Dear Author,

Here are the proofs of your article.

- You can submit your corrections **online**, via **e-mail** or by **fax**.
- For **online** submission please insert your corrections in the online correction form. Always indicate the line number to which the correction refers.
- You can also insert your corrections in the proof PDF and email the annotated PDF.
- For fax submission, please ensure that your corrections are clearly legible. Use a fine black pen and write the correction in the margin, not too close to the edge of the page.
- Remember to note the journal title, article number, and your name when sending your response via e-mail or fax.
- Check the metadata sheet to make sure that the header information, especially author names and the corresponding affiliations are correctly shown.
- Check the questions that may have arisen during copy editing and insert your answers/ corrections.
- Check that the text is complete and that all figures, tables and their legends are included. Also check the accuracy of special characters, equations, and electronic supplementary material if applicable. If necessary refer to the *Edited manuscript*.
- The publication of inaccurate data such as dosages and units can have serious consequences. Please take particular care that all such details are correct.
- Please **do not** make changes that involve only matters of style. We have generally introduced forms that follow the journal's style. Substantial changes in content, e.g., new results, corrected values, title and authorship are not allowed without the approval of the responsible editor. In such a case, please contact the Editorial Office and return his/her consent together with the proof.
- If we do not receive your corrections within 48 hours, we will send you a reminder.
- Your article will be published **Online First** approximately one week after receipt of your corrected proofs. This is the **official first publication** citable with the DOI. **Further changes are, therefore, not possible.**
- The printed version will follow in a forthcoming issue.

Please note

After online publication, subscribers (personal/institutional) to this journal will have access to the complete article via the DOI using the URL: http://dx.doi.org/[DOI].

If you would like to know when your article has been published online, take advantage of our free alert service. For registration and further information go to: <u>http://www.link.springer.com</u>.

Due to the electronic nature of the procedure, the manuscript and the original figures will only be returned to you on special request. When you return your corrections, please inform us if you would like to have these documents returned.

Metadata of the article that will be visualized in OnlineFirst

ArticleTitle	Fast norm computation in smooth-degree Abelian number fields	
Article Sub-Title		
Article CopyRight	The Author(s) (This will be the copyright line in the final PDF)	
Journal Name	Research in Number Theory	
Corresponding Author	FamilyName	Bernstein
	Particle	
	Given Name	Daniel J.
	Suffix	
	Division	Department of Computer Science
	Organization	University of Illinois at Unicago
	Address	Unicago, USA Hanni Cinica Instituto fan IT Coounity
	Oromization	Horst Gortz Institute for 11 Security
	Address	Rohum Germany
	Division	Research Center for Information Technology Innovation
	Organization	Academia Sinica
	Address	Taipei. Taiwan
	Phone	· I ·)
	Fax	
	Email	djb@cr.yp.to
	URL	
	ORCID	
Schedule	Received	12 Aug 2022
	Revised	
	Accepted	28 Sep 2022
Abstract	This paper presents a fast method to compute algebraic norms of integral elements of smooth-degree cyclotomic fields, and, more generally, smooth-degree Galois number fields with commutative Galois groups. The typical scenario arising in <i>S</i> -unit searches (for, e.g., class-group computation) is computing a $\Theta(n\log n)$ -bit norm of an element of weight $n^{1/2+o(1)}$ in a degree- <i>n</i> field; this method then uses $n(\log n)^{3+o(1)}$ bit operations. An $n(\log n)^{O(1)}$ operation count was already known in two easier special cases: norms from power-of-2 cyclotomic fields via towers of power-of-2 cyclotomic subfields, and norms from multiquadratic fields via towers of multiquadratic subfields. This paper handles more general Abelian fields by identifying tower-compatible integral bases supporting fast multiplication; in particular, there is a synergy between tower-compatible Gauss-period integral bases and a fast-multiplication idea from Rader. As a baseline, this paper also analyzes various standard norm-computation techniques that apply to arbitrary number fields, concluding that all of these techniques use at least $n^2(\log n)^{2+o(1)}$ bit operations in the same scenario, even with fast subroutines for continued fractions and for complex FFTs. Compared to this baseline, algorithms dedicated to smooth-degree Abelian fields find each norm $n/(\log n)^{1+o(1)}$ times faster, and finish norm computations inside <i>S</i> -unit searches $n^2/(\log n)^{1+o(1)}$ times faster.	
Mathematics Subject Classification (separated by '-')	Primary 11Y40 - 11Y	16 - Secondary 68W30 - 11R18
Footnote Information	This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) as part of the Excellence Strategy of the German Federal and State Governments—EXC 2092 CASA—390781972 "Cyber Security in the Age of Large-Scale Adversaries"; by the U.S. National Science Foundation under Grant 1913167; by the Taiwan's Executive Yuan Data Safety and Talent Cultivation Project (AS-KPQ-109-DSTCP); and by the Cisco University Research Program "Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation" (or other funding agencies). Permanent ID of this document: 6c338bc06ca0c734f22cb48bacc4b65ea666cd81.	

RESEARCH

Check for updates

1

2

3

Fast norm computation in smooth-degree Abelian number fields



*Correspondence:

djb@cr.yp.to

³Horst Görtz Institute for IT Security, Ruhr University Bochum, Bochum, Germany Full list of author information is available at the end of the article This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) as part of the Excellence Strategy of the German Federal and State Governments—EXC 2092 CASA—390781972 "Cyber Security in the Age of Large-Scale Adversaries"; by the U.S. National Science Foundation under Grant 1913167: by the

Security in the Age of Large-Scale Adversaries"; by the U.S. National Science Foundation under Grant 1913167; by the Taiwan's Executive Yuan Data Safety and Talent Cultivation Project (AS-KPQ-109-DSTCP); and by the Cisco University Research Program. "Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation" (or other funding agencies). Permanent ID of this document: 6c338bc06ca0c734f22cb

48bacc4b65ea666cd81

🖄 Springer

Abstract

This paper presents a fast method to compute algebraic norms of integral elements of smooth-degree cyclotomic fields, and, more generally, smooth-degree Galois number fields with commutative Galois groups. The typical scenario arising in S-unit searches (for, e.g., class-group computation) is computing a $\Theta(n \log n)$ -bit norm of an element of weight $n^{1/2+o(1)}$ in a degree-*n* field; this method then uses $n(\log n)^{3+o(1)}$ bit operations. An $n(\log n)^{O(1)}$ operation count was already known in two easier special cases: norms from power-of-2 cyclotomic fields via towers of power-of-2 cyclotomic subfields, and norms from multiquadratic fields via towers of multiquadratic subfields. This paper handles more general Abelian fields by identifying tower-compatible integral bases supporting fast multiplication; in particular, there is a synergy between tower-compatible Gauss-period integral bases and a fast-multiplication idea from Rader. As a baseline, this paper also analyzes various standard norm-computation techniques that apply to arbitrary number fields, concluding that all of these techniques use at least $n^2(\log n)^{2+o(1)}$ bit operations in the same scenario, even with fast subroutines for continued fractions and for complex FFTs. Compared to this baseline, algorithms dedicated to smooth-degree Abelian fields find each norm $n/(\log n)^{1+o(1)}$ times faster, and finish norm computations inside S-unit searches $n^2/(\log n)^{1+o(1)}$ times faster.

Mathematics Subject Classification: Primary 11Y40, 11Y16, Secondary 68W30, 11R18

1 Introduction

- Consider the element $\alpha = 3 + \zeta_{2048}^{271} + 4\zeta_{2048}^{828}$ of the cyclotomic field $K = \mathbb{Q}(\zeta_{2048})$; here ζ_m , for any positive integer *m*, means the complex number $\exp(2\pi i/m)$. Write det $\overset{K}{\square} \alpha$ for the
- ⁵ determinant of multiplication by α as a Q-linear map from K to K, i.e., for the algebraic
- ⁶ norm of α from *K* down to \mathbb{Q} . (See Sect. 1.4 regarding notation choices.) The following
- ⁷ Sage commands print out det ${}^{K}_{\mathbb{O}} \alpha$, while measuring how long the computation takes:

K.<zeta> = CyclotomicField(2048); alpha = 3+zeta^271+4*zeta^828
%time alpha.norm()

¹⁰ Sage 9.5 (the January 2022 version of Sage [83]) takes 61 milliseconds on one core of a ¹¹ 3.5GHz Intel Xeon E3-1275 v3 (Haswell) CPU, around 0.21×10^9 CPU cycles.

© The Author(s) 2023. This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

One has det ${}^{K}_{\mathbb{Q}} \alpha = \prod_{c \in \{1,3,5,\dots,2047\}} (3 + \zeta_{2048}^{271c} + 4\zeta_{2048}^{828c})$. The absolute value of the complex number $3 + \zeta_{2048}^{271c} + 4\zeta_{2048}^{828c}$ is below 8, and one might guess that it is typically somewhere around 4, i.e., that det ${}^{K}_{\mathbb{Q}} \alpha$ has absolute value around $4^{1024} = 2^{2048}$. Sage computes the exact value of det ${}^{K}_{\mathbb{Q}} \alpha$, an integer $272 \dots 618 \approx 0.842 \cdot 2^{2048}$.

Inside Sage, PARI [76] finds $\det_{\mathbb{Q}}^{K} \alpha$ as the resultant of two polynomials in $\mathbb{Z}[x]$. The first polynomial is the minimal polynomial of ζ_{2048} over \mathbb{Q} , namely $x^{1024} + 1$. The second polynomial is $3 + x^{271} + 4x^{828}$. One can skip Sage's number-field machinery and directly compute $\det_{\mathbb{Q}}^{K} \alpha$ as a polynomial resultant:

```
20 ZZx.<x> = ZZ[]
21 Phi = x<sup>1</sup>024+1; g = 3+x<sup>2</sup>71+4*x<sup>828</sup>
22 %time Phi.resultant(g)
```

²³ Sage, when asked for a resultant of two polynomials instead of a norm of a numberfield element, calls FLINT [55] instead of PARI, and now takes 1.12×10^9 cycles. The resultant subroutine has a proof=False option allowing randomized algorithms; this option doesn't save time. What does save time is using NTL [87] polynomials instead of FLINT polynomials:

```
28 ZZx.<x> = PolynomialRing(ZZ,'x', implementation='NTL')
```

```
29 Phi = x^{1024+1}; g = 3+x^{271+4}x^{828}
```

```
30 %time Phi.resultant(g)
```

This takes 0.15×10^9 cycles.

Why does it take so many cycles to compute a 2048-bit resultant of two input poly-32 nomials that have, in dense format, a few thousand small coefficients? The issue is not 33 the number of cycles required per bit for basic arithmetic: for example, Sage takes about 34 20,000 cycles to multiply two 2048-bit integers. The issue is that standard fast-continued-35 fraction techniques for computing the resultant of two polynomials in $\mathbb{Z}[x]$ have cost 36 growing as essentially the *product* of the number of input bits and the number of output 37 bits. The resultant of $x^n + 1$ and a sparse *n*-coefficient input, with $n^{1/2+o(1)}$ coefficients 38 ± 1 , will typically have $\Theta(n \log n)$ bits; these resultant algorithms then $\cot n^2(\log n)^{3+o(1)}$. 39 For some inputs, the output is much smaller and the algorithms are much faster, but this 40 is not the typical case. 41

It is not a new observation that one can do much better by exploiting transitivity of determinants through a tower of subfields of *K*. Take *F* as, e.g., the field $\mathbb{Q}(\zeta_{1024})$. Then *F* is a subfield of *K*: specifically, $\zeta_{1024} = \zeta_{2048}^2$, so *F* is the fixed field of the unique

```
automorphism of K that maps \zeta_{2048} to -\zeta_{2048}. Hence
```

$$\det_{F}^{K} \alpha = \left(3 + \zeta_{2048}^{271} + 4\zeta_{2048}^{828}\right) \left(3 + (-\zeta_{2048})^{271} + 4(-\zeta_{2048})^{828}\right)$$

= $9 - \zeta_{1024}^{271} + 24\zeta_{1024}^{414} + 16\zeta_{1024}^{828} = 9 - \zeta_{1024}^{271} - 16\zeta_{1024}^{316} + 24\zeta_{1024}^{414}$

⁴⁷ One can then compute det^{*K*}_{\square} α as det^{*F*}_{*P*} det^{*K*}_{*F*} α :

```
48 ZZx.<x> = PolynomialRing(ZZ,'x', implementation='NTL')
```

```
g = 3 + x^2 7 1 + 4 \cdot x^8 28
```

46

```
so %time g = ZZx(list(g*g(-x))[::2]) % (x^512+1)
```

```
$1 %time (x^512+1).resultant(g)
```

Here $g \star g(-x)$ gives $g \cdot g(-x) = \det_{\mathbb{Z}[x^2]}^{\mathbb{Z}[x]} g$, a polynomial whose value at ζ_{2048} is $\det_F^K \alpha$; list $(g \star g(-x))$ gives the list of coefficients of $g \cdot g(-x)$; [::2] extracts every second

⁵⁴ coefficient; ZZx (. . .) produces the corresponding polynomial, a polynomial whose value

at ζ_{1024} is det^K_F α ; and (x^{512+1}) reduces modulo $x^{512} + 1$. This is, up to sign, also g.adams_operator(2) (x^{512+1}) since deg g > 0, but the "Adams" naming is questionable given that this operator on polynomials was already used (for root-finding) by Dandelin in [39, page 49] in 1826; see generally [60]. Sage reports that the evaluation of det^K_F takes 0.01×10^9 cycles and that the evaluation of det^F_Q takes 0.09×10^9 cycles. One can save more time by recursively decomposing det^F_Q via transitivity, and exploiting the special form of the power-of-2 cyclotomic polynomials

⁶² to convert each modular reduction into subtraction:

```
63 ZZx.<x> = PolynomialRing(ZZ,'x',implementation='NTL')
64 g = 3+x^271+4*x^828
65 %time for d in 512,256,128,64,32,16,8,4,2,1: \
66 L = list(g*g(-x))[::2]; \
67 g = ZZx(L[:d])-ZZx(L[d:])
```

This reduces the total time to just 0.011×10^9 cycles. Appendix B removes more overhead and takes just 0.0012×10^9 cycles. The important point to keep in mind is that the typical algorithm cost has dropped from $n^{2+o(1)}$ to $n^{1+o(1)}$.

71 1.1 Contributions of this paper

As a baseline, Sect. 3 analyzes the costs of various standard det^K_Q techniques that work for arbitrary number fields. The special case of power-of-2 cyclotomics, as in the $\mathbb{Q}(\zeta_{2048})$ example above, suffices for seeing that these techniques are not competitive asymptotically, so Sect. 3 focuses on this case. The main conclusion of Sect. 3 is that, in the typical case of $\Theta(n \log n)$ -bit outputs for field degree *n* (see Sect. 2 for why this is typical), all of these techniques use at least $n^2(\log n)^{2+o(1)}$ bit operations.

Section 4 explores the question of which number fields allow lower-cost evaluation of det^{*K*}_Q via transitivity, in particular reducing $n^2(\log n)^{2+o(1)}$ to $n(\log n)^{3+o(1)}$. It is natural to ask for the field degree to be smooth—a product of small primes—and for the field to have a correspondingly long tower of subfields. The challenge addressed in Sect. 4 is to build algorithms to multiply efficiently on tower-compatible bases for these subfields. Note that this is easy for power-of-2 cyclotomics: standard polynomial bases are compatible with the tower $\mathbb{Q} \subset \mathbb{Q}(\zeta_4) \subset \mathbb{Q}(\zeta_8) \subset \cdots$ and are well known to allow fast multiplication.

Section 5 analyzes applications to one of the standard techniques for computing class 85 groups, unit groups, etc. The technique is to enumerate small elements of the ring of 86 integers, and filter those elements to see which ones are S-units, where S is the set of infinite 87 places and small finite places. This filtering is typically handled by an Eratosthenes-type 88 sieving procedure when degrees are small and discriminants are large, as in the number-89 field sieve for integer factorization; but if degrees are relatively large, as in the cyclotomic 90 case, then it seems best to compute det ${}^K_{\mathbb{O}} \alpha$ for each element α and then check which det ${}^K_{\mathbb{O}} \alpha$ 91 factor as desired. The smooth-degree Abelian case uses fewer det $^{K}_{\mathbb{O}} \alpha$ computations (since 92 one can search for S-units modulo automorphisms of the field) and speeds up each det ${}^{K}_{\mathbb{O}} \alpha$ 93 computation, overall speeding up the sequence of det ${}^{\mathcal{K}}_{\mathbb{O}} \alpha$ computations by a factor close 94 to n^2 . The Abelian case also speeds up the factorizations. 95

96 1.2 Previous work on speedups using transitivity

97 The fact that transitivity of determinants saves effort is standard textbook material. For

example, a standard exercise starts with the degree-4 field $K = \mathbb{Q}(\zeta_5)$ and the real subfield

- ⁹⁹ $F = \mathbb{R} \cap K = \mathbb{Q}(\sqrt{5})$, and computes $\det_{\mathbb{Q}}^{K} \alpha$ as $\det_{\mathbb{Q}}^{F} \det_{F}^{K} \alpha$. But such small examples give ¹⁰⁰ little information regarding how much effort is saved in larger examples.
- For any power-of-2 cyclotomic field *K*, Gentry and Halevi [51, Section 4] used a tower of power-of-2 cyclotomic subfields to compute det^{*K*}_{\mathbb{Q}} α in essentially linear time, as in the $\mathbb{Q}(\zeta_{2048})$ example given above. Bauch, Bernstein, de Valence, Lange, and van Vredendaal [9, Section 3.4], for the case of multiquadratic fields $K = \mathbb{Q}(\sqrt{d_1}, \sqrt{d_2}, \dots, \sqrt{d_t})$, computed det^{*K*}_{\mathbb{Q}} α in essentially linear time using a tower of multiquadratic subfields.
- Gentry and Halevi also computed $\operatorname{tr}_{\mathbb{Q}}^{K}(1/\alpha)$ for $\alpha \neq 0$. One can easily obtain the inversetrace algorithm in [51] by applying the following simple general-purpose conversion, an example of automatic differentiation, to the $\operatorname{det}_{\mathbb{Q}}^{K}$ algorithm in [51]: for any K, take any algebraic algorithm for $\alpha \mapsto \operatorname{det}_{\mathbb{Q}}^{K} \alpha$, tensor with the jet plane $\mathbb{Q}[\epsilon]/\epsilon^2$ over \mathbb{Q} , and apply the resulting algorithm to $\alpha + \epsilon$ to obtain

$$\det_{\mathbb{Q}[\epsilon]/\epsilon^2}^{K[\epsilon]/\epsilon^2}(\alpha+\epsilon) = (\det_{\mathbb{Q}}^K \alpha)(1+\epsilon \operatorname{tr}_{\mathbb{Q}}^K(1/\alpha)).$$

This conversion loses a small constant factor in performance. This is not how the inversetrace algorithm in [51] is described, but one can, with some effort, check that this is what
the algorithm does.

115 1.3 Fast-multiplication subroutines

There is a huge literature on FFT-based algorithms to multiply two elements of $R[x]/\varphi$, for any monic $\varphi \in R[x]$ with deg $\varphi = n$, using $n(\log n)^{1+o(1)}$ operations on coefficients in R. See generally [16].

These algorithms are faster by a constant factor when φ is "FFT-friendly". This con-119 stant factor is not visible in the $n(\log n)^{1+o(1)}$ asymptotics, but it becomes visible if one 120 applies the same idea recursively to multiply in a ring presented as a tower such as 121 $(\cdots ((R[x_1]/\varphi_1)[x_2]/\varphi_2)\cdots)[x_t]/\varphi_t$. In a "multidimensional FFT", each φ_i is FFT-friendly 122 (e.g., a size-*n* Hadamard–Walsh transform has $\varphi_j = x_i^2 - 1$ and $n = 2^t$), and the cost 123 is $\Theta(n \log n)$ for *n* coefficients; see Sect. 4.12.1. For general φ_i , a constant-factor loss *c* at 124 each level of the tower turns into a loss c^t for t levels, increasing costs by a factor $n^{\Theta(1)}$ if 125 $t \in \Theta(\log n).$ 126

As van der Hoeven and Lecerf pointed out in [59], if one modifies a tower to force 127 $t \in o(\log n)$, by replacing any constant-degree steps with superconstant-degree steps, 128 then the c^t overhead factor mentioned above is $n^{o(1)}$, and one obtains total cost $n^{1+o(1)}$ 129 for multiplication. There is some tension between the idea of reducing t and the idea of 130 exploiting towers to save time in $\det_{\mathbb{O}}^{K}$ computation; but note that if there are t levels, each 131 of relative degree $n^{O(1/t)}$, then there are $n^{O(1/t)}$ multiplications at each level, so reaching 132 total cost $n^{1+o(1)}$ for det^K_{\square} simply requires *t* to be superconstant. A closer look shows that 133 one can do better—as an analogy, FFTs are asymptotically better than Toom's method for 134 univariate multiplication, even though both take essentially linear time—but one should 135 not think that short towers are useless. 136

For multiquadratic fields $\mathbb{Q}(\sqrt{d_1}, \sqrt{d_2}, \dots, \sqrt{d_t})$, the multiplication algorithm in [9, Section 3.3] selects enough moduli *p* for which all of d_1, d_2, \dots, d_t are squares modulo *p*, and then uses Hadamard–Walsh transforms twisted by $\sqrt{d_1}, \sqrt{d_2}, \dots, \sqrt{d_t}$ modulo *p*.

One can, with effort, extract from a paper by Arita and Handa [6, Sections 3.3, 3.4, and 4.3] an essentially-linear-time algorithm to multiply on Gauss-period bases of primeconductor Abelian fields, i.e., subfields of $\mathbb{Q}(\zeta_p)$ (beyond \mathbb{Q}) where *p* is prime. This algorithm can be viewed as a simple "folding" of an FFT algorithm that Rader introduced
in [82], an algorithm having a different flavor from conventional FFTs; see Sect. 4.8. Section 4.12 handles more general Abelian fields, unifying the idea of folding with an extension
of Winograd's generalization [96] of Rader's idea.

147 1.4 Notation and terminology

Wessel [95] and independently Argand [3] introduced a geometric description of each complex number a + bi as a line in the plane from (0, 0) to (a, b). Wessel [95, page 469] referred to $(a^2 + b^2)^{1/2}$ as the length of the line ("Længde" in Danish). Argand [4, page 208] referred to $(a^2 + b^2)^{1/2}$ as the absolute size ("grandeur absolue" in French) and the modulus ("module" in French) of a + bi. Gauss [49, page 103] referred to $a^2 + b^2$ as the norm ("norma" in Latin) of a + bi for $a, b \in \mathbb{Z}$. (One meaning of "norma" in Latin is a carpenter's square used to measure right angles.)

Subsequent literature reused the "norm" terminology for generalizations (1) to algebraic norms, typically called just "norms", but also (2) to 2-norms such as the ℓ^2 norm and the L^2 norm, and beyond that to further generalizations of the concept of length, also typically called just "norms". Algebraic norms and 2-norms coincide for a + bi, aside from quibbles about $a^2 + b^2$ vs. $(a^2 + b^2)^{1/2}$, but differ in general.

This wouldn't be problematic if there were a clear dividing line between papers in number theory saying "norm" for algebraic norms and papers in analysis saying "norm" for those other things. The reality, however, is that those other things appear constantly in number theory (and not just in analytic number theory): consider lattices, for example, or Weil height. Perhaps it's time for number theorists to consider ending the conflict: put the "norm" word down gently and back away.

¹⁶⁶ What, then, should algebraic norms be called? Nobody actually says "algebraic norms" ¹⁶⁷ or "field norms" except for disambiguation. Meanwhile there is a well known, standard, ¹⁶⁸ unambiguous name for a more general concept: "determinant". We refer to the trace ¹⁶⁹ of multiplication by α as the trace of α ; shouldn't we also refer to the determinant of ¹⁷⁰ multiplication by α as the determinant of α ?

There are parameters, of course: in the case of fields, there's an input field and an output field, with the input field having finite degree over the output field. We all know what the trace map is from the input field to the output field; it's not a big leap to talk about the determinant map from the input field to the output field.

As for notation, Dirichlet [40, page 295] wrote N(a + bi) for $a^2 + b^2$; there were no 175 parameters in the N map. Subsequent literature sometimes uses N_F for a norm to F (as in, 176 e.g., [58, page 204] and [57, page 125]) and sometimes uses N_F for a norm from F (as in, 177 e.g., [30]). The extra effort of writing $N_{L/K}$ resolves the ambiguity (if L and K aren't objects 178 with a quotient that could be reasonably plugged into the N_F notation), but anyone who 179 has taken a course in differential geometry, the study of superscripts and subscripts, will 180 see that there's a better place to put the input field. This is not a new idea: see, e.g., [69, 181 page 16] and [41]. 182

As a separate matter, the short name "N" is fine for local notation (notation where brevity is prioritized over broad readability), but it doesn't work well as global notation given all the other common uses of "N". Again determinants come to the rescue: the global notations "det" and "tr" are well established. Adding a K subscript for K-linear maps, and an *L* superscript for taking inputs in *L* as linear maps from *L* to *L*, gives this paper's notation det $_{K}^{L} \alpha$.

This paper does not attempt to avoid the following common abbreviations: "Rings" are commutative rings. If *R* is a ring and *S* is a set then R^S is the ring of *S*-indexed vectors with entries in *R* and coordinatewise operations. If *R* is a ring and *H* is a finite commutative group then R[H] is the group ring of *H* over *R*, the ring of *H*-indexed vectors with entries in *R* and convolution as multiplication.

If *R* is a ring and *m* is a positive integer then a primitive *m*th root of 1 in *R* means an element $\zeta \in R$ such that (1) $\zeta^m = 1$; (2) $\zeta^{m/p} - 1$ is invertible in *R* for all primes *p* dividing *m*; and (3) *m* is invertible in *R* (which one can deduce from the other conditions). The notation ζ_m is specifically the complex number $\exp(2\pi i/m)$.

If *B* is a basis (of, e.g., a vector space) then the set of entries in *B* is called a "basis set";
this is not to be confused with *B* itself, which is a sequence.

200 2 Sizes

Consider all weight-*w* elements $\alpha = \alpha_0 + \alpha_1 \zeta_m + \dots + \alpha_{n-1} \zeta_m^{n-1}$ of the ring of integers $R = \mathbb{Z}[\zeta_m]$ of the power-of-2 cyclotomic field $K = \mathbb{Q}(\zeta_m)$. Here n = m/2, and "weight *w*" means 2-norm $w^{1/2}$, i.e., $\sum_j \alpha_j^2 = w$. This section analyzes the distribution of sizes of the integers $|\det_{\mathbb{Q}}^K \alpha|$.

These sizes illustrate the det^K_Q α sizes of interest in Sects. 3 and 4; those sections include analyses of the performance of various det^K_Q algorithms, and the analyses depend on the number of bits in det^K_Q α . The distribution considered in this section arises naturally in the standard *S*-unit search in Sect. 5, which enumerates small-weight elements $\alpha \in R$ and checks whether det^K_Q α factors into small primes; presumably det^K_Q α is more likely to factor appropriately if it is smaller.

One can also ask about the distribution of det^{*K*}_{$\square \alpha$} for other cyclotomic fields (and other 211 Abelian fields), but the power-of-2 case suffices as an example of what to expect. I haven't 212 found literature directly on point. There are analyses of the distribution of det ${}^{K}_{\Box} \alpha$ inside 213 the number-field sieve (see, e.g., [11, eprint version, Section 5.1; journal version, Section 214 2.2]), but NFS considers fields of relatively low degree compared to the discriminant. The 215 analysis below is conceptually similar to [9, Section 8.1], which heuristically analyzes the 216 coefficients of a Dirichlet log vector of a small element of a real multiquadratic field K on 217 a unit basis obtained from fundamental units of quadratic subfields; but the details here 218 are more complex than in [9], since the embeddings $K \to \mathbb{C}$ here are not embeddings 219 $K \to \mathbb{R}.$ 220

221 **2.1 Notation**

Throughout this section, $n \in \{2, 4, 8, 16, ...\}$; m = 2n; w is a positive integer; $K = \mathbb{Q}(\zeta_m)$; and $R = \mathbb{Z}[\zeta_m]$. For each odd integer c, the function $\sigma_c : K \to \mathbb{C}$ is the unique ring morphism taking ζ_m to ζ_m^c .

225 2.2 Upper bounds

One has $|\zeta_m| = 1$, so $|\alpha| \le \sum_j |\alpha_j| \le \sum_j \alpha_j^2 = w$. (For w > n, one can do better by replacing the inequality $\sum_j |\alpha_j| \le w$ with Cauchy's inequality $\sum_j |\alpha_j| \le n^{1/2} w^{1/2}$; but the

case of interest in Sect. 5 is that *w* is asymptotically bounded by $n^{1/2+o(1)}$.) More generally, $|\zeta_m^c| = 1$, so $|\sigma_c(\alpha)| \le w$. Hence $|\det_{\mathbb{O}}^K \alpha| = \prod_{c \in \{1,3,5,\dots,m-1\}} |\sigma_c(\alpha)| \le w^n$.

230 2.3 The circular approximation to the distribution

²³¹ If w = 1 then the above upper bound is achieved, but for larger w one expects $\sigma_c(\alpha) = \sum_j \alpha_j \zeta_m^{cj}$ to have summands $\alpha_j \zeta_m^{cj}$ pointing in different directions in \mathbb{C} , usually with sum considerably smaller than the upper bound.

Define the *circular approximation* to the distribution of $\log |\det_{\mathbb{Q}}^{K} \alpha|$ as a normal distribution with mean $n(\log w - \gamma)/2 \approx n(\log w)/2 - 0.28860783245n$, where γ is Euler's constant, and variance $n\pi^2/24 \approx 0.411233516712n$. Notice that this mean is under half of the upper bound $n \log w$ from Sect. 2.2, although the ratio converges up to 1/2 as $w \to \infty$. The following paragraphs explain how the circular approximation arises from a heuristic analysis of the size of $\log |\det_{\mathbb{Q}}^{K} \alpha|$.

The first step is to model each $\sigma_c(\alpha)$ as $\sum_j \alpha_j \exp 2\pi i \rho_{c,j}$ where each $\rho_{c,j}$ is an independent uniform random element of \mathbb{R}/\mathbb{Z} . The distribution of $\sum_j \alpha_j \exp 2\pi i \rho_{c,j}$ is invariant under rotation; the basic strategy here is to recover this distribution from its real part.

To analyze the real part of $\sum_{j} \alpha_{j} \exp 2\pi i \rho_{c,j}$, note that the variance of $\cos 2\pi \rho$ for uniform random $\rho \in \mathbb{R}/\mathbb{Z}$ is $\int_{0}^{1} (\cos 2\pi \rho)^{2} d\rho = 1/2$. The variance of $\sum_{j} \alpha_{j} \cos 2\pi \rho_{c,j}$ is $\sum_{j} \alpha_{j}^{2}/2 = w/2$ since, by independence, the summands are uncorrelated.

Now apply the heuristic that sums are normally distributed to conclude that $\sum_{j} \alpha_{j} \exp 2\pi i \rho_{c,j}$ has a complex normal distribution with mean 0 and variance ν for some ν . By definition of the complex normal distribution, the real and imaginary parts are independent normal random variables with variance $\nu/2$, so $\nu = w$.

If *N* is a complex normal random variable with mean 0 and variance 1 then $\log |N|$ has mean $-\gamma/2 \approx -0.28860783245$ and variance $\pi^2/24 \approx 0.411233516712$. If *N* is a complex normal random variable with mean 0 and variance *w* then $\log |N|$ has mean $(\log w - \gamma)/2$ and variance $\pi^2/24$. If N_1, \ldots, N_n are *n* uncorrelated complex normal random variables with mean 0 and variance *w* then $\log |N_1 \cdots N_n|$ has mean $n(\log w - \gamma)/2$ and variance $n\pi^2/24$. Finally, the sums-are-normally-distributed heuristic says that $\log |N_1 \cdots N_n|$ is normally distributed.

257 2.4 Objections to the heuristics

As *m* increases, the powers ζ_m^c for uniform random $c \in \{1, 3, ..., m-1\}$ approach a 258 uniform distribution on the unit circle in the following sense: for each $\operatorname{arc} A$ of the circle, 259 $\lim_{m\to\infty} \Pr[\zeta_m^c \in A]$ is the fraction of the circle contained in A. The same argument 260 applies to ζ_m^{cj} for any odd *j*. One can object, however, that this argument breaks down as 261 more and more powers of 2 appear in *j*: as an extreme case, ζ_m^{cj} is always 1 for j = 0. If 262 *w* is small then there is a noticeable chance that $\alpha \in F$ for a proper subfield $F \subset K$, and 263 then $\det_{\mathbb{Q}}^{K} \alpha = (\det_{\mathbb{Q}}^{F} \alpha)^{\deg_{F} K}$, with distribution determined by the distribution of $\det_{\mathbb{Q}}^{F} \alpha$. 264 (For the application to recognizing S-units, one can save time in these cases by simply 265 computing det $^{F}_{\mathbb{O}} \alpha$ and checking its factorization.) 266

Even for odd *j*, one can object to modeling ζ_m^{cj} as pointing in independent directions as the pair (*c*, *j*) varies. For example, if j' = j + m/4, then the ratio $\zeta_m^{cj'}/\zeta_m^{cj} = \zeta_m^{c(j'-j)} = i^c$ is limited to the set $\{i, -i\}$. If $\alpha \in \zeta_m F$ for a proper subfield $F \subset K$ then $\det_{\mathbb{Q}}^K \alpha = (\det_{\mathbb{Q}}^F (\alpha/\zeta_m))^{\deg_F K}$, again with distribution determined by the $\det_{\mathbb{Q}}^F$ output distribution. Furthermore, even with the uniform random directions in $\sum_{j} \alpha_{j} \exp 2\pi i \rho_{c,j}$, one can object to the heuristic of treating this sum as having a normal distribution—especially when *w* is small, since there are at most *w* nonzero summands. One can similarly object to treating $\log |N_1 \cdots N_n|$ as having a normal distribution. The sums-are-normallydistributed heuristic is only a crude approximation to the central-limit theorem.

- Compute the distribution of $\sum_{j} \alpha_{j} \cos 2\pi \rho_{c,j}$ by convolving scaled cosine distributions. A complication here is that one needs to combinatorially enumerate possibilities for $\#\{j : |\alpha_{j}| = 1\}, \#\{j : |\alpha_{j}| = 2\}$, etc.; but, for large *n* and relatively small *w*, the probabilities are dominated by the first few possibilities, those where $|\alpha_{j}|$ is rarely above 1.
- Recover the rotationally invariant distribution of $\sum_{i} \alpha_{i} \exp 2\pi i \rho_{c,i}$ from the distribu-283 tion of $\sum_{i} \alpha_{i} \cos 2\pi \rho_{c,i}$. The point is that any rotationally invariant random variable 284 can be written in polar coordinates as $r \exp 2\pi i \tau$ where τ is a uniform random ele-284 ment of \mathbb{R}/\mathbb{Z} independent of *r*; so take the distribution of the real part $r \cos 2\pi \tau$, 286 compute the Mellin transform of the density function, divide by the Mellin transform 287 of the density function of $\cos 2\pi \tau$, and compute an inverse Mellin transform to obtain 288 the density function of r. (As noted by Epstein [43], multiplying independent random 289 variables corresponds to multiplying Mellin transforms of density functions.) 290
- Compute the distribution of $\log |N_1 \cdots N_n|$ as a convolution of *n* copies of the *r* distribution.
- ²⁹³ But this still would not handle the actual directions of ζ_m^{cj} .

One can also object that the circular approximation to $\log |\det_{\mathbb{Q}}^{K} \alpha|$ cannot be exactly correct: for each (n, w), the distribution of $\log |\det_{\mathbb{Q}}^{K} \alpha|$ is discrete, while a normal distribution is continuous; also, $\log |\det_{\mathbb{Q}}^{K} \alpha|$ is bounded between 0 and $n \log w$, whereas a normal distribution is unbounded.

298 2.5 Numerical evidence

Table 1 presents, for various choices of (m, w), the mean and variance of log $|\det_{\mathbb{Q}}^{K} \alpha|$ across two sets of 65536 experiments. The set where "double" is "yes" chooses α uniformly at random from weight-*w* elements where $|\alpha_0| = 2$ and $|\alpha_j| \in \{-1, 0, 1\}$ for all other *j*. The set where "double" is "no" instead takes weight-*w* elements where $|\alpha_j| \in \{-1, 0, 1\}$ for all *j*.

These experiments were carried out with the Sage script shown in Fig. 1. The script uses deterministic seeds for reproducibility. The multicore.py used in the script is from [1]. The script also checks the integrals that account for the appearance of $\gamma/2$ and $\pi^2/24$ in this section.

Table 1 suggests that the actual mean divided by *n* is always larger than the circular approximation $(\log w - \gamma)/2$, with a gap of roughly 1/4m + 1/8w for the non-double cases, converging down to 0 as *m* and *w* jointly increase. The variance divided by *n* is consistently below the circular approximation $\pi^2/24$, indicating an anti-correlation not captured by the approximation.

m	W	Double	Mean/n	Circular	Variance/n	Circular
16	8	No	0.784449649	0.751112938	0.250840752	0.411233517
32	8	No	0.775048231	0.751112938	0.297784952	0.411233517
64	8	No	0.771626119	0.751112938	0.300073045	0.411233517
128	8	No	0.769201091	0.751112938	0.297076268	0.411233517
256	8	No	0.767971656	0.751112938	0.302569974	0.411233517
512	8	No	0.767479526	0.751112938	0.304084689	0.411233517
1024	8	No	0.767090049	0.751112938	0.303938674	0.411233517
16	8	Yes	0.818907856	0.751112938	0.225903248	0.411233517
32	8	Yes	0.812924644	0.751112938	0.230691245	0.411233517
64	8	Yes	0.806721757	0.751112938	0.237712502	0.411233517
128	8	Yes	0.804556061	0.751112938	0.248258970	0.411233517
256	8	Yes	0.803026844	0.751112938	0.253884971	0.411233517
512	8	Yes	0.802299755	0.751112938	0.254374189	0.411233517
1024	8	Yes	0.801913665	0.751112938	0.255755664	0.411233517
32	16	No	1.114195770	1.097686529	0.334337997	0.411233517
64	16	No	1.109688552	1.097686529	0.311985149	0.411233517
128	16	No	1.107739880	1.097686529	0.311517559	0.411233517
256	16	No	1.106948700	1.097686529	0.306271284	0.411233517
512	16	No	1.105902631	1.097686529	0.307222592	0.411233517
1024	16	No	1.105780026	1.097686529	0.309983682	0.411233517
32	16	Yes	1.120215430	1.097686529	0.311499371	0.411233517
64	16	Yes	1.116518517	1.097686529	0.301666920	0.411233517
128	16	Yes	1.114480430	1.097686529	0.296678360	0.411233517
256	16	Yes	1.113406893	1.097686529	0.295366668	0.411233517
512	16	Yes	1.113021383	1.097686529	0.295927515	0.411233517
1024	16	Yes	1.112847196	1.097686529	0.293394668	0.411233517
64	32	No	1.452868312	1.444260119	0.321582083	0.411233517
128	32	No	1.450717062	1.444260119	0.317665867	0.411233517
256	32	No	1.449325700	1.444260119	0.313848217	0.411233517
512	32	No	1.448456593	1.444260119	0.316434954	0.411233517
1024	32	No	1.448386058	1.444260119	0.317395974	0.411233517
64	32	Yes	1.453645761	1.444260119	0.318097521	0.411233517
128	32	Yes	1.451760998	1.444260119	0.314215624	0.411233517
256	32	Yes	1.450895244	1.444260119	0.311772454	0.411233517
512	32	Yes	1.450557535	1.444260119	0.310832695	0.411233517
1024	32	Yes	1.450065121	1.444260119	0.311119663	0.411233517
128	64	No	1.794740422	1.790833709	0.319886453	0.411233517
256	64	No	1.794126237	1.790833709	0.321972595	0.411233517
512	64	No	1.793239181	1.790833709	0.319304636	0.411233517
1024	64	No	1.792934800	1.790833709	0.319631699	0.411233517
128	64	Yes	1.794696805	1.790833709	0.320421343	0.411233517
256	64	Yes	1.794007734	1.790833709	0.318422104	0.411233517
512	64	Yes	1.793524077	1.790833709	0.317942481	0.411233517
1024	64	Yes	1.793224334	1.790833709	0.318389943	0.411233517

Table 1 Mean and variance of log $|\det_{\mathbb{Q}}^{K} \alpha|$ for 65,536 random choices of α in each line

See text for details

```
experiments = 65536
mlist = 16,32,64,128,256,512,1024
wlist = 8, 16, 32, 64
import multicore
assert (1/\exp(x)).integral(x,0,oo) == 1
assert (log(x)/exp(x)).integral(x,0,oo) == -euler_gamma
assert (log(x)^2/exp(x)).integral(x,0,oo) == euler_gamma^2 + pi^2/6
gamma = euler_gamma.n()
pi224 = (pi^2/24).n()
ZZx.<x> = PolynomialRing(ZZ, 'x', implementation='NTL')
Phi = {m:ZZx(cyclotomic_polynomial(m)) for m in mlist}
def doit(experiment):
  results = []
  with seed(experiment):
    for m in mlist:
      n = Phi[m].degree()
      for w in wlist:
        for double in 'no', 'yes':
           if double == 'yes':
            if w-3 > n: continue
            alpha = [2]+[1]*(w-4)+[0]*(n-w+3)
           else:
             if w > n: continue
             alpha = [1] * w + [0] * (n-w)
           shuffle(alpha)
           for j in range(n):
             if randrange(2):
               alpha[j] *= -1
           assert len(alpha) == n
           assert sum(alphaj<sup>2</sup> for alphaj in alpha) == w
           N = abs(Phi[m].resultant(ZZx(alpha)))
           results += [(m,n,double,w,experiment,log(RR(N)))]
  return results
data = \{\}
for results in multicore.map(doit,range(experiments)):
  for m,n,double,w,experiment,logN in results:
    print('m',m,'w',w,'double',double,'experiment',experiment,logN)
    key = w,double,m,n
    if key not in data: data[key] = []
    data[key] += [logN]
for key in sorted(data):
  w,double,m,n = key
  print('m',m,'w',w,'double',double,
     'mean/n',mean(data[key])/n,'meanheuristic/n',log(RR(w))/2-gamma/2,
     'variance/n',variance(data[key],bias=True)/n,'varianceheuristic/n',pi224)
Fig. 1 Sage script for experiments used in Table 1
```



Figure 2 plots the distribution (as a transposed cdf) of log $|\det_{\mathbb{Q}}^{K} \alpha|$ observed in the same non-double experiments (blue curve), and, for comparison, plots the circular approximation (black curve). Both curves are on a vertical scale chosen so that the circular

- approximation runs from 4 standard deviations below the mean to 4 standard deviations
- ³¹⁶ above the mean, so the circular approximation always has the same visual shape; note that
- this scale covers an interval of length only $8\sqrt{n\pi^2/24}$ within the interval [0, $n \log w$].

Journal: 40993 Article No.: 0402 TYPESET DISK LE CP Disp.: 2023/3/4 Pages: 58 Layout: BMC-OneCol

318 3 Quadratic techniques

This section reviews various standard algorithms that, for arbitrary number fields K, evaluate $\alpha \mapsto \det_{\mathbb{Q}}^{K} \alpha$. This section analyzes the cost of these algorithms applied to powerof-2 cyclotomics $K = \mathbb{Q}(\zeta_m)$, specifically for the scenario motivated in Sect. 2: namely, α is a nonzero element of $\mathbb{Z}[\zeta_m]$ of weight $n^{1/2+o(1)}$ where n = m/2, and $\det_{\mathbb{Q}}^{K} \alpha$ has $\Theta(n \log n)$ bits.

In short, the modular continued-fraction approach costs $n^2(\log n)^{3+o(1)}$ bit operations,

although there are occasional inputs α where a factor log *n* disappears because of a short

remainder sequence. The complex-embeddings approach costs $n^2(\log n)^{2+o(1)}$ bit opera-

327 tions.

328 3.1 Why resultant computation is so slow, part 1: big integers

Recall that the Euclid–Stevin algorithm to compute polynomial gcd repeatedly replaces (*f*, *g*) with (*g*, *f* mod *g*) as long as $g \neq 0$. Tracking degrees and leading coefficients of the remainders *f*, *g*, *f* mod *g*, . . . reveals the resultant. The point here is that

resultant(
$$f, g$$
) = $(-1)^{(\deg f) \deg g}$ (leadcoeff g)^{deg f-deg(f mod g)} resultant($g, f \mod g$)

if $g \neq 0$ and $f \mod g \neq 0$. There are two base cases: one has resultant(f, g) = $g^{\deg f}$ if deg g = 0, and one has resultant(f, g) = 0 if deg g > 0 and $f \mod g = 0$.

If deg $f = n > \deg g$ then the remainder sequence $f, g, f \mod g, \ldots$ inside the Euclid– Stevin algorithm has $O(n^2)$ coefficients and the quotient sequence $\lfloor f/g \rfloor, \ldots$ has O(n)coefficients. One can compute the quotient sequence in time at most $n(\log n)^{2+o(1)}$; see, e.g., [16, Sections 21–22]. Given n and the quotient sequence, one can compute the sequence of remainder degrees, the sequence of remainder leading coefficients, and the resultant. The total time is at most $n(\log n)^{2+o(1)}$.

However, one must be careful with the concept of "time" used in the previous paragraph. This is actually a count of *operations in the base field*: operations in \mathbb{Q} , if the goal is to compute a resultant of polynomials with coefficients in \mathbb{Q} .

If one takes $f = x^{1024} + 1$ and $g = 4x^{828} + x^{271} + 3$ then there are 211 Euclid–Stevin quotients. The 100th quotient is a polynomial whose coefficients have more than 100,000 bits in each numerator and denominator. The final quotient is a linear polynomial whose coefficients have more than 400,000 bits in each numerator and denominator. The cost of computing these quotients is driven much more by the number of bits than by the number of coefficients.

³⁵⁰ Collins [36] showed that simply rescaling the Euclid–Stevin remainders produces poly-³⁵¹ nomials with much smaller coefficient bounds. These polynomials are called "subresul-³⁵² tants": their coefficients are determinants of various portions of Sylvester's resultant ³⁵³ matrix. The determinant description shows that the subresultants are in $\mathbb{Z}[x]$ when ³⁵⁴ $f, g \in \mathbb{Z}[x]$. The bounds in [36] on the coefficients of subresultants come from apply-³⁵⁵ ing Hadamard's determinant inequality to bounds on the coefficients of f and g.

One still cannot escape *some* growth of coefficients. For example, recall that the resultant of $f = x^{1024} + 1$ and $g = 4x^{828} + x^{271} + 3$ has 2048 bits. This growth is not something that suddenly appears at the last moment in the subresultant algorithm: most algorithm steps are, for almost all inputs, working with large integers.

A subsequent paper by Collins [37] suggested a modular approach to computing resultant (f, g) given $f, g \in \mathbb{Z}[x]$. If one assumes schoolbook arithmetic then the modu³⁶² lar approach gives better cost bounds than the subresultant approach; this comparison ³⁶³ appears in, e.g., [86, page 449, top paragraph]. Perhaps fast arithmetic would narrow the ³⁶⁴ gap, but evaluating this would require developing a variant of fast-continued-fraction ³⁶⁵ algorithms that controls coefficient sizes, and the literature does not give any reason to ³⁶⁶ think that this effort would end up with a *faster* algorithm than the modular approach. So ³⁶⁷ let's look at the performance of the modular approach.

368 3.2 Why resultant computation is so slow, part 2: many moduli

The modular approach reconstructs resultant(f, g) from the image of resultant(f, g) in \mathbb{F}_p for enough primes p. This image is the same as the resultant of the images of f, g in $\mathbb{F}_p[x]$, as long as one avoids "bad" primes p, meaning primes that divide the leading coefficients of f and g.

How does one figure out how many primes are enough? One answer is to use Hadamard's inequality to quickly bound the resultant. Another answer, suggested by Monagan [71], is to guess that a few primes suffice, then more primes, and so on, stopping when the output is sufficiently stable; this fails with negligible probability if there is enough randomness in the primes. The Sage resultant documentation says that proof=False "may use a randomized strategy that errors with probability no more than 2⁻⁸⁰".

In the scenario studied in this section, resultant (f, g) has $\Theta(n \log n)$ bits. One can simply choose primes having enough bits for the explicit upper bounds from Sect. 2.2, although the analysis of Sect. 2 suggests that one can usually save a factor above 2 by tuning the number of primes appropriately. Either way, $\prod_p p$ has $\Theta(n \log n)$ bits. The following analysis concludes that the modular approach then costs $n^2(\log n)^{3+o(1)}$, provided that one takes each $\log \log p$ in $(\log n)^{o(1)}$.

Note first that one *can* take each log log p in $(\log n)^{o(1)}$. For example, choose a parameter y, and take all odd primes $p \le y$. By the prime-number theorem, $\prod_{p\le y} p$ reaches the desired $\Theta(n \log n)$ bits for a suitable choice of $y \in \Theta(n \log n)$. One has to skip bad primes, but one can compensate by multiplying 1 + |leadcoeff fg| into the target for $\prod_{p\le y} p$; the limited coefficient size for f and g implies that the target still has $\Theta(n \log n)$ bits. Now each $p \in O(n \log n)$, implying $\log \log p \in (\log n)^{o(1)}$. There is considerable slack in this argument: one can take much larger p and still have $\log \log p \in (\log n)^{o(1)}$.

Each continued-fraction computation in $\mathbb{F}_p[x]$ involves at most $n(\log n)^{2+o(1)}$ operations in \mathbb{F}_p (including initial reduction of f and g modulo p; f and g have small coefficients, so one does not need to batch this reduction across p). The cost of each operation in \mathbb{F}_p is at most $(\log p)(\log \log p)^{1+o(1)}$, i.e., $(\log p)(\log n)^{o(1)}$, and summing across all p gives $n(\log n)^{1+o(1)}$ since $\sum_p \log p \in \Theta(n \log n)$. The total cost is thus at most $n^2(\log n)^{3+o(1)}$.

³⁹⁷ Could the cost actually be lower than this? Strassen [92] pointed out that one log *n* ³⁹⁸ factor in the cost of continued-fraction computation actually arises as the entropy of the ³⁹⁹ list of quotient degrees in the Euclid–Stevin algorithm. The case of sparse *f* and sparse ⁴⁰⁰ *g* should not be confused with the "normal" case of all quotient degrees 1 (for example, ⁴⁰¹ if $f = x^{1024} + 1$ and $g = x^{999} + x + 1$ then there are just 28 divisions), but experiments ⁴⁰² suggest that the entropy is usually $\Theta(\log n)$ once *g* has at least 3 terms.

403 3.3 Complex embeddings

Another way to compute $\det_{\mathbb{Q}}^{K} \alpha$ for any degree-*n* number field *K* and any $\alpha \in K$ is as $\prod_{\sigma} \sigma(\alpha)$, where σ runs through all ring morphisms $K \to \mathbb{C}$. If each complex number $\sigma(\alpha)$ is represented as a floating-point number with $\Theta(n \log n)$ bits of precision then the product $\prod_{\sigma} \sigma(\alpha)$ also has $\Theta(n \log n)$ bits of precision—the n - 1 multiplications lose, in total, just $\Theta(\log n)$ bits of precision—and if the Θ constant is adjusted appropriately then this is enough precision to recover the integer $\det_{\omega}^{K} \alpha$.

Belabas [10, Section 5.2] recommended using complex embeddings to compute det^{*K*}_{\mathbb{O}} α 410 whenever det^K_{\mathbb{O}} α is "relatively small". The following paragraphs quantify the cost of this 411 approach, including the quantification from [10] but also including speedups beyond [10]. 412 The input $\alpha \in K$ is given in what Cohen [34, Section 4.2] calls the "standard represen-413 tation" of K: α is represented as a polynomial $g \in \mathbb{Z}[x]$ with $g(\theta) = \alpha$ and deg $g < \deg K$. 414 Here θ is a fixed integral primitive element of K, a root of a monic irreducible polynomial 415 $f \in \mathbb{Z}[x]$; one can think of the complex-embedding approach as another way of comput-416 ing resultant (*f*, *g*). For the power-of-2-cyclotomic case $K = \mathbb{Q}(\zeta_m)$, one takes $\theta = \zeta_m$ and 417 $f = x^n + 1.$ 418

The first step is to compute $\sigma(\alpha) = g(\sigma(\theta))$ for each embedding σ . Belabas views this as multiplying the vector of coefficients of g by a precomputed matrix with complex entries $\sigma(1), \sigma(\theta), \sigma(\theta^2), \ldots$; this is, e.g., the matrix of powers ζ_m^{cj} in the case of power-of-2 cyclotomics, where c runs through $\{1, 3, \ldots, m-1\}$. Belabas says that this multiplication costs $O(n^2 M(B))$ where n is the field degree, B is the number of bits of precision required, and M(B) is the cost of B-bit multiplication.

Three speedups noted in [10] are as follows. First, often one already knows a divisor 425 of det^K_{\square} α , so one can reduce the required precision accordingly. Second, in multiplying 426 complex numbers to obtain det ${}_{\mathbb{O}}^{K} \alpha$, one should begin by multiplying the numbers for 427 complex-conjugate σ , so that subsequent multiplications are in \mathbb{R} ; see Sect. 3.4 below. 428 Third, low-precision complex computations suffice to determine the approximate value 429 of det^K_{\square} α , pinpointing how much precision is required for an exact computation—in 430 particular, recognizing cases where det $^{K}_{\mathbb{O}} \alpha$ is unusually small. (One can also use this 431 to avoid the guesswork described above regarding how many primes are required for a 432 modular computation of a resultant.) 433

Beware that small det^K_Q α does not immediately imply that small *B* suffices: if any particular $\sigma(\alpha)$ is close to 0 then the precision obtained for $\sigma(\alpha)$ is lower than the initial precision of $\sigma(1), \sigma(\theta), \sigma(\theta^2), \ldots$, so one needs to recompute $\sigma(\alpha)$ in higher precision. The main case of interest in this section is that det^K_Q α has $\Theta(n \log n)$ bits, and then one can see that $B \in \Theta(n \log n)$ suffices as follows: each $|\sigma(\alpha)|$ is $n^{O(1)}$ as in Sect. 2.2, but $|\det^K_Q \alpha|$ is at least 1 (since $\alpha \neq 0$), so each $|\sigma(\alpha)|$ is at least $1/n^{O(n)}$. So assume $B \in \Theta(n \log n)$; the $n^2 M(B)$ from [10] is then $n^3(\log n)^{2+o(1)}$.

An asymptotically better way to compute $g(\sigma(\theta))$ for all σ , not noted in [10], is by multipoint evaluation, precomputing a tree of products of $x - \sigma(\theta)$ and then computing a tree of remainders of g modulo those products. Schönhage [84, Section 2] used segmentation to reduce multiplication in $\mathbb{C}[x]$ to multiplication in \mathbb{Z} , obtaining a cost bound $nB(\log nB)^{1+o(1)}$ for *n*-coefficient polynomials with *B* bits in each coefficient and all coefficients on the same scale; this cost bound is $n^2(\log n)^{2+o(1)}$ for $B \in \Theta(n \log n)$. Schönhage [84, Section 4] obtained the same cost bound for division in $\mathbb{C}[x]$, assuming that one is dividing by polynomials whose roots in \mathbb{C} are O(1). A multipoint-evaluation tree has $\Theta(\log n)$ layers, for total cost $n^2(\log n)^{3+o(1)}$.

A particularly efficient form of remainder tree is an FFT tree—exactly what is needed here, since *K* is assumed to be a power-of-2 cyclotomic. Even simpler than building a tree is using Bluestein's trick from [24,25] to reduce DFT to convolution. Schönhage [84, Section 3] used Bluestein's trick to obtain a cost bound $nB(\log nB)^{1+o(1)}$ for a size-*n* DFT with *B* bits of precision; i.e., cost $n^2(\log n)^{2+o(1)}$ for $B \in \Theta(n \log n)$.

The subsequent n - 1 multiplications of $\sigma(\alpha)$ values, each to *B* bits of precision, cost $nB(\log B)^{1+o(1)}$. If $B \in \Theta(n \log n)$ then the overall cost is $n^2(\log n)^{2+o(1)}$. This is, for most inputs, asymptotically better than the continued-fraction approach: it avoids an extra $(\log n)^{1+o(1)}$ factor.

459 **3.4** Complex conjugation on complex embeddings

As noted above, inside the complex-embeddings approach, Belabas suggested first multiplying complex-conjugate pairs of complex numbers. In the $3 + \zeta_{2048}^{271} + 4\zeta_{2048}^{828}$ example, this means computing the real number $(3 + \zeta_{2048}^{271c} + 4\zeta_{2048}^{828c})(3 + \zeta_{2048}^{-271c} + 4\zeta_{2048}^{-828c})$ for each pair {*c*, *-c*}. Then the subsequent multiplications are multiplications in \mathbb{R} , which, for any given precision, one expects to be at least twice as fast as multiplications in \mathbb{C} .

 A_{465} A 2× speedup is not visible at the level of detail of the analyses in this section. However, it is useful to consider what this speedup is accomplishing algebraically, for comparison to the transitivity of determinants exploited in Sect. 4.

The original problem is to evaluate $\det_{\mathbb{Q}}^{\mathbb{Q}[x]/f}$. Complex embeddings tensor with \mathbb{C} over \mathbb{Q} , reducing the original problem to the problem of evaluating $\det_{\mathbb{C}}^{\mathbb{C}[x]/f}$. The ring $\mathbb{C}[x]/f$ factors as $\prod_c \mathbb{C}[x]/(x-\zeta_m^c)$, and $\det_{\mathbb{C}}^{\mathbb{C}[x]/f}$ factors correspondingly as $\prod_c \det_{\mathbb{C}}^{\mathbb{C}[x]/(x-\zeta_m^c)}$. The image in $\mathbb{C}[x]/(x-\zeta_m^c)$ of $g \in \mathbb{C}[x]/f$ is simply $g(\zeta_m^c)$, with determinant $g(\zeta_m^c)$. Multiplying these *n* complex numbers $g(\zeta_m^c)$ produces the desired $\det_{\mathbb{C}}^{\mathbb{C}[x]/f} g = \det_{\mathbb{Q}}^{\mathbb{Q}[x]/f} g$.

The complex-conjugation speedup instead tensors with \mathbb{R} over \mathbb{Q} . The ring $\mathbb{R}[x]/f$ factors as a product of rings $\mathbb{R}[x]/((x-\zeta_m^c)(x-\zeta_m^{-c}))$, and det $\mathbb{R}^{\mathbb{R}[x]/f}$ g factors correspondingly as a product of det $\mathbb{R}^{\mathbb{R}[x]/((x-\zeta_m^c)(x-\zeta_m^{-c}))}$ g, exactly the real numbers multiplied above.

These real numbers, in turn, are computed as follows: tensor with \mathbb{C} over \mathbb{R} , and then compute the desired $\det_{\mathbb{C}}^{\mathbb{C}[x]/((x-\zeta_m^c)(x-\zeta_m^{-c}))}g$ as the product of $\det_{\mathbb{C}}^{\mathbb{C}[x]/(x-\zeta_m^c)}g$ and $\det_{\mathbb{C}}^{\mathbb{C}[x]/(x-\zeta_m^{-c})}g$, i.e., the product of $g(\zeta_m^c)$ and $g(\zeta_m^{-c})$. One can, alternatively, suppress the role of \mathbb{C} here: reduce g modulo $(x - \zeta_m^c)(x - \zeta_m^{-c}) \in \mathbb{R}[x]$ and directly compute a determinant down to \mathbb{R} .

481 3.5 Complex conjugation on the original field

Complex conjugation was used above as an automorphism of \mathbb{C} with fixed field \mathbb{R} . A different way to use complex conjugation is to restrict it to the field $K = \mathbb{Q}(\zeta_m)$. This restriction is an easy-to-compute automorphism of K, namely the ring morphism that maps ζ_m to ζ_m^{-1} . The corresponding automorphism $x \mapsto x^{-1}$ of $\mathbb{Q}[x]/f$ maps 1, $x, x^2, x^3, \ldots, x^{n-1}$, the usual basis for $\mathbb{Q}[x]/f$ as a \mathbb{Q} -vector space, to 1, $-x^{n-1}, -x^{n-2}, -x^{n-3}, \ldots, -x$ respectively. This automorphism is an easy linear map to apply.

The field $\mathbb{R} \cap K$ is the fixed field of complex conjugation on K, since \mathbb{R} is the fixed field of complex conjugation on \mathbb{C} . This field $\mathbb{R} \cap K$ has degree n/2 if $m \ge 4$, with \mathbb{Q} -basis $1, \zeta_m + \zeta_m^{-1}, \zeta_m^2 + \zeta_m^{-2}, \ldots, \zeta_m^{n/2-1} + \zeta_m^{-(n/2-1)}$. The corresponding subfield F of $\mathbb{Q}[x]/f$ has ⁴⁹¹ Q-basis 1, $x - x^{n-1}$, $x^2 - x^{n-2}$, ..., $x^{n/2-1} - x^{n/2+1}$. This is the subfield of $\mathbb{Q}[x]/f$ fixed by ⁴⁹² the automorphism $x \mapsto x^{-1}$ of $\mathbb{Q}[x]/f$; the latter automorphism is also called complex ⁴⁹³ conjugation.

Given $g \in \mathbb{Q}[x]/f$, write *h* for the product of *g* and its complex conjugate $g(x^{-1})$. Then $h = \det_F^{\mathbb{Q}[x]/f} g \in F$. One can use transitivity of determinants to compute $\det_{\mathbb{Q}}^{\mathbb{Q}[x]/f} g$ as $\det_{\mathbb{P}}^F h$, which in turn is a product of various values $h(\zeta_m^c) = g(\zeta_m^c)g(\zeta_m^{-c})$.

⁴⁹⁷ This is the same as the product of $g(\zeta_m^c)g(\zeta_m^{-c})$ values in Sect. 3.4. The difference is in ⁴⁹⁸ how the values $g(\zeta_m^c)g(\zeta_m^{-c})$ are computed: as $\det_{\mathbb{R}}^{\mathbb{C}}g(\zeta_m^c)$, or as a real embedding $h(\zeta_m^c)$ of ⁴⁹⁹ $h = \det_F^{\mathbb{Q}[x]/f} g$.

Beware that subfields of general number fields do not capture the full power of multiplying complex conjugates. For example, the field $\mathbb{Q}(\sqrt[3]{2})$ is isomorphic to $\mathbb{Q}[x]/(x^3-2)$ and has no subfields other than \mathbb{Q} and itself; but tensoring with \mathbb{C} produces $\mathbb{C}[x]/(x^3-2)$, which has two complex-conjugate factors. Conversely, multiplying complex conjugates does not capture the full power of subfields; see Sect. 4.

505 3.6 More morphisms

A common theme in computational number theory is avoiding the hassle of Archimedean 506 precision tracking by switching to the *p*-adics for a suitable prime *p*, or a product of *p*-adics. 507 Let *p* be a prime number that is totally split in *K*, i.e., a prime number for which *f* has *n* 508 distinct roots in \mathbb{F}_p . One can rapidly recognize this case by seeing that $x^p - x$ modulo f is 0. 509 Standard root-finding algorithms—or multipoint evaluation of f on \mathbb{F}_p if p is small—then 510 find the roots. One can do even better for special types of *f*: in particular, for $K = \mathbb{Q}(\zeta_m)$, 511 one can take any prime number $p \in 1 + m\mathbb{Z}$, and there are very fast algorithms to find all 512 primitive *m*th roots of 1 in \mathbb{F}_n . 513

The set of ring morphisms $x \mapsto \rho$ from $\mathbb{Z}[x]/f$ to \mathbb{F}_p , as ρ runs through roots of fin \mathbb{F}_p , is analogous to the set of complex embeddings σ used above. Evaluating all these ring morphisms on a given input $g \in \mathbb{Z}[x]/f$ is a simple matter of multipoint evaluation, assuming the roots have been precomputed; the vector of outputs can be viewed as a limited-precision representation of the input. The product of outputs is the image in \mathbb{F}_p of det $\mathbb{Z}^{\mathbb{Z}[x]/f} g = \det_{\mathbb{Q}}^K g(\theta)$, where as before θ is a root of f in K. Repeating for enough primes p (or one large enough p or any intermediate possibility) then determines det $\mathbb{C}^K_{\mathbb{Q}} g(\theta)$.

Overall this approach has similar asymptotics to the continued-fraction approach. Montgomery noted in [72, Section 4.2] that remainder trees seem to be a constant factor more efficient than continued-fraction computations for most inputs.

More generally, to compute resultant (f, g) where f has a known factorization, one can use a remainder tree to reduce g modulo each factor, and then compute resultant (f, g) as a corresponding product. Whether one should take the time to *search* for factors of f (or for primes p where f factors better) is a different question: this depends on the distribution of f and on how often f will be reused for resultants. In the applications motivating this paper (see Sect. 5), $\det_{\mathbb{Q}}^{K}$ is evaluated on many inputs in K, so many K-dependent precomputations are worthwhile.

3.7 The Galois case: exploiting automorphisms

Another convenient field where f splits completely is K itself—assuming that K is Galois.

Say $f = (x - \rho_1) \cdots (x - \rho_n)$ in K[x]. One can evaluate $g(\rho_1), \dots, g(\rho_n)$ with a remainder tree, and then use a product tree to compute the product $g(\rho_1) \cdots g(\rho_n)$, which is exactly det $\overset{K}{\square} g(\theta)$.

In the case of a power-of-2 cyclotomic $K = \mathbb{Q}(\zeta_m)$, one has $\{\rho_1, \ldots, \rho_n\} = \{\zeta_m, \zeta_m^3, \ldots, \zeta_m^{m-1}\}$ with n = m/2, since $x^n + 1 = (x - \zeta_m)(x - \zeta_m^3) \cdots (x - \zeta_m^{m-1})$. Computing $g(\zeta_m^c)$ is simply rearranging and negating coefficients. There are still *n* choices of *c*, with *n* coefficients to handle for each *c*, and then more work is required for a product.

⁵⁴⁰ Consider a 4-factor product $g(\rho_1)g(\rho_2)g(\rho_3)g(\rho_4)$. By assumption g has weight $n^{1/2+o(1)}$, ⁵⁴¹ so one expects the maximum coefficient of this product to have $\Theta(\log n)$ bits, so com-⁵⁴² puting this product costs $n(\log n)^{2+o(1)}$. There are $\Theta(n)$ such products, together costing

⁵⁴³ $n^2(\log n)^{2+o(1)}$. There are then $\Theta(\log n)$ layers in the product tree, but one can achieve ⁵⁴⁴ total cost $n^2(\log n)^{2+o(1)}$ by arranging the tree to have inputs in $\mathbb{Q}(\zeta_4)$ for the final multi-⁵⁴⁵ plication, $\mathbb{Q}(\zeta_8)$ for the multiplications on the previous layer, etc.

This approach is using some of the structure that will be exploited in Sect. 4, but still costs $n^{2+o(1)}$ because of the computations—and then multiplications—of *n* different conjugates of the input.

549 4 Linear techniques

The fast det^{*K*}_{\mathbb{Q}} α computation in Sect. 1 started with an element α of a power-of-2 cyclotomic field $K = \mathbb{Q}(\zeta_m)$ with m/2 small coefficients, computed a product $\beta = \alpha \sigma(\alpha)$ in $\mathbb{Q}(\zeta_{m/2})$ with m/4 double-size coefficients, computed a product $\beta \tau(\beta)$ in $\mathbb{Q}(\zeta_{m/4})$ with m/8quadruple-size coefficients, etc. The amount of data at each layer is essentially linear unlike the techniques in Sect. 3, which expand each of the m/2 input coefficients to a volume of data comparable to the number of output bits. This section explores the question of how general this speedup is.

557 4.1 Towers

Consider any tower $\mathbb{Q} = K_0 \subseteq K_1 \subseteq \cdots \subseteq K_t$ of number fields, with absolute degrees n_0, n_1, \ldots, n_t and relative degrees d_1, \ldots, d_t . Then $n_0 = 1, n_1 = d_1, n_2 = d_1d_2$, and so on through $n_t = d_1d_2 \cdots d_t$. For simplicity assume $d_j \ge 2$ for all j, eliminating trivial steps in the tower.

⁵⁶² Consider an algorithm that, given $\alpha \in K_t$, computes successively

$$\begin{aligned} \alpha_t &= \det_{K_t}^{K_t} \alpha & \text{as } \alpha, \\ \alpha_{t-1} &= \det_{K_{t-1}}^{K_t} \alpha & \text{as } \det_{K_{t-1}}^{K_t} \alpha_t, \\ \alpha_{t-2} &= \det_{K_{t-2}}^{K_t} \alpha & \text{as } \det_{K_{t-2}}^{K_{t-1}} \alpha_{t-1}, \\ \vdots & \\ \alpha_1 &= \det_{K_1}^{K_t} \alpha & \text{as } \det_{K_1}^{K_2} \alpha_2, \\ \alpha_0 &= \det_{K_0}^{K_t} \alpha &= \det_{\mathbb{Q}}^{K_t} \alpha & \text{as } \det_{K_0}^{K_1} \alpha_1. \end{aligned}$$

563

⁵⁶⁴ I'll assume from now on that the desired input field *K* is exactly K_t , so the output α_0 is ⁵⁶⁵ det^{*K*}_{\mathbb{O}} α . Also write $n = n_t$.

(One could, more generally, take *K* to be any subfield of K_t . If $\alpha \in K$ then one can compute $\pm \det_{\mathbb{Q}}^K \alpha$ as $\alpha_0^{1/e}$ where $e = (\deg K_t)/\deg K$. The sign of $\det_{\mathbb{Q}}^K \alpha$ is clear if *e* is odd; one can use low-precision complex embeddings to recover the sign in the general case, ⁵⁶⁹ if the sign matters for the application. However, so far I haven't found any cases where ⁵⁷⁰ allowing e > 1 saves time compared to reducing to the case e = 1, i.e., replacing each K_j ⁵⁷¹ with $K_i \cap K$, obtaining a tower for K.)

To analyze how costs scale, let's postulate the following scenario: each α_j has $O(n \log n)$ bits across its n_j coefficients, with $O((n/n_j) \log n)$ bits in each coefficient; α_j has $\Theta(n \log n)$ bits whenever $n_j < n/2$; α_0 has $\Theta(n \log n)$ bits. The idea that this is a typical scenario is an extrapolation from Sect. 2.

⁵⁷⁶ One might now hypothesize, extrapolating from Sect. 3, that computing α_{j-1} from α_j ⁵⁷⁷ has cost growing as d_j^2 times the number of bits in α_{j-1} times $(\log n)^{e+o(1)}$, where e = 1⁵⁷⁸ for "FFT-friendly" choices of K and e = 2 for other choices of K. The total cost in the ⁵⁷⁹ above scenario is then at most $(d_1^2 + \dots + d_t^2)n(\log n)^{e+1+o(1)}$. (The only reason for saying ⁵⁸⁰ "at most" under these hypotheses is that not all α_j are assumed to have $\Theta(n \log n)$ bits; in ⁵⁸¹ particular, α_t is merely assumed to have $O(n \log n)$ bits.)

The sum $d_1^2 + \dots + d_t^2$ is at least $tn^{2/t}$. It is exactly $tn^{2/t}$ if $d_1 = \dots = d_t$. For example, it is $4 \log_2 n$ if $d_1 = \dots = d_t = 2$; $8 \log_2 n$ if $d_1 = \dots = d_t = 4$; and $64 \log_2 n$ if $d_1 = \dots = d_t = 16$. On the other hand, it is $4(t-1) + n^2/4^{t-1}$ if $d_1 = d_2 = \dots = d_{t-1} = 2$ and $d_t = n/2^{t-1}$.

Define a *smooth tower* as one where $d_j \in (\log n)^{o(1)}$ for each *j*. This does not require t to grow as $\Theta(\log n)$: for example, one could have $d_j \in (\log n)^{\Theta(1/\log \log \log n)}$ for each *j*, and $t \in \Theta((\log n)(\log \log \log n)/\log \log n)$. For a smooth tower, the total cost above assuming the scenario described above, and assuming the hypothesized cost of each step of the computation—is $n(\log n)^{e+2+o(1)}$.

591 4.2 Why multiplication is perceived to be fast

Let's see whether it's possible to justify the above hypothesis regarding the cost of computing α_{j-1} from α_j . Note that, for a smooth tower, d_j^2 is a minor cost factor, and larger powers of d_j in the cost would contribute at most (log n)^{o(1)}. The main cost factors to worry about are the n_{j-1} coefficients in α_{j-1} and the number of bits per coefficient, typically giving $\Theta(n \log n)$ bits overall.

⁵⁹⁷ Consider the case that K_j has the form $K_{j-1}(\sqrt{\delta})$, with $d_j = 2$, where δ is a non-square ⁵⁹⁸ in K_{j-1} . Write σ for the unique automorphism of K_j fixing K_{j-1} and mapping $\sqrt{\delta}$ to $-\sqrt{\delta}$. ⁵⁹⁹ Then K_{j-1} is the fixed field of σ , and one can compute α_{j-1} as $\alpha_j \sigma(\alpha_j)$, as in the power-⁶⁰⁰ of-2-cyclotomic example in Sect. 1. How quickly can one multiply two elements of K_j , ⁶⁰¹ namely α_j and $\sigma(\alpha_j)$?

As in Sect. 3.3, let's use the "standard representation" of a number field as $\mathbb{Q}[x]/\varphi$ for some monic irreducible polynomial $\varphi \in \mathbb{Z}[x]$, with elements of $\mathbb{Q}[x]/\varphi$ in turn expressed as elements of $\mathbb{Q}[x]$ of degree below deg φ . One of the reasons for the popularity of this representation is that it reduces number-field computations to polynomial computations, which in turn are well known to have fast algorithms.

In particular, multiplying two elements of $\mathbb{Z}[x]/\varphi$ —let's not get distracted here by the possibility of integral elements having denominators in this representation—means multiplying two integer polynomials and then reducing the product modulo φ . There are well-known fast algorithms for each step, and it is easy to prove bounds on the output coefficients.

Specifically, let *n* and *H* be positive integers. The product of two *n*-coefficient polynomi-612 als g, $h \in \mathbb{Z}[x]$, with each coefficient of g, h in the interval [-H, H], is a (2n-1)-coefficient 613 polynomial with each coefficient in $[-nH^2, nH^2]$. One way to compute this product is 614 by segmentation: multiply the integers $g(2^e)$ and $h(2^e)$ where $e = \lfloor \log_2(4nH^2) \rfloor$, and 615 recover gh from $g(2^e)h(2^e)$. The integers $g(2^e)$ and $h(2^e)$ have $O(n \log 2nH)$ bits. Standard 616 integer-multiplication algorithms take time essentially linear in $n \log 2nH$. This approach 617 of combining segmentation with fast multiplication was used by Schönhage [84, Section 618 2], as noted in Sect. 3. 619

For reduction, one can multiply *gh* by a sufficiently precise approximation of the power 620 series $1/\varphi \in \mathbb{Z}[[x^{-1}]]$, round down to obtain $|fg/\varphi|$, multiply by φ , and subtract from fg621 to obtain fg mod φ . All of this is fast when φ and the approximation to $1/\varphi$ have small 622 coefficients. For example, if φ is the 1009th cyclotomic polynomial $(x^{1009} - 1)/(x - 1)$, then 623 a sufficiently precise approximation to $1/\varphi$ is $x^{-1008} + x^{-1009}$. As a variant, rather than 624 reducing gh after recovering it from $g(2^e)h(2^e)$, one can first reduce $g(2^e)h(2^e)$ modulo 625 $\varphi(2^e)$ and then recover *gh* mod φ , provided that *e* is chosen large enough; see, e.g., [44, 626 Proposition 1]. 627

There are more details to fill in regarding the cost of computing σ , how to handle $d_j > 2$, etc., but the above description might make it seem plausible that one can use any smooth tower for K to quickly compute det^K_Q α . A closer look shows, however, that multiplication is not so easy.

4.3 The challenge of multiplying quickly in subfields

The computation at hand isn't simply multiplying in one field $\mathbb{Q}[x]/\varphi$. It's computing an element of a subfield of, say, half degree, and then continuing recursively with fast operations *in that subfield*.

⁶³⁶ Write $K = \mathbb{Q}[x]/\varphi$. Assume that $\theta \in K$ generates a subfield F with deg $F = (\deg K)/2$. ⁶³⁷ Write ψ for the minimal polynomial of θ , and write E for the field $\mathbb{Q}[y]/\psi$. The ring ⁶³⁸ morphism $y \mapsto \theta$ from $\mathbb{Q}[y]$ to F induces an isomorphism from E to F. Applying the ⁶³⁹ inverse of this isomorphism to det^K_F $g \in F$ produces an element of E, reducing the problem ⁶⁴⁰ of evaluating det^K_Q to the half-degree problem of evaluating det^E_Q. But how fast is this ⁶⁴¹ inverse isomorphism?

⁶⁴² Cohen's second textbook [35, page 65, top paragraph] considers this problem (mention-⁶⁴³ ing, as an example, taking a "relative trace or norm" from $\mathbb{Q}[x]/\varphi$ down to F and repre-⁶⁴⁴ senting it as an element of $\mathbb{Q}[y]/\psi$) and suggests falling back to linear algebra, treating the ⁶⁴⁵ isomorphism as a \mathbb{Q} -module isomorphism and inverting the matrix for this isomorphism. ⁶⁴⁶ Simply looking at the matrix inverse already involves a quadratic number of matrix ⁶⁴⁷ entries. One might hope for a fast inversion method exploiting the structure of this matrix; ⁶⁴⁸ but, no, the situation is even worse.

Take, for example, $\varphi = (x^{1009} - 1)/(x - 1)$, and consider the subfield *F* of $K = \mathbb{Q}[x]/\varphi$ generated by x + 1/x. The minimal polynomial $\psi \in \mathbb{Q}[y]$ of x + 1/x is

$$y^{504} + y^{503} - 503y^{502} - \dots - 917285588550945621669039$$

9659548581991981915884700658789713569504909034649084227
6029300596063250047752380 y^{226} - \dots + 2667126y^3 - 31878y^2 - 252y + 1

⁶⁵² with coefficients as large as 346 bits.

651

The effects that force large coefficients in this polynomial ψ also force the inverse matrix mentioned above to have many large entries. The typical outputs of the inverse isomorphism are correspondingly large, no matter what method is used to compute them. Take, e.g., the small element $x^{500} + 1/x^{500} = x^{500} + x^{509}$ of *K*. This has, under the isomorphism, preimage $2T_{500}(y/2)$ where T_j is the *j*th Chebyshev polynomial of the first kind; 102 coefficients of this polynomial have more than 300 bits each, including 344-bit coefficients of y^{220} , y^{222} , y^{224} , y^{226} .

To summarize, the elements of $\mathbb{Q}[y]/\psi$ being multiplied won't normally have small coefficients, and ψ doesn't have small coefficients. This breaks multiple steps in the argument that arithmetic in this field is fast. If the original input $g \in \mathbb{Q}[x]/\varphi$ has very large coefficients, then there isn't much impact from the extra size of ψ etc., but the applications motivating this paper (see Sect. 5) start with very small coefficients.

665 4.4 The superfield representation

The inconvenience of working with "the standard representation" of a real-cyclotomic field $\mathbb{R} \cap \mathbb{Q}(\zeta_m) = \mathbb{Q}(\zeta_m + \zeta_m^{-1})$, such as working with $\mathbb{Q}[y]/\psi$ for the degree-504 polynomial ψ shown above in the case m = 1009, is not a new observation. The literature on computations in $\mathbb{R} \cap \mathbb{Q}(\zeta_m)$ typically represents field elements as elements of the larger field $\mathbb{Q}(\zeta_m)$, which in turn is represented as $\mathbb{Q}[x]/\Phi_m$ where Φ_m is the *m*th cyclotomic polynomial.

However, this representation is redundant, for example with only 504 degrees of freedom in the 1008 coefficients for m = 1009. Multiplying elements represented in this way is correspondingly redundant. One cannot simply dismiss this effect as a constant-factor slowdown: if elements of a degree- n_j field K_j are represented as elements of a degree- n_t field K_t then there are n_t/n_j times as many coefficients as desired, and n_t/n_j can be on the scale of n.

678 4.5 Relative representations

⁶⁷⁹ Cohen's second textbook includes a chapter [35, Chapter 2] on "basic relative number ⁶⁸⁰ field algorithms", saying [35, Section 2.1.1] that, compared to representing a number field ⁶⁸¹ L as an extension of \mathbb{Q} , representing L as an extension of a nontrivial subfield K is "usually ⁶⁸² preferable". Two reasons stated in [35] for this preference are that

- The defining polynomial of *L* over *K* is of "lower degree" and
- "The *K*-structure on *L* gives considerably more arithmetical information than considering *L* on its own".

For example, consider again the case that $K_j = K_{j-1}(\sqrt{\delta})$, where δ is a non-square in K_{j-1} . The K_{j-1} -relative representation of K_j (to be more precise, of the pair $(K_j, \sqrt{\delta})$) is as $K_{j-1}[x]/(x^2 - \delta)$: the polynomial $\alpha_0 + \alpha_1 x$ in $K_{j-1}[x]$ represents the element $\alpha_0 + \alpha_1 \sqrt{\delta}$ of K_j . The specified generators of K_j as a \mathbb{Q} -vector space are simply the generators of K_{j-1} followed by $\sqrt{\delta}$ times the same generators. Extracting α_0, α_1 from this representation is simply extracting the first half and the second half of the coefficients.

How quickly can one multiply two elements in the relative representation of K_j ? A standard Karatsuba-type multiplication of $\alpha_0 + \alpha_1 x$ by $\beta_0 + \beta_1 x$ in $K_{j-1}[x]$ involves three multiplications in K_{j-1} . One also incurs a multiplication by δ to reduce modulo x^2 –

- ⁶⁹⁵ δ, although often one can choose δ to make this multiplication very fast. At best this ⁶⁹⁶ algorithm reduces a multiplication problem to three half-size multiplication problems. If ⁶⁹⁷ this is applied recursively in a 2-power tower then degree-*n* multiplication involves $n^{\log_2 3}$
- multiplications in \mathbb{Q} , not counting the δ multiplications.
- As noted in Sect. 1, van der Hoeven and Lecerf [59] suggested choosing towers with superconstant relative degrees d_j so as to reduce the number of base-field multiplications to $n^{1+o(1)}$. One can obtain $\cot n(\log n)^{O(1)}$ for multiplications in a tower represented in this way by requiring each layer to have multiplication overhead $(\log n)^{O(1/t)}$ for relative degree $n^{\Theta(1/t)}$; this is easy for $t \in \Theta(1)$, but seems hard for $t \in \Theta((\log n)/\log \log n)$. See also the discussion of open problems in [59, Conclusion]. I don't see how this approach can obtain $\cot n(\log n)^{O(1)}$ for computing $\det_{\mathbb{O}}^{K} \alpha$ with $\Theta(n \log n)$ bits.

706 4.6 The Abelian case

From now on, let's focus on Abelian number fields, i.e., Galois number fields with commutative Galois groups, and see whether this added structure gives faster algorithms.

The Kronecker–Weber theorem states that each Abelian number field is a subfield of some cyclotomic field $\mathbb{Q}(\zeta_m)$; see, e.g., [94, Theorem 14.1]. Conversely, subfields of cyclotomic fields are certainly Abelian. The smallest positive integer *m* such that $K \subseteq \mathbb{Q}(\zeta_m)$ is called the conductor of *K*.

(In this paper, as in [69, page 9], a "number field" is a subfield of $\mathbb C$ having finite dimension 713 as a \mathbb{Q} -vector space. Often the literature defines "number field" more broadly as a field 714 containing \mathbb{O} and having finite dimension as a \mathbb{O} -vector space; but this broader definition 714 breaks typical statements of the Kronecker–Weber theorem, such as [94, Theorem 14.1]. 716 As a workaround, one could say that each number field in this broader sense is isomorphic 717 to a number field in the strict sense, so each Abelian number field in this broader sense 718 is isomorphic to a subfield of $\mathbb{Q}(\zeta_m)$ for some *m*; or, as in [61, Theorem 5.9], one could 719 say that each Abelian number field in this broader sense has a superfield of the form $\mathbb{Q}(\zeta)$ 720 where ζ is a root of unity.) 721

In analyses of the costs of algorithms below, I'll ignore the cost of various per-field 722 precomputations. Formally, this is most easily described as existence of an algorithm 723 A_K for each suitable field K; the cost of evaluating $K \mapsto A_K$ is irrelevant to the cost 724 of evaluating $\alpha \mapsto A_K(\alpha)$. In reality, optimizing the precomputation cost could be of 725 interest, but only in corner cases where *m* is much larger than *n* or where there are not 726 many computations for each K. Specifically, all of the precomputations below take time 727 $m^{O(1)}$; I'll assume that m is $n^{O(1)}$, so the precomputation time is also $n^{O(1)}$, which is, at 728 least asymptotically, outweighed by the number of $\det_{\mathbb{O}}^{K}$ evaluations in Section 5. 729

730 4.7 The Gauss-period representation for prime conductor

⁷³¹ Within the set of Abelian fields, let's start with the case of odd prime conductor *p*, specif-⁷³² ically by reviewing a standard construction of all of the subfields of $\mathbb{Q}(\zeta_p)$; these all have

- ⁷³³ conductor *p*, except for \mathbb{Q} , which has conductor 1.
- This construction is due to Gauss. The proof that each subfield of $\mathbb{Q}(\zeta_p)$ is one of Gauss's fields is typically presented today as an application of Galois theory, but my impression is
- ⁷³⁶ that this application is merely a restatement in different language of facts that Gauss had

proven in [48]. Gauss stated facts without proof for more general cyclotomic fields (see
[2]); I'll return later to the difficulties that appear in the general case.

739 4.7.1 Example: the Gauss periods for p = 17

Gauss's ruler-and-compass construction of a 17-gon [48, Section 354] exhibited, in essence, a tower of number fields $\mathbb{Q} = K_0 \subset K_1 \subset K_2 \subset K_3 \subset K_4 = \mathbb{Q}(\zeta_{17})$ with $\deg K_j = 2^j$. Explicitly, with ζ_{17} abbreviated as ζ :

- K_4 has \mathbb{Q} -basis $\zeta^{\pm 1}$, $\zeta^{\pm 2}$, $\zeta^{\pm 3}$, $\zeta^{\pm 4}$, $\zeta^{\pm 5}$, $\zeta^{\pm 6}$, $\zeta^{\pm 7}$, $\zeta^{\pm 8}$.
- K_3 has \mathbb{Q} -basis $\zeta + \zeta^{-1}, \zeta^2 + \zeta^{-2}, \zeta^3 + \zeta^{-3}, \zeta^4 + \zeta^{-4}, \zeta^5 + \zeta^{-5}, \zeta^6 + \zeta^{-6}, \zeta^7 + \zeta^{-7}, \zeta^8 + \zeta^{-6}, \zeta^7 + \zeta^{-7}, \zeta^8 + \zeta^{-6}, \zeta^7 + \zeta^{-7}, \zeta^8 + \zeta^{-6}, \zeta^7 + \zeta^{-6}, \zeta^7 + \zeta^{-7}, \zeta^8 + \zeta^{-6}, \zeta^7 + \zeta^{-6}, \zeta^7 + \zeta^{-7}, \zeta^8 + \zeta^{-6}, \zeta^7 + \zeta^{-6}, \zeta^7 + \zeta^{-7}, \zeta^8 + \zeta^{-6}, \zeta^7 + \zeta^{-6}, \zeta^7 + \zeta^{-7}, \zeta^8 + \zeta^{-6}, \zeta^7 +$
- ζ^{-8} . Each basis element displayed here has exponents c, -c modulo 17 for some c; note that $\{1, -1\}$ is the unique subgroup of $(\mathbb{Z}/17)^*$ of order 2.
- K_2 has Q-basis $\zeta + \zeta^4 + \zeta^{-4} + \zeta^{-1}$, $\zeta^2 + \zeta^8 + \zeta^{-8} + \zeta^{-2}$, $\zeta^3 + \zeta^{-5} + \zeta^5 + \zeta^{-3}$, $\zeta^6 + \zeta^7 + \zeta^{-7} + \zeta^{-6}$. Each basis element displayed here has exponents c, 4c, -4c, -cmodulo 17 for some c; note that {1, 4, -4, -1} is the unique subgroup of ($\mathbb{Z}/17$)* of order 4.

• K_1 has \mathbb{Q} -basis $\zeta + \zeta^2 + \zeta^4 + \zeta^8 + \zeta^{-8} + \zeta^{-4} + \zeta^{-2} + \zeta^{-1}$, $\zeta^3 + \zeta^6 + \zeta^{-5} + \zeta^7 + \zeta^{-2} + \zeta^{-7} + \zeta^5 + \zeta^{-6} + \zeta^{-3}$. These are $(-1 + \sqrt{17})/2$ and $(-1 - \sqrt{17})/2$; the field K_1 is $\mathbb{Q}(\sqrt{17})$.

These basis elements are called "Gauss periods" (or "Gaussian periods"), not to be confused with "Gauss sums", which are Fourier transforms of Gauss periods. Beware that the literature sometimes uses the terminology "Gauss sums" for Gauss periods; see, e.g., [12].

757 4.7.2 Constructing Gauss periods for any odd prime p

- Let *p* be an odd prime number. The field $\mathbb{Q}(\zeta_p)$ has Galois group isomorphic to $(\mathbb{Z}/p)^*$; each element $c \in (\mathbb{Z}/p)^*$ corresponds to the unique automorphism σ_c of $\mathbb{Q}(\zeta_p)$ mapping ζ_p to ζ_p^c . Note that this automorphism permutes the \mathbb{Q} -basis $\zeta_p, \zeta_p^2, \ldots, \zeta_p^{p-1}$ of $\mathbb{Q}(\zeta_p)$.
- The group $(\mathbb{Z}/p)^*$ is a cyclic group with $\#(\mathbb{Z}/p)^* = p 1$, so, for each divisor d of p 1, there is a unique subgroup H of $(\mathbb{Z}/p)^*$ with #H = d. The fixed field of the corresponding group of automorphisms is the unique subfield F of $\mathbb{Q}(\zeta_p)$ of degree (p-1)/d.

Explicitly, the Gauss period $\zeta_p^j + \zeta_p^{c^j} + \zeta_p^{c^{2j}} + \cdots + \zeta_p^{c^{d-1}j}$, where $j \in (\mathbb{Z}/p)^*$ and d is the order of c in $(\mathbb{Z}/p)^*$, is in the fixed field F of σ_c ; it is exactly $\operatorname{tr}_F^{\mathbb{Q}(\zeta_p)} \zeta_p^j$. The set $\{\operatorname{tr}_F^{\mathbb{Q}(\zeta_p)} \zeta_p^j : j \in (\mathbb{Z}/p)^*\}$ is a \mathbb{Q} -basis set for F. This follows from the fact that $\{\zeta_p^j : j \in (\mathbb{Z}/p)^*\}$ is a \mathbb{Q} -basis set for $\mathbb{Q}(\zeta_p)$. The point is that σ_c maps $\alpha = \sum_{j \in (\mathbb{Z}/p)^*} \alpha_j \zeta_p^j$ to $\sum_{j \in (\mathbb{Z}/p)^*} \alpha_j \zeta_p^{cj} = \sum_{j \in (\mathbb{Z}/p)^*} \alpha_{j/c} \zeta_p^j$, so σ_c fixes α exactly when $\alpha_j = \alpha_{j/c}$ for each j, i.e., exactly when $j \mapsto \alpha_j$ is constant on orbits of multiplication by c.

The \mathbb{Q} -basis $\zeta_p, \zeta_p^2, \ldots, \zeta_p^{p-1}$ of $\mathbb{Q}(\zeta_p)$ is an integral basis: its \mathbb{Z} -span is $\mathbb{Z}[\zeta_p]$, the ring of integers of $\mathbb{Q}(\zeta_p)$. Consequently the Gauss-period basis for each subfield of $\mathbb{Q}(\zeta_p)$ is an integral basis for that subfield.

Another important feature of the Gauss-period basis is that one can efficiently compute conjugates of field elements represented as \mathbb{Q} -linear combinations of Gauss periods. For the same reason, the representation is subfield-compatible (and hence compatible with any given tower of subfields): if $K \subseteq L$ are subfields of $\mathbb{Q}(\zeta_p)$ then one can efficiently (1) map an element of K from K's Gauss-period representation to L's Gauss-period representation, and (2) invert this map on its image.

779 4.8 Multiplication algorithms for the Gauss-period representation for prime conductor

⁷⁸⁰ Bach and Shallit [7, Section 4, "period basis"], crediting Lenstra, used the Gauss-period ⁷⁸¹ basis for computations in an Abelian field K of prime conductor p. The multiplication ⁷⁸² algorithm in [7] for this basis has cost cubic in the degree of K.

Gao, von zur Gathen, and Panario [47, Section 3] instead used the superfield representation of elements of *K* as elements of $\mathbb{Q}(\zeta_p)$, giving multiplication cost essentially linear in *p*. This cost can be much smaller than cubic in the degree of *K*, but can also be much larger: consider, as an extreme example, the quadratic field $K = \mathbb{Q}(\sqrt{p})$.

The general issue here was noted in [7, page 206, "we would like to avoid using the larger 787 field"]; the issue for $\mathbb{Q}(\sqrt{p})$ was noted in [29, Section 7, first paragraph]. This was not a 788 big issue for [47]. The goal of [47] was to multiply in \mathbb{F}_q for a given prime power q; the 789 strategy in [47] was to represent \mathbb{F}_q as a quotient of a degree-*n* subring of $\mathbb{Q}(\zeta_p)$; in this 790 context, one can safely restrict attention to the case that p - 1 is not much larger than n. 791 However, if the goal is instead to evaluate $\det_{\mathbb{Q}}^{\mathbb{Q}(\zeta_p)}$ via a smooth tower of subfields of $\mathbb{Q}(\zeta_p)$ 792 then one ends up considering subfields of many degrees, with similar data volume in each 793 degree, so one cannot ignore the gap between the degree and p-1. 794

Let's look more closely at known essentially-linear-time multiplication algorithms for $\mathbb{Q}(\zeta_p)$. A conventional FFT modulo $x^m - 1$, where *m* is a power of 2 above 2p, works with the additive structure of the exponent group \mathbb{Z}/m . A different essentially-linear-time DFT algorithm, introduced by Rader in [82], instead works with the multiplicative structure of $(\mathbb{Z}/p)^*$ —and we'll see in a moment that Rader's algorithm can easily take advantage of the symmetries of the Gauss-period basis for a subfield of $\mathbb{Q}(\zeta_p)$.

⁸⁰¹ See Appendix A for software to double-check the main algorithms presented here.

802 4.8.1 Rader's FFT

The goal is to compute a size-*p* DFT over a ring *R* where *p* is an odd prime: i.e., to compute $g(1), g(\zeta), \ldots, g(\zeta^{p-1})$ given $g \in R[x]$ with deg g < p, where ζ is a primitive *p*th root of 1 in *R*.

Write g as $g_0 + g_1 x + \dots + g_{p-1} x^{p-1}$. Rader handles g_0 separately (simply adding g_0 to each output), and handles g(1) separately, easily reducing to the problem of computing $g(\zeta), \dots, g(\zeta^{p-1})$ where $g = g_1 x + \dots + g_{p-1} x^{p-1}$. Let's now focus on that problem.

View *g* as an element of the group ring $R[\mathbb{Z}/p]$; i.e., view the indices of *g* as elements of Z/p. Let ω be a generator of $(\mathbb{Z}/p)^*$. Then

811
$$g(\zeta^{\omega^b}) = \sum_{j \in (\mathbb{Z}/p)^*} g_j \zeta^{\omega^b j} = \sum_{a \in \{0,1,\dots,p-2\}} g_{\omega^{-a}} \zeta^{\omega^{b-a}}.$$

In other words, $O_b = \sum_a I_a Z_{b-a}$, where $I_a = g_{\omega^{-a}}$, $Z_b = \zeta^{\omega^b}$, and $O_b = g(\zeta^{\omega^b})$: the output sequence O is a length-(p-1) cyclic convolution of the input sequence I and the constant sequence Z, i.e., a product in the group ring $R[\mathbb{Z}/(p-1)]$. Rader concludes by pointing to essentially-linear-time subroutines for cyclic convolution.

816 4.8.2 Inverting Rader's FFT

The standard principle that a DFT with exponents negated is an inverse DFT, aside from scaling by a constant factor, means that one can use a DFT algorithm for an inverse DFT without inspecting the details of the algorithm. However, seeing how to merge this principle into the details of Rader's algorithm turns out to be useful for the generalized
algorithms in Sects. 4.8.4 and 4.12.4.

The details are as follows. Again handle g_0 and g(1) separately, easily reducing to the problem of recovering $g = g_1 x + \dots + g_{p-1} x^{p-1}$ given $g(\zeta), \dots, g(\zeta^{p-1})$, i.e., recovering the above sequence I from the above sequence O. Define $Z'_b = (Z_{(p-1)/2-b} - 1)/p$. The following calculation, where the indices a, b range over $\mathbb{Z}/(p-1)$, shows that Z has a convolution inverse, specifically Z':

827

836

$$p\sum_{a} Z_{a} Z_{b-a}' = \sum_{a} \zeta^{\omega^{a}} \left(\zeta^{\omega^{(p-1)/2+a-b}} - 1 \right)$$
$$= \sum_{a} \zeta^{\omega^{a}} \zeta^{-\omega^{a-b}} - \sum_{a} \zeta^{\omega^{a}} = \sum_{a} \zeta^{(1-\omega^{-b})\omega^{a}} - \sum_{a} \zeta^{\omega^{a}}.$$

This last quantity is, as desired, p if b = 0, else 0; the point is that $\sum_{a} \zeta^{s\omega^{a}}$ is p - 1 if s = 0, else -1. Hence convolution with Z' is deconvolution with Z; in particular, I is the convolution of O and Z'.

4.8.3 Exploiting input symmetries in Rader's FFT

If *g* is real, meaning that $g_j = g_{-j}$ for each $j \in (\mathbb{Z}/p)^*$, then $g(\zeta^c) = g(\zeta^{-c})$. In other words, if the input sequence *I* is periodic with period (p-1)/2, then the output sequence *O* is also periodic with period (p-1)/2. One can exploit this twofold symmetry by folding the *Z* sequence: one has

$$\begin{aligned} O_b &= \sum_a I_a Z_{b-a} = \sum_{0 \le a < (p-1)/2} I_a Z_{b-a} + \sum_{(p-1)/2 \le a < p-1} I_a Z_{b-a} \\ &= \sum_{0 \le a < (p-1)/2} I_a (Z_{b-a} + Z_{b-a + (p-1)/2}) = \sum_{0 \le a < (p-1)/2} I_a Y_{b-a} \end{aligned}$$

where $Y_b = Z_b + Z_{b+(p-1)/2}$. This expresses the first half of *O* as a length-((p-1)/2) cyclic

⁸³⁸ convolution of Y and the first half of I.

More generally, fix a positive integer d dividing p-1, and say $g_j = g_{j\omega^d}$ for all $j \in (\mathbb{Z}/p)^*$. (The previous paragraph is the case d = (p-1)/2.) Then the input and output sequences are periodic with period d, and are determined by their first d entries, i.e., the entries at positions 0 through d-1. The first d entries of the output O are a length-d cyclic convolution of Y and the first d entries of I, where now $Y_b = Z_b + Z_{b+d} + \cdots + Z_{b+p-1-d}$. The number of operations in this convolution, after precomputation of the Y sequence, is essentially linear in d: more precisely, $d(\log d)^{1+o(1)}$.

This folded generalization of Rader's FFT algorithm is not new. Arita and Handa [6, Sections 3.3–3.4] considered the Gauss-period basis of a subfield of $\mathbb{Q}(\zeta_p)$ (not mentioning that these are Gauss periods), considered DFTs (under another name) of elements of the subfield, and expressed these DFTs as convolutions (not mentioning Rader's algorithm).

4.8.4 Inverting a folded Rader FFT

⁸⁵¹ Define $Y'_b = Z'_b + Z'_{b+d} + \dots + Z'_{b+p-1-d} = (Y_{(p-1)/2-b} - (p-1)/d)/p$. Then Y' is the ⁸⁵² convolution inverse of Y: the folding map from the group ring $R[\mathbb{Z}/(p-1)]$ to $R[\mathbb{Z}/d]$ ⁸⁵³ maps Z to Y, maps Z' to Y', and maps the equation ZZ' = 1 to the equation YY' = 1. ⁸⁵⁴ Convolution with Y' thus inverts the folded Rader algorithm from Sect. 4.8.3.

In other words: Consider the problem of recovering, from the input described in a moment, a polynomial $g \in R[x]$ with deg g < p, with g(0) = 0, and with the periodicity 863

 $g_j = g_{j\omega^d}$ for all $j \in (\mathbb{Z}/p)^*$, where indices are again interpreted as elements of \mathbb{Z}/p . The 857 input consists of the first *d* entries of the *d*-periodic sequence *O* defined by $O_b = g(\zeta^{\omega^b})$; 858 i.e., the values of g at ζ , ζ^{ω} , ζ^{ω^2} , ..., $\zeta^{\omega^{d-1}}$. 859

To solve this problem, simply apply a length-*d* cyclic convolution of the input sequence 860

with Y', obtaining the first d entries of the sequence I defined by $I_a = g_{\omega^{-a}}$. These entries 861 are the coefficients of g on the *R*-basis 862

 $\dots n-1-d$

$$x^{\omega^{0}} + x^{\omega^{d}} + x^{\omega^{2d}} + \dots + x^{\omega^{p-1-2d}} + x^{\omega^{p-1-d}},$$

$$x^{\omega^{-1}} + x^{\omega^{d-1}} + x^{\omega^{2d-1}} + \dots + x^{\omega^{p-2-2d}} + x^{\omega^{p-2-d}}$$

$$\vdots$$

$$w^{1-d} + w^{1} + w^{d+1} + \dots + w^{p-3d} + w^{p-2d}$$

$$x^{-} + x^{-} + x^{-} + \cdots + x^{-} + x^{-}$$

Note that replacing x with ζ_p in these formulas produces the Gauss periods. 864

n = 1 = 2d

Combining the folded Rader FFT with the inverse folded Rader FFT produces a fast 865 multiplication algorithm for this type of periodic polynomial. One is given two peri-866 odic polynomials f, g; one uses the folded Rader FFT to evaluate the polynomials at 867 $\zeta, \zeta^{\omega}, \zeta^{\omega^2}, \ldots, \zeta^{\omega^{d-1}}$; one then multiplies pointwise and uses the inverse folded Rader 868 FFT to interpolate, obtaining a periodic polynomial h with the same values as f, g. The 860 periodicity implies that h has the same values as fg at all powers of ζ , so h = fg in 870 $R[x]/((x^p-1)/(x-1))$. All of this takes just $d(\log d)^{1+o(1)}$ operations in *R*, after precom-871 putation of the *Y* sequence. 872

4.8.5 Integers as a base ring 873

...0

Again fix a positive integer *d* dividing p - 1. Consider *d*-periodic polynomials $g \in \mathbb{Z}[x]$, 874 defined as polynomials $g \in \mathbb{Z}[x]$ satisfying deg g < p, g(0) = 0, and $g_i = g_{i\omega^d}$ for all 875 $j \in (\mathbb{Z}/p)^*$, with indices interpreted as elements of \mathbb{Z}/p . As above, represent *d*-periodic 876 polynomials on the period basis: i.e., represent g as the sequence $g_1, g_{0,-1}, \ldots, g_{0,1-d}$. Con-877 sider the problem of multiplying *d*-periodic polynomials: given *d*-periodic *f*, *g*, find *d*-878 periodic h with h = fg in $\mathbb{Z}[x]/((x^p - 1)/(x - 1))$. 879

This problem for \mathbb{Z} reduces immediately to the same problem for \mathbb{Z}/M , if the modulus 880 *M* is chosen large enough to ensure that the coefficients of *h* in \mathbb{Z} can be recovered from 881 their images in \mathbb{Z}/M . An easy way to measure "large enough" is to note that the maximum 882 possible coefficient of h in absolute value is 2p - 3 times the maxima for f and g; the 883 factor 2p - 3 fits into $\Theta(\log p)$ bits, and the same factor across all O(p) coefficients fits into 884 $O(p \log p)$ bits, a bound sufficiently small for this paper's analyses. One can, with more 885 work, compute bounds that are better for most inputs—for example, one can evaluate f886 and g at 1, and more generally use low-precision complex embeddings to estimate sizes, as 887 mentioned in Sect. 3.3—but a logarithmic factor is to be expected, as explained in Sect. 2. 888 The reason to reduce to \mathbb{Z}/M is that one can also choose *M* to ensure that \mathbb{Z}/M has the 889 primitive roots of 1 needed for the folded Rader FFT. Concretely, take M as a product of 890 distinct primes $q \in 1 + pd\mathbb{Z}$. Then \mathbb{Z}/M contains a primitive pth root of 1 for defining 891 the DFT in the first place, and, less importantly, contains a primitive dth root of 1 so 892 that the length-d cyclic convolutions inside a folded Rader FFT can in turn be handled 893 by traditional FFTs when d is smooth. Given the goal of using folded Rader FFTs, this 894 reduction is the obvious adaptation of a widely used reduction suggested by Pollard [78], 894 independently Nicholson [74, page 532], and independently Schönhage-Strassen [85], 896

namely reducing modulo products of primes $q \in 1 + 2^k \mathbb{Z}$ to support traditional size- 2^k FFTs.

There is a logarithmic inefficiency in this reduction when d and M are both large. Specifically, there are $d(\log d)^{1+o(1)}$ multiplications in \mathbb{Z}/M , and each multiplication in \mathbb{Z}/M uses $b(\log b)^{1+o(1)}$ bit operations if M has b bits, so there are two logarithmic factors in the total cost on top of the output size bd. Recall that the usual scenario considered in this paper is $bd \in \Theta(n \log n)$; very often d is between $n^{0.1}$ and $n^{0.9}$, implying $bd(\log b)^{1+o(1)}(\log d)^{1+o(1)} = n(\log n)^{3+o(1)}$. Let's see how to do better, saving a logarithmic factor and reaching cost $bd(\log bd)^{1+o(1)} = n(\log n)^{2+o(1)}$.

What does not seem to save this logarithmic factor is working separately modulo each 906 prime factor q of M, along with choosing each q small enough to have log $\log q \in (\log n)^{o(1)}$, 907 as in Sect. 3.2. This *would* reduce the total cost of the folded Rader FFTs across each \mathbb{Z}/q 908 to $n(\log n)^{2+o(1)}$; but how does one reduce \mathbb{Z}/M to $\prod_{q}(\mathbb{Z}/q)$ in the first place? Standard 909 algorithms for this reduction, and for the corresponding interpolation at the end of the 910 multiplication, have two logarithmic factors, one for multiplications and one for the height 911 of a product tree. See, e.g., [16, Sections 18 and 23]. This issue did not arise in Sect. 3.2: 912 each reduction there started from small input coefficients, and interpolation used only 913 one large coefficient. 914

What does save a logarithmic factor, as in Sects. 3.3 and 4.2, is segmentation. It is important here that the folded Rader FFT is simply carrying out a convolution. Segmentation converts length-*d* convolution over \mathbb{Z}/M , where *M* has *b* bits, into O(bd)-bit multiplication, costing $bd(\log bd)^{1+o(1)}$ bit operations. It suffices here to take *M* as a product of distinct primes $q \in 1 + p\mathbb{Z}$.

920 4.8.6 Application to det evaluation

⁹²¹ If *K* is a degree-*n* subfield of $\mathbb{Q}(\zeta_p)$, and if *K* has a smooth tower (i.e., if *n* factors into small enough primes), then computing det^{*K*}_{\mathbb{Q}} α costs $n(\log n)^{3+o(1)}$ in the same $\Theta(n \log n)$ -bit scenario. Each step through the tower costs $n(\log n)^{2+o(1)}$ (see Sect. 4.8.5), and there are $(\log n)^{1-o(1)}$ steps.

As a concrete example, the prime p = 1009 has $p - 1 = 2 \cdot 2 \cdot 2 \cdot 2 \cdot 3 \cdot 3 \cdot 7$, so one can compute $\det_{\mathbb{Q}}^{K} \alpha$ for $K = \mathbb{Q}(\zeta_{1009})$ by a series of multiplications in subfields of K of degrees 1008, 504, 252, 126, 63, 21, 7, using the Gauss-period representation of each subfield, using cyclic convolutions of lengths 1008, 504, 252, 126, 63, 21, 7 respectively to compute the underlying DFTs.

(In this example, one could also use the prime factors of p - 1 in the opposite order, or any other order. In general, for each K, all smooth towers for K have the same performance at the level of detail of this paper's analysis. This is not saying that the towers have *exactly* the same performance; the analysis absorbs all $(\log n)^{o(1)}$ factors.)

This application of folded Rader FFTs to fast $\det_{\mathbb{Q}}^{K} \alpha$ computation seems to be new. A helpful speedup in this context is to push conjugation and subfield extraction through the DFTs: to compute $\det_{K_{j-1}}^{K_j} \alpha_j$, apply a K_j -folded DFT to α_j , fold the result d_j times, and apply a K_{j-1} -folded inverse DFT.

If one is starting from a very large p and a relatively small subfield of $\mathbb{Q}(\zeta_p)$ then the Yprecomputation stated above could be a bottleneck. Each Y_b is a Gauss period, with $\zeta \in R$ substituted for ζ_p ; presumably it is possible in time $n^{O(1)}$ to identify the defining equation of the subfield in question and solve for a suitable system of Y_b values (a weak form of

reciprocity) without even computing ζ . But such speedups are not necessary for this paper:

recall from Sect. 4.6 that m is assumed to be $n^{O(1)}$, and that per-field precomputation cost

is not included.

945 4.9 Arbitrary conductor: difficulties and desiderata

⁹⁴⁶ What about arbitrary Abelian number fields, i.e., subfields of arbitrary cyclotomic fields ⁹⁴⁷ $\mathbb{Q}(\zeta_m)$, without the constraint of *m* being prime?

The general case is not as easy as the prime case. The group $(\mathbb{Z}/m)^*$ is not cyclic in general, although if one splits *m* into prime-power components then non-cyclic components can appear only for the power of 2. More fundamentally, there are many *m* for which $\{\zeta_m^j : j \in (\mathbb{Z}/m)^*\}$ is not a \mathbb{Q} -basis set for $\mathbb{Q}(\zeta_m)$: for example, $\zeta_4 = i$ and $\zeta_4^3 = -i$ do not form a \mathbb{Q} -basis for $\mathbb{Q}(\zeta_4)$, and $\zeta_8, \zeta_8^3, \zeta_8^5, \zeta_8^7$ do not form a \mathbb{Q} -basis for $\mathbb{Q}(\zeta_8)$. If one starts with a basis 1, $\zeta_8, \zeta_8^2, \zeta_8^3$ for $\mathbb{Q}(\zeta_8)$ and takes traces down to $\mathbb{Q}(\zeta_4)$ then one obtains 2, 0, 2 ζ_4 , 0; the nonzero traces 2, 2 ζ_4 form a \mathbb{Q} -basis for $\mathbb{Q}(\zeta_4)$ but not an integral basis.

Traces from $\mathbb{Q}(\zeta_m)$ to *K* can behave suboptimally even when *K* has conductor *m*. Take, 955 for example, m = 8 and $K = \mathbb{R} \cap \mathbb{Q}(\zeta_8)$. The integral basis 1, ζ_8 , ζ_8^2 , ζ_8^3 of $\mathbb{Q}(\zeta_8)$ has trace 956 2, $\zeta_8 + \zeta_8^{-1}$, $\zeta_8^2 + \zeta_8^{-2}$, $\zeta_8^3 + \zeta_8^{-3}$, which is not exactly an integral basis of K. On the other 957 hand, replacing each trace with the sum of *distinct* conjugates replaces 2 with 1, giving 958 an integral basis. Breuer [29], crediting Hiss and Lenstra, gave an explicit integral basis 959 for every Abelian number field; see [29, Corollary 2] for cases handled by the trace, [29, 960 Lemma 4] for cases handled by sums of distinct conjugates (and not by the trace), and 961 [29, Lemma 3] for the fact that these cover all cases. See Sect. 4.12 for more on this 962 construction. 963

Breuer's stated objective [29, Section 1] was to find an integral basis of each Abelian field that allows efficient arithmetic and efficiently finding "for an arbitrary element of a cyclotomic field the basis representation in the smallest possible field". This description considers only moving from $\mathbb{Q}(\zeta_m)$ to a subfield, but it is natural to consider moving more generally from K to L and from L to K whenever $K \subseteq L$ are subfields of $\mathbb{Q}(\zeta_m)$. In det^K evaluation via towers, it is important to be able to efficiently move from a subfield K_j to a smaller subfield K_{j-1} .

971 4.10 A sub-cyclotomic-field-compatible integral basis for each cyclotomic field

This subsection reviews one component of the construction from [29]: an integral basis for $\mathbb{Q}(\zeta_m)$ introduced by Zumbroich [97] and independently Bosma [26]. As Bosma put it [26, Section 1], this basis allows one to efficiently find "the smallest cyclotomic field in which a given sum of roots of unity lies". Arithmetic using this basis was implemented in, respectively, CAS, which was later superseded by GAP, and Cayley, which was later superseded by Magma. The GAP implementation is available in Sage as UniversalCyclotomicField.

For each prime p and each positive integer e, choose a set $I_{p,e}$ of p^{e-1} consecutive integers: e.g., the set $\{1, 2, ..., p^{e-1}\}$. Define $S_{p^e} \subseteq \mathbb{Z}/p^e$ as the complement of the image of $I_{p,e}$ in \mathbb{Z}/p^e . The specified integral basis set for $\mathbb{Q}(\zeta_{p^e})$ is $\{\zeta_{p^e}^j : j \in S_{p^e}\}$. In other words, starting from $\zeta_{p^e}^{\mathbb{Z}}$, one removes some arc consisting of 1/p of the circle. More generally, for each positive integer m, define $S_m \subseteq \mathbb{Z}/m$ as the set of images in \mathbb{Z}/m of all $j \in \mathbb{Z}$ such that, for each prime divisor p of m, the set $j - (m/p^e)I_{p,e} =$ $\{j - (m/p^e)s : s \in I_{p,e}\}$ is disjoint from $p^e\mathbb{Z}$, where $e = \operatorname{ord}_p m$ (i.e., $m \in p^e\mathbb{Z}$ and $m \notin p^{e+1}\mathbb{Z}$). The specified integral basis set for $\mathbb{Q}(\zeta_m)$ is $\{\zeta_m^j : j \in S_m\}$. This no longer has the arc description.

For example, take m = 12, and choose $I_{p,e} = \{0, 1, ..., p^{e-1} - 1\}$; in particular, $I_{2,2} = \{0, 1\}$ and $I_{3,1} = \{0\}$. The allowed exponents j then avoid $4\mathbb{Z} + 3I_{2,2} = 4\mathbb{Z} + \{0, 3\}$, and avoid $3\mathbb{Z} + 4I_{3,1} = 3\mathbb{Z}$, so $S_{12} = \{1, 2, 5, 10\}$.

There are two steps in showing that this is an integral basis set. First, this set has the right number of elements. To see this, observe that if p is a prime divisor of m and $e = \operatorname{ord}_p m$ then the set $(m/p^e)I_{p,e}$ consists of p^{e-1} distinct integers modulo p^e . The condition $p^e \mathbb{Z} \cap (j - (m/p^e)I_{p,e}) = \{\}$ thus excludes exactly p^{e-1} choices of j modulo p^e . These conditions are independent across p, leaving exactly $\prod_p (p^e - p^{e-1}) = \#(\mathbb{Z}/m)^*$ choices of $j \in S_m$; and the map $j \mapsto \zeta_m^j$ on S_m is injective.

Second, one can express any element of $\mathbb{Z}[\zeta_m]$ as a \mathbb{Z} -linear combination of ζ_m^j for $j \in S_m$. The proof is constructive. Write the input as $\sum_{a \in \mathbb{Z}/m} \alpha_a \zeta_m^a$. For each prime divisor p of m (in, say, increasing order), define $e = \operatorname{ord}_p m$, and eliminate all $a \in \mathbb{Z}/m$ such that $a - (m/p^e)I_{p,e}$ includes a multiple of p^e as follows: use the identity $1 = -\zeta_p - \cdots - \zeta_p^{p-1}$, together with $\zeta_p = \zeta_m^{m/p}$, to rewrite ζ_m^a as $-\sum_b \zeta_m^b$ where b ranges over

¹⁰⁰² $\{a + m/p, ..., a + (p - 1)m/p\}$. Here is why this works:

• The set $b - (m/p^e)I_{p,e}$ cannot include a multiple of p^e for any of the new exponents b. (If it does then the difference $(b - a) - (m/p^e)(I_{p,e} - I_{p,e})$ includes a multiple of p^e , say $(b - a) - (m/p^e)\Delta$ where $\Delta \in I_{p,e} - I_{p,e}$. This is also a multiple of m/p^e —since by construction b - a is a multiple of m/p—and hence a multiple of m, i.e., 0 in \mathbb{Z}/m . Hence $(m/p^e)\Delta$ is a multiple of m/p, so Δ is a multiple of p^{e-1} ; but the only multiple of p^{e-1} in $I_{p,e} - I_{p,e} = \{-p^{e-1} + 1, \dots, p^{e-1} - 1\}$ is 0, so $\Delta = 0$, so b = a, but by construction $b \neq a$, contradiction.)

• This property is preserved by any subsequent rewrites, i.e., replacements of ζ^{b} with ζ^{c} where c - b is a multiple of m/p' for a prime $p' \neq p$. (Indeed, c - b is a multiple of p^{e} , so if $c - (m/p^{e})I_{p,e}$ includes a multiple of p^{e} then $b - (m/p^{e})I_{p,e}$ also includes a multiple of p^{e} .)

For each p, there are m/p exponents $a \in \mathbb{Z}/m$ that require rewrites (if they appear in the input), and each rewrite takes O(p) operations, for a total of $\sum_p O(m)$ operations; this is $O(m(\log m)/\log \log m)$ by the prime-number theorem. Also, each layer of rewriting converts *B*-bit coefficients into at most (B + 1)-bit coefficients, and there are at most $\log_2 m$ layers of rewriting.

The choice $I_{p,e} = \{1, 2, ..., p^{e-1}\}$ has the properties $I_{p,e} \cap p^e \mathbb{Z} = \{\}$ and $pI_{p,e} = p\mathbb{Z} \cap I_{p,e+1}$. One can then see that the basis for any divisor of *m* is included in the basis for *m*, making it trivial to recognize elements of smaller cyclotomic fields. These bases are what Bosma [26] calls "canonical bases for cyclotomic fields".

GAP instead takes $I_{p,e}$ as $\{-(p^{e-1}-1)/2, \dots, (p^{e-1}-1)/2\}$ for odd p, and $\{2^{e-1}, \dots, 2^e - 1\}$ for p = 2. The choice $0 \in I_{p,1}$ for odd p gives, e.g., basis $\zeta_p, \zeta_p^2, \dots, \zeta_p^{p-1}$ for $\mathbb{Q}(\zeta_p)$. This does not include a basis element for \mathbb{Q} : recognizing the subfield \mathbb{Q} requires checking for equal coefficients as in [29, Corollary 3]. The advantage of this basis, as in
Sect. 4.8, is that conjugation is as easy as possible.

One can freely use one choice of $I_{p,e}$ to simplify conjugation and another choice of $I_{p,e}$ to simplify subfield detection, since it is efficient to rewrite any input using any given choice of $I_{p,e}$. Also, given a rewriting function for one choice of $I_{p,e}$, one can conjugate the input by a power of ζ_m to obtain a rewriting function for a rotated basis, i.e., a rotated choice of $I_{p,e}$.

1033 4.11 Cyclotomic fields of smooth conductor

Consider computing det^{*K*}_{\mathbb{Q}} via a smooth tower of cyclotomic fields $\mathbb{Q} = \mathbb{Q}(\zeta_{m_0}) \subset \mathbb{Q}(\zeta_{m_1}) \subset \cdots \subset \mathbb{Q}(\zeta_{m_t}) = K$. This setup requires m_t to be smooth, which is more restrictive than merely requiring $\mathbb{Q}(\zeta_{m_t})$ to have smooth degree; on the other hand, this avoids the prime-conductor requirement from Sect. 4.8.

If each $K_j = \mathbb{Q}(\zeta_{m_j})$ is represented as in Sect. 4.10 then it is easy to convert elements of K_{j-1} from the K_j representation to the K_{j-1} representation. It is also easy to convert the input, a small element of $\mathbb{Z}[\zeta_{m_t}]$, to the K_t representation. One obvious way to compute det $K_{j-1} \alpha_j$, given $\alpha_j \in K_j$, is to multiply $\sigma(\alpha_j)$ across the automorphisms σ of K_j that fix K_{j-1} , as in Sect. 3.7. The remaining question is how long conjugation and multiplication take in this K_j representation.

Expanding the allowed set of exponents from S_m in Sect. 4.10 to \mathbb{Z}/m makes conjugation easy, simply permuting \mathbb{Z}/m , and reduces multiplication to the problem of multiplying modulo $x^m - 1$, using $m(\log m)^{1+o(1)}$ coefficient operations. The rewriting operation from Sect. 4.10, reducing the set of exponents from \mathbb{Z}/m to S_m , uses at most $m(\log m)^{1+o(1)}$ coefficient operations.

In terms of the degree n of $\mathbb{Q}(\zeta_m)$, these costs are $n(\log n)^{1+o(1)}$, since the ratio n/m is $(\log n)^{o(1)}$. (More precisely, $n/m \ge \Theta(1/\log \log n)$. This is a standard calculation that runs as follows. First, n/m is the product of 1 - 1/p over prime divisors p of m. Choose a positive integer y so that the number of primes $p \le y$ is the number of prime divisors of m; then n/m is at least the product of 1 - 1/p over primes $p \le y$, which is $\Theta(1/\log y)$ by Mertens's theorem, while m is at least $\prod_{p \le y} p$, so $\log m$ is at least $\Theta(y)$ by the prime-number theorem. This gives $n/m \ge \Theta(1/\log \log m)$, also implying $\Theta(\log \log n) = \Theta(\log \log m)$.)

As before, this gives total cost $n(\log n)^{3+o(1)}$ in the $\Theta(n \log n)$ -bit scenario. This case does not need Rader's FFT.

1058 4.12 Using more subgroups

Let's now unify the ideas of Sects. 4.8 and 4.11: handling $K = \mathbb{Q}(\zeta_m)$ for any *m* using any tower of subgroups of $(\mathbb{Z}/m)^*$, without requiring the subgroups to correspond to cyclotomic subfields.

1062 4.12.1 Multicyclic convolution

The following paragraphs review a standard unification of conventional FFTs with fast
 Hadamard–Walsh transforms.

Let *R* be a ring. Let *t* be a nonnegative integer. Let d_1, d_2, \ldots, d_t be integers with $d_j \ge 2$. Write $n = d_1 d_2 \cdots d_t$. Let $\zeta \in R$ be a primitive *n*th root of 1. Let e_1, e_2, \ldots, e_t be

nonnegative integers. Assume that $e_1 \in n\mathbb{Z}$ if $t \ge 1$. Let x_0 be a unit in R. The goal here is to multiply quickly in the ring

1068
$$R_t = R[x_1, x_2, \dots, x_t] / \left(x_1^{d_1} - x_0^{e_1}, x_2^{d_2} - x_1^{e_2}, \dots, x_t^{d_t} - x_{t-1}^{e_t} \right),$$

with the conventional representation of ring elements as polynomials of degree below d_j in x_j .

If d_j has a prime factor $p < d_j$ then one can replace the modulus $x_j^{d_j} - x_{j-1}^{e_j}$ with the moduli $y^p - x_{j-1}^{e_j}, x_j^{d_j/p} - y$ where y is a new variable, obtaining a problem of the same form with d_j replaced by $p, d_j/p$. So it suffices to consider the case that $d_1, d_2, ..., d_t$ are primes. Primality does not matter for the algorithm statement, but the cost analysis says that smaller d_j is better.

The multicyclic case, which is the case used in subsequent sections, is the case that $e_j = 0$ for all *j*; in other words, multiplication in the group ring R[G], where *G* is any finite commutative group. But it is also important to consider non-multicyclic cases to enable the speedup from the previous paragraph.

The algorithm applies an FFT, a fast isomorphism from R_t to R^n ; multiplies in R^n , which is simply *n* separate multiplications in *R*; and applies an inverse FFT to recover the product in R_t .

For t = 0, there is nothing to do, so assume $t \ge 1$. The first layer of the FFT algorithm proceeds as follows.

Consider the ring morphism $R[z]/(z^{d_1}-1) \rightarrow R^{d_1}$ where coordinate c of the output, for $0 \leq c < d_1$, maps z to ζ^{cn/d_1} . This is a textbook size- d_1 DFT, straightforwardly computable using $\Theta(d_1^2)$ operations in R. This is also straightforwardly invertible using $\Theta(d_1^2)$ operations in R, since d_1 is invertible in R by definition of primitive roots.

(One can improve these $\Theta(d_1^2)$ operation counts by substituting more complicated DFT algorithms. However, this paper will apply multicyclic convolution to smooth towers, and then d_1 is $(\log n)^{o(1)}$, so d_1^2 is also $(\log n)^{o(1)}$. At that level of detail, the exponent of d_1 in the operation count does not matter.)

By assumption $e_1 \in n\mathbb{Z}$, so in particular $e_1 \in d_1\mathbb{Z}$. The ring morphism $R[x_1] \rightarrow R[z]/(z^{d_1}-1)$ mapping x_1 to $x_0^{e_1/d_1}z$ is invertible since x_0 is a unit, and induces a ring morphism $R[x_1]/(x_1^{d_1}-x_0^{e_1}) \rightarrow R[z]/(z^{d_1}-1)$, straightforwardly computable and invertible using $\Theta(d_1)$ operations in R once the necessary powers of x_0 have been precomputed. Composing this morphism with the DFT gives a ring morphism $R[x_1]/(x_1^{d_1}-x_0^{e_1}) \rightarrow R^{d_1}$ that maps x_1 to $x_0^{e_1/d_1} \zeta^{cn/d_1}$ in the *c*th coordinate.

1099 Applying this layer to the whole ring

11

100
$$R_t = R[x_1, x_2, \dots, x_t] / \left(x_1^{d_1} - x_0^{e_1}, x_2^{d_2} - x_1^{e_2}, x_3^{d_3} - x_2^{e_3}, \dots, x_t^{d_t} - x_{t-1}^{e_t} \right),$$

uses $\Theta(d_1 n)$ operations and gives a product of d_1 rings of the form

$$R[x_2,...,x_t]/\left(x_2^{d_2}-x_0^{e_2e_1/d_1}\zeta^{e_2cn/d_1},x_3^{d_3}-x_2^{e_3},...,x_t^{d_t}-x_{t-1}^{e_t}\right)$$

Each of these rings now has the same structure as R_t , except for t being reduced by 1. The point is that $x_0^{e_2e_1/d_1}\zeta^{e_2cn/d_1}$ can be expressed as a $d_2 \cdots d_t$ th power, since e_2e_1/d_1 and e_2cn/d_1 are multiples of $d_2 \cdots d_t$; also, ζ^{d_1} is a primitive $d_2 \cdots d_t$ th root of 1. The rest of the DFT proceeds recursively, using a total of $\Theta((d_1 + \cdots + d_t)n)$ operations in R. The inverse also uses $\Theta((d_1 + \cdots + d_t)n)$ operations in R. The cost is at least $\Theta(n \log n)$. If each d_j is bounded by, say, s then the cost is $O(sn \log n)$. In particular, multiplication in R[G], where G is any finite commutative group whose cardinality n = #G factors into primes at most s, uses $O(sn \log n)$ operations in R.

Often a primitive *n*th root of 1 is overkill; it is easy to compute the roots that are actually required given the sequence $d_1, d_2, ..., d_t, e_1, e_2, ..., e_t$. In the multicyclic case, the algorithm uses only a primitive *r*th root of 1 for $r = \text{lcm}\{d_1, d_2, ..., d_t\}$. In the balanced multicyclic case, where $d_1 = d_2 = \cdots = d_t$, one has $r = d_1$, usually much smaller than $n = d_1^t$. The Hadamard–Walsh transform is a multicyclic transform with $d_1 = d_2 = \cdots = d_t = 2$, and needs only a primitive 2nd root of 1: i.e., the root -1, with 2 invertible in *R*.

1118 **4.12.2** Multicyclic convolution with large coefficients

Consider now the problem of multicyclic convolution over \mathbb{Z} , i.e., the problem of multiplying in $\mathbb{Z}[x_1, \ldots, x_t]/(x_1^{d_1} - 1, \ldots, x_t^{d_t} - 1)$. Again write $n = d_1 \cdots d_t$.

One can reduce multicyclic convolution over \mathbb{Z}/M to this problem of multicyclic convolution over \mathbb{Z} , at the expense of reducing each output coefficient separately modulo M, which costs $O(nb \log b)$ if M has b bits. Conversely, one can reduce multicyclic convolution over \mathbb{Z} to multicyclic convolution over a quotient \mathbb{Z}/M selected (1) to be sufficiently large to recover the output coefficients in \mathbb{Z} and (2) to have appropriate primitive roots of 1 for the standard multicyclic FFTs from Sect. 4.12.1.

However, even in the smooth case, those multicyclic FFTs involve $n(\log n)^{1+o(1)}$ opera-1127 tions in \mathbb{Z}/M . Many of those operations are multiplications (except in extreme cases such 1128 as Hadamard–Walsh transforms), each taking $b(\log b)^{1+o(1)}$ operations if M has b bits. 1129 As in Sect. 4.8.5, there is an inefficiency here when both b and n are large. A straight-1130 forward use of segmentation replaces the two logarithmic factors with one logarithmic 1131 factor times something exponential in t; this was satisfactory in Sect. 4.8.5, with t = 1, 1132 but is not satisfactory in general. One can try to reduce t by replacing d_1, \ldots, d_t with the 1133 elementary divisors of the group $\mathbb{Z}/d_1 \times \cdots \times \mathbb{Z}/d_t$, but this is not helpful for, e.g., the 1134 case $d_1 = \cdots = d_t = 3$. 1135

To do better, one can inspect standard algorithms for fast integer multiplication (see, e.g., [13]) and observe that many, if not all, of the same ideas naturally support multicyclic convolution. State-of-the-art multicyclic convolution as in [56] is generally more complicated than necessary for this paper, since this paper, unlike [56], freely allows $(\log n)^{o(1)}$ factors; the following paragraphs explain a simpler algorithm. No claims of novelty are made here.

One of the Schönhage–Strassen [85] algorithms to multiply in \mathbb{Z} is as follows. There are 1142 three parameters: a positive integer κ , a positive integer $c \in \Theta(\log \kappa)$, and a prime number 1143 $p \in 1 + \kappa \mathbb{Z}$ having $\Theta(\log \kappa)$ bits. Map \mathbb{Z} to $\mathbb{Z}[\gamma]/(2^c - \gamma)$ and lift to $\mathbb{Z}[\gamma]$, splitting each input 1144 into coefficients between -2^{c-1} and 2^{c-1} ; map to $\mathbb{Z}[y]/(y^{\kappa}-1)$; map to $(\mathbb{Z}/p)[y]/(y^{\kappa}-1)$; 1145 use a length- κ FFT to multiply in $(\mathbb{Z}/p)[\gamma]/(\gamma^{\kappa}-1)$. Each product coefficient has absolute 1146 value at most $2^{2c-2}\kappa$, so if $p > 2^{2c-1}\kappa$ (which is compatible with p having $\Theta(\log \kappa)$ bits) 1147 then one easily recovers the product in $\mathbb{Z}[y]/(y^{\kappa}-1)$, and if the output polynomial needed 1148 *y*-degree at most $\kappa - 1$ then one easily recovers the original product in \mathbb{Z} . 1149

Typically κ is chosen as a power of 2, so that a traditional power-of-2 FFT uses $\Theta(\kappa \log \kappa)$ operations in \mathbb{Z}/p . Each operation in \mathbb{Z}/p uses $(\log \kappa)^{1+o(1)}$ bit operations, since p has 11

 $\Theta(\log \kappa)$ bits. The overall cost is thus $\kappa(\log \kappa)^{2+o(1)}$ for outputs fitting into $\Theta(\kappa \log \kappa)$ bits; i.e., $b(\log b)^{1+o(1)}$ for outputs fitting into b bits.

To handle multicyclic convolution in the same way, take a positive integer κ , a positive integer $c \in \Theta(\log \kappa n)$, and a prime number $p \in 1 + \kappa n\mathbb{Z}$ having $\Theta(\log \kappa n)$ bits, with $p > 2^{2c-1}\kappa n$. Map the ring $\mathbb{Z}[x_1, \ldots, x_t]/(x_1^{d_1} - 1, \ldots, x_t^{d_t} - 1)$ to the ring $(\mathbb{Z}[y]/(2^c - y))[x_1, \ldots, x_t]/(x_1^{d_1} - 1, \ldots, x_t^{d_t} - 1)$ and lift to the ring $\mathbb{Z}[y, x_1, \ldots, x_t]/(x_1^{d_1} - 1, \ldots, x_t^{d_t} - 1)$, splitting each of the input coefficients into polynomials in y with coefficients between -2^{c-1} and 2^{c-1} . Then map to the ring $\mathbb{Z}[y, x_1, \ldots, x_t]/(y^{\kappa} - 1, x_1^{d_1} - 1, \ldots, x_t^{d_t} - 1)$ and further to the ring

$$(\mathbb{Z}/p)[y, x_1, \ldots, x_t]/(y^{\kappa} - 1, x_1^{d_1} - 1, \ldots, x_t^{d_t} - 1).$$

Each product coefficient has absolute value at most $2^{2c-2}\kappa n$, so one recovers the product in $\mathbb{Z}[y, x_1, \ldots, x_t]/(y^{\kappa} - 1, x_1^{d_1} - 1, \ldots, x_t^{d_t} - 1)$, and thus the product in $\mathbb{Z}[y, x_1, \ldots, x_t]/(x_1^{d_1} - 1, \ldots, x_t^{d_t} - 1)$ assuming it has *y*-degree at most $\kappa - 1$, and thus the product in $\mathbb{Z}[x_1, \ldots, x_t]/(x_1^{d_1} - 1, \ldots, x_t^{d_t} - 1)$.

The product in $(\mathbb{Z}/p)[y, x_1, ..., x_t]/(y^{\kappa} - 1, x_1^{d_1} - 1, ..., x_t^{d_t} - 1)$ can be handled as explained in Sect. 4.12.1 if $p \in 1 + \kappa n \mathbb{Z}$. This uses $O(s\kappa n \log \kappa n)$ operations in \mathbb{Z}/p if κn factors into primes at most *s*. Each operation in \mathbb{Z}/p uses $(\log \kappa n)^{1+o(1)}$ bit operations, so the overall cost is at most $s\kappa n(\log \kappa n)^{2+o(1)}$ bit operations for *n* output coefficients each fitting into $\Theta(\kappa \log \kappa n)$ bits. For this paper, one should think of *s* as being small: the smooth case emphasized in this paper is that *n* factors into small primes, and one can easily choose κ within any desired range to also factor into small primes.

The applications below focus on reducing various types of FFTs over rings R to mul-1173 ticyclic convolutions, generalizing Sect. 4.8. The application to det computation takes 1174 rings R of the form \mathbb{Z}/M , where M is generally chosen larger and larger as one moves 1175 down a tower; one always has $b > \Theta(\log n)$, where b is the number of bits in M. One 1176 can handle the multicyclic convolutions by the algorithm in the previous two paragraphs, 1177 taking κ as $b/\Theta(\log bn)$, so κn is $bn/\Theta(\log bn) = bn/\Theta(\log \kappa n)$. The overall cost is then 1178 $sbn(\log bn)^{1+o(1)}$ bit operations for *n* output coefficients each fitting into $\Theta(b)$ bits. This 1179 saves a log factor as desired. 1180

1181 4.12.3 A primitive size-m FFT

Let *m* be a positive integer. Let *R* be a ring. Let ζ be a primitive *m*th root of 1 in *R*. Write $G = (\mathbb{Z}/m)^*$. Consider the ring morphism $R[x] \to R^G$ that maps *g* to the vector $b \mapsto g(\zeta^b)$. The main objective here is to efficiently apply this morphism to $g = g_0 + g_1 x + \dots + g_{m-1} x^{m-1}$, given $g_0, g_1, \dots, g_{m-1} \in R$.

This morphism is a "primitive DFT". This is, for m > 1, different from a traditional "full DFT". The difference is that a full DFT evaluates g at ζ^b for all $b \in \mathbb{Z}/m$, while a primitive DFT evaluates g at ζ^b only for $b \in (\mathbb{Z}/m)^*$, i.e., only for b coprime to m.

Note that multiplication in \mathbb{Z}/m restricts to an action of the group G on \mathbb{Z}/m , with one orbit $(m/d)(\mathbb{Z}/d)^*$ for each positive divisor d of m: for example, this orbit is $(\mathbb{Z}/m)^*$ for d = m, and {0} for d = 1. The notation $(m/d)(\mathbb{Z}/d)^*$ here uses multiplication by m/d as notation for the map from \mathbb{Z}/d to \mathbb{Z}/m induced by multiplication by m/d as a map from \mathbb{Z} to \mathbb{Z} .

For each positive divisor *d* of *m*, define I_d as the element $\sum_{a \in (\mathbb{Z}/d)^*} g_{(m/d)a}a^{-1}$ of the group ring $R[(\mathbb{Z}/d)^*]$, and Z_d as the element $\sum_{b \in (\mathbb{Z}/d)^*} \zeta^{(m/d)b}b$ of the group ring; i.e., these are the vectors $a \mapsto g_{(m/d)a^{-1}}$ and $b \mapsto \zeta^{(m/d)b}$. Write O_d for the product $I_d Z_d$. Then

$$O_{d} = \sum_{a \in (\mathbb{Z}/d)^{*}} \sum_{b \in (\mathbb{Z}/d)^{*}} g_{(m/d)a} a^{-1} \zeta^{(m/d)b} b = \sum_{a \in (\mathbb{Z}/d)^{*}} \sum_{b \in (\mathbb{Z}/d)^{*}} g_{(m/d)a} a^{-1} \zeta^{(m/d)ba} b a$$
$$= \sum_{b \in (\mathbb{Z}/d)^{*}} \sum_{a \in (\mathbb{Z}/d)^{*}} g_{(m/d)a} \zeta^{(m/d)ba} b = \sum_{b \in (\mathbb{Z}/d)^{*}} \sum_{j \in (m/d)(\mathbb{Z}/d)^{*}} g_{j} \zeta^{bj} b;$$

1198

i.e., the entry $(O_d)_b$ in the vector O_d is $\sum_{j \in (m/d)(\mathbb{Z}/d)^*} g_j \zeta^{bj}$.

Now $g(\zeta^b) = \sum_j g_j \zeta^{bj} = \sum_d \sum_{j \in (m/d)(\mathbb{Z}/d)^*} g_j \zeta^{bj} = \sum_d (O_d)_b$ where *d* runs through positive divisors of *m*. The desired vector $b \mapsto g(\zeta^b)$ is thus the sum of O_d across *d*. Care is required in the details of this vector addition, for two reasons:

- O_d is represented as a compressed vector, indexed by elements of $(\mathbb{Z}/d)^*$. One thus needs to synchronize indices for additions.
- There can be many divisors d of m, and adding each O_d directly into O_m would incur $\#(\mathbb{Z}/m)^*$ additions for each d. To more efficiently handle all O_d , work upwards through d < m, adding each O_d into O_{dp} for the smallest prime p dividing m/d and then forgetting about O_d . Each d then incurs $\#(\mathbb{Z}/dp)^* \le dp$ additions, and the sum of dp across divisors d of m is bounded by $m(\log m)^{1+o(1)}$.

These additions require computing O_d in the first place for each positive integer d of m, i.e., multiplying I_d by Z_d in $R[(\mathbb{Z}/d)^*]$; this is a multicyclic convolution as in Sects. 4.12.1 and 4.12.2. For example, if m is prime then these are multiplications in $R[(\mathbb{Z}/m)^*]$ and $R[(\mathbb{Z}/1)^*]$, which are cyclic convolutions of lengths m - 1 and 1 respectively; this special case matches Rader's original FFT.

If $n = \#(\mathbb{Z}/m)^*$ is smooth, meaning all prime factors in $(\log n)^{o(1)}$, then $\#(\mathbb{Z}/d)^*$, a divisor of n, also has all prime factors in $(\log n)^{o(1)}$. The convolution algorithm of Sect. 4.12.1 thus uses $O((\log n)^{o(1)} \#(\mathbb{Z}/d)^* \log \#(\mathbb{Z}/d)^*)$ operations in R, hence $O(((\log n)^{1+o(1)} \#(\mathbb{Z}/d)^*))$ operations in R. The total cost, the sum of costs over d, is $O(m(\log n)^{1+o(1)})$ since $\sum_d \#(\mathbb{Z}/d)^* = m$, hence $O(n(\log n)^{1+o(1)})$ as in Sect. 4.11, hence $n(\log n)^{1+o(1)}$ since the cost for d = m is at least $\Theta(n \log n)$.

When *m* is a power of an odd prime, this generalization of Rader's FFT matches the 1221 primitive part of a DFT algorithm by Winograd [96, Section 4]. For $m = 2^e$, Winograd 1222 uses a conventional size 2^e additive FFT to directly solve the original DFT problem, rather 1223 than using convolutions to compute primitive DFTs for the multiplicative group $(\mathbb{Z}/m)^*$. 1224 For general *m*, Winograd first decomposes a size-*m* DFT into prime-power DFTs, and 1225 then reduces each odd-prime-power DFT to its primitive part. The advantage of working 1226 directly with the primitive part of a size-*m* DFT is that it allows more choices of subgroups, 1227 not requiring the subgroups to align with the prime-power decomposition of *m*. 1228

1229 4.12.4 Inversion

Looking only at the primitive part of a DFT—evaluating $g(\zeta^b)$ only for $b \in (\mathbb{Z}/m)^*$, rather than for all $b \in \mathbb{Z}/m$ —raises the question of how to recover g from these values. One cannot hope to recover m coefficients of g from only $\#(\mathbb{Z}/m)^*$ values for m > 1, but if one restricts the allowed g indices as in Sect. 4.10 then there is no obvious obstacle to recovering g.

Recall the principle that a full DFT with exponents negated is an inverse full DFT. This implies the same principle for a primitive DFT. Given $\nu \in R^G$, define $h_j = \sum_{a \in G} \nu_a \zeta^{-aj}$ for each $j \in \mathbb{Z}/m$; then $\sum_{j \in \mathbb{Z}/m} h_j \zeta^{bj} = \sum_{a \in G} v_a \sum_{j \in \mathbb{Z}/m} \zeta^{(b-a)j} = mv_b$ for $b \in G$. Hence the polynomial $(\sum_j h_j x^j)/m$ has values v_b as desired; note that m is invertible in R since there is a primitive mth root of 1. Among the preimages of v under the map $R[x]/(x^m - 1) \to R^G$, this polynomial is characterized by having value 0 at ζ^b if $b \notin G$.

The inverse primitive DFT has a different shape from the forward primitive DFT: it computes *m* values from $\#(\mathbb{Z}/m)^*$ values, rather than the other way around. The inverse can again be reduced to a convolution in the group ring $R[(\mathbb{Z}/d)^*]$ for each positive integer *d* dividing *m*: the element $H_d = \sum_{b \in (\mathbb{Z}/d)^*} h_{(m/d)b}b$ of the group ring is the product of $Z_d = \sum_{b \in (\mathbb{Z}/d)^*} \zeta^{(m/d)b}b$ and $V_d = \sum_{a \in G} v_{-1/a}(a \mod d)$, where *a* mod *d* means the image of *a* in $(\mathbb{Z}/d)^*$.

To efficiently compute V_d for all d, work downwards through d (reversing how O_d was handled in the forward transform), obtaining V_d for each d < m via V_{dp} for the smallest prime p dividing m/d. The sum of dp is again bounded by $m(\log m)^{1+o(1)}$. This procedure computes all h_j at similar speed to the forward transform.

The following paragraphs explain how to tweak the above procedure to produce an output polynomial where all exponents *j* are guaranteed to have $gcd\{m, j\}$ dividing *m*/rad *m*; here rad *m* means the radical of *m*, the product of prime divisors *p* of *m*. One can then skip any *d* not divisible by rad *m*, rather than subsequently eliminating exponents for those *d* as in Sect. 4.10. For example, when *m* is prime, the tweaked inversion procedure uses a convolution only for $(\mathbb{Z}/m)^*$ (skipping a convolution for $(\mathbb{Z}/1)^*$), and produces exponents $1, \ldots, m - 1$ (skipping 0), exactly as in Sect. 4.8.4.

Consider any positive integer r dividing rad m. Abbreviate $gcd\{m, r^{\infty}\}$ as m_r , and choose $s_r \in m_r \mathbb{Z} \cap (1 + (m/m_r)\mathbb{Z})$. These quantities s_r have three critical properties:

- If r = 1 then $s_r \in 1 + m\mathbb{Z}$.
- If r > 1 then $gcd\{m, s_r\} > 1$.

• If a prime divisor p of m does not divide r then the difference $s_r - s_{rp}$ is in $(m/p^{\operatorname{ord}_p m})\mathbb{Z}$. (Both s_r and s_{rp} are in $m_r\mathbb{Z}$ and in $1 + (m/m_{rp})\mathbb{Z}$, so $s_r - s_{rp}$ is in $m_r\mathbb{Z}$ and in $(m/m_{rp})\mathbb{Z}$; m_r is coprime to m/m_{rp} , so $s_r - s_{rp}$ is in $(mm_r/m_{rp})\mathbb{Z}$; and $m_{rp}/m_r = p^{\operatorname{ord}_p m}$.)

Define $h'_{j,r} = \sum_{a \in G} v_a \zeta^{-ajs_r}$. Then $h'_{j,1} = \sum_{a \in G} v_a \zeta^{-aj} = h_j$ since $s_1 \in 1 + m\mathbb{Z}$, so $\sum_{j \in \mathbb{Z}/m} h'_{j,1} \zeta^{bj} = mv_b$ for $b \in G$. If r > 1 then $\sum_{j \in \mathbb{Z}/m} h'_{j,r} \zeta^{bj} = 0$ for $b \in G$ since $\operatorname{gcd}\{m, s_r\} > 1$.

Define $h'_{j} = \sum_{r} \mu(r) h'_{j,r}$ where μ is the Möbius function. (For example, $h'_{j} = h'_{j,1} - h'_{j,p}$ 1268 $h'_{i,q} + h'_{i,pq}$ if rad *m* is the product of primes *p*, *q*.) Then $\sum_{j \in \mathbb{Z}/m} h'_j \zeta^{bj} = mv_b$ for $b \in \hat{G}$. 1269 The polynomial $(\sum_i h'_i x')/m$, like the previous polynomial $(\sum_i h_i x')/m$, thus has the 1270 desired values v_b . This polynomial also has the extra feature described above: h'_i can be 1271 nonzero only if $gcd\{m, j\}$ divides m/rad m. (Indeed, consider any j for which $gcd\{m, j\}$ 1272 does not divide m/rad m, i.e., for which $\operatorname{ord}_p j \ge \operatorname{ord}_p m$ for some prime p dividing m. If p 1273 does not divide *r* then $s_r - s_{rp} \in (m/p^{\operatorname{ord}_p m})\mathbb{Z}$, so $js_r - js_{rp} \in m\mathbb{Z}$, so $\zeta^{-ajs_r} = \zeta^{-ajs_{rp}}$ for 1274 each $a \in G$, so $h'_{i,r} = h'_{i,rp}$; hence $h'_i = 0$.) 1275

Fast computation of this polynomial works the same way as fast computation of the previous polynomial: the desired $H'_d = \sum_{b \in (\mathbb{Z}/d)^*} h'_{(m/d)b} b$ is the product of Z'_d and V_d in the group ring $R[(\mathbb{Z}/d)^*]$, after a precomputation of $Z'_d = \sum_r \mu(r) \sum_{b \in (\mathbb{Z}/d)^*} \zeta^{s_r(m/d)b} b$. If *d* is not divisible by rad *m* then $Z'_d = 0$; again, the point of this tweak is to skip such values of *d*. 1281 4.12.5 Folding: the symmetric case

Now let *H* be a subgroup of *G*, and write *K* for the subfield of $\mathbb{O}(\zeta_m)$ fixed by $\{\sigma_c : c \in H\}$. 1282 The objective here is to save a factor essentially #H for arithmetic on elements of this 1283 subfield. The special case of prime *m* was handled by the folded Rader FFT from Sect. 4.8. 1284 As a starting point, if K has conductor smaller than m, then one can replace m with the 1284 conductor, and replace H with the corresponding subgroup for the new conductor. For 1286 each change of conductor, one also needs to correspondingly change the input represen-1287 tation; Sect. 4.10 explained how to do this for full cyclotomic fields, and this conversion 1288 is compatible with the symmetries described below. So assume from now on that K has 1280 conductor m. 1290

Define $H_0 = H \cap (1 + (\operatorname{rad} m)(\mathbb{Z}/m))$. Section 4.12.7 explains how to handle the possibility that $\#H_0 > 1$. Assume for now that $\#H_0 = 1$. For example, if *m* is squarefree, then rad m = m, so $\#H_0 = 1$.

¹²⁹⁴ By [29, Lemma 3(1) and Corollary 2], $\mathbb{Q}(\zeta_m)$ has an integral basis set $B \subseteq \zeta_m^{\mathbb{Z}}$ such that ¹²⁹⁵ the usual action of H on $\zeta_m^{\mathbb{Z}}$ restricts to a free action of H on B. This implies, as noted in ¹²⁹⁶ [29, page 281, top paragraph], that {tr_K^{$\mathbb{Q}(\zeta_m)$} $\beta : \beta \in B$ } is an integral basis set for K.

These traces $\operatorname{tr}_{K}^{\mathbb{Q}(\zeta_{m})} \beta$ for $\beta \in B$ generalize the Gauss periods from Sect. 4.7. It seems reasonable to refer to these generalized basis elements as Gauss periods: the periodicity is immediately visible in the coefficients of each trace, thanks to H acting freely on B.

Now represent elements of K as linear combinations of these Gauss periods, generalizing the case of prime m. So far this matches what is proposed in [29, Section 5]. What is not addressed in [29] is how to multiply quickly.

Each input to multiplication is a polynomial $g = g_0 + g_1 x + \dots + g_{m-1} x^{m-1}$ in $\mathbb{Z}[x]/(x^m - 1)$ representing $g(\zeta_m)$. This polynomial is not represented on the length *m* basis 1, x, \dots, x^{m-1} , but rather on a basis of length just $\#(\mathbb{Z}/m)^*/\#H$ representing the Gauss periods: the basis set is $\{\sum_{c \in H} x^{bc} : b \in \mathbb{Z}/m, \zeta_m^b \in B\}$. In other words, $g_j = g_{jc}$ for all $j \in \mathbb{Z}/m$ and all $c \in H$, and $g_j = 0$ when ζ_m^j is outside B.

Move as usual from \mathbb{Z} to a quotient ring R containing the primitive roots of 1 needed for all FFTs that appear. Each input to multiplication is then a polynomial in $R[x]/(x^m - 1)$, again represented on the basis set $\{\sum_{c \in H} x^{bc} : b \in \mathbb{Z}/m, \zeta_m^b \in B\}$.

The critical point here is that the *H* symmetry in the *x* exponents passes directly through every step in the generalization of Rader's algorithm from Sect. 4.12.3, producing a generalized folded Rader FFT. For each positive divisor *d* of *m*, the input group-ring element $I_{d} = \sum_{a \in (\mathbb{Z}/d)^*} g_{(m/d)a}a^{-1}$ has entries invariant under the action of *H* on $(m/d)(\mathbb{Z}/d)^*$, so one replaces $(\mathbb{Z}/d)^*$ with the corresponding quotient group, precomputing a folded version of Z_d in that group. Similar comments apply to the inverse transform from Sect. 4.12.4.

1317 4.12.6 Folding: cost analysis of the symmetric case

Note that sometimes $B \cap \zeta_m^{(m/d)(\mathbb{Z}/d)^*} = \{\}$ so *d* can simply be skipped in the forward transform. For example, for a prime power $m = p^e$, the basis *B* in Sect. 4.10 skips some arc of 1/p of the circle: if the arc is chosen to contain 1 then d = 1 can be skipped.

For the case m = p, this leaves just d = p, which is why the folded Rader FFT in Sect. 4.8 works exclusively with $(\mathbb{Z}/p)^*$. On the other hand, skipping the arc is somewhat deceptive when one tries to generalize; if e > 1 then one encounters powers of ζ_{p^e} having different orders. From this perspective, for the case m = p there is expository value in considering the entire circle—as in Rader's original algorithm, which allows $g_0 \neq 0$, doing extra work for a Rader of the lost arc.

For general *m*, the construction of *B* in [29, Section 3] avoids any exponent divisible by $p^{\text{ord}_p m}$ for any prime divisor *p* of *m*; the construction reviewed in Sect. 4.10 also works this way if one chooses the sets $I_{p,e}$ in that section to contain 0. This ensures that the divisors *d* of *m* that appear are all divisible by rad *m*. The tweaked inverse transform in Sect. 4.12.4 makes the same guarantee.

If the starting conductor-*m* field *K* is a proper subfield of $\mathbb{Q}(\zeta_m)$ then requiring the degree *n* of *K* to be smooth does not necessarily mean that the degree $#(\mathbb{Z}/m)^*$ of $\mathbb{Q}(\zeta_m)$ is smooth. Fortunately, what appears in convolution is not the group $(\mathbb{Z}/m)^*$, but the quotient group $(\mathbb{Z}/m)^*/H$, which has cardinality *n*, and, more generally, quotients $((m/d)(\mathbb{Z}/d)^*)/H$ having cardinality dividing *n*, so each cardinality is smooth, giving fast convolution by the algorithm of Sect. 4.12.1.

One also has to check that the total size of the groups $((m/d)(\mathbb{Z}/d)^*)/H$ that appear is $n(\log n)^{o(1)}$. The point here is that the action of H on $(m/d)(\mathbb{Z}/d)^*$ is free for $d = \operatorname{rad} m$ (since by assumption $\#H_0 = 1$ where $H_0 = H \cap (1 + (\operatorname{rad} m)(\mathbb{Z}/m)))$, and thus for each positive divisor d of m divisible by rad m. Each element of the quotient $((m/d)(\mathbb{Z}/d)^*)/H$ thus corresponds to #H elements of $(m/d)(\mathbb{Z}/d)^*$, and the sets $(m/d)(\mathbb{Z}/d)^*$ are disjoint subsets of \mathbb{Z}/m as d varies, so the total size of the groups is at most $\lfloor m/\#H \rfloor$. One has $m/\#(\mathbb{Z}/m)^* \in (\log n)^{o(1)}$ as in Sect. 4.11, and $\#(\mathbb{Z}/m)^*/\#H = n$.

Moving down through a tower of subfields with conductor *m* corresponds to moving up through a tower of subgroups *H* of $(\mathbb{Z}/m)^*$. If *B* is chosen so that the largest subgroup *H* in the tower acts freely on *B* then the smaller subgroups in the tower will also act freely on *B*. Moving from the basis for a smaller field to the basis for a larger field is then simply repeating coefficients, and moving the other way (as in det^{*K*}_Q evaluation) is removing redundant coefficients.

1351 4.12.7 Folding: the almost-symmetric case

What happens if instead $\#H_0 > 1$? One then has $\#H_0 = 2$ by [29, Lemma 3(2)], with Hfactoring as a direct product of H_0 and another group H_1 , and [29, Lemma 4] constructs a basis of K that is *almost* as symmetric as the Gauss periods. Part of the basis consists of traces of the form $\operatorname{tr}_{K}^{\mathbb{Q}(\zeta_m)} \zeta^b$; the other part consists of traces of the form $\operatorname{tr}_{K_1}^{\mathbb{Q}(\zeta_m)} \zeta^b =$ (1/2) $\operatorname{tr}_{K}^{\mathbb{Q}(\zeta_m)} \zeta^b$, where K_1 is the fixed field of H_1 and b is chosen to have $\operatorname{tr}_{K_1}^{\mathbb{Q}(\zeta_m)} \zeta^b \in K$. All of this relies on K having conductor m.

The easy approach to fast multiplication here is to represent the elements of K as elements of the superfield K_1 , using only the H_1 symmetry. This reduces to the symmetric case handled above. This loses a factor 2 compared to the desired H symmetry, but this loss is not visible at the level of detail of this section's cost analyses; $\Theta(\#H_1)$ is the same as $\Theta(\#H)$.

This approach should not be confused with representing the elements of K as elements of $\mathbb{Q}(\zeta_m)$, losing the variable factor contemplated in Sect. 4.4. These representations coincide only when $K_1 = \mathbb{Q}(\zeta_m)$, i.e., when K has half the degree of $\mathbb{Q}(\zeta_m)$, i.e., when $\#H_1 = 1$.

It is instructive to look at the case $#H_1 = 1$ more closely. The archetypal examples are the following two half-degree subfields of a power-of-2 cyclotomic field $\mathbb{Q}(\zeta_m)$ where $m \ge 8$:

• The subfield fixed by σ_{-1} , i.e., the real-cyclotomic field $\mathbb{R} \cap \mathbb{Q}(\zeta_m)$. 1369

• The subfield fixed by $\sigma_{m/2-1}$. 1370

Both of these have conductor *m*, unlike the half-degree cyclotomic subfield $\mathbb{Q}(\zeta_{m/2})$, which 1371 is the subfield fixed by $\sigma_{m/2+1}$. 1372

In the power-of-2 real-cyclotomic case, the almost-symmetric basis in [29, Lemma 4] 1373

consists of the *H*-traces $\zeta_m + \zeta_m^{-1}, \ldots, \zeta_m^{m/4-1} + \zeta_m^{-m/4+1}$ and one H_1 -trace $\zeta_m^{m/2} = -1$. 1374 Because the H_1 part of the basis is so short, one can productively use H-folded DFTs to

1375

multiply the *H*-symmetric part of the first input by the *H*-symmetric part of the second 1376

input, and then use schoolbook multiplication to handle the missing products. 1377

4.13 Open guestions 1378

Multiquadratic fields $K = \mathbb{Q}(\sqrt{d_1}, \sqrt{d_2}, \dots, \sqrt{d_t})$ are Abelian. One can apply the machin-1379 ery from Sect. 4.12 to this case, starting by writing $\sqrt{d_1}, \ldots, \sqrt{d_t}$ in terms of Gauss periods. 1380 However, the multiplication algorithm in [9, Section 3.3] is considerably simpler, mapping 1381 \mathbb{Z} to quotient rings \mathbb{F}_p for which $\mathbb{F}_p[x_1, \ldots, x_t]/(x_1^2 - d_1, \ldots, x_t^2 - d_t)$ splits into 2^t copies 1382 of \mathbb{F}_p . Can one efficiently handle arbitrary towers by the same technique? 1383

(Note that [59] studied tower performance but did not specifically consider the base 1384 ring \mathbb{Z} , and in particular did not consider switching from \mathbb{Z} to \mathbb{F}_p for a suitably selected *p*. 1385 It would also be interesting to investigate a switch to \mathbb{R} or \mathbb{C} , but using \mathbb{F}_p has the virtue 1386 of avoiding precision questions.) 1387

One might think that the answer is "Yes, of course". Start with $R_0 = \mathbb{Z}$ and $K_0 = \mathbb{Q}$. 1388 For $j \in \{1, 2, ..., t\}$, select a monic irreducible polynomial $\varphi_i \in R_{i-1}[x_i]$, define $R_i =$ 1389 $R_{i-1}[x_i]/\varphi_i$, and define $K_i = K_{i-1}[x_i]/\varphi_i$. There are infinitely many primes *p* for which 1390

• φ_1 splits into linear factors in $\mathbb{F}_p[x_1]$; 1391

• φ_2 , with each of the roots of φ_1 in \mathbb{F}_p substituted for x_1 , splits into linear factors in 1392 $\mathbb{F}_{p}[x_{2}];$ 1393

• etc.; 1394

and then $\mathbb{F}_p[x_1, \ldots, x_t]/(\varphi_1, \ldots, \varphi_t)$ is isomorphic to \mathbb{F}_p^n , where $n = \prod_j \deg \varphi_j$. To com-1395 pute det^K₀ α for a small element α of the ring of integers \mathcal{O} of $K = K_t$, multiply α by a 1396 denominator *D* known to have $D\mathcal{O} \subseteq R_t$, and then divide D^n out of det^{*K*}_{\mathbb{O}} $D\alpha$. 1397

One obvious question is how large the denominator D is. For multiquadratics with 1398 coprime squarefree d_1, \ldots, d_t , denominator $n = 2^t$ suffices; what about other fields? But 1399 this question doesn't seem to matter much for performance, even though D appears to 1400 an *n*th power in det^K_{\mathbb{O}} $D\alpha$: in deciding how large $\prod_{p} p$ needs to be, one can disregard 1401 the known divisor D^n of det^K_{\mathbb{O}} $D\alpha$ and consider only the size of det^K_{\mathbb{O}} α . This is the non-1402 Archimedean version of a suggestion from Belabas [10, Section 5.2] mentioned in Sect. 3.3, 1403 namely using known divisors to limit the precision of complex embeddings. 1404

A more worrisome question is how large p is. The standard proof that there are infinitely 1405 many suitable primes p runs via Chebotarev's density theorem. This theorem includes the 1406 statement that the primes splitting completely in K have density 1/#G, where G is the 1407 Galois group of K. (An older density theorem due to Frobenius suffices here; see generally 1408 [89].) Since the density is nonzero, there are infinitely many such primes. One might have 1409 to skip primes that divide D, but there are only finitely many such primes. The density 1410 provides enough information to formulate reasonable conjectures regarding the size of p. 1411

```
K.<a> = NumberField(x<sup>2</sup>-3)
Kpoly.<y> = K[]
L.<b> = K.extension(y<sup>2</sup>-(a+4))
Lpoly.<z> = L[]
M.<c> = L.extension(z<sup>2</sup>-(a*b+5*b+9*a+2))
Mpoly.<t> = M[]
N.<d> = M.extension(t<sup>2</sup>-(6*a*b*c+5*b*c+3*a*c+5*c+8*a*b+9*b+7*a+9))
for F in K,L,M,N:
num = 0
for p in primes(2,oo):
if len(F.primes_above(p)) == F.absolute_degree():
print(F.absolute_degree(),p)
num += 1
if num >= 3: break
Fig. 3 Sage script to build a random-looking tower of fields of degrees 2.4.8.16 and find the first 3
```

completely split primes in each field

The reason this question is worrisome is that Galois groups are generally huge. For example, if *n* is prime then one expects degree-*n* fields to have #G = n!, and then one expects *p* to have $\Theta(n \log n)$ bits—which (1) raises the question of how to find such a *p* and (2) forces cost $n^{2+o(1)}$ for *n*-coefficient multiplications. Requiring a smooth tower should bias #G downwards, but how much? One might guess that usually $\#G = 2^{n-1}$ for a degree-*n* field having a power-of-2 tower.

As a data point, Figure 3 builds a random-looking tower of fields of degrees 2, 4, 8, 16 and prints out the first 3 primes that split completely in each field. These primes are 11, 13, 23 for degree 2; 61, 157, 181 for degree 4; 181, 647, 1907 for degree 8; and 1,331,339, 1,384,861, 1,570,633 for degree 16. For comparison, 97, 193, 257 split completely in $\mathbb{Q}(\zeta_{32})$, a cyclotomic field of degree 16.

For the case of *K* being Galois (whether or not Abelian), one has #G = n, so completely split primes *p* appear with density 1/n. If the goal is to find $n^{1+o(1)}$ such primes, enough primes to have $\Theta(n \log n)$ bits in the product, then one can reasonably conjecture that the maximum prime has only $(2 + o(1)) \log_2 n$ bits. For weaker bounds assuming GRH, see, e.g., [54, Section 2].

More needs to be done even in the Galois case: fast multiplication in each subfield requires a tower representation that keeps coefficient sizes under control, avoiding the blowups illustrated in Sect. 4.3. For the Abelian case, generalizing Gauss periods as in [29] provides explicit small-coefficient integral bases for each field, and generalizing Rader's FFT as in this paper provides fast multiplication directly on these bases. Are there explicit subfield-compatible integral bases supporting fast multiplication for Galois number fields beyond Abelian fields?

Another open question is whether one can do better than $n^{2+o(1)}$ for a degree-*n* Abelian field when *n* is prime. Perhaps one can achieve $n^{1.5+o(1)}$, analogously to how group structure is used to save an $n^{0.5+o(1)}$ factor in [28,79,90], and, in the elliptic case [20].

1438 **5 Enumerating small S-units**

The primary motivation for this paper comes from the role of det ${}^{K}_{\mathbb{Q}} \alpha$ computation inside one of the fundamental tools in computational algebraic number theory: namely, passing

1469

all small elements $\alpha \in \mathcal{O}$, where \mathcal{O} is the ring of integers of K, through a filter that outputs the *S*-units α .

This section reviews parameter choices for this tool, applications of this tool, and the conjectured performance of standard algorithms that work for arbitrary number fields *K*. This section then analyzes the impact of speedups for the case of smooth-degree cyclotomic fields $K = \mathbb{Q}(\zeta_m)$.

1447 5.1 Parameter choices

Beyond the choice of number field K, there is a choice of the set S. Typically an application specifies K, whereas S is something for the algorithm designer to optimize.

The traditional objective in choosing *S* is to minimize the time required to find filtered *S*-units that generate the *S*-unit group, i.e., the time required for a search of all small elements of \mathcal{O} to identify small *S*-units in \mathcal{O} that generate the *S*-unit group. For each *S*, the group generated by the filtered *S*-units is the full *S*-unit group once the search space is large enough, i.e., once the bound on "small" elements of \mathcal{O} is large enough; but this raises a quantitative question of how large.

For simplicity, let's take *S* specifically as $\infty \cup \{P : \#(\mathcal{O}/P) \le y\}$, where ∞ is the set of infinite places, *P* runs through nonzero prime ideals of \mathcal{O} , and *y* is a parameter to be optimized. For algorithms to compute *S*, see, e.g., [34, Sections 4.8.2 and 6.2].

A nonzero element $\alpha \in \mathcal{O}$ is an *S*-unit if and only if the ideal $\alpha \mathcal{O}$ has the form $\prod_{P:\#(\mathcal{O}/P) \leq y} P^{e(P)} \text{ for some function } e: \{P: \#(\mathcal{O}/P) \leq y\} \to \mathbb{N}. \text{ The } S\text{-unit group is the set}$ of nonzero elements $\alpha \in K$ such that the fractional ideal $\alpha \mathcal{O}$ has the form $\prod_{P:\#(\mathcal{O}/P) \leq y} P^{e(P)}$ for some function $e: \{P: \#(\mathcal{O}/P) \leq y\} \to \mathbb{Z}.$

For example, in the case $K = \mathbb{Q}$, a nonzero rational number is an *S*-unit if and only if it is *y*-smooth, i.e., has the form $\pm \prod_{p \le y} p^{e(p)}$. There is an extensive literature on the distribution of *y*-smooth integers; for surveys see, e.g., [53,73,75]. If integers are chosen independently and uniformly at random from the interval [1, H] then $(\log y)^2$ is conventionally chosen as $(1/2 + o(1))(\log H) \log \log H$, giving chance $1/y^{1+o(1)}$ of *y*-smoothness and giving total time

$$y^{2+o(1)} = \exp\left(\left(\sqrt{2} + o(1)\right) (\log H)^{1/2} (\log \log H)^{1/2}\right)$$

to find $y^{1+o(1)}$ integers that are *y*-smooth. See, e.g., [31, Theorem 10.1].

For general number fields, much less has been proven. It is still conventional to choose $(\log y)^2 \in (1/2 + o(1))(\log H) \log \log H$, where now H is an estimate for the typical size of det^K_Q α . Various applications are then conjectured to find $y^{1+o(1)}$ *S*-units in total time $y^{2+o(1)}$. See [31, Section 10] for a review of several such conjectures. These conjectures start from the heuristic that, in the words of [31, Section 10], "the auxiliary numbers that 'would be smooth' are just as likely to be smooth as random integers of the same approximate magnitude".

The reason for asking for $y^{1+o(1)}$ *S*-units is that the rank of the *S*-unit group is #S - 1, which, for reasonably large *y*, consists mainly of the number of finite places *P* in *S*. By Landau's prime-ideal theorem [65, Section 5], the number of *P* with $\#(\mathcal{O}/P) \le y$ is $(1 + o(1))y/\log y$.

Some caution is required here. First, Landau's theorem is a statement as $y \to \infty$ for a fixed *K*, not a statement regarding the conventional choice of *y* as *K* varies. Furthermore, finding $y^{1+o(1)}$ *S*-units is not a guarantee of generating the full *S*-unit group. For the number field $\mathbb{Q}(\sqrt{2}, \sqrt{3}, \sqrt{5}, \sqrt{1000003})$, a small search with some notions of smallness will find only elements of $\mathbb{Q}(\sqrt{2}, \sqrt{3}, \sqrt{5})$; Cohen gives a warning in [34, page 354, item (3), last sentence] about small *S*-units not being "random".

On the other hand, experiments suggest that, for "balanced" number fields such as $\mathbb{Q}(\zeta_m)$, small filtered *S*-units avoid such conspiracies. See, e.g., Miller's *S*-unit computation [70] proving under GRH that $\mathbb{R} \cap \mathbb{Q}(\zeta_{512})$ has class number 1.

The circular approximation in Sect. 2 says that weight-*w* elements of degree-*n* power-1491 of-2 cyclotomics have $\log H \approx n(\log w - \gamma)/2$. The conventional choice of γ then has 1492 $(\log v)^2 \in n^{1+o(1)}$ under the mild assumption $w \in n^{O(1)}$. To understand more precisely 1403 how large *w* should be, start from the standard conjectures mentioned above, which say 1494 that one needs to search $\exp(n^{1/2+o(1)})$ ring elements to find enough S-units; and then 1495 match this to the number of weight-*w* elements to see that $w \in n^{1/2+o(1)}$. Generating all 1496 small ring elements means also using smaller values of w, but almost all of the filtered 1497 S-units will be found with $w \in n^{1/2+o(1)}$, and almost all of the computation time is spent 1498 with $w \in n^{1/2+o(1)}$. See also [1] for experiments with prime cyclotomics. 1499

1500 5.2 Applications

Filtering small ring elements to find small *S*-units has a long history. The traditional application is to find fundamental invariants of \mathcal{O} , such as the class group Cl \mathcal{O} and the unit group \mathcal{O}^* . If the filtered *S*-units generate the *S*-unit group then linear algebra on the exponent vectors of the factorizations of *S*-units (the vectors *e* above) reveals the unit group (*S*-units with trivial factorization, i.e., with *e* = 0) and, if *S* is large enough, the class group (all integer vectors modulo the subgroup of "class-group relations", i.e., the subgroup of *S*-unit exponent vectors).

For details of this application, see, e.g., Cohen's description in [34, Section 6.5]. Cohen starts with a more general search that filters small elements of any ideal I to find Sgenerators of I, and applies this to ideals I obtained as random products of small prime ideals, but also, in [34, page 354, item (3)], mentions taking $I = \mathcal{O}$ as an "important speedup". For simplicity this section focuses on filtering small elements of \mathcal{O} to find Sunits, but the speedups described below generalize easily to filtering small elements of any I to find S-generators of I.

Starting a few decades ago (see [67]), filtering ring elements to find *S*-units took on new importance as a critical subroutine inside NFS, the number-field sieve for integer factorization. The number-field sieve is conjectured to factor any positive integer *N* into primes in time at most $\exp((\log N)^{1/3+o(1)})$; in this context $\log H \in (\log N)^{2/3+o(1)}$.

A much newer application is "filtered-S-unit attacks" against a problem that has arisen 1519 in cryptography, namely finding very short elements of a "worst-case" ideal I, not just 1520 the moderately short elements that one finds with, e.g., LLL. See generally [1, 17, 23]. The 1521 simplest *S*-unit attacks start with an *S*-generator g of *I* with $g \in I$, and search for shorter 1522 S-generators $gu/v \in I$ where u and v come from a database of S-units. Filtered-S-unit 1523 attacks build the database by filtering small elements of \mathcal{O} , and are conjectured in [17, 1524 page 47] to find very short elements (to be precise, "Hermite factor" at most $n^{1/2+o(1)}$; this 1525 is overkill for the cryptographic applications) in subexponential time. 1526

Class-group computations, unit-group computations, and NFS carry out linear algebra
 on the S-unit exponent vectors. The conventional choice of *y* mentioned above tries to

minimize the cost of finding *S*-units in the first place, without regard to the cost of linear algebra. If linear algebra turns out to be the main bottleneck then algorithm designers can and do improve overall algorithm performance by reducing *y*; see, e.g., [81, page 115, bottom paragraph]. The cost of finding *S*-units is then easily visible in the overall algorithm run time.

1534 5.3 The standard filtering procedure

It is straightforward to enumerate all small elements of \mathcal{O} for various reasonable notions of "small". For example, in Sect. 2, one can try all w up to some bound; for each w, enumerate possibilities for a partition of w as a sum of n squares in nonincreasing order; and, for each possibility, enumerate ways to assign the (positive and negative) square roots to $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$. This rapidly generates small elements $\alpha \in \mathcal{O}$. The big problem is to figure out which of these elements α are *S*-units.

The standard procedure (see, e.g., [34, page 491, step (6)]) computes $N = |\det_{\mathbb{Q}}^{K} \alpha|$ and throws α away if N is not y-smooth. The point here is that if $\alpha \mathcal{O} = \prod_{P:\#(\mathcal{O}/P) \leq y} P^{e(P)}$ then $N = \prod_{P:\#(\mathcal{O}/P) \leq y} \#(\mathcal{O}/P)^{e(P)}$; each #(O/P) is a prime power bounded by y, so N is y-smooth.

As mentioned above, standard conjectures say that α is an S-unit with probability 1545 $1/v^{1+o(1)}$ for the conventional choice of y, and that N is y-smooth with probability $1/v^{1+o(1)}$ 1546 for the conventional choice of y. This does not mean that these events are identical: if N 1547 is y-smooth then the standard procedure still has to check whether α is in fact an S-unit. 1548 Specifically, if N is y-smooth, then, for each prime p dividing N, the standard procedure 1549 runs through each P above p (each nonzero prime ideal P of \mathcal{O} with $p \in P$) having 1550 $\#(\mathcal{O}/P) < \gamma$, computes ord $_{P}\alpha$ as in [34, Section 4.8.3], and checks whether this accounts 1551 for the power of p in N. (The check is simpler if K is Galois: each P above p then has 1552 the same $\#(\mathcal{O}/P)$, so α is an S-unit if and only if, for each prime p dividing N, some P 1553 above p has $\#(\mathcal{O}/P) < \gamma$.) Because N is y-smooth with probability only $1/\gamma^{\Theta(1)}$, this is not 1554 a bottleneck asymptotically; as Cohen puts it in [34, page 491], "this will be done quite 1555 rarely and does not really increase the running time". 1556

1557 5.4 Exploiting automorphisms

If α is an *S*-unit and σ is an automorphism of *K* then $\sigma(\alpha)$ is also an *S*-unit. Rather than searching all small elements of \mathcal{O} , one can search orbits of small elements under the automorphism group of *K*. Typically "small" is defined in a way that is invariant under automorphisms: for example, in Sect. 2, $\sigma(\alpha)$ has weight *w* if and only if α has weight *w*.

In particular, if *K* is Galois, then the automorphism group coincides with the Galois group and has cardinality $n = \deg K$. There are two reasons that this does not imply a speedup factor *n*; on the other hand, for typical examples, the speedup factor does end up as $\Theta(n)$.

The first reason is that, for n > 1, some elements of K are in proper subfields and thus have smaller orbits. For example, in Sect. 2, $\alpha = \sum_{0 \le j < n} \alpha_j \zeta_m^j$ is in a proper subfield of Kif and only if $\alpha_j = 0$ for all odd j. However, this is increasingly rare as w grows.

The second reason is that one has to account for the cost of enumerating orbits. The naive approach is to enumerate all small ring elements and then, for each element, try applying automorphisms to see whether the element is an orbit leader (say, first in its orbit in lexicographic order as a vector on a specified basis). One expects to try only $\Theta(\log n)$ automorphisms on average for recognizing that an element isn't an orbit leader. This might sound fast enough for fields where applying an automorphism costs only $n^{1/2+o(1)}$ in sparse representation—but one wants small costs *per orbit*, not just small costs *per element*. Simply writing down an element in sparse form usually costs $n^{1/2+o(1)}$, so writing down all *n* elements in a size-*n* orbit usually costs $n^{3/2+o(1)}$. Evaluating det^K_Q α for the orbit leader α costs more than this in Sect. 3 but less than this in Sect. 4.

In various concrete examples of interest, one can easily modify the generation of small 1579 ring elements to more efficiently generate orbit leaders. For example, for power-of-2 1580 cyclotomics, an element $\alpha = \sum_{0 \le i \le n} \alpha_i \zeta_m^j$ outside all proper subfields can always be 1581 conjugated to have $\alpha_1 \neq 0$, so one can handle the degree-*n*/2 subfield recursively and then 1582 limit the generation procedure to force $\alpha_1 \neq 0$. This reduces the orbit-enumeration cost 1583 by a factor $n^{1/2+o(1)}$ for $w \in n^{1/2+o(1)}$: each element has at most w nonzero positions, and 1584 position *i* is moved to position 1 by at most two automorphisms. One can impose further 1585 restrictions to further reduce the orbit-enumeration cost; see, e.g., how the cycloshort 1586 module in [1] generates orbit leaders in the case of prime *m*, for orbits not just under 1587 automorphisms but also under multiplications by ζ_m . 1588

1589 5.5 Exploiting subfields

Take again a weight-w integral element α in a degree-n power-of-2 cyclotomic field 1590 K, and assume $w \in n^{1/2+o(1)}$. The circular approximation in Sect. 2 says that det^K_{\square} α 1501 almost always has $\Theta(n \log n)$ bits. Section 1 already explained how to compute det $\overset{\vec{K}}{\Box} \alpha$ 1592 in time $n(\log n)^{3+o(1)}$ in this case using a tower of cyclotomic subfields, and Sect. 4 1593 explained how to reach this cost for any Abelian field whose degree is $(\log n)^{o(1)}$ -smooth, 1594 whereas the best non-subfield methods from Sect. 3 take time $n^2(\log n)^{2+o(1)}$. Combin-1595 ing this $n/(\log n)^{1+o(1)}$ speedup with the $\Theta(n)$ automorphism speedup gives an overall 1596 $n^2/(\log n)^{1+o(1)}$ speedup in the sequence of det^{*K*}_{\mathbb{O}} α computations. 1597

Note that this is a speedup from one type of algorithm to another, with both types applied 1598 to power-of-2 cyclotomics: namely, a speedup from (1) general-purpose algorithms to (2) 1599 algorithms exploiting automorphisms and subfields. The subroutines used for the general-1600 purpose algorithms to reach $n^2(\log n)^{2+o(1)}$ include complex FFTs exploiting the structure 1601 of the roots of $x^m - 1$; see Sect. 3.3. For a field such as $\mathbb{Q}[x]/(x^n - x - 1)$ without this 1602 structure, the best techniques in Sect. 3 cost $n^2(\log n)^{3+o(1)}$ except when the polynomial-1603 remainder sequence is particularly short, so moving from such a field to a power-of-2 1604 cyclotomic of the same degree gives an $n^2(\log n)^{o(1)}$ speedup. This speedup factor drops 1605 to $n^2/(\log n)^{1+o(1)}$ if there is some way to reach cost $n^2(\log n)^{2+o(1)}$ for det $\mathbb{Q}^{[x]/(x^n-x-1)}$. In 1606 any case, given known techniques, it is clear that one should check the field structure, and 1607 in particular should take advantage of automorphisms and subfields. 1608

1609 5.6 Better alternatives for limited-dimension search spaces

NFS uses number fields that, compared to cyclotomics with the same size of H and the same size of y, have relatively low degree and relatively high discriminant. Quantitatively, the NFS field degree is only $(\log y)^{1+o(1)}$ rather than $(\log y)^{2+o(1)}$. Furthermore, small ring elements in NFS are tilted towards having a small number of coefficients. Most of the NFS literature considers just two integer coefficients (α_0, α_1) of an element $\alpha = \alpha_0 + \alpha_1 \theta$ of a selected number field $\mathbb{Q}(\theta)$, with a search space of $y^{2+o(1)}$ elements defined by a range of $y^{1+o(1)}$ choices of α_0 and a range of $y^{1+o(1)}$ choices of α_1 ; see, e.g., [31, Algorithm 11.1, Step 3].

In this setting, one can fix α_1 and view det $\mathcal{M}_{\mathbb{Q}}^K \alpha$ as a polynomial in the integer α_0 . One can 1618 write down successive polynomial values using repeated differences or using asymptot-1619 ically faster multipoint-evaluation subroutines. NFS algorithm statements usually avoid 1620 writing down these values in the first place: instead they observe that the values of α_0 1621 for which det^K_{\mathbb{O}} α is divisible by p consist of a small number of arithmetic progressions 1622 modulo p, and simply mark those positions in an array indexed by α_0 , assuming free access 1623 to RAM. See, e.g., [31, page 57]. This Eratosthenes-like sieving procedure accounts for the 1624 "sieve" part of the name of NFS. 1625

To limit RAM usage, more advanced versions of NFS limit the size of p found in this 1626 way; they then write down det ${}^{K}_{\Omega}\alpha$ and switch over to "cofactorization" to find larger p. 1627 This does not mean that $\det_{\mathbb{O}}^{K}$ evaluation is a bottleneck: these versions of NFS carry 1628 out cofactorization only on the occasions that the product of small p found is above a 1629 specified cutoff. With this type of "early abort", NFS is not carrying out a full search for 1630 all small S-units; but the requirement of having many small p, like the requirement of 1631 having small coefficients, is algebraically compatible with finding the full S-unit group, 1632 and analytically is conjectured to have similar smoothness probabilities. Such conjectures 1633 are again provable for \mathbb{Q} ; see Pomerance's early-abort analysis in [81, Section 4]. 1634

Sometimes the NFS literature considers two-dimensional lattices of integer pairs (α_0, α_1) for which det^{*K*}_{\mathbb{Q}} α is divisible by *p*; see, e.g., [80]. One can also consider NFS variants with three or more coefficients in α , but normally NFS takes fields where the size of θ^j increases rapidly with *j*, so one would expect the optimal lattice dimension to be small; the question of whether three coefficients are useful in NFS appears in, e.g., [15, fourth slide, bottom two lines].

The literature on class-group computation often considers fields of low degree with large coefficient ranges as in NFS, but it also considers fields of high degree with small coefficient ranges such as cyclotomics. The literature on *S*-unit attacks focuses on highdegree fields such as cyclotomics. As the lattice dimension increases and the allowed coefficient size decreases, it seems to become more and more difficult to quickly identify small ring elements in a lattice of ring elements divisible by *p*, except when *p* is very small. Fast det^{*K*}_{\mathbb{O}} evaluation plays an obvious role in these applications when *K* has high degree.

1648 5.7 Exploiting more cyclotomic structure

A useful step in computing the structure of the cyclotomic field $\mathbb{Q}(\zeta_m)$ is to compute the structure of the real-cyclotomic field $\mathbb{R} \cap \mathbb{Q}(\zeta_m)$, which has half the degree if $m \ge 3$.

For the unit group of $\mathbb{R} \cap \mathbb{Q}(\zeta_m)$, one can instantly write down generators of a full-rank subgroup, the group of "cyclotomic units". These generators are also rapidly found by a search of small ring elements.

For example, the set of cyclotomic units is $\zeta_m^{\mathbb{Z}} \prod_{c \in \{1,3,\dots,m-1\}} (1 + \zeta_m^c + \zeta_m^{-c})^{\mathbb{Z}}$ in the power-of-2 case. The group of cyclotomic units is conjectured to be the full unit group in this case. See [23, Appendix C] for a review of evidence for this conjecture.

For general *m*, there is often a gap between the group of cyclotomic units and the full unit group. To test whether the index is divisible by a given prime ℓ , one can use order- ℓ characters to see whether there are products of powers of generators of the known group that are ℓ th powers of units outside the group; if so, one can adjoin the ℓ th roots and repeat.

(This procedure is often called " ℓ -saturation" in the context of unit-group computation.)

After checking all small primes ℓ , one can reasonably hope that the full unit group is known.

To confidently obtain the class group of $\mathbb{R} \cap \mathbb{Q}(\zeta_m)$, the literature uses filtered *S*-units, as in [70]. For any number field, confirming the class number also confirms the full unit group by analytic techniques. For $\mathbb{R} \cap \mathbb{Q}(\zeta_m)$ where *m* is a prime power, these techniques boil down to Kummer's theorem that the class number of $\mathbb{R} \cap \mathbb{Q}(\zeta_m)$ is the index of the cyclotomic units inside the full group of units. See, e.g., [94, Theorem 8.2], and see [88] for a generalization to any *m*.

Starting from the units of $\mathbb{R} \cap \mathbb{Q}(\zeta_m)$, one obtains the units of $\mathbb{Q}(\zeta_m)$ by adjoining ζ_m and, for *m* not a prime power, $1 - \zeta_m$; see [94, page 40]. There is also a standard easy-tocompute formula for the class number of $\mathbb{Q}(\zeta_m)$ in terms of the class number of $\mathbb{R} \cap \mathbb{Q}(\zeta_m)$; see [94, Theorem 4.17]. For *p*-units, meaning *S*-units where the finite places in *S* are the prime ideals over *p*, one can efficiently move from the *p*-units of $\mathbb{R} \cap \mathbb{Q}(\zeta_m)$ to the *p*-units of $\mathbb{Q}(\zeta_m)$ by adjoining Jacobi sums and then taking square roots (2-saturation); see [17]. For further *S*-units of $\mathbb{Q}(\zeta_m)$, filtering appears in [17, page 47] and [1].

In short, cyclotomics make many computations easier, but filtering continues to play an important role: filtering gives *S*-units of $\mathbb{R} \cap \mathbb{Q}(\zeta_m)$, cyclotomic structure then gives the *p*-units of $\mathbb{Q}(\zeta_m)$, and further filtering gives further *S*-units of $\mathbb{Q}(\zeta_m)$.

1680 5.8 Comparison to the cost of smoothness detection

One might think that checking the *y*-smoothness of $N = \det_{\mathbb{Q}}^{K} \alpha$ is, at least asymptotically, much more expensive than computing *N* in the first place. ECM is conjectured to use exp $((\sqrt{2} + o(1))(\log y)^{1/2}(\log \log y)^{1/2})$ multiplications mod *N*; early-abort ECM from [22, Section 3] is conjectured to replace $\sqrt{2}$ with $\sqrt{8/9}$; either way, the cost is exponential in $n^{1/4+o(1)}$ when $\log y \in n^{1/2+o(1)}$.

However, one can merge smoothness tests of many integers N, even without the visible structure from Sect. 5.6. Specifically, if Y is a finite set of primes and S is a finite sequence of positive integers then the batch smoothness-detection algorithm from [14] uses assuming free RAM access— $O(B(\log B)^{2+o(1)})$ bit operations to find all Y-smooth integers in S (integers that factor into the primes in Y), where B is the total number of bits in Yand S. A closer look shows that if one plugs in the Harvey–van der Hoeven integermultiplication algorithm [56] then the number of bit operations is $O(B(\log B)^2)$.

In particular, take $Y = \{p : p \le y\}$; then Y has $\Theta(y)$ bits. Assume that S has $\Theta(y)$ integers, each having $\Theta(n \log n)$ bits. Then $B \in \Theta(yn \log n)$. The conventional choice of yhas $\log y \in \Theta(n^{1/2} \log n)$, so the cost $O(B(\log B)^2)$ is $O(yn^2(\log n)^3)$, i.e., $O(n^2(\log n)^3)$ per integer.

This analysis suggests that, with optimized subroutines for general number fields, $\det_{\mathbb{Q}}^{K}$ evaluation and batch smoothness detection are balanced—up to a constant factor, which could point in either direction—in the number of bit operations. Saving a constant factor in $\det_{\mathbb{Q}}^{K}$ evaluation thus makes the entire *S*-unit search faster by a constant factor. This does not require the work in Sect. 4—the improvement in Sect. 3.3 from $n^{2}(\log n)^{3}$ to $n^{2}(\log n)^{2}$

- for cyclotomics is enough—but further speedups in det^K_{\mathbb{Q}} evaluation, as in Sect. 4, make it easier to see speedups in batch smoothness detection.
- Understanding the real-world impact of these speedups for concrete sizes requires a 1704 much more detailed analysis and optimization of S-unit searches. As an example of the 1705 issues that will arise, the real costs of RAM are a bigger problem for batch smoothness 1706 detection than for det^K_{\square} evaluation. On the other hand, rather than taking $Y = \{p : p \le y\}$, 1707 one can take Y as the set of primes p where some P over p has $\#(\mathcal{O}/P) \leq y$. This is 1708 particularly effective in the Galois case, reducing #Y by a factor n+o(n), which also reduces 1709 RAM requirements by a factor n + o(n). The same change of Y speeds up conventional 1710 trial division. Cyclotomics also provide some speedup for Pollard's ρ method (use the 1711 iteration polynomial $x^m + c$) and Pollard's p - 1 method (if $p \in 1 + m\mathbb{Z}$ then p - 1 is more 1712 likely to factor appropriately), although much less speedup for ECM. 1713 Finally, note that working with orbits under the Galois group as in Sect. 5.4 speeds up a 1714
- sequence of smoothness tests by the same $\Theta(n)$ factor that it speeds up a sequence of det^{*K*}_Q evaluations: there are that many fewer *N* values to handle. It is clear that known algorithms for *S*-unit searches are much faster for cyclotomics than for unstructured number fields; the only question is how much.

Acknowledgements

Thanks to Tanja Lange for her comments. Thanks to Christine van Vredendaal for carrying out initial versions of the
 experiments in Sect. 2.

- 1721 Funding Information Open Access funding enabled and organized by Projekt DEAL.
- 1722 Data availability Data sharing not applicable to this article as no datasets were generated or
- analysed during the current study.

Author details

¹Department of Computer Science, University of Illinois at Chicago, Chicago, USA, ³Horst Görtz Institute for IT Security,

Ruhr University Bochum, Bochum, Germany, ⁴Research Center for Information Technology Innovation, Academia Sinica,
 Taipei, Taiwan.

1727 Appendix A: software to check the Gauss-period algorithms

An open-source software package abelianfields is available from [18] for various

- 1729 tests that the main algorithms described in this paper work as specified. This appendix
- describes the software.

1731 A.1: software readability

- The idea that a test of software S is also a test of algorithm A relies implicitly on the idea that there is a match between S and A. Helping readers check this match was a high priority in the development of the abelianfields software package. This has three important consequences.
- First, this software is written in a high-level language, specifically Sage. This has the disadvantage of incurring Sage overhead for each step. This is a large slowdown when one step is an arithmetic operation in a small finite field, as in Sect. 4.12.2. The software does not attempt to show what can be done in reducing overhead per ring operation. See Appendix B for further software illustrating ways to streamline det^{*K*}_{\mathbb{Q}} evaluation in the case of power-of-2 cyclotomic fields *K*, using a lower-level language.
- Second, abelianfields uses straightforward subroutines for precomputations. Here
- ¹⁷⁴³ "precomputations" refers to algorithm steps that depend only on the number field and the

number of input bits, independently of the specific input at hand. The paper's performance 1744 evaluation assumes that the results of these precomputations are cached, and that there are 1745 enough inputs that the cost of the precomputations does not matter; consequently, various 1746 speedups in the precomputations, such as known techniques to accelerate the search 1747 for prime fields with appropriate primitive roots of 1, are simply ignored. The software 1748 makes a few exceptions for subroutines that seemed likely to cause scaling problems for 1749 experiments and that were easy to rewrite in faster ways. 1750 Third, within the main computation, various standard speedups are suppressed because 1751

they would compromise readability. For example, FFTs are not cached; software prioritizing speed would include FFT caching, even though this does not affect the $n(\log n)^{3+o(1)}$ asymptotic. As another example, the software includes not just tests of its main functions, but also many assertions inside the functions to highlight assumptions and conclusions; eliminating assertions is a standard speedup.

1757 A.2: basic subroutines

The following low-level functions inside abelianfields are used in various ways inside the convolution functions described in Appendix A.3, the prime-conductor det functions described in Appendix A.4, and the general-conductor det functions described in Appendix A.5:

primitive.root_remember, given a ring *R*, a positive integer *n*, and a primitive *n*th root of 1 in *R*, caches that root for future reference. The caching does not currently move across pairs (*R*, *n*), for example to obtain a primitive *n*th root by squaring a previously cached primitive 2*n*th root.

primitive.root returns a primitive *n*th root of 1 in *R*, given *R* and *n*. This function
 tries to find and cache a primitive *n*th root if one has not been previously specified by
 primitive.root_remember; this search is fast when *R* is a prime field.

- units.generator, given *m*, returns a deterministically selected generator of $(\mathbb{Z}/m)^*$, if $(\mathbb{Z}/m)^*$ is cyclic. Note that applying primitive.root to inputs $R = \mathbb{Z}/m$ and $n = \#(\mathbb{Z}/m)^*$ often differs from this: for example, $(\mathbb{Z}/4)^*$ has a generator 3, but $\mathbb{Z}/4$ does not have a primitive 2nd root of 1.
- tree.producttree and tree.remainders are product-tree/remainder-tree
 subroutines. These are copied from [21], except for minor tweaks to upgrade to
 Python 3.
- tree.interpolate is an analogous interpolation-tree subroutine. (Sage's built-in CRT function provides the same results as tree.interpolate aside from details of how inputs are arranged, but uses an algorithm that scales quadratically in most cases. This difference is outside this paper's algorithm analysis, since interpolation is used only in precomputation.)
- auxmodulus.prime, given *B* and *n*, returns the smallest prime number in $\{B, B+1, B+2, \ldots\} \cap (1+n\mathbb{Z}).$
- auxmodulus.product, given *B* and *n*, returns an integer $M \ge B$ and a sequence *P* of distinct prime numbers in $1 + n\mathbb{Z}$ having product *M*. This function also uses primitive.root_remember to remember a primitive *n*th root of 1 in \mathbb{Z}/M , obtained from interpolating (via tree.interpolate) primitive *n*th roots of 1 in \mathbb{Z}/p across *p* in *P*. Currently the prime numbers in *P* are chosen as the smallest prime

1788 1789 1790	numbers in $1 + n\mathbb{Z}$, with the search stopping once the number of bits in the prime numbers minus the number of primes is at least the number of bits in <i>B</i> . More work would usually bring <i>M</i> somewhat closer to <i>B</i> .
1791 1792	The general-conductor det functions also rely on various further manipulations of sub- groups and quotient groups of $(\mathbb{Z}/m)^*$, which are abstracted by units as follows:
1793 1794	 units.group, given <i>m</i>, returns the group (Z/m)*. This group supports elements (see units.element below), iteration, and the following functions:
1795 1796	- cardinality returns $\#(\mathbb{Z}/m)^*$.
1797	- gens returns a vector of independent generators of $(\mathbb{Z}/m)^*$. Calling this function repeatedly always returns the same vector
1798	 ngens returns the number of generators returned by gens.
1800 1801	 gens_orders returns a vector showing the order of each generator returned by gens.
1802 1803	 ring returns Z/m, represented as Sage's Zmod(m). subgroup_1mod, given a positive divisor d of m, returns the kernel (represented
1804 1805	as a units.subgroup; see below) of the natural map from $(\mathbb{Z}/m)^*$ to $(\mathbb{Z}/d)^*$. Internally, this uses a trivial enumeration of $(\mathbb{Z}/m)^*$ for simplicity; one can do
1806	better for large <i>m</i> .
1807 1808	 units.element, given (Z/m)* (represented as a units.group) and an element of Sage's Zmod(m) or ZZ coprime to m (or, alternatively, an exponent vector on
1809 1810	units.group(m).gens()), returns an element of $(\mathbb{Z}/m)^*$, with support for all of Sage's AbelianGroupElement features (e.g., multiplication and exponents)
1811	- modulus returns m.
1813	 inring returns the corresponding element of Zmod (m).
1814	- reduce, given a positive divisor d of m, returns the element of $(\mathbb{Z}/d)^*$ obtained by feeding this element of $(\mathbb{Z}/m)^*$ through the natural map from $(\mathbb{Z}/m)^*$ to
1816	$(\mathbb{Z}/d)^*$.
1817	• units.subgroup, given <i>m</i> and $g_1, g_2, \ldots \in (\mathbb{Z}/m)^*$, returns the subgroup <i>H</i> of $(\mathbb{Z}/m)^*$ generated by $g_1, g_2, \ldots \in (\mathbb{Z}/m)^*$
1818 1819	$(\mathbb{Z}/m)^*$ generated by g_1, g_2, \ldots . This subgroup supports elements (as elements of $(\mathbb{Z}/m)^*$), iteration, and the following functions:
1820	 cardinality returns #H.
1821	– modulus returns <i>m</i> .
1822 1823	- gens, ngens, and gens_orders are as above, but for independent generators of H rather than independent generators of $(\mathbb{Z}/m)^*$.
1824	- fullgroup returns $(\mathbb{Z}/m)^*$.
1825	- quotient returns the quotient group $(\mathbb{Z}/m)^*/H$, represented as a units quotient (see below)
1825	- reduce, given a positive divisor d of m. returns the subgroup of $(\mathbb{Z}/d)^*$ obtained
1828	by feeding <i>H</i> through the natural map from $(\mathbb{Z}/m)^*$ to $(\mathbb{Z}/d)^*$.

1829	- extend, given a vector of elements of $(\mathbb{Z}/m)^*$, returns the subgroup generated by H and those elements.	
1831	- intersect, given a subgroup H' of $(\mathbb{Z}/m)^*$, returns $H \cap H'$.	
1832	- is subgroup of, given a subgroup H' of $(\mathbb{Z}/m)^*$, returns True if H is a sub-	
1833	group of H' , else False.	
1834	- representatives mod, given a subgroup H' of H (as a subgroup of $(\mathbb{Z}/m)^*$),	
1835	returns representatives in $(\mathbb{Z}/m)^*$ of the quotient H/H' .	
1836	- conductor returns the conductor of <i>H</i> .	
1837	- is_radfree returns True if H is <i>rad-free</i> , meaning that H acts freely on	
1838	$(\mathbb{Z}/\mathrm{rad}m)^*$.	
1839 1840	 is_friendly returns True if <i>H</i> is <i>friendly</i>, meaning that (1) <i>m</i> is divisible by 8 and all elements of <i>H</i> are 1 modulo 4 or (2) <i>m</i> is not divisible by 8. 	
	$Hof(\mathbb{Z}/m)^*$ returns the quotient group	
1841	• unites equotient, given <i>m</i> and a subgroup <i>T</i> or (\mathbb{Z}/m) , returns the quotient group $O = (\mathbb{Z}/m)^*/H$ This quotient group supports elements (see	
1842	units.quotientelement below), iteration, and the following functions:	
1844	– cardinality returns #Q.	
1845	- modulus returns <i>m</i> .	
1846	- gens, ngens, gens_orders are as above, but for independent generators of	
1847	Q.	
1848	- fullgroup returns $(\mathbb{Z}/m)^*$.	
1849	- denominator returns H.	
1850	• units.quotientelement, given $Q = (\mathbb{Z}/m)^*/H$ as above and an element of	,
1851	$(\mathbb{Z}/m)^*$ (or, alternatively, an exponent vector on Q . <code>gens()</code>), returns an element of	,
1852	Q, with support for all of Sage's AbelianGroupElement features and the following	
1853	functions:	
1854	– lift returns a preimage of this element under the natural map from $(\mathbb{Z}/m)^*$ to	
1855	Q.	
1856	- reduce, given a positive divisor d of m, returns the element of $(\mathbb{Z}/d)^*/H$	
1857	obtained by feeding this element of Q through the natural map from $(\mathbb{Z}/m)^*/H$	
1858	to $(\mathbb{Z}/d)^*/H$.	
1859	Sage has a built-in Zmod(m).unit_group() that supports some of the above fea-	
1860	tures; abelianfields uses Zmod(m).unit_group() for some tests, and also tests	
1861	units.subgroup.conductor against PARI's computation of conductors of fixed	
1862	fields of subgroups of the Galois group of $\mathbb{Q}(\zeta_m)$.	
1863	A.3: convolution functions	
1864	The convolution module inside abelianfields handles the standard convolution	
1865	techniques from Sects. 4.12.1 and 4.12.2:	
1866	• convolution.multi_fft evaluates an isomorphism to \mathbb{R}^n from the ring $S =$	

- $R[x_1, x_2, \dots, x_t] / (x_1^{d_0} x_0^{e_0}, \dots, x_t^{d_{t-1}} x_{t-1}^{e_{t-1}}), \text{ given a ring } R, \text{ a primitive } n \text{th root } \zeta$
- of 1 in R, a unit x_0 in R, a list d of t positive integers with product n, and a list e of

- t nonnegative integers where *n* divides e_0 if $t \ge 1$. This function also supports an optional reverse=True argument that inverts the isomorphism.
- ¹⁸⁷¹ convolution.multi multiplies in the above ring *S*.
- convolution.multi_cyclic multiplies in *S* in the case $e_0 = e_1 = \cdots = e_{t-1} = 0$.
- convolution.cyclic is the case t = 1, multiplying in $R[x]/(x^d 1)$.

The convolution.multi_fft and convolution.multi functions leave it to the caller to handle replacing d_i with $p, d_i/p$ for speed when d_i is composite and p is a prime divisor of d_i . The *cyclic* functions, which are the functions used elsewhere in abelianfields, handle this replacement automatically.

By default, the *cyclic* functions require *R* to be \mathbb{Z} or \mathbb{Z}/M , and do not require *R* to contain a primitive *n*th root of 1. These functions automatically decompose large coefficients into elements of a small prime field \mathbb{Z}/p , saving a logarithmic factor as described in Sect. 4.12.2. Presumably handling this by segmentation, as in Sect. 4.8.5, would be noticeably faster for small *t*, and in particular for convolution.cyclic.

The *cyclic* functions also support a guaranteed_primitive=True option that requires R to contain a primitive nth root of 1; in this case R is not required to be \mathbb{Z} or \mathbb{Z}/M . Currently guaranteed_primitive=True also disables decomposing large coefficients into elements of a small prime field.

1888 A.4: functions for prime-conductor fields

The prime module inside abelianfields handles number fields of odd prime conductor p, along with \mathbb{Q} . Some of the functions in this module provide size-p FFT algorithms and, more generally, folded Rader FFTs for positive divisors d of p - 1:

• prime.complete is a size-p DFT. This function is given a ring R, an odd prime number p, a primitive pth root ζ of 1 in R, and the coefficients $g_0, g_1, \ldots, g_{p-1}$ of a polynomial $g = g_0 + g_1 x + \cdots + g_{p-1} x^{p-1} \in R[x]$; this function returns $g(1), g(\zeta_p), \ldots, g(\zeta_p^{p-1})$. Internally, this function implements Rader's original FFT algorithm reviewed in Sect. 4.8.1. This function is not used elsewhere; it is provided as a baseline algorithm for comparison.

- prime.folded takes R, p, ζ , a positive integer d dividing p 1, and d coef-1898 ficients $g_0, g_1, \ldots, g_{d-1}$ representing the *d*-periodic polynomial $g = \sum_i g_i (x^{\omega^{-i}} + i)$ 1899 $x^{\omega^{d-j}}+\dots+x^{\omega^{p-1-d-j}})$ where ω is units.generator(p). This function returns 1900 $g(\zeta^{\omega^0}), \ldots, g(\zeta^{\omega^{d-1}})$. Internally, this function uses the folded Rader algorithm 1901 reviewed in Sect. 4.8.3, so it is simply a length-d cyclic convolution with a pre-1903 computed vector. The largest case, d = p - 1, is close to prime.complete, but 1903 prime.complete allows nonzero constant coefficient, also evaluates at 1, and has 1904 a different order of inputs and outputs. 1905
- prime.folded_inverse inverts prime.folded for any particular (*R*, *p*, *ζ*, *d*).
 See Sect. 4.8.4 for the algorithm.

All of these functions currently require R to be \mathbb{Z} or some \mathbb{Z}/M , and internally use convolution.cyclic.

Further prime functions operate on elements of \mathbb{Z}^d , representing *d*-periodic polyno-

mials over \mathbb{Z} , representing (integral) *d*-periodic elements of $\mathbb{Q}(\zeta_p)$, meaning elements of

the degree-*d* subfield of $\mathbb{Q}(\zeta_p)$. Note that, in particular, the element $(z, z, ..., z) \in \mathbb{Z}^d$ represents the element -z of $\mathbb{Q}(\zeta_p)$. Each function is given *p* and *d* along with the further inputs described below:

prime.multiply takes two elements of \mathbb{Z}^d representing *d*-periodic elements 1915 $\alpha, \beta \in \mathbb{Q}(\zeta_n)$, and returns an element of \mathbb{Z}^d representing $\alpha\beta$. Internally, 1916 prime.multiply computes an easy bound on the absolute coefficients of $\alpha\beta$; 1917 uses auxmodulus.product to choose a modulus *M* above twice this bound with 1918 a primitive *p*th root of 1 in \mathbb{Z}/M ; and finishes with two length-*d* folded DFTs over 1919 \mathbb{Z}/M , d multiplications in \mathbb{Z}/M , and one length-d inverse folded DFT over \mathbb{Z}/M . 1920 Each folded DFT boils down to one length-d cyclic convolution with a precomputed 1021 sequence. 1922

• prime.subfield takes a positive integer d_2 dividing d, and an element of \mathbb{Z}^d representing a d-periodic element $\alpha \in \mathbb{Q}(\zeta_p)$. It returns an element of \mathbb{Z}^{d_2} representing α , if α is d_2 -periodic. In other words, it extracts the first d_2 entries of the input sequence, if the input sequence is d_2 -periodic.

• prime.conjugate takes an integer e, and an element of \mathbb{Z}^d representing a dperiodic element $\alpha \in \mathbb{Q}(\zeta_p)$. It returns an element of \mathbb{Z}^d representing $\sigma(\alpha)$, where σ is the unique automorphism of $\mathbb{Q}(\zeta_p)$ mapping ζ_p to $\zeta_p^{\omega^e}$, where again ω is units.generator(p). In other words, prime.conjugate rotates the input sequence to the left by e positions.

prime.det_relative evaluates the determinant map from the degree-d sub-1932 field of $\mathbb{Q}(\zeta_p)$ down to the degree- d_2 subfield of $\mathbb{Q}(\zeta_p)$, where d_2 is a positive inte-1933 ger dividing d. This function takes d_2 and an element of \mathbb{Z}^d representing a d-1934 periodic element $\alpha \in \mathbb{Q}(\zeta_p)$, and returns an element of \mathbb{Z}^{d_2} . Internally, this function 1935 includes two implementations (tested against each other): the default implementa-1936 tion saves time by pushing conjugation and subfield extraction through the DFTs 1937 as explained in Sect. 4.8.6, but there is also a simpler reference implementation, 1938 enabled by ref=True, that multiplies conjugates by calling prime.conjugate 1939 and prime.multiply as black boxes. 1940

• prime.det_absolute evaluates the determinant map from the degree-*d* subfield of $\mathbb{Q}(\zeta_p)$ down to \mathbb{Q} . Internally, this function automatically factors *d* into primes and repeatedly applies prime.det_relative. One could alternatively modify prime.det_relative to internally perform this factorization; either way, the factorization is essential for the speed of this paper's algorithms.

1946 A.5: functions for arbitrary-conductor fields

The general module inside abelianfields has a similar structure to the prime module but supports arbitrary conductor. Some of the functions provide the construction from [29, Section 5] of an *H*-normal basis:

• general.normalbasis, given a rad-free subgroup H of $(\mathbb{Z}/m)^*$, returns B/Hand a list of rewrite rules. Here B is an H-normal integral basis of $\mathbb{Q}(\zeta_m)$ consisting of roots of 1, and the rewrite rules specify how to rewrite arbitrary roots of 1 in terms of B. Internally, the rewriting follows the construction from [97] and [26] reviewed in Sect. 4.10.

- general.canonicalize expresses an *H*-periodic polynomial as an *H*-periodic
 polynomial on the basis *B* from general.normalbasis, rewriting any other expo-
- ¹⁹⁵⁷ nents that appear. The input and output use exponents in $(\mathbb{Z}/m)/H$ having additive
- order divisible by rad m; the output is guaranteed to be supported on B, which is
- automatic if *m* is squarefree but not for general *m*.
- general.from_conventional also produces an *H*-periodic polynomial on the
- basis B, but takes input in its conventional (not H-folded) form as a list of m exponents, with no requirements on the additive order.
- ¹⁹⁶³ Some of the functions provide generalized Rader FFTs and generalized folded Rader FFTs:
- As a warmup without folding, general.primitive is a primitive size-*m* DFT. This
- function is given a ring *R*, a positive integer *m*, a primitive *m*th root ζ of 1 in *R*, and the coefficients $g_0, g_1, \ldots, g_{m-1}$ of a polynomial $g = g_0 + g_1 x + \cdots + g_{m-1} x^{m-1} \in R[x]$. This function returns $g(\zeta^c)$ for each $c \in \{0, 1, \ldots, m-1\}$ with $gcd\{c, m\} = 1$. Internally,
- function returns $g(\zeta^c)$ for each $c \in \{0, 1, ..., m-1\}$ with $gcd\{c, m\} = 1$. Internally this function uses the generalized Rader FFT from Sect. 4.12.3.
- general.primitive_inverse is an inverse primitive size-*m* DFT. Internally, this uses the inversion algorithm (and tweak) from Sect. 4.12.4.
- general.folded is an *H*-folded DFT, as in Sect. 4.12.5. This function is given R, m, ζ , a subgroup *H* of $(\mathbb{Z}/m)^*$, and an *H*-periodic polynomial; it evaluates the
- ¹⁹⁷³ polynomial at ζ^c for each *c* in $(\mathbb{Z}/m)^*/H$. The subgroup *H* is required to be rad-free.
- general.folded_inverse is an inverse *H*-folded DFT.
- Finally, as in prime, there are further functions to operate on *H*-periodic polynomials over \mathbb{Z} , representing elements of the subfield of $\mathbb{Q}(\zeta_m)$ fixed by *H*. Each function is given *m* and *H* along with the further inputs described below:
- general.multiply takes two *H*-periodic polynomials *f*, *g* and returns *fg*. Internally, this uses *H*-folded DFTs.
- general.conjugate takes an element $c \in (\mathbb{Z}/m)^*$ and an *H*-periodic polynomial f, and returns $\sigma_c(f)$, where σ_c is the unique automorphism of $\mathbb{Q}(\zeta_m)$ mapping ζ_m to ζ_m^{∞} .
- general.subfield changes representation from the subfield K of $\mathbb{Q}(\zeta_m)$ fixed by 1983 *H* to the subfield *F* of $\mathbb{Q}(\zeta_{\ell})$ fixed by a subgroup *S* of $(\mathbb{Z}/\ell)^*$, where $F \subseteq K$, assuming 1984 the input represents an element of F. This function takes as input a subgroup H_2 of 1984 $(\mathbb{Z}/m)^*$ containing *H*; the function defines ℓ as the conductor of *H*₂, and *S* as the 1986 reduction of H_2 to $(\mathbb{Z}/\ell)^*$. This function also takes as input an *H*-periodic polynomial 1987 f, and produces as output an S-periodic polynomial. The exact set of pairs (H, S)1988 supported by this function has a complicated description from details of how the 1989 function steps through conductors and subgroups; both H and S are required to be 1990 rad-free, and none of the default abelianfields tests include any cases where this 1991 condition is insufficient. 1992
- general.det_relative has the same input-output format as
 general.subfield but evaluates det^K_F rather than evaluating the identity map on
 *F.*Internally, this function simply multiplies conjugates using general.conjugate
 and general.multiply.

¹⁹⁹⁷ • general.det_absolute takes an *H*-periodic polynomial representing an ele-

- ¹⁹⁹⁸ ment α of the subfield K of $\mathbb{Q}(\zeta_m)$ fixed by H, and returns det^K_Q α . This function
- requires *H* to be friendly.

A subroutine general.tower constructs the tower used in general.det_absolute. Internally, general.tower starts with $(\mathbb{Z}/m)^*$ and builds a chain of subgroups down towards *H*. It tries to insert one prime at a time into the field degree so as to obtain a maximum-length tower, but, for $m \in 8\mathbb{Z}$, sometimes intersects the current subgroup with $1 + 4\mathbb{Z}$ to preserve the rad-free condition, in effect losing one degree-2 step in the tower. There is no attempt in general.tower to search for alternative chains.

2006 A.6: output of the tests

Running all internal abelianfields tests on Linux systems with Sage installed is a simple matter of typing make in the package directory. Various notes are printed regarding which test size is in progress, but not regarding all the individual tests. Any failure is printed as a Sage assertion failure.

Along with correctness tests, the tests of the prime module include reporting, for each 2011 small prime p, the sizes of convolutions used in computing determinants down to \mathbb{Z} of five 2012 random small elements of $\mathbb{Z}[\zeta_p]$, specifically elements with $\lceil (p-1)^{1/2} \rceil$ coefficients ± 1 2013 and all remaining coefficients 0. For example, a typical computation for p = 29 involved 2014 convolutions of lengths 28 (for a forward DFT) and 14 (for an inverse DFT) over \mathbb{Z}/M 2015 where *M* has 14 bits, convolutions of lengths 14 and 7 over \mathbb{Z}/M where *M* has 23 bits, 2016 and convolutions of lengths 7 and 1 over \mathbb{Z}/M where *M* has 83 bits, as reflected by the 2017 following output line: 2018

2019 p 29 totalbits 1735 Mbits,n: 14,28 14,14 23,14 23,7 83,7 83,1

The product of *M* bits and *n* adds up to 1735 in this example. Sizes can vary from one element to another; see generally Sect. 2. To skip most tests and simply see these convolution sizes, one can run make sagelibs and then, inside Sage, run the following, although this still tests the results against Sage's resultant subroutine:

```
2024 import prime
2025 prime.test_sizes(29)
```

The totalbits quantity is not monotonic in p: for example, the quantity is typically 9090 or 9480 for p = 59 and typically just 3099 or 3177 or 3249 for p = 61, reflecting the fact that $\mathbb{Q}(\zeta_{61})$ has a much nicer tower than $\mathbb{Q}(\zeta_{59})$ does.

Five experiments with $p = 193 = 1+3 \cdot 2^6$ had totalbits being 16739, 17157, 17168, 17228, 16859; five experiments with $p = 769 = 1 + 3 \cdot 2^8$ had totalbits being 83286, 83368, 82804, 82962, 82964. The 5× growth in totalbits from p = 193 to p = 769 is, as expected, slightly larger than the 4× growth in p: the number of tower levels is growing logarithmically, and one expects another near-logarithmic factor for reasons explained in Sect. 2.

(Similar comments apply to the general module, with larger sizes since general does not push conjugation and folding through DFTs. For example, five experiments from general.test_sizes(193) had totalbits being 36981, 36711, 38196, 36981, 35703, and five experiments from general.test_sizes(769) had totalbits being 176103, 212355, 176922, 176103, 176067.)

Beyond totalbits, there is a third logarithmic factor in the run time of this paper's 2040 algorithm, reflecting the cost of convolution per bit. However, one cannot expect to see 2041 this logarithmic factor in wall-time measurements for this software. Operations in the 2043 small prime fields in Sect. 4.12.2 have inherent cost growing with the size of the prime (which is also roughly logarithmic, with some bumps for the distribution of primes), but 204/ in Sage the cost is instead dominated by prime-independent overhead. Wall time was 204 monitored experimentally as a sanity check (with repeated runs so that precomputations 2046 were cached) and grew approximately $5 \times$ from p = 193 to p = 769, but this should not 2047 be taken as a predictor of the wall-time ratio for optimized software. 2048

2049 Appendix B: faster software for the case of power-of-2 cyclotomics

For the numerical example $\alpha = 3 + \zeta_{2048}^{271} + 4\zeta_{2048}^{828}$ with $K = \mathbb{Q}(\zeta_{2048})$, Sect. 1 compared the performance of traditional det^{*K*}_{\mathbb{Q}} α computation—namely 0.21 × 10⁹ cycles via PARI, or 0.15 · 10⁹ cycles via NTL—to the performance of exploiting a tower of subfields of *K*—more than 10× faster, namely 0.011 × 10⁹ cycles.

A closer look shows that this underestimates the speed advantage of exploiting a tower of subfields. The overhead in Python and Sage for computing g(-x), extracting coefficients, etc. turns out to account for most of the 0.011×10^9 cycles. This issue is essentially nonexistent for the scripts using PARI and NTL: those scripts are bottlenecked by resultant subroutines written in C.

This appendix describes cyclo2power (available from [19] as an accompaniment to this paper), an open-source library to compute det ${}^{K}_{\mathbb{Q}} \alpha$ for integral elements α of powerof-2 cyclotomic fields *K*. The library is written in C and, like NTL and PARI, uses GMP [52] for integer arithmetic. For the numerical example from Sect. 1, cyclo2power uses just 0.0012 × 10⁹ cycles on the same machine, more than 100× faster than NTL.

2064 B.1: representing polynomials as integer values

Segmentation (sometimes called "Kronecker substitution"), also used in Sects. 3.3, 4.2, and 4.8.5, multiplies two polynomials $f, g \in \mathbb{Z}[x]$ by multiplying the integers $f(\rho), g(\rho)$. Here $\rho \in \mathbb{Z}$ is chosen to be a power of 10 for traditional hand computation or a power of 2068 2 for software, so it is easy to read off the coefficients of h = fg from the digits or bits of 2069 $h(\rho)$. One can think of the integers $f(\rho), g(\rho), h(\rho)$ as representations of the polynomials 2070 f, g, h respectively.

For example, one way to multiply $3x^2 + x + 4$ by $2x^2 + 7x + 1$ is to choose $\rho = 1000$ and multiply the integers 3001004 and 2007001, obtaining the integer 6023018029004, from which one easily reads off the polynomial product $6x^4 + 23x^3 + 18x^2 + 29x + 4$. Of course, for this to work, ρ has to be large enough compared to the polynomial coefficients.

It might seem pointless to reduce polynomial arithmetic to integer arithmetic given that integers, in turn, are normally represented as values of polynomials—for example, the notation "6023018029004" above refers, by definition, to the value at 10 of the polynomial $6y^{12} + 2y^{10} + 3y^9 + y^7 + 8y^6 + 2y^4 + 9y^3 + 4$. But if one is given GMP for integer arithmetic then segmentation is well known to be an easy way to carry out polynomial arithmetic.

2080 B.2: tower compatibility of segmentation

Let $n \ge 2$ be a power of 2. Write m = 2n. As usual, represent each $\alpha \in \mathbb{Z}[\zeta_m]$ as the unique $g \in \mathbb{Z}[x]/(x^n + 1)$ with $g(\zeta_m) = \alpha$. As above, represent g by its value $g(\rho) \in \mathbb{Z}/(\rho^n + 1)$ where ρ is a power of 2. One can recover g and thus α from this value if the coefficients of g are small enough compared to ρ .

The product $g(\rho)g(-\rho)$ now represents $g \cdot g(-x)$, which in turn represents $\det_{\mathbb{Z}[x]/(x^{n+1})}^{\mathbb{Z}[x]/(x^{n+1})} \alpha$.

This product $g(\rho)g(-\rho)$ is the same as $G(\rho^2)$ for the unique $G \in \mathbb{Z}[x]/(x^{n/2}+1)$ with

 $G(x^2) = g \cdot g(-x)$. Now replace (g, ρ) with (G, ρ^2) and repeat.

This is how cyclo2power works, with ρ chosen as the smallest power of 2^{16} such that the maximum absolute value of the coefficients of the input *g* is below $\rho/2n$. Each coefficient of *G* is a sum of *n* products (sometimes negated) of coefficients of *g*, and is thus below $\rho^2/4n$ in absolute value. Taking $\rho/2n$ instead of ρ/n simplifies the negativecoefficient rewriting described below, and requiring a power of 2^{16} rather than just a power of 2 simplifies the coefficient handling more broadly.

For example, if n = 1024 and g has small coefficients then cyclo2power chooses $\rho = 2^{16}$, so $\rho^n = 2^{16384}$. This is several times more bits than necessary to represent typical outputs; see Sect. 2. Presumably one could obtain some further speed by (1) allowing the initial ρ to be, e.g., 2^4 , despite the cost of handling unaligned data, and (2) reducing ρ partway through the computation, despite the cost of having to switch to a different representation.

Note that multiplying $g(\rho)$ by $g(-\rho)$ is multiplying in $\mathbb{Z}/(\rho^n + 1)$. Many integermultiplication methods benefit from moduli of the special form $\rho^n + 1$ (see generally [16, Section 3]), gaining about a factor 2 in performance compared to multiplying the same inputs in \mathbb{Z} . However, the documented GMP interface does not provide any of these special multiplication functions.

2105 **B.3: negative coefficients**

Consider the polynomial $f = 5x^3 + 6x^2 - 7x - 8$. The value of f at $\rho = 1000$ is $f(\rho) = 5005992992$. One can recover the coefficients of f from $f(\rho)$ by rewriting the bottom 992 as 1000 - 8, producing coefficient -8 and quotient 5005993; then rewriting 993 as 1000 - 7, producing coefficient -7 and quotient 5006; etc.

This rewriting is a considerable part of the cyclo2power code. One cannot reasonably avoid negative coefficients in the context of this paper: this would limit the pool of inputs α in the applications in Sect. 5, and would almost always prohibit converting g into g(-x). This complication disappears if one instead represents integers using a balanced digit set, such as the redundant digit set $\{-5, -4, ..., 4, 5\}$ in radix 10: one can then simply negate each coefficient. However, GMP does not represent integers in this way.

2116 B.4: speedups over NTL

For small *n* (e.g., n = 4) and small coefficients (e.g., $\{-1, 0, 1\}$), cyclo2power is about twice as fast as NTL's resultant computation applied to $x^n + 1$. The cyclo2power advantage grows rapidly as *n* increases. For example, for n = 1024 and coefficients -1, 0, 1 with probability 1/4, 1/2, 1/4 respectively, NTL takes about 0.66×10^9 cycles while cyclo2power takes about 0.0014×10^9 cycles, more than 400 times faster. The cyclo2power advantage also grows somewhat as the number of bits per coefficient

- increases, although this is less striking than the growth with n and less relevant to the
- ²¹²⁴ applications considered in this paper.
- There are also functions in cyclo2power for handling power-of-2 real-cyclotomic
- fields instead of power-of-2 cyclotomic fields. This boils down to skipping a final squaring.
- This speedup is most noticeable for small n.
- 2128 Received: 12 August 2022 Accepted: 28 September 2022

2129 References

- 1. S-unit attacks: filtered. Web page with software, 17 December 2021 (2021). https://s-unit.attacks.cr.yp.to/filtered.html
- Anderson, L., Chahal, J.S., Top, J.: The last chapter of the Disquisitiones of Gauss. Hardy-Ramanujan J. 44, 152–159 (2021)
- 21333.Argand, J.-R.:Essai sur une manière de representer lesquantités imaginaires dans les construc-2134tions géométriques (1806).Reprinted in [5].http://www.bibnum.education.fr/mathematiques/geometrie/2135essai-sur-une-maniere-de-representer-des-quantites-imaginaires-dans-les-cons
- Argand, J.-R.: Reflexions sur la nouvelle théorie desimaginaires, suivies d'une application à la demonstration d'untheorème d'analise. Annales de mathématiques pures et appliqué, 5, 197–209 (1814). Reprinted on pages 112–123 of [5]. https://babel.hathitrust.org/cgi/pt?id=uc1.%24c126479&view=1up&seq=209
- Argand, J.-R.: Essai sur une manière de representer lesquantités imaginaires dans les constructions géométriques, 2nd edn. Gauthier-Villars (1874). https://archive.org/details/essaisurunemani00julegoog/page/n152/mode/2up. See [3, 4]
- Arita, S., Handa, S.: Subring homomorphic encryption. In: ICISC 2017 [64], pp. 112–136 (2017). https://eprint.iacr.org/ 2017/066
- 2144 7. Bach, E., Shallit, J.: Factoring with cyclotomic polynomials. Math. Comput. **52**, 201–219 (1989)
- Baeza-Yates, R.A., Goles, E., Poblete, P.V. (eds.). LATIN'95: Theoretical Informatics, Second Latin American Symposium, Valparaíso, Chile, April 3–7, 1995, Proceedings, vol. 911. Springer (1995). ISBN 3-540-59175-3. See [47]
- Bauch, J., Bernstein, D.J., de Valence, H., Lange, T., van Vredendaal, C.: Short generators without quantum computers: the case of multiquadratics. In: Eurocrypt 2017 [38], pp. 27–59 (2017). https://eprint.iacr.org/2017/404
- 2149 10. Belabas, K.: Topics in computational algebraic number theory. Journalde théorie des nombres de Bordeaux 16, 19–63
 (2004)
- Benger, N., Charlemagne, M., Chen, K.: A note on the behaviour of the number field sieve in the medium prime case: smoothness of norms. J. Shanghai Jiaotong Univ. (Sci.) 23, 138–145 (2018)
- 12. Berndt, B.C., Evans, R.J.: The determination of Gauss sums. Bull. Am. Math. Soc. New Ser. 5, 107–129 (1981)
- 13. Bernstein, D.J.: Multidigit multiplication for mathematicians (2001). https://cr.yp.to/papers.html#m3
- 14. Bernstein, D.J.: How to find smooth parts of integers (2004). https://cr.yp.to/papers.html#smoothparts
- Bernstein, D.J.: Polynomial selection for the number-field sieve, part 2: polynomial merit (2005). https://cr.yp.to/talks.
 html#2005.07.19
- Bernstein, D.J.: Fast multiplication and its applications, in [32], pp. 325–384 (2008). https://cr.yp.to/papers.html#
 multapps
- 2160 17. Bernstein, D.J.: S-unit attacks (2021). https://cr.yp.to/talks.html#2021.08.20
- 18. Bernstein, D.J.: Abelianfields (software package) (2022). https://s-unit.attacks.cr.yp.to/norms.html#abelianfields
- 19. Bernstein, D.J.: Cyclo2power (software package) (2022). https://s-unit.attacks.cr.yp.to/norms.html#cyclo2power
- Bernstein, D.J., De Feo, L., Leroux, A., Smith, B.: Faster computation of isogenies of large prime degree, in [46], pp.
 39–55 (2020). https://velusort.isogeny.org
- 2165 21. Bernstein, D.J., Heninger, N., Lange, T.: FactHacks: RSAfactorization in the real world (2012). https://facthacks.cr.yp.to
- 2166 22. Bernstein, D.J., Lange, T.: Batch NFS, in SAC 2014 [62], pp. 38–58 (2014). https://cr.yp.to/papers.html#batchnfs
- 2167 23. Bernstein, D.J., Lange, T.: Non-randomness of S-unit lattices (2021). https://cr.yp.to/papers.html#spherical
- Bluestein, L.I.: A linear filtering approach to the computation of the discrete Fourier transform. IEEE Northeast Electron.
 Res. Eng. Meet. **10**, 218–219 (1968)
- Bluestein, L.I.: A linear filtering approach to the computation of discrete Fourier transform. IEEE Trans. Audio Electroacoust. 18, 451–455 (1970)
- 2172 26. Bosma, W.: Canonical bases for cyclotomic fields. Appl. Algebra Eng. Commun. Comput. 1, 125–134 (1990)
- 2173 27. Bostan, A.: Computing the N-th term of a q-holonomic sequence, in [42], pp. 46–53 (2020). https://hal.inria.fr/ 2174 hal-02882885
- Bostan, A., Yurkevich, S.: Fast computation of the N-th term of a q-holonomic sequence and applications (2020); see also older version [27]. https://hal.inria.fr/hal-03084680
- 2177 29. Breuer, T.: Integral bases for subfields of cyclotomic fields. Appl. Algebra Eng. Commun. Comput. **8**, 279–289 (1997)
- Browning, T.D., Matthiesen, L.: Représentations de normes decorps de nombres arbitraires par des produits de polynômeslinéaires, Annales Scientifiques de l'École Normale Supérieure. Quatrième Série 50, 1383–1446 (2017)
- Buhler, J., Lenstra, H.W., Jr., Pomerance, C.: Factoring integers with the number field sieve, in [67], pp. 50–94 (1993).
 http://www.math.leidenuniv.nl/~hwl/PUBLICATIONS/1993e/art.pdf
- Buhler, J.P., Stevenhagen, P. (eds.): Surveys in Algorithmic Number Theory, vol. 44. Mathematical Sciences Research
 Institute Publications, Cambridge University Press, New York (2008). See [16]
- Calmet, J. (ed.): Computer Algebra, EUROCAM '82, European Computer Algebra Conference, Marseille, France, 5–7
 April, 1982, Proceedings, vol. 144. Springer (1982). ISBN 3-540-11607-9. See [84]
- Cohen, H.: A Course in Computational Algebraic Number Theory. Graduate Texts in Mathematics, vol. 138. Springer (1993). ISBN 3-540-55640-0

Cohen, H.: Advanced Topics in Computational Number Theory, Graduate Texts in Mathematics, vol. 193. Springer 2188 35. (2000) ISBN 0-387-98727-4 2189 Collins, G.E.: Sub-resultants and reduced polynomial remainder sequences. J. ACM 14, 128–142 (1967), https://doi 2190 36 ora/10.1145/321371.321381 2191 Collins, G.E.: The calculation of multivariate polynomial resultants. J. ACM 18, 515-532 (1971). https://doi.org/10. 2102 37 1145/321662.321666 2193 Coron, J.-S., Nielsen, J.B. (eds.): Advances Incryptology-EUROCRYPT 2017-36th Annual International Conference 38 2194 on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30-May 4, 2017, Proceedings, Part I 2195 (2017). ISBN 978-3-319-56619-1. See [9] 2196 39 Dandelin, G.P.: Recherches sur la reísolution des eíguationsnumeírigues. Nouveaux mémoires de l'académie royale 2197 des sciences etbelles-lettres de Bruxelles, vol. 3 (1826). https://babel.hathitrust.org/cgi/pt?id=uc1.%24b54345 2198 2199 40. Dirichlet, G.L.: Recherches sur les formes quadratiques àcoëfficients et à indéterminées complexes, Journal fürdie reine und angewandte Mathematik 24, 291-371 (1842) 2200 Ellenberg, J.S., Venkatesh, A.: The number of extensions of a number field with fixed degree and bounded discriminant. 41 2201 Ann. Math. 163, 723–741 (2006) 2202 Emiris, I.Z., Zhi, L. (eds.): ISSAC '20: International Symposium on Symbolic and Algebraic Computation, Kalamata, 42. 2203 2204 Greece, July 20–23, 2020, ACM (2020), ISBN 978-1-4503-7100-1, See [27] Epstein, B.: Some applications of the Mellin transform in statistics. Ann. Math. Stat. 19, 370–379 (1948). https://doi. 2205 43. org/10.1214/aoms/1177730201 2206 Fee, G., Granville, A.: The prime factors of Wendt's binomial circulant determinant. Math. Comput. 57, 839–848 (1991) 2207 44 Fukuda, K., van der Hoeven, J., Joswig, M., Takayama, N. (eds.): Mathematical Software—ICMS 2010, Third International 45. 2208 Congresson Mathematical Software, Kobe, Japan, September 13–17, 2010, Proceedings, vol. 6327. Springer (2010). 2200 ISBN 978-3-642-15581-9. See [55] 2210 Galbraith, S. (ed.): ANTS XIV: Proceedings of the Fourteenth Algorithmic Number Theory Symposium, Auckland 2020. 2211 46. Open Book Series, vol. 4. Mathematical Sciences Publishers (2020). ISBN 978-1-935107-07-1. See [20] 2212 Gao, S., von zur Gathen, J., Panario, D.: Gauss periods and fast exponentiation in finite fields (extended abstract), in [8], 47. 2213 pp. 311-322 (1995) 2214 Friedrich, C.: Gauss, Disquisitiones arithmeticae (1801). https://archive.org/details/disquisitionesa00gaus 48. 2215 2216 49. Friedrich, C.; Gauss, Theoria residuorum biguadraticorum, Commentatiosecunda, in [50], pp. 93–148 (1876), Paper says "Societati Regiae Tradita 1831 Apr 15, Commentationes societatis regiae scientiarum Gottingensisrecentiores, 2217 vol. VII, Gottingae MDCCCXXXII". https://ia902808.us.archive.org/21/items/117771763 002/117771763 002.pdf 2218 2219 50. Friedrich, C.: Gauss, Werke, Band II (1876). https://ia902808.us.archive.org/21/items/117771763_002/117771763_002. pdf. See [49] 2220 Gentry, C., Halevi, S.: Implementing Gentry's fully-homomorphic encryption scheme, in Eurocrypt 2011 [77], 129–148 2221 51. (2010). https://eprint.iacr.org/2010/520 2222 Granlund, T.: The GMP development team, GNU MP version 6.2.1 (2020). https://gmplib.org/ 52 2223 2224 53. Granville, A.: Smooth numbers: computational number theory and beyond, pp. 267-323 (2008). https://dms. umontreal.ca/~andrew/PDF/msrire.pdf 2225 Grenié, L., Giuseppe, M.: Explicit smoothed prime ideals theorems under GRH. Math. Comput. 85, 1875–1899 (2016) 2226 54 Hart, W.B.: Fast library for number theory: an introduction. In ICMS 2010 [45], pp. 88–91 (2010). https://flintlib.org 55. 2227 Harvey, D., van der Hoeven, J.: Integer multiplication in time O(nlog n). Ann. Math. Second Ser. 193, 563–617 (2021) 2228 56. 2229 57. Hecke, E.: Lectures on the Ttheory of Algebraic Numbers, Graduate Texts in Mathematics, vol. 77. Springer (1981) Hilbert, D.: Die Theorie der algebraischen Zahlkörper. Jahresbericht der Deutschen Mathematiker-Vereinigung 4, 58. 2230 2231 175-546 (1897) 59 van der Hoeven, J., Lecerf, G.: Accelerated tower arithmetic. J. Complex. 55, 101402 (2019) 2233 60. Householder, A.S.; Dandelin, Lobačevskii, or Graeffe? Am. Math. Mon. 66, 464–466 (1959) Janusz, G.J.: Algebraic Number Fields, Pure and Applied Mathematics, vol. 55. Academic Press, Cambridge (1973) 2234 Joux, A., Youssef, A.M. (eds.): Selected Areas in Cryptography—SAC 2014—21st International Conference, Montreal, 2235 62. 2236 QC, Canada, August 14-15, 2014, Revised Selected Papers, 8781. Springer (2014). ISBN 978-3-319-13050-7. See [22] Kauers, M. (ed.): Symbolic and Algebraic Computation, International Symposium ISSAC 2005, Beijing, China, July 63. 2237 24-27, 2005, Proceedings. Association for Computing Machinery (2005). ISBN 1-59593-095-7. See [71] 2238 Kim, H., Kim, D.-C. (eds.): Information Security and Cryptology—ICISC 2017—20th International Conference, Seoul, 2239 South Korea, November 29-December 1, 2017, Revised Selected Papers, vol. 10779. Springer (2018). ISBN 978-3-319-2240 78555-4. See [6] 2241 Landau, E.: Neuer Beweis des Primzahlsatzes und Beweis des Primidealsatzes. Mathematische Annalen 56, 645–670 2242 65. 2243 (1903)van Leeuwen, J. (ed.): Algorithms-ESA '94, Second Annual European Symposium, Utrecht, the Netherlands, Septem-66. 2244 ber 26–28, 1994, Proceedings, vol. 855. Springer (1994). ISBN 3-540-58434-X. See [86] 2245 Lenstra, A.K., Lenstra, H.W., Jr. (eds.): The Development of the Number Field Sieve, Lecture Notes in Mathematics, vol. 2246 67. 1554, Springer (1993), ISBN 3-540-57013-6, See [31, 80] 2247 Lenstra, H.W., Jr., Tijdeman, R. (eds.): Computational Methods in Number Theory I, vol. 154. Mathematical Centre Tracts. 2248 68. Mathematisch Centrum, Amsterdam (1982). ISBN 90-6196-248-X. MR 84c:10002. See [81] 2249 69. Marcus, D.A.: Number Fields, 2nd edn. Springer, Berlin (2018) 2250 Miller, J.C.: Class numbers of totally real fields and applications to the Weber class number problem. Acta Arith. 164, 2251 381-397 (2014), arXiv:1405.1094 2252 2253 71. Monagan, M.B.: Probabilistic Algorithms for Computing Resultants. In: ISSAC [63], pp. 245–252 (2005) Montgomery, P.L.: An FFT extension of the elliptic curve method of factorization. Ph.D. Thesis, University of California 72. 2254 at Los Angeles (1992). https://cr.yp.to/bib/1992/montgomery.pdf 2255 2256 73. Moree, P.: Psixyology and diophantine equations. Ph.D. Thesis, Universiteit Leiden (1993)

- 2257 Nicholson, P.J.: Algebraic theory of finite Fourier transforms. J. Comput. Syst. Sci. 5, 524–547 (1971) 74.

Journal: 40993 Article No.: 0402 TYPESET DISK LE CP Disp.:2023/3/4 Pages: 58 Layout: BMC-OneCol

- Norton, K.K.: Numbers with Small Prime Factors, and the Least kth Power Non-residue. Memoirs of the American Mathematical Society. American Mathematical Society (1971)
- 2260 76. The PARI Group, PARI/GP version 2.11.2 (2019). https://pari.math.u-bordeaux.fr
- Paterson, K.G. (ed.): Advances in Cryptology—EUROCRYPT2011—30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15–19, 2011, Proceedings. Springer (2011). ISBN 978-3-642-20464-7. See [51]
- 2264 78. Pollard, J.M.: The fast Fourier transform in a finite field. Math. Comput. 25, 365–374 (1971)
- Pollard, J.M.: Theorems on factorization and primality testing. Math. Proc. Camb. Philos. Soc. **76**, 521–528 (1974).
 https://doi.org/10.1017/S0305004100049252
- 2267 80. Pollard, J.M.: The lattice sieve, in [67], pp. 43–49 (1993)
- Pomerance, C.: Analysis and comparison of some integer factoring algorithms, in [68], pp. 89–139 (1982). MR 84i:10005.
 https://math.dartmouth.edu/~carlp/PDF/analysiscomparison.pdf
- Rader, C.M.: Discrete Fourier transforms when the number of samples is prime. In: Proceedings of the IEEE 56, pp. 1107–1108 (1968). ISSN0018-9219. https://cr.yp.to/bib/entries.html#1968/rader
- The Sage Developers (ed.): SageMath, the Sage Mathematics Software System (Version 9.5), (2022). https://www.
 sagemath.org
- Schönhage, A.: Asymptotically fast algorithms for the numerical multiplication and division of polynomials with complex coefficients. In: EUROCAM '82 [33], pp. 3–15 (1982)
- 85. Schönhage, Arnold: Strassen, Volker: Schnelle Multiplikationgroßer Zahlen. Computing 7, 281–292 (1971)
- Schönhage, A., Vetter, E.: A new approach to resultant computations and other algorithms with exact division. In: ESA
 1994 [66], pp. 448–459 (1994)
- 2279 87. Shoup, V.: NTL version 11.4.3 (2020). https://libntl.org
- 2280 88. Sinnott, W.M.: On the Stickelberger ideal and the circular units of a cyclotomic field. Ann. Math. Second Ser. 108, 107–134 (1978)
- 89. Stevenhagen, P., Lenstra, H.W., Jr.: Chebotarëv and his density theorem. Math. Intell. 18, 26–37 (1996)
- Strassen, V.: Einige resultate über Berechnungskomplexität. Jahresbericht der Deutschen Mathematiker-Vereinigung
 78, 1–8 (1976)
- Strassen, V.: The computational complexity of continued fractions, In: SYM-SAC 1981 [93], pp. 51–67 (1981); see also newer version [92]
- Strassen, V.: The computational complexity of continued fractions. SIAM J. Comput. 12, 1–27 (1983); see also older version [91]. ISSN 0097-5397
- Wang, P.S. (ed.): SYM-SAC '81: Proceedings of the 1981 ACM Symposium on Symbolic and Algebraic Computation, Snowbird, Utah, August 5–7, 1981. Association for Computing Machinery, New York (1981). ISBN0-89791-047-8. See
 [91]
- 94. Washington, L.C.: Introduction to Cyclotomic Fields, 2nd edn., vol. 83 (1997). ISBN 0-387-94762-0
- Wessel, C.: Om Directionens analytiske Betegning, et Forsog, anvendtfornemmelig til plane og sphæriske Polygoners
 Oplosning, Nye Samling afdet Kongelige Danske Videnskabernes Selskabs Skrifter 5, pp. 469–518 (1799). https://
 babel.hathitrust.org/cgi/pt?id=ien.35556000979690&view=1up&seq=561
- 96. Winograd, S.: On computing the discrete Fourier transform. Math. Comput. **32**, 175–199 (1978)
- 2297 97. Zumbroich, M.: Grundlagen einer Arithmetik inKreisteilungskörpern und deren Implementation in CAS. RWTH Aachen
 (1989). Diplomarbeit

2299 Publisher's Note

2300 Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Journal: 40993	
Article: 402	

Author Query Form

Please ensure you fill out your response to the queries raised below and return this form along with your corrections

Dear Author

During the process of typesetting your article, the following queries have arisen. Please check your typeset proof carefully against the queries listed below and mark the necessary changes either directly on the proof/online grid or in the 'Author's response' area provided below

Query	Details required	Author's response
1.	Please confirm if the inserted city names in all the affiliations are correct. Amend if necessary.	
2.	As keywords are mandatory for this chapter, please provide 3-6 keywords.	
3.	Kindly check and confirm whether the Mathematics Subject Classification has been processed correctly.	