

# Modular forms in Sage

Martin Raum  
Max Planck Institute for Mathematics  
Bonn, Germany

2011-03-24

- Sage had good support for modular forms right from the beginning.
- But it is sometimes surprisingly slow. (This was one project for these Sage days)
- Most users wanted modular forms for elliptic curves etc.
- So multiplication was implemented much later.

All this is only true for integral weight forms

- Sage had good support for modular forms right from the beginning.
- But it is sometimes surprisingly slow. (This was one project for these Sage days)
- Most users wanted modular forms for elliptic curves etc.
- So multiplication was implemented much later.

All this is only true for integral weight forms

- Sage had good support for modular forms right from the beginning.
- But it is sometimes surprisingly slow. (This was one project for these Sage days)
- Most users wanted modular forms for elliptic curves etc.
- So multiplication was implemented much later.

All this is only true for integral weight forms

- Sage had good support for modular forms right from the beginning.
- But it is sometimes surprisingly slow. (This was one project for these Sage days)
- Most users wanted modular forms for elliptic curves etc.
- So multiplication was implemented much later.

All this is only true for integral weight forms

- Sage had good support for modular forms right from the beginning.
- But it is sometimes surprisingly slow. (This was one project for these Sage days)
- Most users wanted modular forms for elliptic curves etc.
- So multiplication was implemented much later.

**All this is only true for integral weight forms**

Magma has half-integral weight forms implemented:

```
ModularForms(Gamma0(4), 1/2);
```

So does Sage, but they are not as conveniently implemented as integral weight forms are:

```
half_integral_weight_modform_basis(  
    DirichletGroup(16)(1), 7, 10)
```

And there is no support for weight  $\frac{1}{2}$ .

Magma has half-integral weight forms implemented:

```
ModularForms(Gamma0(4), 1/2);
```

So does Sage, but they are not as conveniently implemented as integral weight forms are:

```
half_integral_weight_modform_basis(  
    DirichletGroup(16)(1), 7, 10)
```

And there is no support for weight  $\frac{1}{2}$ .

Magma has half-integral weight forms implemented:

```
ModularForms(Gamma0(4), 1/2);
```

So does Sage, but they are not as conveniently implemented as integral weight forms are:

```
half_integral_weight_modform_basis(  
    DirichletGroup(16)(1), 7, 10)
```

And there is no support for weight  $\frac{1}{2}$ .

- Magma v2.8 has Hilbert modular forms for general weight and general fields (Dembel , Voight),
- In PSage we've got the fastest implementation ever of parallel weight over  $\mathbb{Q}(\sqrt{5})$ .
- Very recently, support for modular forms for  $\Gamma \subseteq \mathrm{SL}_2(\mathcal{O})$  with  $\mathcal{O} \subseteq \mathbb{Q}(\sqrt{D})$ ,  $D < 0$  has been provided.

- Magma v2.8 has Hilbert modular forms for general weight and general fields (Dembel , Voight),
- In PSage we've got the fastest implementation ever of parallel weight over  $\mathbb{Q}(\sqrt{5})$ .
- Very recently, support for modular forms for  $\Gamma \subseteq \mathrm{SL}_2(\mathcal{O})$  with  $\mathcal{O} \subseteq \mathbb{Q}(\sqrt{D})$ ,  $D < 0$  has been provided.

- Magma v2.8 has Hilbert modular forms for general weight and general fields (Dembel , Voight),
- In PSage we've got the fastest implementation ever of parallel weight over  $\mathbb{Q}(\sqrt{5})$ .
- Very recently, support for modular forms for  $\Gamma \subseteq \mathrm{SL}_2(\mathcal{O})$  with  $\mathcal{O} \subseteq \mathbb{Q}(\sqrt{D})$ ,  $D < 0$  has been provided.

- Magma v2.8 has Hilbert modular forms for general weight and general fields (Dembel , Voight),
- In PSage we've got the fastest implementation ever of parallel weight over  $\mathbb{Q}(\sqrt{5})$ .
- Very recently, support for modular forms for  $\Gamma \subseteq \mathrm{SL}_2(\mathcal{O})$  with  $\mathcal{O} \subseteq \mathbb{Q}(\sqrt{D})$ ,  $D < 0$  has been provided... **in Magma v2.15.**

- The implementation of the latter uses Sharblies, that Skoruppa intended to use for Jacobi forms.
- But he has recently told me that he probably won't have time for this :-).

- The implementation of the latter uses Sharblies, that Skoruppa intended to use for Jacobi forms.
- But he has recently told me that he probably won't have time for this :-).

## So what are our strong sides?

Trac # 8701: implement scalar-valued Siegel modular forms on  $\mathrm{Sp}(4, \mathbb{Z})$

- This is an implementation of Siegel modular forms with Hecke operators,
- that is provided by Skoruppa, Ryan, Ghitza, Tornara and me. (started by Skoruppa)
- Also vector-valued forms are available.

## So what are our strong sides?

**Trac # 8701:** implement scalar-valued Siegel modular forms on  $\mathrm{Sp}(4, \mathbb{Z})$

- This is an implementation of Siegel modular forms with Hecke operators,
- that is provided by Skoruppa, Ryan, Ghitza, Tornara and me. (started by Skoruppa)
- Also vector-valued forms are available.

## So what are our strong sides?

**Trac # 8701:** implement scalar-valued Siegel modular forms on  $\mathrm{Sp}(4, \mathbb{Z})$

- This is an implementation of Siegel modular forms with Hecke operators,
- that is provided by Skoruppa, Ryan, Ghitza, Tornara and me. (started by Skoruppa)
- Also vector-valued forms are available.

## So what are our strong sides?

**Trac # 8701:** implement scalar-valued Siegel modular forms on  $\mathrm{Sp}(4, \mathbb{Z})$

- This is an implementation of Siegel modular forms with Hecke operators,
- that is provided by Skoruppa, Ryan, Ghitza, Tornara and me. (started by Skoruppa)
- Also vector-valued forms are available.

There is no theory of modular symbols or Sharblies for rank 2 groups.

- We cannot expect general congruence subgroups to work,
- but many conjectures are unsolves (and interesting) even for the full modular group.

There is no theory of modular symbols or Sharblies for rank 2 groups.

- We cannot expect general congruence subgroups to work,
- but many conjectures are unsolves (and interesting) even for the full modular group.

There is no theory of modular symbols or Sharblies for rank 2 groups.

- We cannot expect general congruence subgroups to work,
- but many conjectures are unsolves (and interesting) even for the full modular group.

# Strategie

Implement as many constructions for Siegel modular forms as possible and provide generators for the spaces that we know.

This comprises

- Maaß lifts,
- theta series,
- Borcherds products and
- Satoh brackets and Rankin-Cohen brackets.

It is a systematic approach, but don't forget the adhoc strategies suggested by Poor and Yuen (I will come back to this at the end).

# Strategie

Implement as many constructions for Siegel modular forms as possible and provide generators for the spaces that we know.

This comprises

- Maaß lifts,
- theta series,
- Borcherds products and
  - Satoh brackets and Rankin-Cohen brackets.

It is a systematic approach, but don't forget the adhoc strategies suggested by Poor and Yuen (I will come back to this at the end).

# Strategie

Implement as many constructions for Siegel modular forms as possible and provide generators for the spaces that we know.

This comprises

- Maaß lifts,
- theta series,
- Borcherds products and
- Satoh brackets and Rankin-Cohen brackets.

It is a systematic approach, but don't forget the adhoc strategies suggested by Poor and Yuen (I will come back to this at the end).

# Strategie

Implement as many constructions for Siegel modular forms as possible and provide generators for the spaces that we know.

This comprises

- Maaß lifts,
- theta series,
- Borcherds products and
- Satoh brackets and Rankin-Cohen brackets.

It is a systematic approach, but don't forget the adhoc strategies suggested by Poor and Yuen (I will come back to this at the end).

## Example

Igusa found

$$\mathbb{Q}[I_4, I_6, I_{10}, I_{12}] = [\mathrm{Sp}_2(\mathbb{Z}), \det^\bullet],$$

and Satoh gives six generators of

$$[\mathrm{Sp}_2(\mathbb{Z}), \det^\bullet \otimes \mathrm{sym}^2]$$

over  $[\mathrm{Sp}_2(\mathbb{Z}), \det^\bullet]$ .

## Example

Igusa found

$$\mathbb{Q}[l_4, l_6, l_{10}, l_{12}] = [\mathrm{Sp}_2(\mathbb{Z}), \det^\bullet],$$

and Satoh gives six generators of

$$[\mathrm{Sp}_2(\mathbb{Z}), \det^\bullet \otimes \mathrm{sym}^2]$$

over  $[\mathrm{Sp}_2(\mathbb{Z}), \det^\bullet]$ .

Indeed, Siegel modular forms should be in Sage **very soon**:  
Ryan, Skoruppa and Tornara are at the MSRI.

There are many more types of modular forms:

- Siegel modular forms of genus  $> 2$ ,
- Jacobi forms (with lattice index),
- paramodular forms,
- orthogonal modular forms
  - hermitian modular forms
  - quaternion modular forms
  - $O(2, 5)$ : kind of restricted quaternion modular forms

There are many more types of modular forms:

- Siegel modular forms of genus  $> 2$ ,
- Jacobi forms (with lattice index),
- paramodular forms,
- orthogonal modular forms
  - hermitian modular forms
  - quaternion modular forms
  - $O(2, 5)$ : kind of restricted quaternion modular forms

There are many more types of modular forms:

- Siegel modular forms of genus  $> 2$ ,
- Jacobi forms (with lattice index),
- paramodular forms,
- orthogonal modular forms
  - hermitian modular forms
  - quaternion modular forms
  - $O(2, 5)$ : kind of restricted quaternion modular forms

## A framework for all expansions

The modular forms Fourier expansion framework provides fast and convenient implementation for general types of modular forms.

- It is not (yet) in Sage or PSage.
- It has  $> 30,000$  lines, so PSage seems more realistic.

## A framework for all expansions

The modular forms Fourier expansion framework provides fast and convenient implementation for general types of modular forms.

- It is not (yet) in Sage or PSage.
- It has  $> 30,000$  lines, so PSage seems more realistic.

## A framework for all expansions

The modular forms Fourier expansion framework provides fast and convenient implementation for general types of modular forms.

- It is not (yet) in Sage or PSage.
- It has  $> 30,000$  lines, so PSage seems more realistic.

- Almost all types mentioned above are provided, but most of these implementations are not quite complete.
- Focus on performance for L-series, so other aspects might be slow.
- The implementations can be used for research computations, but until now only few people have used it to investigate:
  - L-series,
  - Sturm bounds and the vanishing cone,
  - Structure of spaces and
  - congruences.

- Almost all types mentioned above are provided, but most of these implementations are not quite complete.
- Focus on performance for L-series, so other aspects might be slow.
- The implementations can be used for research computations, but until now only few people have used it to investigate:
  - *L*-series,
  - Sturm bounds and the vanishing cone,
  - Structure of spaces and
  - congruences.

- Almost all types mentioned above are provided, but most of these implementations are not quite complete.
- Focus on performance for L-series, so other aspects might be slow.
- The implementations can be used for research computations, but until now only few people have used it to investigate:
  - $L$ -series,
  - Sturm bounds and the vanishing cone,
  - Structure of spaces and
  - congruences.

# Modeling your Fourier expansion

You need to fix

- $S$ : a monoid of Fourier indices,
- $G \curvearrowright S$ : a group,
- $f_\lambda, \lambda \in \Lambda$ : precisions;  $G$ -finite subsets with multiplicatively closed complement in  $S$ ,
- $A$ : an  $R$ -module with  $G$  action and
- $\hat{\chi} : C \rightarrow \text{Hom}(G, \overline{\mathbb{Q}}^\times)$ : a monoid of characters.

# Modeling your Fourier expansion

You need to fix

- $S$ : a monoid of Fourier indices,
- $G \curvearrowright S$ : a group,
- $f_\lambda, \lambda \in \Lambda$ : precisions;  $G$ -finite subsets with multiplicatively closed complement in  $S$ ,
- $A$ : an  $R$ -module with  $G$  action and
- $\hat{\chi} : C \rightarrow \text{Hom}(G, \overline{\mathbb{Q}}^\times)$ : a monoid of characters.

# Modeling your Fourier expansion

You need to fix

- $S$ : a monoid of Fourier indices,
- $G \curvearrowright S$ : a group,
- $f_\lambda, \lambda \in \Lambda$ : precisions;  $G$ -finite subsets with multiplicatively closed complement in  $S$ ,
- $A$ : an  $R$ -module with  $G$  action and
- $\hat{\chi} : C \rightarrow \text{Hom}(G, \overline{\mathbb{Q}}^\times)$ : a monoid of characters.

You provide an implementation of

- the  $G$ -action and  $\hat{\chi}$ ,
- representatives in  $S$  w.r.t. the  $G$  action,
- the filter  $f_\lambda$ , that is,
  - enumeration of representatives in  $S$  and
  - $\{s_1 s_2 = s\}$  for given  $s$ ,

and some helper functions.

From this you get an implementation of Fourier expansions.

You provide an implementation of

- the  $G$ -action and  $\hat{\chi}$ ,
- representatives in  $S$  w.r.t. the  $G$  action,
- the filter  $f_\lambda$ , that is,
  - enumeration of representatives in  $S$  and
  - $\{s_1 s_2 = s\}$  for given  $s$ ,

and some helper functions.

From this you get an implementation of Fourier expansions.

You provide an implementation of

- the  $G$ -action and  $\hat{\chi}$ ,
- representatives in  $S$  w.r.t. the  $G$  action,
- the filter  $f_\lambda$ , that is,
  - enumeration of representatives in  $S$  and
  - $\{s_1 s_2 = s\}$  for given  $s$ ,

and some helper functions.

From this you get an implementation of Fourier expansions.

You provide an implementation of

- the  $G$ -action and  $\hat{\chi}$ ,
- representatives in  $S$  w.r.t. the  $G$  action,
- the filter  $f_\lambda$ , that is,
  - enumeration of representatives in  $S$  and
  - $\{s_1 s_2 = s\}$  for given  $s$ ,

and some helper functions.

From this you get an implementation of Fourier expansions.

You provide an implementation of

- the  $G$ -action and  $\hat{\chi}$ ,
- representatives in  $S$  w.r.t. the  $G$  action,
- the filter  $f_\lambda$ , that is,
  - enumeration of representatives in  $S$  and
  - $\{s_1 s_2 = s\}$  for given  $s$ ,

and some helper functions.

From this you get an implementation of Fourier expansions.

You provide an implementation of

- the  $G$ -action and  $\hat{\chi}$ ,
- representatives in  $S$  w.r.t. the  $G$  action,
- the filter  $f_\lambda$ , that is,
  - enumeration of representatives in  $S$  and
  - $\{s_1 s_2 = s\}$  for given  $s$ ,

and some helper functions.

From this you get an implementation of Fourier expansions.

You provide an implementation of

- the  $G$ -action and  $\hat{\chi}$ ,
- representatives in  $S$  w.r.t. the  $G$  action,
- the filter  $f_\lambda$ , that is,
  - enumeration of representatives in  $S$  and
  - $\{s_1 s_2 = s\}$  for given  $s$ ,

and some helper functions.

From this you get an implementation of Fourier expansions.

You provide an implementation of

- the  $G$ -action and  $\hat{\chi}$ ,
- representatives in  $S$  w.r.t. the  $G$  action,
- the filter  $f_\lambda$ , that is,
  - enumeration of representatives in  $S$  and
  - $\{s_1 s_2 = s\}$  for given  $s$ ,

and some helper functions.

**From this you get an implementation of Fourier expansions.**

## Example: Symmetric algebraic power series

Reduction of element of  $S$

```
ssorted = sorted(zip(s, range(1, self.__n + 1)))  
  
return ( tuple([e for (e,_) in ssorted]),  
         self.group()([p for (_,p) in ssorted]) )
```

Decomposition of elements in  $S$

```
for t in iterprod(*(map(lambda i: xrange(i+1), s))) :  
    yield (t, tuple(map(operator.sub, s, t)))  
  
raise StopIteration
```

## Example: Symmetric algebraic power series

Reduction of element of  $S$

```
ssorted = sorted(zip(s, range(1, self.__n + 1)))
```

```
return ( tuple([e for (e,_) in ssorted]),  
        self.group()([p for (_,p) in ssorted]) )
```

Decomposition of elements in  $S$

```
for t in iterprod(*(map(lambda i: xrange(i+1), s))) :  
    yield (t, tuple(map(operator.sub, s, t)))
```

```
raise StopIteration
```

## Example: Symmetric algebraic power series

Reduction of element of  $S$

```
ssorted = sorted(zip(s, range(1, self.__n + 1)))
```

```
return ( tuple([e for (e,_) in ssorted]),  
        self.group()([p for (_,p) in ssorted]) )
```

Decomposition of elements in  $S$

```
for t in iterprod(*(map(lambda i: xrange(i+1), s))) :  
    yield (t, tuple(map(operator.sub, s, t)))
```

```
raise StopIteration
```

Containment in  $f_\lambda$ :

```
return all(b is infinity or se < b
           for (se,b) in zip(s, self.__bound))
```

and iteration over reduced elements:

```
dbs = [b - bp
        for (b,bp) in zip(self.__bound, [0] + self.__bound)]
for s in iterprod(*(map(xrange, dbs))):
    yield tuple([sum(s[:i])
                 for i in range(1,self.__n + 1)])
```

```
raise StopIteration
```

Containment in  $f_\lambda$ :

```
return all(b is infinity or se < b
           for (se,b) in zip(s, self.__bound))
```

and iteration over reduced elements:

```
dbs = [b - bp
        for (b,bp) in zip(self.__bound, [0] + self.__bound)]
for s in iterprod(*(map(xrange, dbs))):
    yield tuple([sum(s[:i])
                 for i in range(1,self.__n + 1)])
```

```
raise StopIteration
```

Trivial characters and trivial representations on  $A$  are provided by the framework.

So let's initialize the ring:

```
EquivariantMonoidPowerSeriesRing(NNnMonoid(n, True),  
    TrivialCharacterMonoid(SymmetricGroup(n), A),  
    TrivialRepresentation(SymmetricGroup(n), A))
```

Trivial characters and trivial representations on  $A$  are provided by the framework.

So let's initialize the ring:

```
EquivariantMonoidPowerSeriesRing(NNnMonoid(n, True),  
    TrivialCharacterMonoid(SymmetricGroup(n), A),  
    TrivialRepresentation(SymmetricGroup(n), A))
```

# Modular forms and generators of graded rings

- Often Maaß lifts give a set of generators (the easiest to implement).
- In principal, theta series are sufficient (this is slow!!).
- Borcherds products are too slow if implemented naively, but a new algorithm is currently beeing implemented.

# Modular forms and generators of graded rings

- Often Maaß lifts give a set of generators (the easiest to implement).
- In principal, theta series are sufficient (this is slow!!).
- Borcherds products are too slow if implemented naively, but a new algorithm is currently being implemented.

# Modular forms and generators of graded rings

- Often Maaß lifts give a set of generators (the easiest to implement).
- In principal, theta series are sufficient (this is slow!!).
- Borcherds products are too slow if implemented naively, but a new algorithm is currently beeing implemented.

- Get this framework into PSage .
- Find Karatsuba type multiplication for equivariant monoid power series.
- Develop modular symbols for Siegel modular forms or anything that works equally well (Poor's and Yuen's conjecture is proved!).

- Get this framework into PSage **and use it!**
- Find Karatsuba type multiplication for equivariant monoid power series.
- Develop modular symbols for Siegel modular forms or anything that works equally well (Poor's and Yuen's conjecture is proved!).

- Get this framework into PSage **and use it!**
- Find Karatsuba type multiplication for equivariant monoid power series.
- Develop modular symbols for Siegel modular forms or anything that works equally well (Poor's and Yuen's conjecture is proved!).

- Get this framework into PSage **and use it!**
- Find Karatsuba type multiplication for equivariant monoid power series.
- Develop modular symbols for Siegel modular forms or anything that works equally well (Poor's and Yuen's conjecture is proved!).

# Questions?