Theta functions in FLINT

Jean Kieffer

FLINT workshop, Palaiseau, January 30, 2025

History and motivation

- In 2022, FLINT/Arb already had `acb_modular.h`. Numerical evaluation of Jacobi theta functions, and related computations with elliptic curves and (classical) modular forms.

History and motivation

- In 2022, FLINT/Arb already had `acb_modular.h`. Numerical evaluation of Jacobi theta functions, and related computations with elliptic curves and (classical) modular forms.
- I was interested in computations with higher-dimensional abelian varieties and (Siegel) modular forms. This requires evaluating the Riemann theta functions in any dimension g, a generalization of Jacobi's θ ($g = 1$), usually much more expensive to manipulate.

History and motivation

- In 2022, FLINT/Arb already had `acb_modular.h`. Numerical evaluation of Jacobi theta functions, and related computations with elliptic curves and (classical) modular forms.
- I was interested in computations with higher-dimensional abelian varieties and (Siegel) modular forms. This requires evaluating the Riemann theta functions in any dimension g, a generalization of Jacobi's θ ($g = 1$), usually much more expensive to manipulate.
- In 2022-2023, we discovered a new algorithm to evaluate Riemann theta functions in quasi-linear time in the required precision in a joint work with Noam D. Elkies. It should also be faster than existing methods in practice, including for $g = 1$.

History and motivation

- In 2022, FLINT/Arb already had `acb_modular.h`. Numerical evaluation of Jacobi theta functions, and related computations with elliptic curves and (classical) modular forms.
- I was interested in computations with higher-dimensional abelian varieties and (Siegel) modular forms. This requires evaluating the Riemann theta functions in any dimension g, a generalization of Jacobi's θ ($g = 1$), usually much more expensive to manipulate.
- In 2022-2023, we discovered a new algorithm to evaluate Riemann theta functions in quasi-linear time in the required precision in a joint work with Noam D. Elkies. It should also be faster than existing methods in practice, including for $g = 1$.
- Implemented in FLINT 3.1 as `acb_theta.h`. Big rewrite in draft PR #2182.

Mathematics

Riemann theta functions

Arguments/parameters:

- $\tau \in \mathcal{H}_g$: a $g \times g$ symmetric complex matrix with $\mathrm{Im}(\tau)$ positive definite. (If $g = 1$, this is just a complex number with $\mathrm{Im}(\tau) > 0$.)
- $z \in \mathbb{C}^g$
- $a, b \in \{0, 1\}^g$: theta characteristics.

$$\theta_{a,b}(z, \tau) = \sum_{n \in \mathbb{Z}^g + \frac{a}{2}} \exp\Big(\pi i n^T \tau n + 2\pi i n^T \big(z + \tfrac{b}{2}\big)\Big).$$

Evaluating Riemann theta functions

Input: τ, z, and a working precision N.

Output: $\theta_{a,b}(z, \tau)$ as complex numbers to precision N for all a, b.

$$\theta_{a,b}(z,\tau) = \sum_{n\in\mathbb{Z}^g+\frac{a}{2}} \exp\Big(\pi i n^T \tau n + 2\pi i n^T\big(z + \tfrac{b}{2}\big)\Big).$$

Rough Algorithm 1: Summation

1. Collect all vectors $n \in \mathbb{Z}^g + \frac{a}{2}$ whose associated exponential term has absolute value $\geq 2^{-N}$. These lie in a certain ellipsoid E, more precisely a ball for the quadratic form $\mathrm{Im}(\tau)$ of radius $\approx \sqrt{N}$.
2. Compute a partial sum of the series defining $\theta_{a,b}$ over this ellipsoid.
3. Add an error bound from the tail of the series.

Complexity is $\widetilde{O}(N \cdot \#E)$, i.e. $\widetilde{O}(N^{1+g/2})$ in general (depends on τ).
This strategy is used and carefully optimized in `acb_modular.h`.

Duplication

This uses duplication formulas, the main one being

$$\theta_{a,b}(z,\tau)^2 = \sum_{a' \in \{0,1\}^g} (-1)^{a'^T b} \theta_{a',0}(0, 2\tau)\theta_{a+a',0}(2z, 2\tau).$$

Rough Algorithm 2: Duplication

1. Compute $\theta_{a,0}(0, 2\tau)$ and $\theta_{a,0}(2z, 2\tau)$ to precision N using an algorithm of your choice.
2. Evaluate $\theta_{a,b}(z,\tau)$ at low precision using the summation algorithm.
3. Use the duplication formula to get $\theta_{a,b}(z,\tau)^2$ to precision N, and extract the correct square root using the low-precision approximation as a guide.

Complexity (apart from step 1) is $\widetilde{O}(N)$, outside of unlucky cases where $\theta_{a,b}(z,\tau)$ is very close to zero (we can deal with those too). We only lose $O_g(1)$ bits of precision.

Reduction

The Siegel modular group $\mathrm{Sp}_{2g}(\mathbb{Z})$ acts on $\mathcal{H}_g$ (and more generally on $\mathbb{C}^g \times \mathcal{H}_g$), much as the classical modular group $\mathrm{SL}_2(\mathbb{Z})$ acts on $\mathcal{H}_1$.

Given $\tau \in \mathcal{H}_g$, one can always find $\gamma \in \mathrm{Sp}_{2g}(\mathbb{Z})$ such that $\gamma\tau$ is reduced, in particular:

- $\mathrm{Im}(\tau)$ is LLL-reduced, (HKZ would be even better),
- $\mathrm{Im}(\tau_{1,1}) \geq \sqrt{3}/2$.

Rough algorithm 3: Reduction

1. Compute γ. Let $(z', \tau') = \gamma(z, \tau)$.
2. Further reduce z' modulo the lattice $\mathbb{Z}^g + \tau'\mathbb{Z}^g$ to obtain z''.
3. Compute $\theta_{a,b}(z'', \tau')$ to precision N using an algorithm of your choice.
4. Apply the theta transformation formulas to recover $\theta_{a,b}(z, \tau)$.

The cost of reduction is negligible in practice.

Assembling the quasi-linear algorithm

Input: τ, z, and a working precision N.

Output: $\theta_{a,b}(z,\tau)$ as complex numbers to precision N for all a, b.

The quasi-linear algorithm

1. Start applying **Reduction** to obtain a reduced pair (z'', τ').
2. Choose an integer m such that $2^m \operatorname{Im}(\tau'_{1,1}) \approx N$. We have $m = O(\log N)$.
3. Compute $\theta_{a,0}(0, 2^m\tau')$ and $\theta_{a,0}(2^m z'', 2^m\tau')$ using **Summation**. The ellipsoids we compute contain $O(1)$ points, so this costs $\widetilde{O}(N)$.
4. Apply **Duplication** m times to get $\theta_{a,b}(z'', \tau')$. This costs $\widetilde{O}(N)$ too.
5. Finish applying **Reduction** to get $\theta_{a,b}(z, \tau)$.

Dimension-lowering (1)

The previous algorithm is inefficient on reduced matrices $\tau \in \mathcal{H}_g$ whose imaginary part is skewed, such as

$$\mathrm{Im}(\tau) = \begin{pmatrix} 1 & 0 \\ 0 & 100 \end{pmatrix}.$$

In that case, after just a few duplication steps, the ellipsoids containing the points $n = (n_1, n_2) \in \mathbb{Z}^g + \frac{a}{2}$ we would consider in a partial sum become very thin in the direction of n_2.

Can we leverage this?

Dimension-lowering (2)

In that case, when writing

$$\tau = \begin{pmatrix} \tau_1 & x \\ x & \tau_2 \end{pmatrix},$$

we have

$$\theta_{a,b}(z,\tau) = \sum_{n_2 \in \mathbb{Z} + \frac{a_2}{2}} e^{\pi i(\cdots)} \theta_{a_1,b_1}(z_1 + x n_2, \tau_1).$$

In order to get $\theta_{a,b}(z,\tau)$ to precision N, we only need very few values of n_2: we reduced our evaluation in dimension 2 to $O(1)$ evaluations of theta functions in dimension 1.

Depending on the shape of τ, applying this dimension-lowering strategy at well-chosen spots between duplication steps can be very beneficial.

Implementation in FLINT 3.1

Key features (1)

- Manipulate matrices in $\mathrm{Sp}_{2g}(\mathbb{Z})$ (type `fmpz_mat_t`).
- Manipulate elements in $\mathbb{C}^g \times \mathcal{H}_g$. The reduction algorithm is implemented as:

```
void acb_siegel_reduce(fmpz_mat_t mat, const acb_mat_t tau,
                       slong prec)
```

- Manipulate (integer points in) ellipsoids defined by positive-definite quadratic forms. We introduce a type `acb_theta_eld_t`, and construct ellipsoids with

```
int acb_theta_eld_set(acb_theta_eld_t E, const arb_mat_t C,
                      const arf_t R2, arb_srcptr v)
```

  where `C` is the upper-triangular Cholesky matrix, `R2` is the squared radius, and `v` is the center in $\mathbb{R}^g$.
  This `acb_theta_eld_t` structure doesn't contain all the points (but we can ask for them), and is directly input to the summation methods.

Key features (2)

- Run the summation algorithms. For instance:

```
void acb_theta_naive_all(acb_ptr th, acb_srcptr zs, slong nb,
                         const acb_mat_t tau, slong prec)
```

  I implemented most optimizations I could think of (exponential terms are computed by multiplications rather than exponentiations, the precision varies for each term, etc.)

- Run the whole quasi-linear algorithm (with reduction and dimension-lowering):

```
void acb_theta_all(acb_ptr th, acb_srcptr z,
                   const acb_mat_t tau, int sqr, slong prec)
```

  One can somewhat tune how many duplication steps are performed, and when dimension-lowering is applied, by modifying `acb_theta_ql_a0_nb_steps`.

Key features (3)

- Also compute derivatives of Riemann theta functions, either by direct summation or from finite differences on the output of `acb_theta_all`:

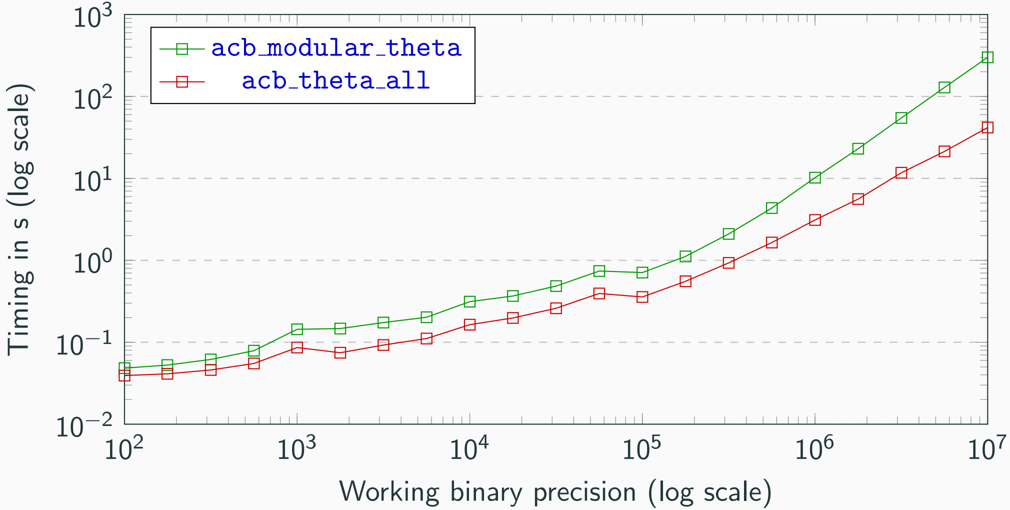
```
void acb_theta_jet_naive_all(acb_ptr dth, acb_srcptr z,
            const acb_mat_t tau, slong ord, slong prec)
void acb_theta_jet_all(acb_ptr dth, acb_srcptr z,
            const acb_mat_t tau, slong ord, slong prec)
```

- Evaluate Siegel modular forms for $g = 2$ at a given point $\tau \in \mathcal{H}_g$ by writing them in terms of theta functions, in the spirit of `acb_modular_delta`: e.g.

```
void acb_theta_g2_chi10(acb_t res, acb_srcptr th2, slong prec)
```

Performance comparison

Time to evaluate $\theta_{a,b}(z,\tau)$ with $z = 0.1 + 0.2i$ and $\tau = 0.3 + 0.8i$:



Proposed changes in PR #2182

Context structures in summation algorithms

We introduce context structures attached to $\tau \in \mathcal{H}_g$ and $z \in \mathbb{C}^g$ in summation algorithms, of type acb_theta_ctx_tau_t and acb_theta_ctx_z_t respectively.

```
void acb_theta_ctx_tau_set(acb_theta_ctx_tau_t ctx,
                           const acb_mat_t tau, slong prec)
```

They store things like $\exp(\pi i \tau_{1,1})$, etc. that would otherwise get recomputed at each call to summation algorithms. We can also duplicate $\tau \mapsto 2\tau$ directly (using squarings):

```
void acb_theta_ctx_tau_dupl(acb_theta_ctx_tau_t ctx, slong prec)
```

This removes some overhead when computing the required low-precision approximations in the duplication formula.

The signatures (and names) of summation functions have changed.

Supporting several vectors z in the quasi-linear algorithm

`acb_theta_all` and similar functions have a different signature to allow for several values of z:

```
void acb_theta_all(acb_ptr th, acb_srcptr zs, slong nb,
                   const acb_mat_t tau, int sqr, slong prec);
```

Recall that using the duplication formulas at any z requires computing theta values at $z = 0$ anyway. We now mutualize them.

This also greatly improves the efficiency of dimension-lowering, which almost always leads to several evaluations for the same matrix τ.

Better control on the algorithm structure

The new function

```
int acb_theta_ql_nb_steps(slong * pattern, const acb_mat_t tau,
                          int cst, slong prec)
```

completely determines how many duplication steps will be applied, and when to use the dimension-lowering strategy, through the whole algorithm.

The output `pattern` is then used as input to the function running the quasi-linear algorithm (`acb_theta_ql_exact`). I spent most of this week profiling that function with varying patterns to see what the best choices are depending on the shape of τ.

Further changes

- Introduce functions like `acb_theta_all_notransform` in the spirit of `acb_modular_theta_notransform`.
- Simplify the management of error bounds by assuming that some internal functions always get exact input.
- Introduce functions `acb_theta_ql_lower_dim` and `acb_theta_ql_recombine` to implement the dimension-lowering strategy that we test independently.
- In the $g = 1$ summation functions, rely on `acb_modular_theta_sum` instead of `acb_modular_theta`. This is to allow `acb_modular_theta` to possibly point to `acb_theta_all` in the future.

To Do

- Make sure there are no regressions compared to the previous version. As of now it seems there are: apparently `acb_modular_theta` got slower (?) for $g = 1$, and we sometimes get NaN results.
- Use quasi-linear algorithms in some functions that don't use them yet (e.g. `acb_theta_00`)