Modular forms in SAGE, a (short) status report

Craig Citro

September 30, 2007
The basics ...

Here's a sample session doing something you'd expect:

```
[citro@sage ~] $ magma
Magma V2.13-5 Sat Sep 29 2007 20:46:07 on sage
Type ? for help.  Type <Ctrl>-D to quit.
> M := ModularSymbols(12,4);
> S := CuspidalSubspace(M);
> NewformDecomposition(S);
[ Modular symbols space for Gamma_0(12) of weight 4 and dimension 2 over Rational Field,
  Modular symbols space for Gamma_0(12) of weight 4 and dimension 4 over Rational Field
]
> qExpansionBasis(S);
[ q - 6*q^5 - 4*q^7 + O(q^8),
  q^2 - 2*q^4 - 3*q^6 + O(q^8),
  q^3 - 4*q^5 + 4*q^7 + O(q^8)
]
```
The basics ...

The corresponding code works quite well in SAGE:

-------------------------
| SAGE Version 2.8.5.1, |
-------------------------

```
sage: M = ModularSymbols(12,4) ; S = M.cuspidal_subspace()
sage: S.decomposition()
[
   Modular Symbols subspace of dimension 2 of Modular Symbols space
   Modular Symbols subspace of dimension 4 of Modular Symbols space
]
```

```
sage: S.q_expansion_basis(8)
[
   q - 6*q^5 - 4*q^7 + O(q^8),
   q^2 - 2*q^4 - 3*q^6 + O(q^8),
   q^3 - 4*q^5 + 4*q^7 + O(q^8)
]
```

```
sage:
```
MAGMA also has good code for working directly with modular forms:

```plaintext
> Newforms(ModularForms(12,4));
[* [*
   q + 3*q^3 - 18*q^5 + 8*q^7 + O(q^8)
*]/*]
```

SAGE has more of an issue with this:

```plaintext
sage: ModularForms(12,4).new_subspace()
-------------------------------------------
Traceback (most recent call last)
...
<type 'exceptions.NotImplementedError'>:
```
So let’s say you want some bigger examples: this isn’t somewhere we have any sort of optimized code.

Magma V2.13-5 Sun Sep 30 2007 07:31:19 on sage
Type ? for help. Type <Ctrl>-D to quit.
> time M := ModularSymbols(30,4);
Time: 0.110

> time M := ModularSymbols(30,16);
Time: 3.940

> time M := ModularSymbols(30,8);
Time: 0.340
> time M := ModularSymbols(300,8);
Time: 165.650
The basics ...

```python
sage: time M = ModularSymbols(30,4)
CPU times: user 0.13 s, sys: 0.02 s, total: 0.15 s
Wall time: 0.15 ( vs. 0.110 )

sage: time M = ModularSymbols(30,16)
CPU times: user 41.30 s, sys: 2.08 s, total: 43.38 s
Wall time: 43.52 ( vs. 3.940 )

sage: time M = ModularSymbols(30,8)
CPU times: user 6.58 s, sys: 0.34 s, total: 6.91 s
Wall time: 6.91 ( vs. 0.340 )
sage: time M = ModularSymbols(300,8)
### gave up after ~1.5 hours, vs. 165.650
```
It’s not all bad.

Of course, we’re currently comparing things that MAGMA has heavily optimized to things that aren’t in SAGE: in the example above, in order to compute a presentation of the space of Manin symbols, MAGMA is using hand-coded C, and SAGE is using code William wrote in Python. If you profile one of those:

```
sage: prun M = ModularSymbols(50,6)

68240 function calls (68234 primitive calls) in 2.890 CPU

Ordered by: internal time

ncalls   tottime    percall   cumtime    percall   filename:lineno(function)
1        2.212    2.212      2.289    2.289   relation_matrix.py:183(gens_to_basis_matrix)
...
```

- Craig Citro

Modular forms in SAGE, a (short) status report
Warning!

So the code for modular forms in SAGE is divided into two parts: modular forms code and modular symbols code. Anything based on modular symbols is trustworthy . . . not as much can be said about any of the code that works directly with modular forms. There is at least one bug on trac about this, and I’m sure there are more.

Here’s what I know about the history of this code. William wrote basically all of this code, much of it in the very early days of SAGE, and this sometimes shows. The errors in the modular forms code I mentioned above I think date from this era: the code looks like William wrote it with certain expectations about how various things would be implemented in SAGE, and then things ended up getting developed differently.
What should we do?

There are other places in the code where we *should* beat MAGMA, at least asymptotically. For instance, we’re faster to compute matrices of Hecke operators, and we’re asymptotically faster for computing charpolys:

```plaintext
> M := ModularSymbols(NextPrime(2000), 2, 1);
> time t := HeckeOperator(M, 2);
Time: 0.260
> time f := CharacteristicPolynomial(t);
Time: 0.150
```

```plaintext
sage: time M = ModularSymbols(next_prime(2000), 2, sign=1)
sage: time t = M.hecke_matrix(2)
CPU times: user 0.13 s, sys: 0.00 s, total: 0.13 s
Wall time: 0.13
sage: time f = t.charpoly()
CPU times: user 0.92 s, sys: 0.00 s, total: 0.92 s
Wall time: 0.95
```
What should we do?

> M := ModularSymbols(NextPrime(10000),2,1);
> time t := HeckeOperator(M, 2);
  Time: 3.310
> time f := CharacteristicPolynomial(t);
  Time: 20.860

sage: time M = ModularSymbols(next_prime(10000),2,sign=1)
sage: time t = M.hecke_matrix(2)
CPU times: user 2.30 s, sys: 0.05 s, total: 2.35 s
Wall time: 2.35
sage: time f= t.charpoly()
CPU times: user 367.68 s, sys: 2.52 s, total: 370.20 s
Wall time: 370.53
What should we do?

I think the only serious problem is that this code hasn’t seen enough attention over time. William wrote almost all of it, and things only seem to get added when someone needs a specific function added.

- **sage/modular/modform**: This code needs to be looked over somewhat closely, and parts rewritten to use modular symbols for actual computation of spaces of modular forms.

- **sage/modular/modsym**: This code all works quite well, but needs to be optimized. Case in point: the creation of spaces that we saw timings on above.

There’s also just a lot missing:

```
[craigcitro@dhcp140 /sage/devel/sage-cc/sage/modular] $ grep "NotImplementedError" *.py */*/py | wc -l
128
```
What should we do?

There’s very little in the way of doctests; here’s `sage/modular/modform`:

<table>
<thead>
<tr>
<th>BEFORE</th>
<th>AFTER</th>
<th>FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>100%</td>
<td>ambient.py</td>
</tr>
<tr>
<td>100%</td>
<td>100%</td>
<td>ambient_eps.py</td>
</tr>
<tr>
<td>30%</td>
<td>100%</td>
<td>constructor.py</td>
</tr>
<tr>
<td>10%</td>
<td>100%</td>
<td>cuspidal_submodule.py</td>
</tr>
<tr>
<td>20%</td>
<td>100%</td>
<td>eis_series.py</td>
</tr>
<tr>
<td>2%</td>
<td>31%</td>
<td>eisenstein_submodule.py</td>
</tr>
<tr>
<td>5%</td>
<td>60%</td>
<td>element.py</td>
</tr>
<tr>
<td>100%</td>
<td>100%</td>
<td>half_integral.py</td>
</tr>
<tr>
<td>0%</td>
<td>100%</td>
<td>hecke_operator_on_qexp.py</td>
</tr>
<tr>
<td>8%</td>
<td>100%</td>
<td>numerical.py</td>
</tr>
<tr>
<td>15%</td>
<td>100%</td>
<td>space.py</td>
</tr>
<tr>
<td>100%</td>
<td>100%</td>
<td>theta.py</td>
</tr>
<tr>
<td>100%</td>
<td>100%</td>
<td>vm_basis.py</td>
</tr>
</tbody>
</table>
Discussion

What do you all need?
Where I’m starting:

So here’s where I’ll be starting:

- Start making `sage/modular/modform` produce correct answers
- Get doctests in `sage/modular/modform` passing `sage -coverage`
- Start optimizing `sage/modular/modsym` code