Guide to using the Stark-Heegner point package

Adam Logan

Overview

This package is aimed at the numerical verification of the conjectures on "Stark-Heegner points" described in chapters 7 and 8 of [D]. Let us briefly recall the basic setup in those lectures. We suppose given an elliptic curve E with everywhere good reduction over a quadratic field F (this is more restrictive than what is assumed there, but we only consider these cases) and an almost totally real extension K of F—that is, we fix a real place ∞_0 of F and consider a quadratic extension K/F in which ∞_0 becomes complex while ∞_1, the other real place of F, splits into two real places.

From the theory of L-functions, then, we predict that the sign in the functional equation of $L(E, K)$ should be -1, so that the curve should have odd rank. On the other hand, the rank of E over F should be even. In other words, we expect that there should be a point of infinite order over K taken to its negative by the nontrivial automorphism of K/F.

The conjecture in chapter 8 of [D] proposes a way to find such a point. Namely, by evaluating certain integrals of the Hilbert modular form attached to the elliptic curve, one obtains a complex number z, and some multiple of it is expected to correspond to the desired point of infinite order under the Weierstrass uniformization of the curve. See [D] for more details, and [DL] for a full account of the calculations that were performed using this software.

Installation and basics

After downloading and unzipping, please begin by reading the enclosed license file (`license.txt`). Place all of the files into a directory: if you have a .gprc file specifying a path to GP data or program files, that might be a good choice. Otherwise, you should run GP from the directory containing the program files.

At present, only three curves are supported: one over each of the fields $\mathbf{Q}(\sqrt{29}), \mathbf{Q}(\sqrt{37}), \mathbf{Q}(\sqrt{41})$. Begin by entering the command `\rn`, where $n \in \{29, 37, 41\}$. GP will inform you that it is reading data for the field of your choice; it should soon be finished. The package defines some variables (including w, the standard generator of $\mathcal{O}_{\mathcal{F}}$) and a number of functions.

Using the package

The function of principal interest to most users is `test`, which attempts to test the conjecture. To use it, first write your ATR extension in the form $K = F(\sqrt{a + bw})$. Then invoke the command `test(a+b*y)`. (You may use your own variable name in place of `y`, except that `x` is inadmissible because of the way GP handles relative extensions of number fields.) GP will the do the following:

1. GP will initialize the number field K, find units in it, and make continued fractions for those units. It will decide on a unit likely to allow relatively rapid computation of the integral. At this stage you will be warned if the strict class number of K is greater than 1. Also, the program may choose to conjugate your generator to bring it into agreement with its conventions for the archimedean embeddings. This step produces a substantial amount of text (unless you are running in silent mode, for which see below).
2. The program will then construct and display the sequence of limits for evaluating the integrals.
3. Then the program will try to calculate the integrals of ω^+ along the path corresponding to the unit in question. By default it will go to about 14 places of accuracy. If the continued fraction is unfavourable, this can take an extremely long time. For each partial integral, the program will print a "progress report" every 500 terms, so that you know it is still awake. When it finishes a partial integral, it will print a message: `done one integral, returning` (a value). Every two limits in the sequence require the computation of eight partial integrals.
4. This done, the total value will be presented, and a warning message will appear unless the real part of this value is recognized as a small rational multiple of the real period of the curve. If the message does appear, one should reassure oneself that the ratio shown could in fact be a rational number: in case of unexpected loss of precision, the message could appear spuriously.

5. The program will then attempt to find a nontorsion point on E defined over F. Unfortunately the algorithm used falls somewhat short of the ideal, and sometimes it is necessary to find the point directly from the value of the integral.
6. The program will print the imaginary part of the integral, the imaginary part of the point on $\mathbf{C}/\Lambda$ corresponding to the point found by searching, and the imaginary period, and give a guessed linear dependence for these. In case of doubt, one may try to find the linear dependence with a slightly lower precision or find the integral to a higher accuracy.

In addition, no reasonable linear dependence may be expected when $h_F > 1$. In this case, the dependence will generally exist only with points defined over the Hilbert class field of F, and the program will not attempt to find these. Sometimes this can be done by hand, especially when the class group is an elementary abelian 2-group.

The other function that might be useful is `testminus`, which attempts to test the conjecture for the integral of ω^-. This is somewhat similar, but since the result of that calculation is expected to be a torsion point when the class number of the quartic field is equal to its narrow class number, the program simply attempts to recognize it as such rather than to compare it to a point of infinite order.

Both the `test` and `testminus` commands have two optional parameters. The first parameter indicates the requested precision (default = 14 digits); higher values of the second (up to 3) ask GP to print less text.

Examples

\r 29
test(2+y); \\2 + y is the fundamental unit. The relative discriminant has norm -16.
... *much deleted* ...
```
integration produced z-value -2.22688252024*I
searching produced z-value 5.17663896185*I
imaginary period is 34.2340042602*I
suggested linear dependence:  [-15, 20, -4]
```

test(2+y,40); \\requests 40-digit accuracy
... *much deleted* ...
```
integration produced z-value -2.2268825202458366276522208719429335632*I
searching produced z-value 5.176638961858435533262905837420423412 2*I
imaginary period is 34.234004260214065020010357456888117923*I
suggested linear dependence:  [-15, 20, -4]
```

testminus(2+y);
... *much deleted* ...
```
found z value -4.734809523373574118 E-15 - 13.693601704085626004843219 53*I by integration
periods:  [10.8794721724, 5.43973608624 + 17.1170021301*I]
suggested linear dependence on real and imaginary periods:  [-5, 2, -4]
```

test(19+9*y,25); \\this field has class number 2
... *much deleted* ...
```
integration produced z-value 0.654118639020392218072 31*I
searching produced z-value 4.83524613596531606066 21*I
imaginary period is 34.2340042602140650200 10*I
suggested linear dependence:  [-11856341, -20022231, 3054503]
```
findpoint(ev,y-1) \\generating the Hilbert class field
...
```
14.037060623267928880081297049953163671088907610074864712958962 2...*I
```
\p 28
lindep([0.654118639020392218072 31*I,4.835246135965316060662 21*I,34.234004260214065020010*I,
14.037060623267928880081297 0*I])
```
[15, 10, -14, 30]
```

Other functions

Some of the other functions in this package might be of use for other purposes. Here is a selection:
`cf(v):` where `v = [m,n]` and m, n are quads, this returns a continued fraction of m/n. This is not promised to work except when m and n are quads in $\mathbf{Q}(\sqrt{29})$, $\mathbf{Q}(\sqrt{37})$, or $\mathbf{Q}(\sqrt{41})$.
`ellaq(e,n,f,p,slow=0):` where `e` $= [0, a_2, 0, a_4, a_5]$, the polynomial f is irreducible over $\mathbf{Z}/p\mathbf{Z}$ and so defines a finite field of characteristic $p > 2$, and n is a nonresidue in this field, determines the number of points of the elliptic curve defined by `e` over the field $\mathbf{Z}[x]/(f, p)$ using Shanks-Mestre (much faster than Schoof in the range we need, and also a lot easier to code).
`findpoint(e,q):` tries (rather inefficiently) to find a point on the elliptic curve given by `e` over the quadratic extension of the field in use given by adjoining a square root of `q`.
`nfissquare(a,v):` returns the list of square roots of `v` in the number field `a`, where `a` is obtained from `*nfinit` of a polynomial in a variable other than `x` and `v` is a polynomial in the variable of `a`.
`qgcd(a,b):` returns a GCD of the quads a, b, assumed to be in one of the usual three fields.
`quadissquare(q):` returns the list of square roots of the quad `q`.
`sqrff(sq,nr,q):` takes a square root of `sq` in the finite field defined by the polynomial which is the base of the polymods `sq` and `nr`. For the programmer's convenience, `q` explicitly gives the order of the field; and `nr` is a nonsquare in the field, which can be found using `nonres`.

Bugs and potential extensions

1. A memory leak (possibly in GP itself, possibly in my program) prevents very long computations. On my 1200 mHz Pentium system with 256 M of memory, for example, after about 18 hours of computation the program spends most of its time paging and hardly any time computing.
2. Even in the least silent mode, the program should produce less text.
3. The point-finding routines should be made more effective. Also, they should try to divide points symbolically rather than numerically.
4. The program could ask for confirmation before embarking on what looks like a very long computation.
5. It would be nice if the program tried to deal automatically with at least some situations where $h > 1$, rather than leaving it all for the user.
6. The program would certainly be better if it didn't assume that the base field is quadratic. (Changing this could be a big job, though.)
7. Much of the code is, frankly, a mess, and rewriting it from scratch wouldn't be a bad idea.

References

[D] H. Darmon. *Rational points on modular elliptic curves.* NSF-CBMS notes. To appear.

[DL] H. Darmon, A. Logan. *Periods of Hilbert modular forms and rational points on elliptic curves.* To appear in IMRN.