f-1}}.$$

Theorem 18. For every $s > 1$ and every $P \geq 2$, we have

$$\zeta_P(s; q, \mathcal{A}) = \prod_{\substack{p + q\mathbb{Z} \in \mathcal{A}, \\ p \geq P}} (1 - p^{-s})^{-1} = \prod_{m \geq 1} \prod_{K \in \mathscr{G}} \left(\prod_{\chi \in K^\perp} L_P(ms, \chi) \right)^{C_{\mathscr{A}}(K, m, 1-X)/m}.$$

Proof. This is a simple consequence of Corollary *15*. Indeed, we may shuffle our series to our fancy by the absolute summability ensured by the condition $s > 1$ and the bounds $|C_{\mathscr{A}}(K, k)/k| \leq |G|$, as well as $|\mathscr{G}| \leq |G|$. This last bound follows from the fact that there are at most as many cyclic subgroups as there are possible generators.

1.2.3 3. Proof of Theorem 2 in general

Let us recall the Witt decomposition. The readers will find in Lemma 1 of *[9]* a result of the same flavour. We have simply modified the proof and setting as to accommodate polynomials having real numbers for coefficients.

Lemma 19. Let $F(t) = 1 + a_1 t + \ldots + a_\delta t^\delta \in \mathbb{R}[t]$ be a polynomial of degree δ. Let $\alpha_1, \ldots, \alpha_\delta$ be the inverses of its roots. Put $s_F(k) = \alpha_1^k + \ldots + \alpha_\delta^k$. The $s_F(k)$ are integers and satisfy the Newton-Girard recursion

$$s_F(k) + a_1 s_F(k-1) + \ldots + a_{k-1} s_F(1) + k a_k = 0, \tag{1.12}$$

where we have defined $a_{\delta+1} = a_{\delta+2} = \ldots = 0$. Put

$$b_F(k) = \frac{1}{k} \sum_{d | k} \mu(k/d) s_F(d). \tag{1.13}$$

Let $\beta \geq 1$ be such that $\beta \geq \max_j |1/\alpha_j|$. When t belongs to any segment $\subset (-\beta, \beta)$, we have

$$F(t) = \prod_{j=1}^\infty (1 - t^j)^{b_F(j)} \tag{1.14}$$

where the convergence is uniform in the given segment.

Proof. Since we follow the proof of Lemma 1 of *[9]*, we shall be rather sketchy. We write $F(t) = \prod_i (1 - \alpha_i t)$. By logarithmic differentiation, we obtain

$$\frac{tF'(t)}{F(t)} = \sum_i \frac{\alpha_i t}{1 - \alpha_i t} = \sum_{k \geq 1} s_F(k) t^k.$$

This series is absolutely convergent in any disc $|t| \leq b < 1/\beta$ where $\beta = \max_j(1/|\alpha_j|)$. We proceed by expressing s_F in terms of b_F via (1.13) in a disc of radius $b < 1/\beta$. After some shuffling of the terms, we reach the expression

$$\frac{tF'(t)}{F(t)} = \sum_{j \geq 1} b_F(j) \frac{jt^j}{1 - t^j}.$$

The lemma follows readily by integrating the above relation.

How does the mathematician E. Witt enter the scene? In the paper *[21]*, (11) therein is a decomposition that is the prototype of the above expansion.

Lemma 20. We use the hypotheses and notation of Lemma *19*. Let $\beta \geq 2$ be larger than the inverse of the modulus of all the roots of $F(t)$. We have

$$|b_F(k)| \leq 2 \deg F \cdot \beta^k / k.$$

Proof. We clearly have $|s_F(j)| \leq \deg F \cdot \beta^j$, so that

$$\begin{aligned} |b_F(k)| &\leq \frac{\deg F}{k} \sum_{1 \leq j \leq k} \beta^j \leq \frac{\deg F}{k} \beta \frac{\beta^k - 1}{\beta - 1} \\ &\leq \frac{\deg F}{k} \frac{\beta^k}{1 - 1/\beta} \leq 2 \deg F \frac{\beta^k}{k}. \end{aligned}$$

There are numerous easy upper estimates for the inverse of the modulus of all the roots of $F(t)$ in terms of its coefficients. Here is a simplistic one.

Lemma 21. Let $F(X) = 1 + a_1 X + \ldots + a_\delta X^\delta$ be a polynomial of degree δ. Let ρ be one of its roots. Then either $|\rho| \geq 1$ or $1/|\rho| \leq |a_1| + |a_2| + \ldots + |a_\delta|$.

Proof. On noticing that

$$(1/\rho)^\delta = -a_1 (1/\rho)^{\delta - 1} - a_2 (1/\rho)^{\delta - 2} - \ldots - a_\delta,$$

the conclusion follows.

Lemma 22. The sum $\sum_{L \in \mathcal{L}} \mu(|L|/|K|)$ where

$$\mathcal{L} = \{L \in \mathscr{G} / L^{[t]} = \langle \mathscr{A} \rangle \text{ and } K \subset L\}$$

depends only on $\gcd(t, \varphi(q))$.

Proof. Let us call this quantity $r_0(t)$. We first check that it depends only on $t \mod \varphi(q)$: this follows from the fact that the map $x \mapsto x^{\varphi(q)}$ reduces to the identity over G. Secondly, any prime factor of t, say p', that is prime to $\varphi(q)$, may be removed from t, i.e. $r_0(t) = r_0(t/p')$: the map $x \mapsto x^{p'}$ is one-to-one in L.

The lemma is an immediate consequence of these two remarks.

Proof of Theorem 2 . The proof requires several steps. The very first one is a direct consequence of (1.14), which leads to the identity

$$\frac{F(t)}{H(t)} = \prod_{j=\Delta}^{\infty} (1-t^j)^{b_F(j)-b_H(j)}. \tag{1.15}$$

The absence of the term with $j < \Delta$ is due to our assumption that $(F(X) - H(X))/X^\Delta \in \mathbb{R}[X]$. Up to this point (1.15) is only established as a formal identity. Our second step is to establish (1.15) for all $t \in \mathbb{C}$ with $|t| < 1/\beta$. By Lemma *20*, we know that $|b_F(j) - b_H(j)| \leq 4\max(\deg F, \deg H)\beta^j/j$. Therefore, for any bound J, we have

$$\sum_{j\geq J+1} |t^j||b_F(j) - b_H(j)| \leq 4\max(\deg F, \deg H)\frac{|t\beta|^{J+1}}{(1-|t\beta|)(J+1)}, \tag{1.16}$$

as soon as $|t| < 1/\beta$. We thus have

$$\frac{F(t)}{H(t)} = \prod_{\Delta\leq j\leq J} (1-t^j)^{b_F(j)-b_H(j)} \times I_1, \tag{1.17}$$

where $|\log I_1| \leq 4\max(\deg F, \deg H)|t\beta|^{J+1}/[(1-|t\beta|)(J+1)]$. Now that we have the expansion (1.17) for each prime p, we may combine them. We readily get

$$\prod_{\substack{p\geq P,\\ p+q\mathbb{Z}\in\mathcal{A}}} \frac{F(1/p^s)}{H(1/p^s)} = \prod_{\substack{p\geq P,\\ p+q\mathbb{Z}\in\mathcal{A}}} \prod_{\Delta\leq j\leq J} (1-p^{-js})^{b_F(j)-b_H(j)} \times I_2,$$

where I_2 satisfies

$$\begin{aligned}
\log I_2| &\leq 4\max(\deg F, \deg H) \sum_{p\geq P} \frac{\beta^{J+1}}{1-\beta/P^s}\frac{1}{(J+1)p^{(J+1)s}} \\
&\leq \frac{4\max(\deg F, \deg H)\beta^{J+1}}{(1-\beta/P^s)(J+1)}\left(\frac{1}{P^{(J+1)s}} + \int_P^\infty \frac{dt}{t^{(J+1)s}}\right) \\
&\leq \frac{4\max(\deg F, \deg H)(\beta/P^s)^J\beta}{(1-\beta/P^s)(J+1)}\left(\frac{1}{P^s} + \frac{1}{Js+s-1}\right),
\end{aligned}$$

since $P \geq 2$ and $J \geq 3$. Letting J go to infinity, we see that when $P^s > \beta$ and $s > 1/\Delta$,

$$\prod_{\substack{p\geq P,\\ p+q\mathbb{Z}\in\mathcal{A}}} \frac{F(1/p^s)}{H(1/p^s)} = \prod_{j\geq\Delta} \prod_{\substack{p\geq P,\\ p+q\mathbb{Z}\in\mathcal{A}}} (1-p^{-js})^{b_F(j)-b_H(j)} = \prod_{j\geq 2} \zeta_P(js; q, \mathscr{A})^{b_H(j)-b_F(j)}$$

in the notation of Theorem *18*. We use this theorem to infer that

$$\prod_{\substack{p\geq P,\\ p+q\mathbb{Z}\in\mathcal{A}}} \frac{F(1/p^s)}{H(1/p^s)} = \prod_{j\geq\Delta} \prod_{m\geq 1} \prod_{K\in\mathscr{G}} \Big(\prod_{\chi\in K^\perp} L_P(mjs,\chi)\Big)^{\frac{C_{\mathscr{A}}(K,m,1-X)}{m}(b_H(j)-b_F(j))}.$$

Notice that we have $s_H(j) - s_F(j) = 0$ (and hence $b_H(j) - b_F(j) = 0$) when $j < \Delta$ by our assumption on Δ. Let us glue the variables m and j in n. On using the definitions (1.11) and (1.13), we see that the functions $m \mapsto C_{\mathscr{A}}(K, m, 1-X)/m$ and $j \mapsto (b_H(j) - b_F(j))$ are of the form $(1 \star r)(m)/m$, respectively $(\mu \star (s_H - s_F))(j)/j$. Hence

$$n \sum_{jm=n} \frac{C_{\mathscr{A}}(K, m, 1-X)}{m}(b_H(j) - b_F(j)) = \sum_{td=n} r(t)\big(s_H(d) - s_F(d)\big).$$

We replace $r(t)$ by its value to conclude that this sum is $C_{\mathscr{A}}(K,m,F/H)$, as defined by (1.4). We have reached

$$\prod_{\substack{p\geq P,\\ p+q\mathbb{Z}\in\mathcal{A}}}\frac{F(1/p^s)}{H(1/p^s)}=\prod_{n\geq\Delta}\prod_{K\in\mathscr{G}}\Big(\prod_{\chi\in K^\perp}L_P(ns,\chi)\Big)^{\frac{C_{\mathscr{A}}(K,n,F/H)}{n}}. \tag{1.18}$$

The final task is to control the tail of this product, but prior to that, we change the variable n in (1.18) in m again. To control the tail, we check that, by Lemma *17*,

$$\begin{aligned}
\pm\log\prod_{m\geq M+1}\prod_{K\in\mathscr{G}}\Big(\prod_{\chi\in K^\perp}L_P(ms,\chi)\Big)^{\frac{C_{\mathscr{A}}(K,m,F/H)}{m}}&\\
&\leq\sum_{m\geq M+1}\sum_{K\in\mathscr{G}}\frac{|C_{\mathscr{A}}(K,m,F/H)|}{m}|G/K|\frac{ms-1+P}{P^{ms}}\\
&\leq\sum_{m\geq M+1}\sum_{K\in\mathscr{G}}\sum_{t|m}\mu^2(t)|\mathscr{G}|(\deg F+\deg H)\beta^{m/t}\frac{ms-1+P}{mP^{ms}}\\
&\leq(\deg F+\deg H)|\mathscr{G}|^2\sum_{m\geq M+1}\frac{\beta^m}{1-(1/\beta)}\frac{s+P}{P^{ms}}\\
&\leq(\deg F+\deg H)|\mathscr{G}|^2\frac{\beta(s+P)}{\beta-1}\frac{1}{1-(\beta/P^s)}\Big(\frac{\beta}{P^s}\Big)^{M+1}\\
&\leq 4(\deg F+\deg H)|\mathscr{G}|^2(s+P)\Big(\frac{\beta}{P^s}\Big)^{M+1}.
\end{aligned}$$

1.2.4 4. Link with two other sets of inequalities

In this section, we develop some elements that are contiguous to our topic.

A formula

Lemma 23. Let $q>1$ be a modulus. We set G_0 to be a subgroup of $G=(\mathbb{Z}/q\mathbb{Z})^\times$ and $G_0^\perp$ be the subgroup of characters that take the value 1 on G_0. For any integer b, we define $\langle b\rangle$ to be the subgroup generated by b modulo q. We have

$$\prod_{\chi\in G_0^\perp}L_P(s,\chi)=\prod_{G_0\subset K\subset G}\prod_{\substack{p\geq P,\\ \langle p\rangle G_0=K}}\Big(1-p^{-|K/G_0|s}\Big)^{-|G/K|}.$$

The right-hand side of this formula contains products of the kind we seek and, if we were to start from such a set of formulas, the problem would be to *invert* them in some sense.

Proof. We note that $\prod_{\chi\in G_0^\perp}(1-\chi(p)z)^{\chi(a)}=\prod_{\psi\in\hat{L}}(1-\psi(p)z)^{f(\psi)}$ when $\langle p\rangle=L$ and where

$$f(\psi)=\sum_{\substack{\chi\in G_0^\perp,\\ \chi|L=\psi}}\chi(a). \tag{1.19}$$

The condition $\chi\in G_0^\perp$ can also be written as $\chi|G_0=1$, hence we can assume that $\psi|(L\cap G_0)=1$. We write

$$\prod_{\chi\in G_0^\perp}(1-\chi(p)z)^{\chi(a)}=\prod_{\substack{\psi'\in\widehat{LG_0},\\ \psi'|G_0=1}}(1-\psi(p)z)^{f'(\psi')},$$

where

$$f'(\psi') = \sum_{\substack{\chi \in G_0^\perp, \\ \chi | LG_0 = \psi}} \chi(a). \tag{1.20}$$

When a lies outside LG_0, this sum vanishes; otherwise it equals $|G/(LG_0)|\psi'(a)$. The characters of LG_0 that are trivial on G_0 are canonically identified with the characters of the cyclic group $(LG_0)/G_0$. We thus have

$$\prod_{\substack{\psi' \in \widehat{LG_0}, \\ \psi' | G_0 = 1}} (1 - \psi(p)z) = 1 - z^{|(LG_0)/G_0|},$$

and this proves our formula.

Notes on the scope of Lemma 23

From a metholodogical viewpoint, a moment's thought discloses that two residue classes modulo q that fall inside the same lattice-invariant class cannot be distinguished by the set of identities of Lemma *23*. This implies that we indeed extract the maximum information from our setting. This could be formalized in the following manner: consider the vector space $\mathscr{F}[G]$ of functions from G to $\mathbb{C}$, and the sub-space generated by $(1_{G_0})_{G_0 \in \mathscr{G}}$. This subspace is clearly included in the subspace generated by $(1_{\mathcal{A}})_{\mathcal{A} \in G^\sharp}$. These two spaces can be shown to be equal. We end this discussion here, as we do not need this fact.

Link with abelian field theory

The case $G_0 = \{1\}$ in the identity of Lemma *23* is classical in Dedekind zeta function theory for the field $\mathbb{Q}(\zeta_q)$, where $\zeta_q = \exp(2i\pi/q)$, and can be found in Proposition 13 of *[13]* in a rephrased form. For the general case, we follow Chapter 8 of *[11]* by Narkiewicz. The Dedekind zeta-function associated with an abelian field K is given by

$$\zeta_K(s) = \prod_{\chi \in X(K)} L(s, \chi) \tag{1.21}$$

as per Theorem 8.6 of *[11]*. The group $X(K)$ is the group of characters attached to K, see Proposition 8.4 of *[11]*. This equality (1.21) is proved prime per prime, and we can restrict to ideals whose norm is prime to some integer. In particular, we can restrict it to the primes that are prime to q, which excludes at least the ramified primes. Let $H_q(K)$ be the subgroup of the integers $r \mod q$ that are such that the automorphism of $\mathbb{Q}(\zeta_q)$ defined by $\zeta_q \mapsto \zeta_q^r$ is the identity on K. The sets $X(K)$ and $H_q(K)^\perp$ are almost equal: $X(K)$ is made only of primitive characters associated to the characters in $H_q(K)^\perp$. We may select $G_0 = H_q(K)$ in Lemma *23<23>*. Some work involving the decomposition law in abelian number fields, which may for instance be found in Theorem 8.2 *[11]*, gives us, when the prime factors of q are all at most P, that

$$\prod_{\chi \in X(K)} L_P(s, \chi) = \prod_{H_q(K) \subset K \subset G_q} \prod_{\substack{p \geq P, \\ \langle p \rangle H_q(K) = K}} \left(1 - p^{-|K/H_q(K)|s}\right)^{-|G_q/K|}.$$

The proof we provide of Lemma *23* is much simpler, but the above analysis establishes that the identities stemming from both approaches are the same.

1.2.5 5. Timing and implementation notes

Let $s > 1$ be a real number and $P \geq 2$ be a parameter. We consider the vector, for any positive integer t:

$$\Gamma_{P,s}(t) = \left(\log \prod_{\chi \in G_0^\perp} L_P(ts, \chi)\right)_{G_0 \in \mathscr{G}}. \tag{1.22}$$

The rows of $\Gamma_{P,s}(t)$ are indexed by the cyclic subgroups of G. An approximate value of this vector is provided by the function `ComponentStructure.get_gamma` from the values of the Hurwitz zeta function. We next define

$$V_s(t) = \big(\zeta_P(ts; q, \mathcal{A})\big)_{\mathcal{A}\in G^\sharp}. \tag{1.23}$$

The class `LatticeInvariant` gives the two lists: the one of the cyclic subgroups and the one of their generators, ordered similarly and in increasing size of the subgroups.

The algorithm (function `get_vs`):

Input

Input the four parameters `q`, `s`, `nbdecimals` and `big_p` as well as the two parameters that control the output `verbose` and `with_laTeX`.

Precomputation-1

Compute and store the algebraic quantities that we need: the tuple of cyclic subgroups of $G = (\mathbb{Z}/q\mathbb{Z})^\times$, the tuple of its lattice-invariant classes, the exponent of G, its character group, an enumeration of the elements of G and, for each cyclic subgroup of G, the set of characters of G that are trivial on it. This is done by the class `ComponentStructure`.

Initialization

Find `M` so that the right-hand side of (1.6) is less than $10^{-\texttt{nbdecimals}-10}$.

Precomputation-2

Build the set $\mathscr{M}$ of integers m such that $m \le M$ and all the prime factors of m divide q. Then compute the constants $(C_{\mathscr{A}}(K, m, 1-X))$ for every possible class $\mathscr{A}$ and every m in $\mathscr{M}$.

Main Loop

For $m \in \mathscr{M}$, add the contribution of this index to the sum approximating $V_s(1)$ from the right-hand side of (1.5) with $:P = \texttt{big\_p}$.

Post-computation

Complete the products with the values for primes $p <$ `big_p`.

Output

Return the tuple of lattice-invariant classes and the tuple of couples of lower/upper bounds for the wanted Euler products.

Once the script is loaded, a typical call will be

```
get_vs(12, 2, 100, 110)
```

to compute modulo 12 the possible constants with $s = 2$, asking for 100 decimal digits and using $P = 110$. The output is self explanatory. The number of decimal digits asked for is roughly handled and one may lose precision in between, but this is indicated at the end. Note that we expect the final result to be of size roughly unity, so what we ask for is not the relative precision but the number of decimals. Hence, in the function `get_gamma`, we replace by an approximation of 0 the values that we know are insignificantly small. This is a true time-saver.

There are two subsequent optional parameters `verbose` and `with_laTeX`. The first one may take the values 0, 1 and 2; when equal to 0, the function will simply do its job and return the tuple of the invariant classes and the one of the computed lower and upper values. When equal to 1, the time taken will also be printed. And when equal to 2, its default value, some information on the computation is given. When the parameter `verbose` is at least 2 and `with_laTeX` is 1, the values of the constants will be further presented in a format suitable for inclusion in a LaTeX-file. The digits presented in LaTeX-format when `with_laTeX` $= 1$ are always accurate. For instance, the call `get_vs(12, 2, 100,`

`100, 2, 1)` is the one used to prepare the addendum *[2]* in which we give the first hundred decimal digits of every Euler product over a lattice invariant class when the modulus is at most 16.

The computations of the Euler products of Theorem *2* (with $P = 2$, the parameter `big_p` being used to decide from which point onwards we use the usual Euler product or the expression of the theorem) is implemented in:

The parameter `big_p` may be increased by the script to ensure that $P \geq 2\beta$ (a condition that is usually satisfied). We reused the same structure as the one for the function `get_vs` except that the set of indices m is now a full interval. Since the coefficients $|b_F(j) - b_G(j)|$ may increase like β^j, we increase the working precision by $J \log \beta / \log 2$.

Checking

The values given here have been checked in several ways. The co-authors of this paper have run several independent scripts. We also provide the function `get_vs_checker(q, s, borne = 10000)` which computes approximate values of the same Euler products by simply truncating the Euler product representation. We checked with positive result the stability of our results with respect of the variation of the parameter P. This proved to be a very discriminating test.

Furthermore, approximate values for Shanks' and Lal's constants are known (Finch in *[4]* gives 10 digits) and we agree with those. Finally, the web site *[7]* by X. Gourdon and P. Sebah, or the attached postscript file on the same page, gives in section 4.4 the first fifty digits of the constant they call A and which are

$$\frac{\pi^2}{2} \prod_{p \equiv 1[4]} \left(1 - \frac{4}{p}\right) \left(\frac{p+1}{p-1}\right)^2 = \begin{array}{l} 1.95049\,11124\,46287\,07444\,65855\,65809\,55369 \\ \quad 25267\,08497\,71894\,30550\,80726\,33188\,94627 \\ \quad 61381\,60369\,39924\,26646\,98594\,38665\cdots \end{array} \tag{1.24}$$

Our result matches that of *[7]*.

Some observations on the running time and complexity

We tried several large computations to get an idea of the limitations of our script with the choice $s = 2$ in Corollary *8*. We present five tables:

- A first table for $3 \leq q \leq 100$ with the uniform choice $P = 100$ and asking for 100 decimal digits.
- Three further tables obtained with the choice $P = 200$ and asking for a thousand decimal digits. The cases retained are $q \leq 16$, $91 \leq q \leq 100$ and $200 \leq q \leq 220$. This last interval contains the first integer q such that $r = \omega(\varphi(q)) = 4$, namely $q = 211$.
- And finally a table for $q \in \{3, 5\}$ and asking for 5000 decimal digits. The running time is given with different choices of the parameter P.

Since we did not run each computation hundred times to get an average timing, these tables have to be taken with a pinch of salt. The processor was an Intel Core i5-2500 at 3.30 GHz. The first half of Table *2* may be reproduced with the call:

```
table_performance(3, 51, 100, 100)
```

In these tables, $r = \omega(\varphi(q))$ is the number of distinct prime divisors of $\varphi(q)$ as in Corollary *8*. The time is given in tenth of a second, indicated by "s/10". The column with the tag "$\#m's$" contains the number of indices $m \leq M$ such that $m|\varphi(q)^\infty$. We otherwise follow the notation of Theorem *2*.

It seems likely, when looking at Tables *2*, *3*, *4* and *5* that the number of values of the Hurwitz zeta-function to be computed is the main determining factor of the time consumption. This number is controlled by $\varphi(q)$, since this is the number of characters, and by the number of m's required, a value that is on the whole controlled by $r = \omega(\varphi(q))$.

Table 1: Time used when asking for 1000 digits for $q \leq 16$

q	$\varphi(q)$	r	$\#m's$	$\|G^\sharp\|$	M	Time (s/10)
3	2	1	8	2	218	10
4	2	1	8	2	218	7
5	4	1	8	3	218	14
7	6	2	26	4	218	69
8	4	1	8	4	218	12
9	6	2	26	4	218	67
11	10	2	19	4	218	81
12	4	1	8	4	218	14
13	12	2	26	6	218	135
15	8	1	8	6	218	26
16	8	1	8	6	218	24

Table 2: Time used when asking for 1000 digits for $90 < q \leq 100$

q	$\varphi(q)$	r	$\#m's$	$\|G^\sharp\|$	M	Time (s/10)
91	72	2	26	30	219	910
92	44	2	14	8	218	286
93	60	3	47	16	219	1388
95	72	2	26	18	218	912
96	32	1	8	16	218	114
97	96	2	26	12	218	1257
99	60	3	47	16	219	1399
100	40	2	19	12	218	363

Table 3: Time used when asking for 1000 digits for $200 \leq q \leq 220$

q	$\varphi(q)$	r	$\#m's$	$\|G^\sharp\|$	M	Time (s/10)
200	80	2	19	24	218	759
201	132	3	37	16	218	2543
203	168	3	42	24	219	3767
204	64	1	8	20	218	240
205	160	2	19	28	219	1573
207	132	3	37	16	218	2520
208	96	2	26	40	219	1259
209	180	3	47	24	219	4552
211	210	4	69	16	219	8406
212	104	2	14	12	218	743
213	140	3	31	16	218	2271
215	168	3	42	24	219	3807
216	72	2	26	24	219	930
217	180	3	47	40	219	4517
219	144	2	26	24	219	1970
220	80	2	19	24	218	753

Table *6* gives some data about the running time when asking for 5000 decimal digits, which essentially sets the horizon of the present method. The time is counted in minutes.

Table 4: Time used when asking for 5000 digits

q	P	time
3	200	80m
3	400	35m
3	500	35m
5	500	72m
5	1000	70m
5	5000	72m

[1] H. Cohen, *High precision computations of Hardy-Littlewood constants*, preprint (1996), 1–19.

[2] S. Ettahri, O. Ramaré and L. Surel, *Some Euler Products*, Preprint (2020), 4p, Addendum to 'Fast multi-precision computation of some Euler products'.

[3] J.-H. Evertse, *On sums of S -units and linear recurrences*, Compositio Math. **53** (1984), no. 2, 225–244. MR 766298

[4] S. R. Finch, *Mathematical constants*, Encyclopedia of Mathematics and its Applications, vol. 94, Cambridge University Press, Cambridge, 2003. MR 2003519

[5] P. Flajolet and I. Vardi, *Zeta function expansions of classical constants*, preprint (1996), 1–10.

[6] É. Fouvry, C. Levesque and M. Waldschmidt, *Representation of integers by cyclotomic binary forms*, Acta Arith. **184** (2018), no. 1, 67–86. MR 3826641

[7] X. Gourdon and P. Sebah, *Constants from number theory*, http://numbers.computation.free.fr/Constants/constants.html (2010).

[8] M. Lal, *Primes of the form $n^4 + 1$*, Math. Comp. **21** (1967), 245–247. MR 0222007

[9] P. Moree, *Approximation of singular series constant and automata. with an appendix by Gerhard Niklasch.*, Manuscripta Matematica **101** (2000), no. 3, 385–399.

[10] P. Moree, *On the average number of elements in a finite field with order or index in a prescribed residue class*, Finite Fields Appl. **10** (2004), no. 3, 438–463. MR 2067608

[11] W. Narkiewicz, *Elementary and analytic theory of algebraic numbers*, third ed., Springer Monographs in Mathematics, Springer-Verlag, Berlin, 2004. MR 2078267 (2005c:11131)

[12] OEIS Foundation Inc., *The on-line encyclopedia of integer sequence*, 2019, http://OEIS.org/.

[13] J.-P. Serre, *Cours d'arithmétique*, Collection SUP: "Le Mathématicien", vol. 2, Presses Universitaires de France, Paris, 1970. MR 0255476

[14] D. Shanks, *On the conjecture of Hardy & Littlewood concerning the number of primes of the form $n^2 + a$*, Math. Comp. **14** (1960), 320–332. MR 0120203

[15] D. Shanks, *On numbers of the form :math: `n^{4}+1 `*, Math. Comput. **15** (1961), 186–189. MR 0120184

[16] D. Shanks, *The second-order term in the asymptotic expansion of $B(x)$*, Math. Comp. **18** (1964), 75–86. MR 0159174

[17] D. Shanks, *Lal's constant and generalizations*, Math. Comp. **21** (1967), 705–707. MR 0223315

[18] L. Tóth, *Menon's identity and arithmetical sums representing functions of several variables*, Rend. Semin. Mat. Univ. Politec. Torino **69** (2011), no. 1, 97–110. MR 2884710

[19] L. Tóth, *On the number of cyclic subgroups of a finite Abelian group*, Bull. Math. Soc. Sci. Math. Roumanie (N.S.) **55(103)** (2012), no. 4, 423–428. MR 2963406

[20] A. J. van der Poorten and H. P. Schlickewei, *Zeros of recurrence sequences*, Bull. Austral. Math. Soc. **44** (1991), no. 2, 215–223. MR 1126359

[21] E. Witt, *Treue Darstellung Liescher Ringe*, J. Reine Angew. Math. **177** (1937), 152–160. MR 1581553

1.3 Installation EULER_PRODUCT

1.3.1 Euler Product for SageMath

Computing Lattice Invariant Euler Products

The **sage-euler-product** package for SageMath adds functionality related to Number Theory. It is based on SageMath https://www.sagemath.org_ and relies heavily on:

- gmp or mpir for arbitrary precision arithmetic
- PARI/GP for number field computations

Prerequisites

Installing sage-euler-product requires a working Sage installation.

Installation from PyPI in an existing Sage installation built from source

The module is distributed on PyPI and is easily installed through the Python package manager pip. Switch to the source directory (SAGE_ROOT) of your Sage installation, and run the following command:

```
$ sage -pip install sage-euler-product [--user]
```

The `--user` option is optional and allows to install the module in your user space (and does not require administrator rights).

If you use Debian or Ubuntu and you installed Sage through the operating system's package manager (that is, the package sagemath), run these two commands:

```
$ source /usr/share/sagemath/bin/sage-env
$ pip install sage-euler-product --user
```

If you use Arch Linux, you need to install from source (see next section).

Installation of the development version from GitHub in an existing Sage installation

This section provides detailed instructions on how to download, modify and install the development version of **sage-euler-product**. In all commands,

PIP has to be replaced by either pip, pip2, or sage -pip PYTHON has to be replaced by either python, python2 or sage -python If you are an Arch Linux user with the sagemath package installed, use PIP=pip2 and PYTHON=python2. If you downloaded SageMath as a tarball or installed it from source use PIP='sage -pip' and PYTHON='sage -python'.

You can install the latest development version in one line with:

```
$ PIP install git+https://github.com/archimede-institut/sage-euler-product [--user]
```

As before, the –user option is optional and when specified will install the module in your user space.

You can also perform a two stage installation that will allow you to modify the source code. The first step is to clone the repository:

```
$ git clone https://github.com/archimede-institut/sage-euler-product
```

The above command creates a repository sage-euler-product with the source code, documentation and miscellaneous files. You can then change to the directory thus created and install the surface dynamics module with:

```
$ cd sage-euler-product
$ PIP install . [--user]
```

Do not forget the . that refers to the current directory.

When you don't want to install the package or you are testing some modifications to the source code, a more convenient way of using **sage-euler-prodct** is to do everything locally. Once done, you can import the sage-euler-product module. To check that you are actually using the right module (i.e. the local one) you can do in a SageMath session:

```
    sage: import euler_product
    sage: euler_product.__path__        # random
    ['/home/you/sage-euler-product/euler_product/']
```

The result of the command must correspond to the path of the repository created by the command git clone given above.

If you wish to install your custom version of sage-euler-product just use PIP as indicated before.

Installation in a virtual Python environment (no prior Sage installation required)

Create and activate a virtual environment:

```
python3 -m venv venv-euler-product
. venv-euler-product/bin/activate
```

Install the package in the virtual environment:

```
pip install "sage-euler-product[passagemath] @ git+https://github.com/archimede-institut/
↪sage-euler-product
```

This automatically installs the modularized parts of the Sage library that are needed by the package. (These modularized distributions are provided by https://github.com/passagemath.)

Next, start Sage:

```
rehash
sage
```

At the Sage prompt, load a modularized top-level environment:

```
sage: from sage.all__sagemath_schemes import *
```

Documentation

complete module documentation: https://archimede-institut.github.io/sage-euler-product/

Check

After installing **sage-euler-product**, check that it works by launching Sage and typing the following commands. You should get the same output as below.

```
   sage: from euler_product.all import *
   sage: from euler_product.lattice_invariant_euler_produ import get_euler_products
   sage: get_euler_products(3, 1, 1-x^2,1, 100)
   Computing the structural invariants ...  done.
   We have Delta  = 2 and beta = 2
   We use big_m = 310 , big_p = 300 and working prec = 653 .
   Computing the finite products for p < 300 ...  done.
   Computing C_A(K, m, F/H) ... --------------------
   For p+3ZZ in  frozenset({1})
   For F(x) = -x^2 + 1
   and H(x) = 1
   the product of F(1/p)/H(1/p) is between
   0.
↪9671040753637981066150556834173635260473412207450092130719978569438733967843271277395717230016746853806
   and
   0.
↪9671040753637981066150556834173635260473412207450092130719978569438733967843271277395717230016746853806
   (Obtained:  193  correct decimal digits)
   --------------------
   For p+3ZZ in  frozenset({2})
   For F(x) = -x^2 + 1
   and H(x) = 1
   the product of F(1/p)/H(1/p) is between
   0.
↪7071813747951674302088659938984504109243584468119496848353517677901518159831128643782536704398941052123
   and
   0.
↪7071813747951674302088659938984504109243584468119496848353517677901518159831128643782536704398941052123
   (Obtained:  193  correct decimal digits)
   Time taken:  1.920718120993115 seconds.
   ((frozenset({1}), frozenset({2})),
    ((0.
↪9671040753637981066150556834173635260473412207450092130719978569438733967843271277395717230016746853806
↪
      0.
↪9671040753637981066150556834173635260473412207450092130719978569438733967843271277395717230016746853806
↪
     (0.
↪7071813747951674302088659938984504109243584468119496848353517677901518159831128643782536704398941052123
↪
      0.
↪7071813747951674302088659938984504109243584468119496848353517677901518159831128643782536704398941052123
```

https://github.com/archimede-institut/sage-euler-product Assuming you have the program git on your computer, you can install the development version with the command:

```
   $ sage -pip install git+https://github.com/archimede-institut/sage-euler-product [--
↪user]
```

Authors

Olivier Ramarè: see https://ramare-olivier.github.io/Maths/mcom3630.pdf for complete Mathematical references

Dominique Benielli: maintainerDeveloppement Cell, Institut Archimède Aix-Marseille Université

How to cite this project

If you have used this project for please cite us as described on our zenodo site.

Versions

The first release of sage-euler-product will appear soon as a sagemath spkg.

.

1.4 API Documentation

1.4.1 euler_product.lattice_invariant_euler_products

The main function of this package is `get_euler_products` which computes with interval arithmetic and a proven precision Euler products of rational functions over primes in special sets modulo some fixed `q`. These special sets are the lattice invariant classes modulo `q`, and the software also enables the user to use them through the class `ComponentStructure`.

AUTHORS:

- Olivier Ramaré: initial version
- Dominique Benielli Aix Marseille Université , Integration as SageMath package. Cellule de developpement Institut Archimède

...WARNING:

Needs Sage version at least 9.0 CAREFUL, this is Python 3 code!

EXAMPLES:

```
sage: from euler_product.lattice_invariant_euler_products import get_euler_products
```

euler_product.lattice_invariant_euler_products.**get_euler_products**(*q*, *s*, *f_init*, *h_init*, *nb_decimals=100*, *big_p=300*, *verbose=2*, *with_laTeX=0*, *digits_offset=10*)

Returns the pair ((A), (approx_prod_(p in A mod q) f_init(1/p^s) / h_init(1/p^s))) where (A) is the tuple of the lattice-invariant classes modulo `q` and approx_prod_(p in A mod q) f_init(1/p^s) / h_init(1/ps)) is an arithmetic interval approximation of the product over every prime in the class A modulo q of the quotient f_init(1/p^s) / h_init(1/p^s) given in the form of a pair (lower_bound, upper_bound). We expect the difference upper_bound - lower bound to be < 10^(-nb_decimals) but this is not guaranteed. In case it does not happen, increase `nb_decimals` slightly. We ask at the beginning for *digits_offset* more (binary) digits. We compute directly what happens for primes < `big_p`. We assume that f_init(0) = h_init(0) = 1, that s is a positive real number and that $\Delta s > 1$ where Δ is the order of the zero of f_init-h_init at 0. This last condition is to ensure the Euler products converge absolutely. See Theorem 2 of *the reference file*.

to do

assert F[0] = H[0] = 1

INPUT:

- **q – int**
  a positive integer. The products are taken over classes modulo q.
- **s – int, rat or real number**
  A real number > 0. It should be given with enough precision to enable the computations, so either an exact type or a `RealIntervalField(...)` number, given with enough precision. As this precision is given in binary digits, using `10*nb_decimals` is a safe choice. Notice that, if you want to have `s = 2.1`, better use `21/10`. Additional conditions may be required for the Euler products to be absolutely convergent.
- **f_init – pol**
  a polynomial with real coefficients and such that f_init(0) = 1.
- **h_init – pol**
  a polynomial with real coefficients and such that h_init(0) = 1.
- **nb_decimals – int (default: *100*), optional**
  The number of decimals that are being sought by the final result. The function aims at such a number of decimals but a final tuning may be required.
- **big_p – int (default:*300*), optional**
  This is an internal parameter that is described in the accompanying paper. In short: the Euler products up to `big_p` are computed directly.
- **verbose – int (default: *2*), optional**
  Defines the amount of output shown. It may take the usual values 0, 1, 2, towards more explanations. When `get_vs` is used inside another function, `verbose == 0` is usually what is required. The value -1 is special and the effect is fully described in the tutorial.
- **with_laTeX – int (default: *0*), optional**
  This parameter takes the value either 1 or not 1. As of now, this has effect only when `verbose == 2`.
- **digits_offset – int (default: *10*), optional**
  Not used yet.

OUTPUT:

pair of tuples
The output is a pair whose first component is the tuple of lattice invariant classes (A) and second component is the corresponding tuple of values (prod_(p in A mod q) f_init(1/p^s) / h_init(1/p^s)) where each value is given in interval arithmetic as a pair (lower_bound, upper_bound).

EXCEPTIONS:
ValueError ('non convergent product') ValueError("f_init[0] and h_init[0] must be equal to 1")

EXAMPLES:

```
sage: from euler_product.lattice_invariant_euler_products import get_euler_products
sage: get_euler_products(7, 21/10, 1-x^3, 1+2*x^2, 100, 100, 0)  # doctest:␣
↪+NORMALIZE_WHITESPACE
((frozenset({1}), frozenset({6}), frozenset({2, 4}), frozenset({3, 5})),
 ((0.
↪99999823912367711745827580431839013389423649012356992175226010629313359180602397234537364091027400
↪
   0.
↪99999823912367711745827580431839013389423649012356992175226010629313359180602397234537364091027400
↪
  (0.
↪99995761368844173980775596258481300888856563517407872651122270712171556827250327215896617394812650
```

(continues on next page)

(continued from previous page)

```
↪
   0.
↪999957613688441739807755962584813008888565635174078726511222707121715568272503272158966173948126
↪
  (0.
↪890335106507001059161987036491609346200032054103792800828641436164791111861714900452844442892724
↪
   0.
↪890335106507001059161987036491609346200032054103792800828641436164791111861714900452844442892724
↪
  (0.
↪977268647889313785438818426684454589590611565775849920828973330248423958982660329471898191872225
↪
   0.
↪977268647889313785438818426684454589590611565775849920828973330248423958982660329471898191872225

sage: from euler_product.lattice_invariant_euler_products import get_euler_products
sage: ss = RealIntervalField(1000)(2.1)
sage: get_euler_products(7, ss, 1-x^3, 1+2*x^2, 100, 100, 0)  # doctest: +NORMALIZE_
↪WHITESPACE
((frozenset({1}), frozenset({6}), frozenset({2, 4}), frozenset({3, 5})),
 ((0.
↪999998239123677117459356302984516588894992503080246873187990734037641740944825880497742514543227
↪
   0.
↪999998239123677117459356302984516588894992503080246873187990734037641740944825880497742514543227
↪
 (0.
↪999957613688441739827169019893858024837305107070016588117296855953370246777495422394908263831831
↪
  0.
↪999957613688441739827169019893858024837305107070016588117296855953370246777495422394908263831831
↪
 (0.
↪890335106507001072068827935941757794345031587895501744932220666670675300062403565358528659168504
↪
  0.
↪890335106507001072068827935941757794345031587895501744932220666670675300062403565358528659168504
↪
 (0.
↪977268647889313790103048997724909864420707828425677297780760716081387595772468604769299949053096
↪
  0.
↪977268647889313790103048997724909864420707828425677297780760716081387595772468604769299949053096
```

euler_product.lattice_invariant_euler_products.**get_vs**(*q*, *s*, *nb_decimals=100*, *big_p=100*, *verbose=2*, *with_laTeX=0*, *digits_offset=10*)

Returns the pair ((A), (approx_zeta(s; q, A))) where (A) is the tuple of the lattice-invariant classes modulo q and approx_zeta(s; q, A) is an arithmetic interval approximation of $\zeta(s; q, A) = \prod_{p \in A}(1-p^{-s})^{-1}$ given in the form of a pair (lower_bound, upper_bound).

We expect the difference upper_bound - lower bound to be < 10^(-nb_decimals) but this is not guaranteed. In case it does not happen, increase nb_decimals slightly. We compute directly what happens for primes < big_p.

We ask at the beginning for *digits_offset* more (binary) digits.

INPUT:

- **q – int**
  The products are taken over classes modulo q.
- **s – int, rat or real number**
  A real number > 1. It should be given with enough precision to enable the computations, so either an exact type or a RealIntervalField(...) number, given with enough precision. As this precision is given in binary digits, using 10*nb_decimals is a safe choice. Notice that, if you want to have s = 2.1, better use 21/10.
- **nb_decimals – int (default: *100*)**
  The number of decimals that are being sought by the final result. The function aims at such a number of decimals but a final tuning may be required.
- **big_p – int (default: *100*), optional**
  This is an internal parameter that is described in the accompanying paper. In short: the Euler products up to big_p are computed directly.
- **verbose – int (default: *2*), optional**
  Defines the amount of output shown. It may take the usual values 0, 1, 2, towards more explanations. When get_vs is used inside another function, verbose = 0 is usually what is required. The value -1 is special and the effect is fully described in the tutorial.
- **with_laTeX – int (default: *0*), optional**
  This parameter takes the value 1 or not 1. As of now, this has effect only when verbose == 2.
- **digits_offset – int (default: *10*), optional**
  We ask for some more digits, see above.

OUTPUT:

pair of tuples
The output is a pair whose first component is the tuple of lattice invariant classes (A) and second component is the corresponding tuple of values $(\zeta(s; q, A))$ where each value is given in interval arithmetic as a pair (lower_bound, upper_bound).

EXAMPLES:

```
sage: from euler_product.lattice_invariant_euler_products import get_vs
sage: get_vs(8, 3, 100) # doctest: +NORMALIZE_WHITESPACE
Computing the structural invariants ...  done.
Computing big m ... Computing the finite product for p < 100 ...  done.
done: we use big_m = 18 .
Building indices ... done: there are 5 summands.
-------------------
For p + 8ZZ in frozenset({1})
the product of 1 / (1 - p^{-3}) is between
1.
↪000224871898587088362322133991716493917374715169707098768922160318944604461086152506405263996291
and
1.
↪000224871898587088362322133991716493917374715169707098768922160318944604461086152506405263996291
(Obtained:  104  correct decimal digits)
-------------------
For p + 8ZZ in frozenset({3})
the product of 1 / (1 - p^{-3}) is between
```

(continues on next page)

(continued from previous page)

```
1.
↪039419954424652697264660284148088446555619388245204176694186772658250339289033950950041989947721
and
1.
↪039419954424652697264660284148088446555619388245204176694186772658250339289033950950041989947721
(Obtained:  105  correct decimal digits)
--------------------
For p + 8ZZ in frozenset({5})
the product of 1 / (1 - p^{-3}) is between
1.
↪008599296670352624712823936589306459743031871985271230389156441692272737589887757282575406594017
and
1.
↪008599296670352624712823936589306459743031871985271230389156441692272737589887757282575406594017
(Obtained:  105  correct decimal digits)
--------------------
For p + 8ZZ in frozenset({7})
the product of 1 / (1 - p^{-3}) is between
1.
↪003057245261110788414199619032412511287762245545446425765049343277053803735587622792046765975162
and
1.
↪003057245261110788414199619032412511287762245545446425765049343277053803735587622792046765975162
(Obtained:  105  correct decimal digits)
((frozenset({1}), frozenset({3}), frozenset({5}), frozenset({7})),
 ((1.
↪000224871898587088362322133991716493917374715169707098768922160318944604461086152506405263996291
↪
   1.
↪000224871898587088362322133991716493917374715169707098768922160318944604461086152506405263996291
↪
  (1.
↪039419954424652697264660284148088446555619388245204176694186772658250339289033950950041989947721
↪
   1.
↪039419954424652697264660284148088446555619388245204176694186772658250339289033950950041989947721
↪
  (1.
↪008599296670352624712823936589306459743031871985271230389156441692272737589887757282575406594017
↪
   1.
↪008599296670352624712823936589306459743031871985271230389156441692272737589887757282575406594017
↪
  (1.
↪003057245261110788414199619032412511287762245545446425765049343277053803735587622792046765975162
↪
   1.
↪003057245261110788414199619032412511287762245545446425765049343277053803735587622792046765975162

sage: from euler_product.lattice_invariant_euler_products import get_vs
sage: ss = RealIntervalField(1000)(2.1)
sage: get_vs(7, ss, 100) # doctest: +NORMALIZE_WHITESPACE
```

(continues on next page)

(continued from previous page)

```
Computing the structural invariants ...  done.
Computing big m ... Computing the finite product for p < 100 ...  done.
done: we use big_m = 25 .
Building indices ... done: there are 11 summands.
--------------------
For p + 7ZZ in frozenset({1})
the product of 1 / (1 - p^{-2.
↪1000000000000000888178419700125232338905334472656250000000000000000000000000000000000000000000000000000000000000
↪}) is between
1.
↪0015251649887938725660913688010517101666208733880109315688816926296884362067543474803469725091159
and
1.
↪0015251649887938725660913688010517101666208733880109315688816926296884362067543474803469725091159
(Obtained:  102  correct decimal digits)
--------------------
For p + 7ZZ in frozenset({6})
the product of 1 / (1 - p^{-2.
↪1000000000000000888178419700125232338905334472656250000000000000000000000000000000000000000000000000000000000000
↪}) is between
1.
↪0053143647905533679330453141809294714882943247539536696871604072988921328496577468083535690636615
and
1.
↪0053143647905533679330453141809294714882943247539536696871604072988921328496577468083535690636615
(Obtained:  100  correct decimal digits)
--------------------
For p + 7ZZ in frozenset({2, 4})
the product of 1 / (1 - p^{-2.
↪1000000000000000888178419700125232338905334472656250000000000000000000000000000000000000000000000000000000000000
↪}) is between
1.
↪3163847262351936805824813658507662946352860144404616097232556176854606556798278964523547342562974
and
1.
↪3163847262351936805824813658507662946352860144404616097232556176854606556798278964523547342562974
(Obtained:  102  correct decimal digits)
--------------------
For p + 7ZZ in frozenset({3, 5})
the product of 1 / (1 - p^{-2.
↪1000000000000000888178419700125232338905334472656250000000000000000000000000000000000000000000000000000000000000
↪}) is between
1.
↪1573918393315763316569087551275677116540398269978595705035335829359240632731578772727923341501394
and
1.
↪1573918393315763316569087551275677116540398269978595705035335829359240632731578772727923341501394
(Obtained:  102  correct decimal digits)
((frozenset({1}), frozenset({6}), frozenset({2, 4}), frozenset({3, 5})),
 ((1.
↪0015251649887938725660913688010517101666208733880109315688816926296884362067543474803469725091159
↪
```

(continues on next page)

(continued from previous page)

```
   1.
↪00152516498879387256609136880105171016662087338801093156888169262968843620675434748034697250911599
↪
  (1.
↪00531436479055336793304531418092947148829432475395366968716040729889213284965774680835356906366155
↪
   1.
↪00531436479055336793304531418092947148829432475395366968716040729889213284965774680835356906366155
↪
  (1.
↪31638472623519368058248136585076629463528601444046160972325561768546065567982789645235473425629744
↪
   1.
↪31638472623519368058248136585076629463528601444046160972325561768546065567982789645235473425629744
↪
  (1.
↪15739183933157633165690875512756771165403982699785957050353358293592406327315787727279233415013944
↪
   1.
↪15739183933157633165690875512756771165403982699785957050353358293592406327315787727279233415013944
```

TESTS:

```
sage: from euler_product.lattice_invariant_euler_products import get_vs
sage: get_vs(3, 2, 100)  # doctest: +NORMALIZE_WHITESPACE
Computing the structural invariants ...  done.
Computing big m ... Computing the finite product for p < 100 ...  done.
done: we use big_m = 26 .
Building indices ... done: there are 5 summands.
--------------------
For p + 3ZZ in frozenset({1})
the product of 1 / (1 - p^{-2}) is between
1.
↪03401487541434188053903064441304762857896542848909988641682503842122224587109635804962170798262059
and
1.
↪03401487541434188053903064441304762857896542848909988641682503842122224587109635804962170798262059
(Obtained:  100  correct decimal digits)
--------------------
For p + 3ZZ in frozenset({2})
the product of 1 / (1 - p^{-2}) is between
1.
↪41406439089214763756550181907982937990769506939316217503992496242392810699208849945375485850247511
and
1.
↪41406439089214763756550181907982937990769506939316217503992496242392810699208849945375485850247511
(Obtained:  99  correct decimal digits)
((frozenset({1}), frozenset({2})),
 ((1.
↪03401487541434188053903064441304762857896542848909988641682503842122224587109635804962170798262059
↪
   1.
```

(continues on next page)

(continued from previous page)

```
↪034014875414341880539030644413047628578965428489099886416825038421222245871096358049621707982620
↪
  (1.
↪414064390892147637565501819079829379907695069393162175039924962423928106992088499453754858502475
↪
   1.
↪414064390892147637565501819079829379907695069393162175039924962423928106992088499453754858502475
```

euler_product.lattice_invariant_euler_products.**get_vs_checker**(*q*, *s*, *borne=10000*)

This is a low level sanity check engine described in the tutorial. It is to be used by developers only.

INPUT:

- **q – int**
  The products are taken over lattice invariant classes modulo q.
- **s – real**
  A real number > 1.
- **borne – int (default: *10000*), optional**
  boundary of computation.

EXAMPLES:

```
sage: from euler_product.lattice_invariant_euler_products import get_vs_checker
sage: get_vs_checker(8, 2)
--------------------
For p mod  8  in  frozenset({1})
the product of 1/(1-p^{- 2 }) is about 1.0048326237351608
--------------------
For p mod  8  in  frozenset({3})
the product of 1/(1-p^{- 2 }) is about 1.1394159722583108
--------------------
For p mod  8  in  frozenset({5})
the product of 1/(1-p^{- 2 }) is about 1.0510974216618003
--------------------
 For p mod  8  in  frozenset({7})
 the product of 1/(1-p^{- 2 }) is about 1.0251478255836493
```

euler_product.lattice_invariant_euler_products.**table_performance**(*min_q*, *max_q*, *nb_decimals=100*, *big_p=300*)

The behaviour of this function is described in the attached tutorial.

INPUT:

- **min_q – int**
  The modulus q goes through all the values in `[min_q, max_q]` that are not twice an odd integer.
- **max_q – int**
  The modulus q goes through all the values in `[min_q, max_q]` that are not twice an odd integer.
- **nb_decimals – int (default: *100*), optional**
  Same as in `get_vs`.
- **big_p – int (default: *300*), optional**
  Same as in `get_vs`.

OUTPUT:

str
the table in Latex is issued.

EXAMPLES:

```
sage: from euler_product.lattice_invariant_euler_products import table_performance
sage: table_performance(10, 30) # random
11 102 digits for the first product
12 102 digits for the first product
13 102 digits for the first product
15 102 digits for the first product
16 102 digits for the first product
17 102 digits for the first product
19 102 digits for the first product
20 102 digits for the first product
21 102 digits for the first product
23 102 digits for the first product
24 102 digits for the first product
25 102 digits for the first product
27 102 digits for the first product
28 102 digits for the first product
29 102 digits for the first product
11& 10& 2& 8& 4& 21& 4 \
12& 4& 1& 5& 4& 21& 1 \
13& 12& 2& 10& 6& 21& 5 \
15& 8& 1& 5& 6& 21& 2 \
16& 8& 1& 5& 6& 21& 2 \
17& 16& 1& 5& 5& 21& 4 \
19& 18& 2& 10& 6& 21& 8 \
20& 8& 1& 5& 6& 21& 2 \
21& 12& 2& 10& 8& 21& 6 \
23& 22& 2& 6& 4& 21& 6 \
24& 8& 1& 5& 8& 21& 2 \
25& 20& 2& 8& 6& 21& 8 \
27& 18& 2& 10& 6& 21& 8 \
28& 12& 2& 10& 8& 21& 6 \
29& 28& 2& 7& 6& 21& 9 \
```

1.4.2 euler_product.utils_euler_product

Utils_euler_product utilities for Euler Product

utils_euler_product.py defines functions Main Engines

AUTHORS:

- Olivier Ramaré: initial version
- **Dominique Benielli:**
  Aix Marseille Université, Integration as SageMath package. Cellule de developpement Institut Archimède

Warning

Needs Sage version at least 9.0 CAREFUL, this is Python 3 code!

EXAMPLES:

```
sage: from euler_product.utils_euler_product import  LatticeInvariantClasses
```

class euler_product.utils_euler_product.**ComponentStructure**(*q*)

Bases: object

This class takes a positive integer q and creates the following list of accessors:

- phi_q: the value of the Euler-phi function at q.
- the_exponent: the exponent of the multiplicative group $(\mathbb{Z}/q\mathbb{Z})^*$.
- character_group: the group of Dirichlet characters modulo q, see this function for its description.
- invertibles: the tuple of the integers between 1 and q that are prime to q.
- the_SG_tuple and the_Class_tuple as in the class LatticeInvariantClass.
- nb_class: the number of Lattice Invariant classes.
- invariant_characters: given a subgroup in the_SG_tuple, the tuple of the characters that leaves this subgroup invariant is created. invariant_characters is this list of tuples, arranged as in the_SG_tuple.
- **getr_A_Kt: a method used only for get_CA_Km and get_CA_Km_F_sur_H.**
  The coefficient C(A,K,m, F/H) are a sum on a variable t of s(F/H,m/t) times a function of t, say f(t). The lattice class A in given by its index ind_A in the_Class_tuple, the subgroup K is given by its index ind_K in the_SG_tuple. The function get_r_A_K_t answers a dictionary which to every (ind_A, ind_K, t) associates this f(t) (with the moebius factor). The list of t is of course limited and given as the input parameter of get_r_A_K_t. This is the list of elements that form a divisor-closed subset of integers. This list is the same as the list of necessary values of m.
- **get_CA_Km: a method used for get_vs.**
  The coefficient C(A,K,m) are a sum on a variable t of a function of the value computed by getr_A_K_t. The lattice class A in given by its index ind_A in the_Class_tuple, the subgroup K is given by its index ind_K in the_SG_tuple. The function get_CA_Km answers a dictionary which to every (ind_A, ind_K, m) associates this value.
- **get_CA_Km_F_sur_H: a method used for get_euler_products.**
  The coefficient C(A,K,m, F/H) are a sum on a variable t of s(F/H, m/t) times a function of the value computed by getr_A_K_t. The lattice class A in given by its index ind_A in the_Class_tuple, the subgroup K is given by its index ind_K in the_SG_tuple. The function get_CA_Km_F_sur_H answers a dictionary which to every (ind_A, ind_K, m) associates this value. When F == 1 and H == 1-X, the output of get_CA_Km_F_sur_H is the same as the one of get_CA_Km.
- get_L_values: a method used only for get_gamma.
- **get_gamma: outputs the tuple defined in (22) of the *corresponding paper*.**
  For every cyclic subgroup G_0 in the_SG_tuple, we compute $\sum_{\chi\in G_0^\perp} \log L_P(t * s, \chi)$, where $L_P(x,\chi)$ is the L-series associated to χ, save that we remove the Euler factors for primes below P==big_p. The output is the list of these values computed with prec correct binary digits.

EXAMPLES:

```
sage: from euler_product.utils_euler_product import ComponentStructure
sage: structure = ComponentStructure(3)
```

get_CA_Km(*my_indices*)

get_CA_Km is a method used for get_vs. The coefficient C(A,K,m) are a sum on a variable t of a function of the value computed by getr_A_K_t. The lattice class A in given by its index ind_A in the_Class_tuple,

the subgroup K is given by its index `ind_K` in `the_SG_tuple`. The function `get_CA_Km` answers a dictionary which to every `(ind_A, ind_K, m)` associates this value.

INPUT:

- **my_indices – [int]**
  list of indices (positive integers) `m`. It should be divisor-closed (and include 1) and ordered increasingly.

OUTPUT:

dictionary
outputs the dictionary (ind_A, ind_K, m) –> value, see above.

EXAMPLES:

```
sage: from euler_product.utils_euler_product import ComponentStructure
sage: from collections import OrderedDict
sage: structure = ComponentStructure(3)
sage: OrderedDict(structure.get_CA_Km([1, -4, 4, 2, -4, 1]))  # doctest:␣
↪+ELLIPSIS, +NORMALIZE_WHITESPACE
OrderedDict([((0, 0, 1), 1/2), ((0, 0, -4), 1/2), ((0, 0, 4), 1/2), ((0, 0, 2),␣
↪1/2), ((0, 1, 1), 0), ((0, 1, -4), -1), ((0, 1, 4), -1), ((0, 1, 2), -1),
    ((1, 0, 1), -1/2), ((1, 0, -4), -1/2), ((1, 0, 4), -1/2), ((1, 0, 2), -1/2),
↪ ((1, 1, 1), 1), ((1, 1, -4), 1), ((1, 1, 4), 1), ((1, 1, 2), 1)])
```

get_CA_Km_F_sur_H(*my_indices*, *coeff_sf*, *coeff_sh*)

`get_CA_Km_F_sur_H`: a method used for `get_euler_products`. The coefficient `C(A,K, m, F/H)` are a sum on a variable `t` of `s(F/H, m/t)` times a function of the value computed by ``getr_A_K_t`. The lattice class A in given by its index `ind_A` in `the_Class_tuple`, the subgroup K is given by its index `ind_K` in `the_SG_tuple`. The function `get_CA_Km_F_sur_H` answers a dictionary which to every `(ind_A, ind_K, m)` associates this value. When `F == 1` and `H == 1-X`, the output of `get_CA_Km_F_sur_H` is the same as the one of `get_CA_Km`.

INPUT:

- **my_indices – list[int]**
  list of indices (positive integers) `m`. It should be divisor-closed (and include 1) and ordered increasingly.
- **coeff_sf – list[float]**
  the list of the sum of the m-th power of the inverses of the roots of F.
- **coeff_sh – [type]**
  the list of the sum of the m-th power of the inverses of the roots of H.

OUTPUT

dictionary
outputs the dictionary (ind_A, ind_K, m) –> value, see above.

Examples

sage: from euler_product.utils_euler_product import ComponentStructure sage: structure = ComponentStructure(3) sage: structure.get_CA_Km_F_sur_H([1, 2, 3, 4, 5, 6], [1], [1, 0, -1]) # doctest: +NORMALIZE_WHITESPACE {(0, 0, 1): 0, (0, 0, 2): 1, (0, 0, 3): 0, (0, 0, 4): 1, (0, 0, 5): 0, (0, 0, 6): 0, (0, 1, 1): 0, (0, 1, 2): 0, (0, 1, 3): 0, (0, 1, 4): -2, (0, 1, 5): 0, (0, 1, 6): 0, (1, 0, 1): 0, (1, 0, 2): -1, (1, 0, 3): 0, (1, 0, 4): -1, (1, 0, 5): 0, (1, 0, 6): 0, (1, 1, 1): 0, (1, 1, 2): 2, (1, 1, 3): 0, (1, 1, 4): 2, (1, 1, 5): 0, (1, 1, 6): 0}

`get_L_values`(*m*, *big_p*, *CIF*, *CF*)

for every Dirichlet character χ modulo `q`, we compute the L-series $L_P(m, \chi)$ associated to :math:chi:, save that we remove the Euler factors for primes below `P==big_p`. The output is the list of these values computed with `prec` correct binary digits.

INPUT:

- **`m` – [ComplexIntervalFieldElement]**
  the point where the L-series are computed. The real part should be > 1 .
- **`big_p` – int**
  a positive integer. The Euler products are computed for primes above `big_p`.
- **`CIF` – Complex Interval Field**
  [description]
- **`CF` – Complex Field**
  not used. Only `CR.prec` is used?

OUTPUT:

tuple
the tuple of the values of $L_P(m, \chi)$, where χ varies on the Dirichlet characters, values computed with `prec` correct binary digits.

EXCEPTIONS:

ValueError parameter `m` not in `CIF`

EXAMPLES:

```
sage: from euler_product.utils_euler_product import ComponentStructure
sage: structure = ComponentStructure(10)
sage: CIF = ComplexIntervalField(200)
sage: CF = ComplexField(200 + 1)
sage: m = CIF(2)
sage: structure.get_L_values(m, 200, CIF, CF)
(1.000748102425238619689365450157187702551432318307909367648 0?,
1.000022637797480910480663979089709527419334446685903741889 8? - 0.
↪000013140891690043745487410651569458960644116821995803505 9?*I,
0.999989924051193387296274847969319972395631776846903092249 7? + 4.
↪143927954717328508155998814005883510070027178208295916333 ?e-63*I,
1.000022637797480910480663979089709527419334446685903741889 8? + 0.
↪000013140891690043745487410651569458960644116821995803505 9?*I)
sage: m = CIF(2.1)
sage: structure.get_L_values(m, 200, CIF, CF)
(1.000402927487993369402471491087634699517620972491849258023 9?,
1.000013330852742794601876671961697811977029714891503324800 ? - 7.
↪753895710893476952029795948493461826949999660229676 8?e-6*I,
0.999994764455245450699443791011732548179074675858934972695 9? + 3.
↪155955392795568184998068336536354889461952525441403577 84?e-63*I,
1.000013330852742794601876671961697811977029714891503324800 ? + 7.
↪753895710893476952029795948493461826949999660229676 8?e-6*I)
```

`get_gamma`(*t*, *s*, *big_p*, *prec*)

Outputs the tuple defined in (22) of the *corresponding paper*: for every cyclic subgroup G_0 in `the_SG_tuple`, we compute $\sum_{\chi \in G_0^{\perp}} \log L_P(t * s, \chi)$, where $L_P(x, \chi)$ is the L-series associated to χ, save that we remove the Euler factors for primes below `P==big_p`. The output is the list of these values computed with `prec` correct binary digits.

INPUT:

- **t – int**
  the L-series are computed at t*s.
- **s – float**
  the L-series are computed at t*s. The separation of t and s is only for readability of the code.
- **big_p – int**
  a positive integer. The Euler products are computing for primes larger than big_p.
- **prec – int**
  number of correct binary digits in the output.

OUTPUT:

tuple
the list of values of $\sum_{\chi \in G_0^\perp} \log L_P(t*s, \chi)$, see the function description.

EXAMPLES:

```
sage: from euler_product.utils_euler_product import ComponentStructure
sage: structure  = ComponentStructure(5)
sage: structure.invariant_characters
((0, 1, 2, 3), (0, 2), (0,))
sage: structure.get_gamma(1, 1.2, 20, 100)
(0.41205867484783847538747647 3?, 0.39593264955263085674122241 44?, 0.
↪41136727621318961945202378 06?)
```

getr_A_Kt(*my_indices*)

This method is used only for get_CA_Km and get_CA_Km_F_sur_H. The coefficient C(A,K,m, F/H) are a sum on a variable t of s(F/H,m/t) times a function of t, say f(t). The lattice class A in given by its index ind_A in the_Class_tuple, the subgroup K is given by its index ind_K in the_SG_tuple. The function get_r_A_K_t answers a dictionary which to every (ind_A, ind_K, t) associates this f(t) (with the moebius factor). The list of t is of course limited and given as the input parameter of get_r_A_K_t. This is the list of elements that form a divisor-closed subset of integers. This list is the same as the list of necessary values of m.

INPUT:

- **my_indices – list**
  list of indices (positive integers) t. It should be divisor-closed (and include 1) and ordered increasingly.

OUTPUT:

dictionary
output is a the dictionary (ind_A, ind_K, t) –> value, see above.

EXAMPLES:

```
sage: from euler_product.utils_euler_product import ComponentStructure
sage: structure = ComponentStructure(3)
sage: structure.getr_A_Kt([1, 2, 3, 4, 6])
{(0, 0, 1): 1/2,
(0, 0, 2): 0,
(0, 0, 3): -1/2,
(0, 0, 4): 0,
(0, 0, 6): 0,
```

(continues on next page)

(continued from previous page)

```
(0, 1, 1): 0,
(0, 1, 2): -1,
(0, 1, 3): 0,
(0, 1, 4): 0,
(0, 1, 6): 1,
(1, 0, 1): -1/2,
(1, 0, 2): 0,
(1, 0, 3): 1/2,
(1, 0, 4): 0,
(1, 0, 6): 0,
(1, 1, 1): 1,
(1, 1, 2): 0,
(1, 1, 3): -1,
(1, 1, 4): 0,
(1, 1, 6): 0}
```

class euler_product.utils_euler_product.**LatticeInvariantClasses**

Bases: object

This class takes a modulus q (i.e. a positive integer) and has two named accessors, the_SG_tuple and the_Class_tuple. The SG tuple is the list of the multiplicative subgroups of $(\mathbb{Z}/q\mathbb{Z})^*$ that are generated by a single element. The Class tuple is the list of Lattice Invariant classes, namely the partition of $(\mathbb{Z}/q\mathbb{Z})^*$ made by the smallest non-empty intersections of elements of the_SG_tuple.

EXAMPLES:

```
sage: from euler_product.utils_euler_product import LatticeInvariant
sage: LatticeInvariant(30)
((frozenset({1}),
  frozenset({1, 11}),
  frozenset({1, 19}),
  frozenset({1, 29}),
  frozenset({1, 7, 13, 19}),
  frozenset({1, 17, 19, 23})),
 (frozenset({1}),
  frozenset({11}),
  frozenset({19}),
  frozenset({29}),
  frozenset({7, 13}),
  frozenset({17, 23})))
```

euler_product.utils_euler_product.**get_beta**(*F*)

Outputs the maximum of 1 and of the inverse of the norm of the non-zero roots of the polynomial F.

INPUT:

- **F – pol**
  a polynomial with RealField coefficients.

OUTPUT:

float
the maximum of 1 and of the inverse of the norm of the non-zero roots of F.

EXAMPLES:

```
sage: from euler_product.utils_euler_product import get_beta
sage: R0 = RealField(30)
sage: R0X = R0['x']
sage: (x,) = R0X._first_ngens(1)
sage: F0 = R0X(1 - x^2)
sage: get_beta(F0)
1
```

euler_product.utils_euler_product.**get_beta_rough**(*coeffs_f*)

Outputs the maximum of 1 and of the sum of the norm of the coefficients of the polynomial F, which is precisely given as the list coeffs_f. This is intended to be an easy upper bound when the function get_beta takes too much time.

INPUT:

- **coeffs_f – float**
  a list of floats, supposedly representing a polynomial F.

OUTPUT:

float
Outputs the maximum of 1 and of the sum of the norm of the elements of coeffs_f.

EXAMPLES:

```
sage: from euler_product.utils_euler_product import get_beta_rough
sage: get_beta_rough([1, 3, 4])
8
```

euler_product.utils_euler_product.**get_vector_sf**(*coeffs_f*, *how_many*)

A polynomial F is given by its list of coefficients, the first one being 1. The output is the list $s_F(m)$ for m less than how_many, where $s_F(m)$ is the sum of the m-th power of the inverses of the roots of F.

INPUT:

- **coeffs_f – list[float]**
  coefficients of the polynomial f, starting by 1.
- **how_many – int**
  number of computed coefficients.

OUTPUT:

list
list des coefficient s_f(m) over m <= how_many.

EXAMPLES:

```
sage: from euler_product.utils_euler_product import get_vector_sf
sage: get_vector_sf([1, -1], 5)
[1, 1, 1, 1, 1]
sage: get_vector_sf([1, 1, 1], 10)
[2, -1, -1, 2, -1, -1, 2, -1, -1, 2]
```

euler_product.utils_euler_product.**laTeX_for_number**(*w*, *how_many*, *nb_block_sto_cut*)

Return a character string representing the real number w made of its integer part followed by every decimal up to the``how_many`` -th decimals, where every block of 5 decimal is separated by '\\,', and every succession of how_many blocks is separated by '\n'. The string has a `&` after the decimal point and ends with the string `\\cdots`.

INPUT:

- **w – float**
  w is a real number with a (short) integer part and a floating point.
- **how_many – int**
  number of decimals, separated every 5 of them by '\,' and every block of nb_block_sto_cut, on a different line. '\cdots' ends the string.
- **nb_block_sto_cut – int**
  See above.

OUTPUT:

str
a character string int(w).separated_decimals where separated_decimals is LaTeX formatted version of the decimal expansion of w, see the description of the function.

EXAMPLES:

```
sage: from euler_product.utils_euler_product import laTeX_for_number
sage: laTeX_for_number(22.01234567812345, 100, 8)
'22.&01234\\,56781\\,235\\cdots'
```

euler_product.utils_euler_product.**nb_common_digits**(*a*, *b*, *max_nb_digits=100*)

Returns -1 if floor(a) != floor(b).

INPUT:

- **a – float**
  first float to compare.
- **b – float**
  second float to compare.
- **max_nb_digits – float**
  maximum of number of digits.

OUTPUT:

int
Returns -1 if floor(a) != floor(b), or the number of common digits.

EXAMPLES:

```
sage: from euler_product.utils_euler_product import nb_common_digits
sage: import numpy as np
sage: nb_common_digits(1.33333, 1.334444)
2
sage: nb_common_digits(1.33333, 2.334444)
-1
sage: nb_common_digits(1.33333, np.inf)
-1
sage: nb_common_digits(np.inf, np.nan)
-1
```

euler_product.utils_euler_product.**sub_group_generated**(*n*, *q*)

Return the frozenset of the multiplicative subgroup generated by the powers of n modulo q. It is expected that n and q are coprime.

INPUT:

- **n – int**
  an integer, expected to be coprime to q.
- **q – int**
  a positive integer.

OUTPUT:

frozenset
immutable set of the powers of n modulo q

EXAMPLES:

```
sage: from euler_product.utils_euler_product import sub_group_generated
sage: sub_group_generated(5, 3)
frozenset({1, 2})
```

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e

euler_product.lattice_invariant_euler_products, 26
euler_product.utils_euler_product, 34

INDEX

C

ComponentStructure (*class in euler_product.utils_euler_product*), 35

E

euler_product.lattice_invariant_euler_products module, 26
euler_product.utils_euler_product module, 34

G

get_beta() (*in module euler_product.utils_euler_product*), 39
get_beta_rough() (*in module euler_product.utils_euler_product*), 40
get_CA_Km() (*euler_product.utils_euler_product.ComponentStructure method*), 35
get_CA_Km_F_sur_H() (*euler_product.utils_euler_product.ComponentStructure method*), 36
get_euler_products() (*in module euler_product.lattice_invariant_euler_products*), 26
get_gamma() (*euler_product.utils_euler_product.ComponentStructure method*), 37
get_L_values() (*euler_product.utils_euler_product.ComponentStructure method*), 36
get_vector_sf() (*in module euler_product.utils_euler_product*), 40
get_vs() (*in module euler_product.lattice_invariant_euler_products*), 28
get_vs_checker() (*in module euler_product.lattice_invariant_euler_products*), 33
getr_A_Kt() (*euler_product.utils_euler_product.ComponentStructure method*), 38

L

laTeX_for_number() (*in module euler_product.utils_euler_product*), 40
LatticeInvariantClasses (*class in euler_product.utils_euler_product*), 39

M

module
    euler_product.lattice_invariant_euler_products, 26
    euler_product.utils_euler_product, 34

N

nb_common_digits() (*in module euler_product.utils_euler_product*), 41

S

sub_group_generated() (*in module euler_product.utils_euler_product*), 41

T

table_performance() (*in module euler_product.lattice_invariant_euler_products*), 33