Modular forms in SageMath

Peter Bruin and Sander Dahmen

April 19 till May 17

```
### How to experiment with modular forms in Sage ###

## Before starting with modular forms commands, we briefly look at \
   some basic Python/SageMath syntax ##

# Variables are pretty standard, we get some output using the print \
   command
# To evaluate a code block, press shift-enter
a = 7
b = 8
print a + b

mystring = "Variables can handle anything, "
mystring_number_two = "and are allowed pretty long names."
print mystring + mystring_number_two
```

```
15
Variables can handle anything, and are allowed pretty long names.
```

```
# The if-statement shows how indentation works. The Sage notebook \
   tries to help you by indenting the cursor with 4 spaces when \
   indentation is needed. Also, this examples shows that you can use\
    variables from other code blocks (provided they are evaluated \
   earlier).
if a == b:
    print "We aren't working in the zero ring, are we?"
else:
    if true:
        print "If-statements can be nested."
```

```
If-statements can be nested.
```

```
# A typical while-loop
c=5
while c < 10:
    print c
    c = c + 2
```

```
5
7
9
```

```
# Lists
R=[1,2,4]; print R
S=[1..5]; print S
T=[x^2 for x in S]; print T
print prime_range(17)
print prime_range(7,18)
```

```
[1, 2, 4]
[1, 2, 3, 4, 5]
[1, 4, 9, 16, 25]
[2, 3, 5, 7, 11, 13]
[7, 11, 13, 17]
```

```
# A typical for-loop
S = [3..7]
for x in S:
    print 2*x
```

```
6
8
10
12
14
```

```
## Now we start with modular forms commands ##
```

```
# Create some congruence subgroups
G = SL2Z
G02 = Gamma0(2)
G15 = Gamma1(5)
G16 = Gamma1(6)
G7 = Gamma(7)
```

```
# Print a description of two of the objects we have defined
print G
print G02
```

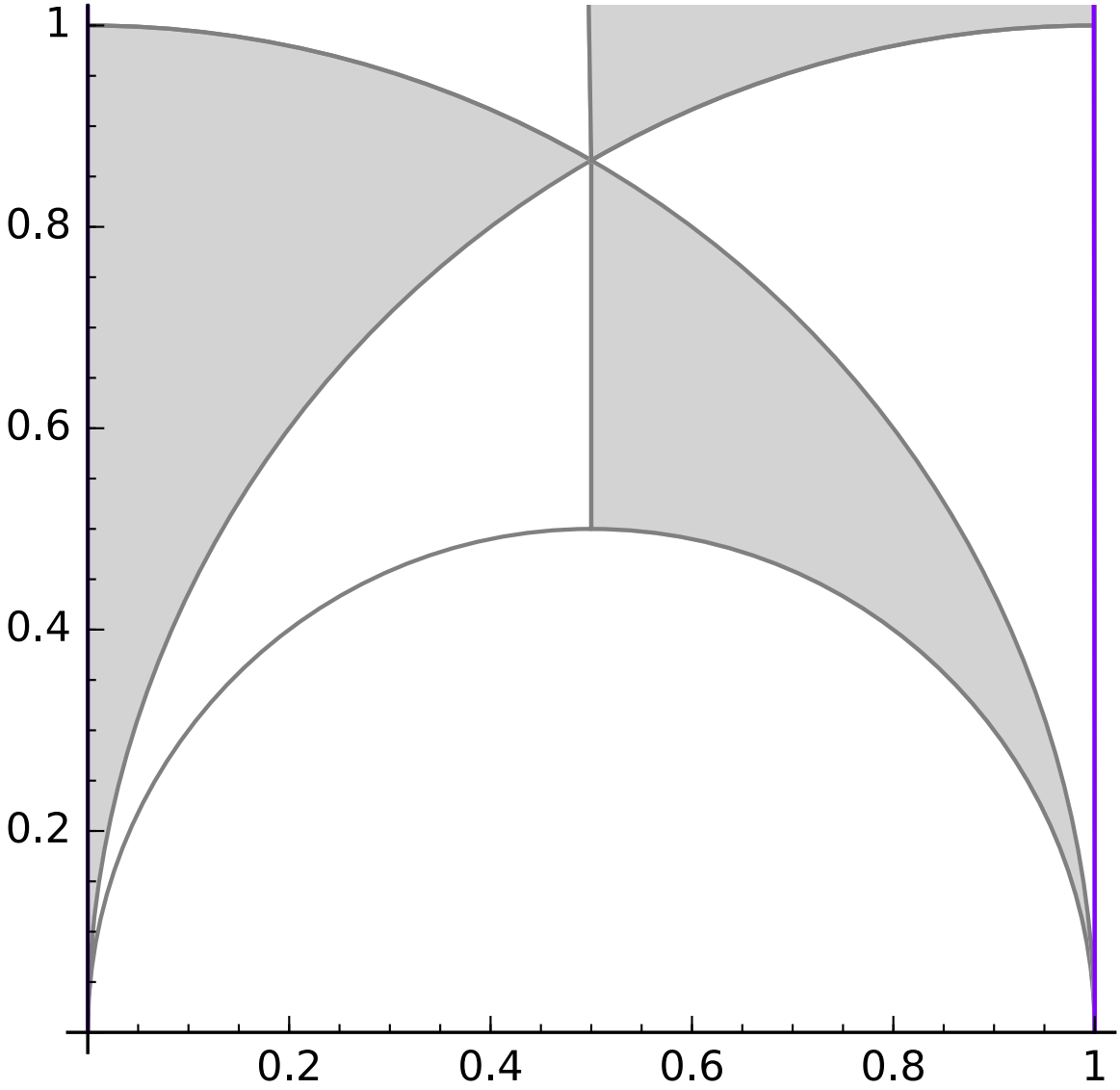
```
Modular Group SL(2,Z)
Congruence Subgroup Gamma0(2)
```

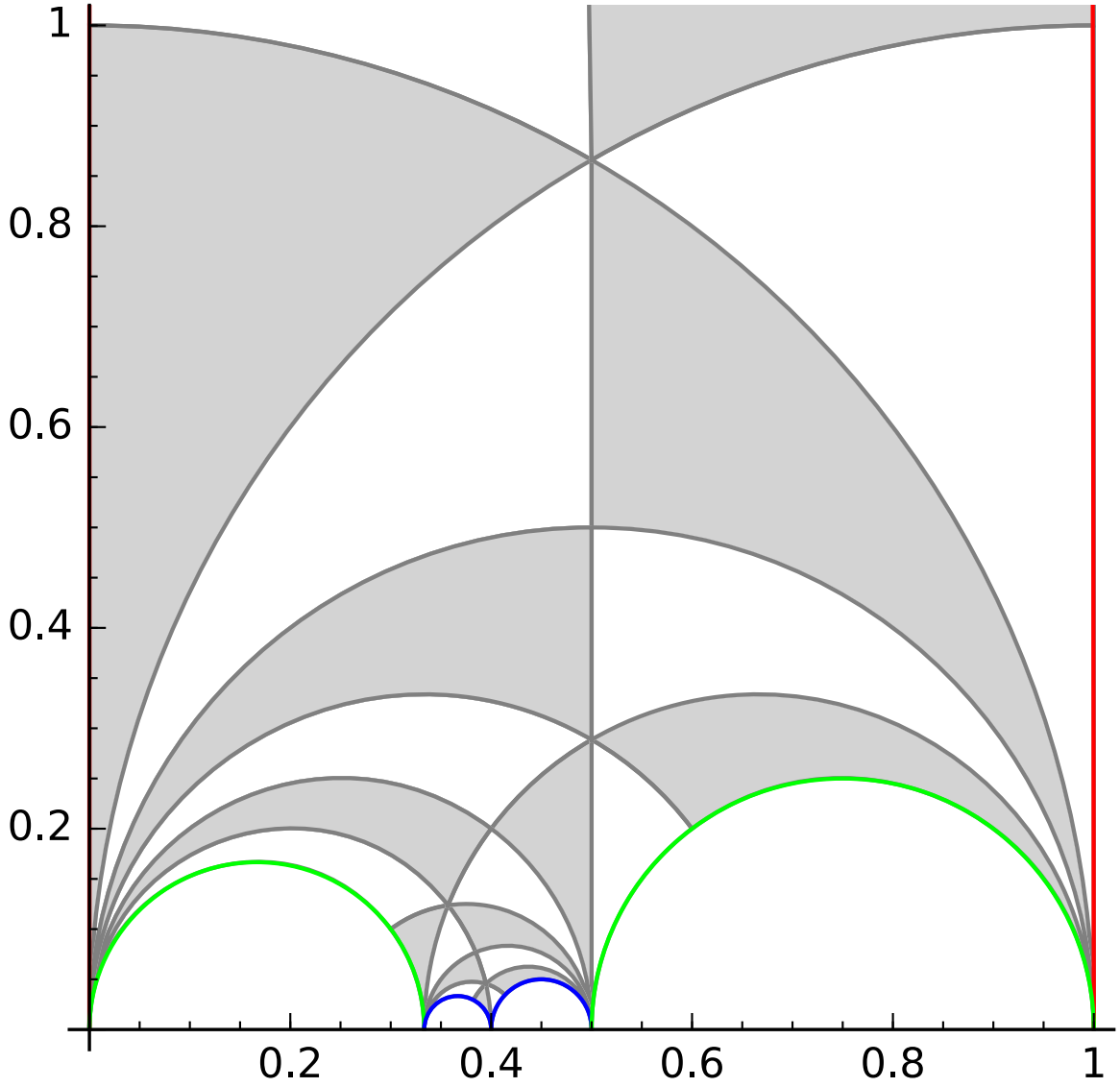
```
# Sage knows about relations between different groups
print G16.is_subgroup(G02)
print G16.index()/G02.index()
```

```
True
8
```

```
# Representatives for the cusps
```

```
print G.cusps()
print G15.cusps()
```
```
[Infinity]
[0, 2/5, 1/2, Infinity]
```
```
# Generators
print "Generators for SL2Z:"
for g in G.gens():
    print g
    print ""
print "Generators for Gamma0(2):"
for g in G02.gens():
    print g
    print ""
```
```
Generators for SL2Z:
[ 0 -1]
[ 1  0]

[1 1]
[0 1]
Generators for Gamma0(2):
[1 1]
[0 1]

[ 1 -1]
[ 2 -1]
```
```
#Coset representatives
print "Coset representatives of Gamma0(2) in SL2Z:"
for g in G02.coset_reps():
    print g
    print ""
```
```
Coset representatives of Gamma0(2) in SL2Z:
[1 0]
[0 1]

[ 0 -1]
[ 1  0]

[1 0]
[1 1]
```
```
#Fundamental domains (the FareySymbol can be considered as a black \
   box)
FareySymbol(G02).fundamental_domain(show_pairing=true)
FareySymbol(G15).fundamental_domain(show_pairing=true)
```


1
0.8
0.6
0.4
0.2
0.2
0.4
0.6
0.8
1




```
# Construct a space of modular forms
M = ModularForms(SL2Z, 12)
print M
```

```
Modular Forms space of dimension 2 for Modular Group SL(2,Z) of weight 12 over Rational
Field
```

```
# Compute a basis and give the q-expansions
print M.basis()
```

```
[
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 + O(q^6),
1 + 65520/691*q + 134250480/691*q^2 + 11606736960/691*q^3 + 274945048560/691*q^4 +
3199218815520/691*q^5 + O(q^6)
]
```

```
# Getting more terms is no problem.
print M.q_expansion_basis(20)
```

```
[
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 - 6048*q^6 - 16744*q^7 + 84480*q^8 - 113643*q^9
- 115920*q^10 + 534612*q^11 - 370944*q^12 - 577738*q^13 + 401856*q^14 + 1217160*q^15 +
987136*q^16 - 6905934*q^17 + 2727432*q^18 + 10661420*q^19 + O(q^20),
1 + 65520/691*q + 134250480/691*q^2 + 11606736960/691*q^3 + 274945048560/691*q^4 +
3199218815520/691*q^5 + 23782204031040/691*q^6 + 129554448266880/691*q^7 +
563087459516400/691*q^8 + 2056098632318640/691*q^9 + 6555199353000480/691*q^10 +
18693620658498240/691*q^11 + 48705965462306880/691*q^12 + 117422349017369760/691*q^13 +
265457064498837120/691*q^14 + 566735214731736960/691*q^15 + 1153203117089652720/691*q^16 +
2245494646076179680/691*q^17 + 4212946097620893360/691*q^18 + 7632441763011374400/691*q^19
+ O(q^20)
]
```

```
# picking out a modular form from the basis
# showing q_expansion and picking out coefficients of q^p with p \
   prime
delta=M.basis()[0]
print delta.q_expansion()
print delta.q_expansion(12)
print [delta[p] for p in prime_range(2,12)]
```

```
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 + O(q^6)
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 - 6048*q^6 - 16744*q^7 + 84480*q^8 - 113643*q^9
- 115920*q^10 + 534612*q^11 + O(q^12)
[-24, 252, 4830, -16744, 534612]
```

```
# Construct a space of cusp forms
S = CuspForms(SL2Z,24)
print S
```

```
Cuspidal subspace of dimension 2 of Modular Forms space of dimension 3 for Modular Group
SL(2,Z) of weight 24 over Rational Field
```

```
# Hecke operators
# Matrix with respect to computed basis, here: S.basis()
# characteristic polynomial of the operator/matrix
T2=S.T(2)
print T2
print T2.matrix()
print T2.charpoly()
```

```
Hecke operator T_2 on Cuspidal subspace of dimension 2 of Modular Forms space of dimension
3 for Modular Group SL(2,Z) of weight 24 over Rational Field
[       0 20468736]
[       1     1080]
x^2 - 1080*x - 20468736
```

```
# Hecke operator on modular form
```

```
print T2(delta*delta)
print T2(delta*delta) == S.basis()[0]+1080*S.basis()[1]
```

```
q + 1080*q^2 + 143820*q^3 + 13246528*q^4 + 28412910*q^5 + O(q^6)
True
```

```
# The diamond operator
MM=ModularForms(Gamma1(7),2)
print MM
d5=MM.diamond_bracket_operator(5)
print d5
print d5.matrix()
```

```
Modular Forms space of dimension 5 for Congruence Subgroup Gamma1(7) of weight 2 over
Rational Field
Diamond bracket operator <5> on Modular Forms space of dimension 5 for Congruence Subgroup
Gamma1(7) of weight 2 over Rational Field
[  -47   -84  -420  -420 -1260]
[  -13   -24  -120  -115  -355]
[    2     4    19    18    57]
[    3     5    27    25    81]
[    1     2     9    10    26]
```

```
%md
26 April
--------

_The Petersson inner product; old and new subspaces_
```

0.1 26 April

The Petersson inner product; old and new subspaces

```
# Eisenstein subspace and cuspidal subspace
M = ModularForms(Gamma1(14), 4)
E = M.eisenstein_submodule()
S = CuspForms(Gamma1(14), 4)
S == M.cuspidal_submodule()
M.dimension()
(E.dimension(), S.dimension())
```

```
True
24
(12, 12)
```

```
# Old and new subspaces
Sold = S.old_submodule()
Snew = S.new_submodule()
(Sold.dimension(), Snew.dimension())
```

```
(6, 6)
```

```
# Check consistency with dimensions of spaces of lower level
CuspForms(Gamma1(2), 4).dimension()
CuspForms(Gamma1(7), 4).dimension()s
```

```
0
3
```

```
S.old_submodule(2)
```

```
Modular Forms subspace of dimension 6 of Modular Forms space of dimension 24 for
Congruence Subgroup Gamma1(14) of weight 4 over Rational Field
```

```
# An example of a non-diagonalisable Hecke operator.  This shows \
   that the result
# stated in the lecture on simultaneous eigenvectors for the T_m \
   with m coprime to
# the level N cannot be generalised directly to the T_m with m not \
   coprime to N.
S = CuspForms(Gamma0(16), 4)
T2 = S.hecke_matrix(2)
T2.jordan_form()
```

```
[0 1|0]
[0 0|0]
[---+-]
[0 0|0]
```

0.2 3 May

Newforms

```
# We can compute the set of newforms (primitive forms) of a given \
   level and weight.
S = CuspForms(Gamma1(15), 2)
N = S.newforms()
N
Newforms(Gamma1(15), 2) == N
```

```
[q - q^2 - q^3 - q^4 + q^5 + O(q^6)]
True
```

```
# An example with multiple newforms
Newforms(Gamma0(26), 2)
```

```
[q - q^2 + q^3 + q^4 - 3*q^5 + O(q^6), q + q^2 - 3*q^3 + q^4 - q^5 + O(q^6)]
```

```
# We have to be careful when the newforms don't have rational \
   coefficients
Newforms(Gamma1(26), 2)
```

```
Error in lines 2-2
Traceback (most recent call last):
  File ‘‘/projects/sage/sage-6.10/local/lib/python2.7/site-
packages/smc_sagews/sage_server.py’’, line 905, in execute
    exec compile(block+'\n', '', 'single') in namespace, locals
  File ‘‘’’, line 1, in <module>
  File ‘‘/projects/sage/sage-6.10/local/lib/python2.7/site-
packages/sage/modular/modform/constructor.py’’, line 454, in Newforms
    return CuspForms(group, weight, base_ring).newforms(names)
  File ‘‘/projects/sage/sage-6.10/local/lib/python2.7/site-
packages/sage/modular/modform/space.py’’, line 1675, in newforms
    raise ValueError(‘‘Please specify a name to be used when generating names for
generators of Hecke eigenvalue fields corresponding to the newforms.’’)
ValueError: Please specify a name to be used when generating names for generators of Hecke
eigenvalue fields corresponding to the newforms.
```

```
# In this case we have to specify a "names" parameter.
N = Newforms(Gamma1(26), 2, names='a')
N
```

```
[q - q^2 + q^3 + q^4 - 3*q^5 + O(q^6), q + q^2 - 3*q^3 + q^4 - q^5 + O(q^6), q + a2*q^2 +
(-a2 - 1)*q^4 - q^5 + O(q^6), q + a3*q^2 - q^3 - q^4 - 3*a3*q^5 + O(q^6)]
```

```
# Every newform has a character
[f.character() for f in N]
```

```
[Dirichlet character modulo 26 of conductor 1 mapping 15 |--> 1, Dirichlet character
modulo 26 of conductor 1 mapping 15 |--> 1, Dirichlet character modulo 26 of conductor 13
mapping 15 |--> -a2 - 1, Dirichlet character modulo 26 of conductor 13 mapping 15 |--> -1]
```

```
# Let us compare our list of newforms with the dimension of the new \
   subspace.
S = CuspForms(Gamma1(26), 2)
Sold = S.old_submodule()
Snew = S.new_submodule()
(Sold.dimension(), Snew.dimension())
```

```
(4, 6)
```

```
# Sage only gives 4 newforms, but the new subspace has dimension 6.
# This is explained by the fact that two of the forms have larger \
   coefficient fields:
[f.base_ring().degree() for f in N]
```

```
[1, 1, 2, 2]
```

```
# The following computation shows that the coefficient fields of the\
    last two forms
# are Q(\sqrt{-3}) and Q(\sqrt{-1}), respectively.
[f.base_ring().discriminant() for f in N]
```

```
[1, 1, -3, -4]
```

```
%md
10 May
------

_$L$-functions_
```

0.3 10 May

*L*-functions

```
# Here is a newform of which we are going to compute the L-series.
# Warning: some things only work for forms with rational \
   coefficients...
N = 14
f = Newforms(Gamma1(N))[0]; f
```

```
q - q^2 - 2*q^3 + q^4 + O(q^6)
```

```
Lf = f.lseries()
Lf
```

```
L-series associated to the cusp form q - q^2 - 2*q^3 + q^4 + O(q^6)
```

```
# We can evaluate L-series both inside and outside the
# right half-plane where the Dirichlet series converges.
Lf(3)
Lf(3+2*I)
Lf(-2-I)
```

```
0.826125962101783
0.995825161298581 + 0.180645100106889*I

1.25737321267029 - 0.432187040382323*I
```

```
# The L-function is holomorphic.
Lf.poles
```

```
[]
```

```
# Like the Riemann zeta function, it has some 'trivial' zeroes.
[Lf(s) for s in [-4..-1]]
```

```
[0.000000000000000, 0.000000000000000, 0.000000000000000, 0.000000000000000]
```

```
# Unfortunately, SageMath is not bug-free:
Lf(0)
```

```
  ***   at top-level: L(0.000000000000000)
  ***                 ^--------------------
  ***   in function L: polcoeff(Lseries(ss,cutoff,de
  ***                           ^--------------------
```

```
  ***   in function Lseries: ...‘‘));LSSeries=sum(k=0,der,Lstar(ss,cutoff,k)*S
  ***                                                  ^--------------------
  ***   in function Lstar: ...cf2,if(cfvec[k],cfvec[k]*G(k*cutoff/vA,ss,der
  ***                                                 ^--------------------
  ***   in function G: ...ss,der]);if(t<GCaseBound,G0(t,ss,der),nn=min(
  ***                                               ^--------------------
  ***   in function G0: ...es(ss,der)/t^(S+ss);gmcf=polcoeff(gmser,der,S
  ***                                                ^--------------------
  *** polcoeff: domain error in polcoeff: degree > -1
Error in lines 2-2
Traceback (most recent call last):
  File ‘‘/projects/sage/sage-6.10/local/lib/python2.7/site-
packages/smc_sagews/sage_server.py’’, line 905, in execute
    exec compile(block+'\n', '', 'single') in namespace, locals
  File ‘‘’’, line 1, in <module>
  File ‘‘/projects/sage/sage-6.10/local/lib/python2.7/site-
packages/sage/lfunctions/dokchitser.py’’, line 405, in __call__
    raise RuntimeError
RuntimeError
```

```
# Sign of the functional equation
Lf.eps
```

```
1
```

```
Lf.conductor
```

```
14
```

```
# Check numerically that the completed L-function
# satisfies the expected functional equation.
def Lambda(s):
    return gamma(s)*N^(s/2)/(2*pi.n())^s * Lf(s)
s = 1.43250982+.435873*I
[Lambda(s), Lambda(2 - s)]
```

```
[0.196288571460192 + 0.0108503534364708*I, 0.196288571460192 + 0.0108503534364707*I]
```

```
# There is also a quicker (but more obscure) way:
Lf.check_functional_equation()  # answer should be a small number
```

```
-1.68051336735253e-18
```

0.4 17 May

Elliptic curves, modularity and the conjecture of Birch and Swinnerton-Dyer

```
# We construct an elliptic curve over Q.
E = EllipticCurve([0, -1, 1, 0, 0]); E
```

```
Elliptic Curve defined by y^2 + y = x^3 - x^2 over Rational Field
```

```
# The conductor of E is 11, so by the modularity theorem
# E corresponds to a cusp form for Gamma0(11) (of weight 2).
N = E.conductor(); N
```
```
11
```

```
# There is only one such cusp form:
nf = Newforms(Gamma0(N), 2); len(nf)
f = nf[0]; f
```
```
1

q - 2*q^2 - q^3 + 2*q^4 + q^5 + O(q^6)
```

```
# Check that the coefficients agree to some precision
all(f[p] == E.ap(p) for p in primes(500))
```
```
True
```

```
# Construct the corresponding L-functions
LE = E.lseries(); LE
Lf = f.lseries(); Lf
```
```
Complex L-series of the Elliptic Curve defined by y^2 + y = x^3 - x^2 over Rational Field
L-series associated to the cusp form q - 2*q^2 - q^3 + 2*q^4 + q^5 + O(q^6)
```

```
# Check that the L-values in s = 1 agree.
# (For L(E, 1), the second number returned is an error bound).
Lf(1)
LE.at1()
```
```
0.253841860855911
(0.253804, 0.000181444)
```

```
# Check that the (strong) BSD conjecture holds for E.
# The output is a list of primes such that L^*(E, 1) equals
# the value predicted by BSD up to a product of powers of
# primes in this list.
E.prove_BSD()
```
```
[]
```

```
# Unfortunately, nothing about BSD is known for elliptic curves
# of higher rank.
E = EllipticCurve('5077a1'); E
E.rank()
E.prove_BSD()
```
```
Elliptic Curve defined by y^2 + y = x^3 - 7*x + 6 over Rational Field

3
```

Set of all prime numbers: 2, 3, 5, 7, ...

```
# Solve the congruent number problem for n
def is_congruent(n):
    return EllipticCurve([0, 0, 0, -n^2, 0]).rank() > 0

[n for n in range(1, 25) if is_congruent(n)]
```

```
[5, 6, 7, 13, 14, 15, 20, 21, 22, 23, 24]
```