ModularFormsModuloTwo.jl

Paul Dubois

February 22, 2020

Contents

Part I

Standard operations

Chapter 1

General

First, we define types and a useful display function.

`Main.ModularFormsModuloTwo.ModularForm` – Type.

> We can represent a modular forms mod 2 by it's coefficients as a polynomial in q or Δ. The routines in this file are made for q-series. Modular forms modulo 2 have coefficients in q-series being 0 most of the times, and 1 otherwise. Thus, we will represent them as sparse 1-dimensional arrays (sparse vectors) of type SparseVector{Int8,Int}.
>
> source

`Main.ModularFormsModuloTwo.ModularFormOrNothingList` – Type.

> Lists of Modular Forms will be useful for storage.
>
> source

`Main.ModularFormsModuloTwo.disp` – Function.

> disp(f[, maxi])
>
> Display details of f, a modular forms mod 2.
>
> Displays what type of data the object id, up to which coefficient is the form known. Then displays the first few coefficients. Coefficients are displayed until maxi (50 by default).
>
> **Example**

```
[f is a modular form mod 2]
julia> disp(f)
```

> source

Chapter 2

Arithmetic

Then, we define the standard arithmetic (arithmetic of modular forms modulo two is more than a trivial implementation).

`Base.:*` – Method.

Compute the multiplication of two modular forms (with mathematical accuracy).

Example

```
[f1 & f2 are modular forms mod 2]
julia> f1*f2
1000-element SparseVector{Int8,Int64} with 86 stored entries:
```

source

`Base.:+` – Method.

Compute the addition of two modular forms (with mathematical accuracy).

Example

```
[f1 & f2 are modular forms mod 2]
julia> f1+f2
1000-element SparseVector{Int8,Int64} with 27 stored entries:
```

source

`Base.:^` – Method.

Compute f^k (with mathematical accuracy).

Example

```
[f is a modular form mod 2]
julia> f^5
1000-element SparseVector{Int8,Int64} with 75 stored entries:
```

source

Main.ModularFormsModuloTwo.sq – Method.

Compute the square of a modular form (with mathematical accuracy).

This is a much more effcient method then computing the square with multiplication. sq(f) is (much) more effcient then f*f, time wise and memory wise.

Example

```
[f is a modular form mod 2]
julia> @time f*f
  0.169466 seconds (37 allocations: 1.127 MiB)

julia> @time sq(f)
```

source

Chapter 3

Equality

It will be usefull to define an equality reation that is lighter than the very strict regular one.

Main.ModularFormsModuloTwo.eq – Method.

Up to maximum coefficient known for both f1 and f2, tell equality.

source

Main.ModularFormsModuloTwo.truncate – Function.

Truncate f to the LENGTH first coefficients with no error.

Example

```
[f is a modular form mod 2]
julia> f
1000-element SparseVector{Int8,Int64} with 16 stored entries:
  [...]

julia> truncate(f, 100)
100-element SparseVector{Int8,Int64} with 5 stored entries:
```

source

Main.ModularFormsModuloTwo.truncate – Function.

Truncate f1 and f2 to LENGTH first coefficients with no error. Truncate to min length of f1 & f2 if LENGTH = -1.

Example

```
[f1 & f2 are modular forms mod 2]
julia> disp(f1)
MF mod 2 (coef to 1000) - 010000000100000000000000010000000000000000000000001...
julia> disp(f2)
MF mod 2 (coef to 100) - 010000000100000000000000010000000000000000000000001...
```

```
julia> f1, f2 = truncate(f1,f2)

julia> disp(f1)
MF mod 2 (coef to 100) - 01000000010000000000000001000000000000000000000001...
julia> disp(f2)
```

source

Chapter 4

Generators

Most of the time, the modular forms modulo two are standards, and we will use generators to create them.

`Main.ModularFormsModuloTwo.Delta_k` – Function.

Create the standard Δ form, with coefficients up to LENGTH => as a Δ-series!

Example

```
julia> disp(Delta(1))
```

source

`Main.ModularFormsModuloTwo.delta` – Function.

Create the standard Δ form, with coefficients up to LENGTH => as a Δ-series!

Example

```
julia> disp(delta())
MF mod 2 (coef to 1000) - 01000000010000000000000001000000000000000000000001...

julia> disp(delta(10^6))
```

source

`Main.ModularFormsModuloTwo.delta_k` – Function.

Create the standard Δ^k form, with coefficients up to LENGTH => as a q-series!

Example

```
julia> disp(delta_k(0))
MF mod 2 (coef to 1000) - 10000000000000000000000000000000000000000000000000...

julia> disp(delta_k(1))
MF mod 2 (coef to 1000) - 01000000010000000000000001000000000000000000000001...
```

```
julia> disp(delta_k(2))
MF mod 2 (coef to 1000) - 00100000000000000001000000000000000000000000000000...

julia> disp(delta_k(3))
MF mod 2 (coef to 1000) - 00010000000100000001000000000000000000000001000000...

julia> disp(delta_k(5))
```

source

Main.ModularFormsModuloTwo.one – Function.

Create a one form of length LENGTH

Example

```
julia> one()
1000-element SparseVector{Int8,Int64} with 1 stored entry:
  [1   ]  =  1

julia> one(1)
1-element SparseVector{Int8,Int64} with 1 stored entry:
```

source

Main.ModularFormsModuloTwo.zero – Function.

Create a zero form of length LENGTH

Example

```
julia> zero()
1000-element SparseVector{Int8,Int64} with 0 stored entries

julia> zero(1)
```

source

Part II

Advanced Operations

Chapter 5

Hecke Operators

Hecke operators represent the hear of the study of modular forms modulo two.

`Main.ModularFormsModuloTwo.Hecke` – Method.

Compute Tp|f (with mathematical accuracy).

Example

```
julia> d=delta()
julia> disp(d)
MF mod 2 (coef to 1000) - 010000000100000000000000010000000000000000000000001...

julia> disp(Hecke(2, d))
MF mod 2 (coef to 500) - 001000000000000000100000000000000000000000000000000...

julia> disp(Hecke(3, d))
```

source

Chapter 6

Recognizer

The functions defined in this section allow he user to switch between the two representations of modular forms modulo two: (capped) infinite q-series and finite Δ-series.

`Main.ModularFormsModuloTwo.drop_error` – Function.

Drops the numerical error that f might have (as long as this error isn't too large).

Example

```
julia> precalculated = loadFormListBinary(10^2, 10^6)
julia> f = delta(10^6) + delta_k(3, 10^6)
julia> disp(f)
MF mod 2 (coef to 1000000) - 01010000010100000001000001000000000000000001000001...

julia> T11f = MFmod2.Hecke(11, f)
julia> disp(T11f)
MF mod 2 (coef to 90909) - 01000000010000000000000001000000000000000000000001...

julia> T11f_exact = drop_error(T11f, precalculated)
julia> disp(T11f_exact)
MF mod 2 (coef to 1000000) - 01000000010000000000000001000000000000000000000001...
```

source

`Main.ModularFormsModuloTwo.to_q` – Function.

Compute the q-series represenstaion of f (using precalculated).

Example

```
julia> precalculated = loadFormListBinary(10^2, 10^6)
julia> df = Delta_k(5)
julia> disp(df)
MF mod 2 (coef to 100) - 00000100000000000000000000000000000000000000000000...

julia> f = to_q(df, precalculated)
julia> disp(f)
MF mod 2 (coef to 1000000) - 00000100000001000000000000000100000001000000010000...
```

source

Main.ModularFormsModuloTwo.to_Δ – Function.

```
to_Δ(f, precalculated)
-- or --
```

Compute the Δ-series represenstaion of f (using precalculated).

Example

```
julia> precalculated = loadFormListBinary(10^2, 10^6)
julia> f = delta(10^6) + delta_k(3, 10^6)
julia> disp(f)
MF mod 2 (coef to 1000000) - 010100000101000000001000001000000000000000001000001...

julia> df = to_delta(f, precalculated)
100-element SparseArrays.SparseVector{Int8,Int64} with 2 stored entries:
  [2  ]  =  1
  [4  ]  =  1

julia> disp(df)
```

source

Part III

Precalculated Data Storage

Chapter 7

Raw Data

Tables of data can be found here:

- Table of primes Hecke operators on modular forms modulo two.

If this is not enoght, remeber that this module allows anyone to generate new data.

Chapter 8

Use of Precalculated Variables

Here is how to use the variables:

- $T_p|\Delta^k$ is `Hecke_primes[p][k+1]`
- $T_3^i T_5^j|\Delta^k$ is `Hecke_powers[i+1,j+1][k+1]`

These are stored as Δ-series. The q-series of powers of Δ are stored as follows:

- Δ^k is `precalculated[k+1]`

Chapter 9

Generating Precalculated Data

The various precalculated data generators may be found in the data subfolder. Note that the user may create new data files, if the provided ones aren't enought. Note as well that there are two implemented ways to store data, we advice the binary for speed purposes.

Chapter 10

Binary Data Reads

`Main.ModularFormsModuloTwo.loadFormListBinary` – Method.

> Loads the list of q-coefficients of Δ powers form file.
>
> source

`Main.ModularFormsModuloTwo.loadHeckePowersListBinary` – Method.

> Loads the list of Δ-coefficients of powers of Hecke operators applied to powers of Δ.
>
> source

`Main.ModularFormsModuloTwo.loadHeckePrimesListBinary` – Method.

> Loads the list of Δ-coefficients of prime Hecke operators applied to powers of Δ.
>
> source

Chapter 11

Text Data Reads

`Main.ModularFormsModuloTwo.loadForm` – Function.

Load the modular form modulo 2 name from file_name.

source

`Main.ModularFormsModuloTwo.loadFormList` – Function.

Load the modular form list from file_name.

source

`Main.ModularFormsModuloTwo.saveForm` – Function.

Save the modular form modulo 2 f as name in file_name.

source

`Main.ModularFormsModuloTwo.saveFormList` – Function.

Save the modular forms modulo 2 f as name in file_name.

source