The Journal of Open Source Software

Aerobus: a C++ template library for polynomials algebra over discrete Euclidean domains

Regis Portalez [1]

1 COMUA, France

DOI: 10.21105/joss.06233

Software

- Review
- Repository
- Archive

Editor: George K. Thiruvathukal

Reviewers:

- @mmoelle1
- @lucaferranti
- @pitsianis

Submitted: 12 October 2023
Published: 07 January 2025

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

Summary

C++ comes with high compile-time computations capability, also known as metaprogramming with templates. Templates are a language-in-the-language which is Turing-complete, meaning we can run every computation at compile time instead of runtime, as long as input data is known at compile time.

Using these capabilities, vastly extended with the latest versions of the standard, we implemented a library for discrete Euclidean domains (commutative and associative), such as $\mathbb{Z}$. We also provide a way to generate the fraction field of such rings (e.g. $\mathbb{Q}$).

We also implemented polynomials over such discrete rings and fields (e.g. $\mathbb{Q}[X]$). Since polynomials are also a ring, the above implementation gives us rational fractions as the field of fractions of polynomials.

In addition, we expose a way to generate Taylor series of analytic functions, known polynomials (e.g. Chebyshev), continued fractions, quotient rings and small degree Conway polynomials to define Galois finite fields.

`Aerobus` was designed to be used in high-performance software, teaching purposes or embedded software. It compiles with major compilers: `gcc`, `clang` and `msvc`. It is quite easily configurable and extensible.

Statement of need

By implementing general algebra concepts such as discrete rings, field of fractions and polynomials, `Aerobus` can serve multiple purposes, mainly polynomial arithmetic at compile time and efficient (Comua, 2024) polynomial evaluation, regardless of the coefficients `ring`.

The main application we want to express in this paper is the automatic (and configurable) generation or Taylor approximation of analytic functions such as `exp` or `sin`, by using polynomial arithmetic at compile time.

Some important software, such as `geographiclib` (Karney, 2013) evaluate polynomials with a simple loop, expected to be unrolled by compiler. It works really well (on arithmetic types such as `double`) but does not provide a way to manipulate polynomials (addition, multiplication, division, modulus) automatically where `aerobus` does at no runtime cost.

Notable libraries such as boost provide polynomial arithmetic, but arithmetic is done at runtime with memory allocations while `aerobus` does it at compile time.

Common analytic functions are usually exposed by the standard library (`<cmath>`) with high (guaranteed) precision. However, in high-performance computing, when not compiled with `-Ofast`, evaluating `std::exp` has several flaws:

- It leads to a `syscall` which is very expensive;
- It doesn't leverage vector units (AVX, AVX2, AVX512 or equivalent in non-intel hardware);

- Results are hardware dependent.

Hardware vendors provide high-performance libraries such as (Wang et al., 2014), but implementation is often hidden and not extensible.

Some others can provide vectorized functions, such as (Wang et al., 2014) does. But libraries like VML are highly tight to one architecture by their use of intrinsic or inline assembly. In addition, they only provide a restricted list of math functions and do not expose capabilities to generate high-performance versions of other functions such as `arctanh`. It is the same for the standard library compiled with `-Ofast`: it generates a vectorized version of some functions (such as `exp`) but with no control of precision and no extensibility. In addition, `fast-math` versions are compiler and architecture dependent, which can be a problem for results reproducibility.

`Aerobus` provides automatic generation of such functions, in a hardware-independent way, as tested on x86 and CUDA platforms. In addition, `Aerobus` provides a way to control the precision of the generated function by changing the degree of Taylor expansion, which can't be used in competing libraries without reimplementing the whole function or changing the array of coefficients.

`Aerobus` does not provide optimal approximation polynomials the way (**?**) does. However, `Sollya` could be used beforehand to feed `aerobus` with appropriate coefficients. `Aerobus` does not provide floating point manipulations (domain normalization) to extend domain of approximation, like it is done in standard library.

Mathematic definitions

For the sake of completeness, we give basic definitions of the mathematical concepts which the library deals with. However, readers desiring complete and rigorous definitions of the concepts explained below should refer to a mathematical book on algebra, such as (Lang, 2012) or (Bourbaki, 2013).

A `ring` $\mathbb{A}$ is a nonempty set with two internal laws, addition and multiplication. There is a neutral element for both, zero and one. Addition is commutative and associative and every element x has an inverse $-x$. Multiplication is commutative, associative and distributive over addition, meaning that $a(b+c) = ab + ac$ for every a, b, c element. We call it `discrete` if it is countable.

In a `field`, in addition to previous properties, each element (except zero), has an inverse for multiplication.

A `integral domain` is a ring with one additional property. For every element a, b, c such as $ab = ac$, then either $a = 0$ or $b = c$. Such a ring is not always a field, such as $\mathbb{Z}$ shows it.

A `euclidean domain` is an integral domain that can be endowed with a euclidean division.

For such a euclidean domain, we can build two important structures:

Polynomials $\mathbb{A}[X]$

Polynomials over $\mathbb{A}$ is the free module generated by a base noted $(X^k)_{k\in\mathbb{N}}$. Practically speaking, it's the set of :

$$a_0 + a_1 X + ... + a_n X^n$$

where $a_n \neq 0$ if $n \neq 0$.

(a_i), the coefficients, are elements of $\mathbb{A}$. The theory states that if $\mathbb{A}$ is a field, then $\mathbb{A}[X]$ is Euclidean. That means notions like division of greatest common divisor (gcd) have a meaning, yielding an arithmetic of polynomials.

Field of fractions

If $\mathbb{A}$ is Euclidean, we can build its field of fractions: the smallest field containing $\mathbb{A}$. We construct it as congruences classes of $\mathbb{A} \times \mathbb{A}$ for the relation $(p, q) \sim (pp, qq)$ iff $p * qq = q * pp$. Basic algebra shows that this is a field (every element has an inverse). The canonical example is $\mathbb{Q}$, the set of rational numbers.

Given polynomials over a field form an Euclidean ring, we can do the same construction and get rational fractions $P(x)/Q(X)$ where P and Q are polynomials.

Quotient rings

In an Euclidean domain $\mathbb{A}$, such as $\mathbb{Z}$ or $\mathbb{A}[X]$, we can define the quotient ring of $\mathbb{A}$ by a principal ideal I. Given that I is principal, it is generated by an element X and the quotient ring is the ring of rests modulo X. When X is `prime` (meaning it has no smallest factors in $\mathbb{A}$), the quotient ring $\mathbb{A}/I$ is a field.

Applied on $\mathbb{Z}$, that operation gives us modular arithmetic and all finite fields of cardinal q where q is a prime number (up to isomorphism). These fields are usually named $\mathbb{Z}/p\mathbb{Z}$. Applied on $\mathbb{Z}/p\mathbb{Z}[X]$, it gives finite Galois fields, meaning all finite fields of cardinal p^n where p is prime (see (Évariste, 1846)).

Software

All types of Aerobus have the same structure.

An englobing type describes an algebraic structure. It has a nested type `val` which is always a template model describing elements of the set.

This is because we want to operate on types more than on values. This allows generic implementation, for example of `gcd` (see below) without specifying what are the values.

Concepts

The library exposes three main `concepts`:

```
template <typename R>
concept IsRing; // see in code or documentation

template <typename R>
concept IsEuclideanDomain; // see in code or documentation

template<typename R>
concept IsField; // see in code or documentation
```

which express the algebraic objects described above. Then, as long as a type satisfies the `IsEuclideanDomain` concept, we can calculate the greatest common divisor of two values of this type using Euclid's algorithm (Heath & others, 1956). As stated above, this algorithm operates on types instead of values and does not depend on the Ring, making it possible for users to implement another kind of discrete Euclidean domain without worrying about that kind of algorithm.

The same is done for the field of fractions: implementation does not rely on the nature of the underlying Euclidean domain but rather on its structure. It's automatically done by templates, as long as Ring satisfies the appropriate concept.

Doing that way, $\mathbb{Q}$ has the same implementation as rational fractions of polynomials. Users could also get the field of fractions of any ring of their convenience, as long as they implement the required concepts.

Native types

Aerobus exposes several pre-implemented types, as they are common and necessary to do actual computations:

- `i32` and `i64` ($\mathbb{Z}$ seen as 32 bits or 64 bits integers)
- `zpz` the quotient ring $\mathbb{Z}/p\mathbb{Z}$
- `polynomial<T>` where T is a ring
- `FractionField<T>` where T is an Euclidean domain

Polynomial exposes an evaluation function, which automatically generates Horner development and unrolls the loop by generating it at compile time. See (Horner, 1815) or (Knuth, 2014) for further developments of this method.

Given a polynomial

$$P = \sum_{i=0}^{i\leq n} a_i X^i = a_0 + a_1 X + ... + a_n X^n$$

we can evaluate it by rewriting it this way:

$$P(x) = a_0 + X(a_1 + X(a_2 + X(... + X(a_{n-1} + Xan))))$$

which is automaticcaly done by `aerobus` using the `polynomial::val::eval` function.

This evaluation function is `constexpr` and therefore will be completely computed at compile time when called on a constant.

Polynomials also expose Compensated Horner scheme like in (Graillat et al., 2006), to gain extra precision when evaluating polynomials close to its roots.

The library also provides built-in integers and functions, such as Bernouilli numbers, factorials or other utilities.

Some well known Taylor series, such as `exp` or `acosh` come preimplemented.

The library comes with a type designed to help the users implement other Taylor series. If users provide a type `mycoeff` satisfying the appropriate template (depending on the `Ring` of coefficients and degree), the corresponding Taylor expansion can be built automatically as a polynomial over this `Ring` and then, evaluated at some value in a native arithmetic type (such as `double`).

Misc

Continued Fractions

Aerobus provides continued fractions, seen as an example of what is possible when you have a proper type representation of the field of fractions. One can get a rational approximation of numbers using their known representation, given by the On-Line Encyclopedia of Integer Sequences (*On-Line Encyclopedia of Integer Sequences*, n.d.). Some useful math constants, such as π or e are provided preimplemented, from which user can have the corresponding rational number by using (for example) `PI_fraction::type` and a computation with `PI_fraction::val`.

Known polynomials

There existe many orthogonal polynomial bases used in various domains, from number theory to quantum physics. Aerobus provide predefined implementation for some of them (Laguerre, Hermite, Bernstein, Bessel, ...). These polynomials have integers coefficients by default, but can be defined (specialized) with coefficients in any `Ring` (such as $\mathbb{Z}/2\mathbb{Z}$).

Quotient rings and Galois fields

If some type meets the `IsRing` concept requirement, Aerobus can generate its quotient ring by a principal ideal generated by some element `X`.

We can then define finite fields such as $\mathbb{Z}/p\mathbb{Z}$ by writing `using Z2Z = Quotient<i32, i32::inject_constant_t<2>>;`.

In $\mathbb{Z}/p\mathbb{Z}[X]$, there are special irreducible polynomials named Conway polynomials (Holt et al., 2005), used to build larger finite fields. `Aerobus` exposes Conway polynomials for p smaller than 1000 and degrees smaller than 20.

To speed up compilation for users who don't use them, they are hidden behing the flag `AEROBUS_CONWAY_IMPORTS`. If this is defined, it's possible define $\mathrm{GF}(p, n) = \mathbb{F}_{p^n}$.

For instance, we can compute $\mathbb{F}_4 = \mathrm{GF}(2, 2)$ by writing:

```
using F2 = zpz<2>;
using PF2 = polynomial<F2>;
using F4 = Quotient<PF2, ConwayPolynomial<2, 2>::type>;
```

Multiplication and addition tables are checked to be those of $\mathbb{F}_4$.

Surprisingly, compilation time is not significantly higher when we include `conwaypolynomials.h`. However, we chose to make it optional.

Acknowledgments

Many thanks to my math teachers, A. Soyeur and M. Gonnord. I also acknowledge indirect contributions from F. Duguet, who showed me the way. I wish also to thank Miss Chloé Gence, who gave me the name of the library.

Reference

Bourbaki, N. (2013). *Algebra II: Chapters 4-7*. Springer Science & Business Media. https://doi.org/10.1007/978-3-642-61698-3

Comua. (2024). *Aerobus benchmarks* (Version v1). Figshare. https://figshare.com/articles/dataset/Aerobus_benchmarks/27222777?file=49774845

Évariste, G. (1846). Mémoire sur les conditions de résolubilité des équations par radicaux. *Journal de Mathématiques Pures Et Appliquées, Ser*, *1*(111846), 417–433.

Graillat, S., Langlois, P., & Louvet, N. (2006). Compensated horner scheme. *Algebraic and Numerical Algorithms and*.

Heath, T. L., & others. (1956). *The thirteen books of euclid's elements*. Courier Corporation.

Holt, D. F., Eick, B., & O'Brien, E. A. (2005). *Handbook of computational group theory*. CRC Press. https://doi.org/10.1201/9781420035216

Horner, W. (1815). A new method of solving numerical equations of all orders, by continuous approximation. *Abstracts of the Papers Printed in the Philosophical Transactions of the Royal Society of London*, *2*, 117–117.

Karney, C. F. (2013). Algorithms for geodesics. *Journal of Geodesy*, *87*, 43–55. https://doi.org/10.1007/s00190-012-0578-z

Knuth, D. E. (2014). *Art of computer programming, volume 2: Seminumerical algorithms*. Addison-Wesley Professional. https://doi.org/10.1137/1012065

Lang, S. (2012). *Algebra* (Vol. 211). Springer Science & Business Media. https://doi.org/10.1007/978-1-4613-0041-0

On-line encyclopedia of integer sequences. (n.d.). https://oeis.org/.

Wang, E., Zhang, Q., Shen, B., Zhang, G., Lu, X., Wu, Q., & Wang, Y. (2014). Intel math kernel library. In *High-performance computing on the intel xeon phi: How to fully exploit MIC architectures* (pp. 167–188). Springer International Publishing. https://doi.org/10.1007/978-3-319-06486-4_7