Coleman Experiments

Sachi Hashimoto and Travis Morrison

December 5, 2019

Abstract

This shows how to use the Coleman Experiments library, which is tailored to batch run Picard curves to produce data for the paper "Chabauty Coleman computations on rank 1 Picard curves". Nonetheless, it has broader functionality that may be useful or adaptable in other scenarios.

Computing rational points and recognizing extra points

To load the code, one must the associated libraries for computing Coleman integrals Balakrishnan and Tuitman's `coleman.m` and our extension `colemanextension.m` available in a forked repository. The files must be loaded in order:

```
load "coleman.m";
load "colemanextensions.m";
load "parsing.m";
load "tests.m";
load "experiments.m";
```

The experiments code is designed to run the Chabauty-Coleman technique on large batch of Picard curves. As coded, it will run Chabauty-Coleman on the first split prime in the number field given by the known infinite order divisor associated to the curve.

To run a batch of curves, first make a text file on your computer with the curves, for example `curves.txt`. The parser reads in files with the format $\Delta : f : g$ where $y^3 - f$ defines a plane model for the Picard curve, Δ is the discriminant of the curve, and $\mathrm{Div}(g) - \deg(g)\infty$ is a known infinite order divisor. Here is a snippet of our text file of Picard curves:

```
47258883:x^4 + 7*x^3 + 8*x^2 - 15*x + 4:x^2 + 4*x - 1
47258883:x^4 + x^3 - 3*x^2 + 1:x
70858800:x^4 + 3*x^3 - x^2 - 4*x + 2:x - 1
83298456:x^4 + 3*x^3 - 16*x^2 + 20*x - 8:x
```

The following command will run Chabauty-Coleman on this file

```
batch_extras("curves.txt","output.txt","outputwithbvals.txt");
```

where one should replace the file names with paths to the files as appropriate. The program will output a file with b-values and one without, this is what the two output files look like. Without b-values, we list the rational points, any p-adic points along with the p and precision, and then a list of p, precision and e at which effective Chabauty succeeded:

```
47258883:x^4 + 7*x^3 + 8*x^2 - 15*x + 4:x^2 + 4*x - 1:[[ 0, 1, 0 ],[ -4, 0, 1 ]]:[]:
[ 11, 15, 40 ]
47258883:x^4 + x^3 - 3*x^2 + 1:x:[[ 0, 1, 0 ],[ 0, 1, 1 ],[ 1, 0, 1 ]]:[]:[ 5, 15, 40 ]
70858800:x^4 + 3*x^3 - x^2 - 4*x + 2:x - 1:[[ 1, 1, 1 ],[ 0, 1, 0 ]]:[]:[ 7, 15, 60 ]
83298456:x^4 + 3*x^3 - 16*x^2 + 20*x - 8:x:[[ 0, 1, 0 ],[ 0, -2, 1 ],[ 2, 2, 1 ],[ 1, 0, 1 ]]:
[[ 7580017738, 5, 15 ]]:[ 5, 15, 40 ]
```

And with b-values:

```
47258883:x^4 + 7*x^3 + 8*x^2 - 15*x + 4:x^2 + 4*x - 1:[[ 0, 1, 0 ],[ -4, 0, 1 ]]:[]:
[ 11, 15, 40 ]:[]
47258883:x^4 + x^3 - 3*x^2 + 1:x:[[ 0, 1, 0 ],[ 0, 1, 1 ],[ 1, 0, 1 ]]:[]:[ 5, 15, 40 ]:[]
70858800:x^4 + 3*x^3 - x^2 - 4*x + 2:x - 1:[[ 1, 1, 1 ],[ 0, 1, 0 ]]:[]:[ 7, 15, 60 ]:[]
83298456:x^4 + 3*x^3 - 16*x^2 + 20*x - 8:x:[[ 0, 1, 0 ],[ 0, -2, 1 ],[ 2, 2, 1 ],[ 1, 0, 1 ]]:
[[ 7580017738, 5, 15 ]]:[ 5, 15, 40 ]:[[ 1 + O(5^15), O(5^7), O(5^14) ]]
```

one can see that the p-adic point found in the last curve is a 5-adic Weierstrass point, since the second b-value is the y-coordinate and is 0.

Batch extras batch computes the function `extras` which can be called on a single piece of curve data like so:

```
>c:=curve_line_parser("4920750000:x^4 + x^3 - 3*x^2 - 2*x:x + 1");
>extras(c,parameters);
> extras(c,parameters);
true [
    [ 58207, 7, 6 ]
]
[
    rec<recformat<x, b, inf, xt, bt, index> |
        x := 0,
        b := [ 1 + O(7^15), 0, 0 ],
        inf := false>,
    rec<recformat<x, b, inf, xt, bt, index> |
        x := 2 + O(7^15),
        b := [ 1 + O(7^15), 2 + O(7^15), 4 + O(7^15) ],
        inf := false>,
    rec<recformat<x, b, inf, xt, bt, index> |
        x := -2 + O(7^15),
        b := [ 1 + O(7^15), 0, 0 ],
        inf := false>,
    rec<recformat<x, b, inf, xt, bt, index> |
        x := -1 + O(7^15),
        b := [ 1 + O(7^15), -1 + O(7^15), 1 + O(7^15) ],
        inf := false>,
    rec<recformat<x, b, inf, xt, bt, index> |
        x := 0,
        b := [ 1 + O(7^15), 0, 0 ],
        inf := true>
]
[
    [ 1 + O(7^15), -2288 + O(7^6), 58388 + O(7^6) ]
]
[ 7, 15, 40 ]
```

The `curve_line_parser` turns the string into a list. The `parameters` are an Associative Array which are set at the top of the experiments document and can be tweaked in the document or terminal according to the user's particular data set or time constraints:

```
> parameters["e"];
40
> parameters["precision"];
15
> parameters["max_e"];
100
> parameters["max_prec"];
25
> parameters["height"];
1000
```

```
> parameters["e_inc"];
20
> parameters["precision_inc"];
5
```

The function `extras` works by first calling the `effective_chabauty_incremental` function, which attempts to run either `effective_chabauty` or `effective_chabauty_with_Qdiv` for the specified parameters `parameters["e"]` and `parameters["precision"]`. If Chabauty fails, it increases e by the e_{inc} value until it is greater than max_e, at which point it resets e and starts to increase the precision, searching in a box for parameters that will succeed.

The height parameter is used to search for rational points on the Picard curve up to a naive height bound. After running Chabauty, extras will run the `compare` function to compare p-adic points with found points.

```
> data:=coleman_data(y^3-f,7,10);
> L,v:=effective_chabauty(data:bound:=1000,e:=50);
> Qpoints:=Q_points(data,1000);
> compare(Qpoints,L);
[ 7786 + O(7^5) ]
[
    [ 1 + O(7^10), -2288 + O(7^5), 7967 + O(7^5) ]
]
```

On this example we have one extra unexplained 7-adic point. We can use the functionality of `tests.m` to understand what kind of point it is. Taking the `extras` data from above we can run

```
> P:=[ 58207, 7, 6 ];
> ws_test(f,P);
false
> torsion_test(f,P,parameters);
false
```

We can use `algebraic_test` to try to realize P as an algebraic point and check for relations in $J(\mathbb{Q})$. This searches for relations smaller than a *deg* parameter.

```
> f:=x^4 + x^3 - 3*x^2 - 2*x;
> x0:=58207;
> Qp:=pAdicField(7,6);
> x0:=Qp!x0;
> deg:=4;
> ht:=100;
> rbound:=5;
> tors:=false;
> algebraic_test(f,x0,deg,ht,rbound,tors:skip_3tor:=true);
[ -3, -5, 3 ]
[
    Place at (2 : 2 : 1),
    Place at (-1 : -1 : 1),
    Place at (a : a^2 + 2*a : 1)
]
x^3 + 3*x^2 + 1
```

The output tells us that our p-adic point P is the algebraic point $(a, a^2 + 2a)$ where a satisfies the minimal polynomial $x^3 + 3x^2 + 1$. Furthermore, because we set `tors := false` (to mark that P is non-torsion) `algebraic_test` has called the function `reln_search` given with parameters `ht`, `rbound` and optional parameter `skip3_tor` to search for the linear relations in $J(\mathbb{Q})$, and it found $-3(2:2:1) - 5(-1:-1:1) + 3(a : a^2 + 2a : 1)$ is principal. We can see how this works by calling the function `reln_search` separately:

```
> K<a>:=NumberField(x^3 + 3*x^2 + 1);
> R:=[a,a^2+2*a,1];
```

```
> reln_search(f,R,K,ht,rbound:skip_3tor:=true);
[ -3, -5, 3 ]
[
    Place at (2 : 2 : 1),
    Place at (-1 : -1 : 1),
    Place at (a : a^2 + 2*a : 1)
]
```

The height gives a bound to search for rational points to use in relations and the rbound gives an absolute value bound up to which to search for coefficients in the relation. The optional parameter `skip_3tor` allows one to skip three torsion points in iterating over possible relations and often makes the search much faster.

For recognizing torsion points, `tests.m` has several functions available. Once we have identified a point as torsion `algebraic_test` with `tors:=true` will automatically look for a minimal polynomial for the x-coordinate of the point and then try to identify the torsion order, like so:

```
> f:=x^4 + 9*x^3 - 30*x^2 + 28*x - 8;
> Qp:=pAdicField(5,6);
> x0:=Qp!2;
> rbound:=10;
> algebraic_test(f,x0,deg,ht,rbound,true);
9 Place at (2 : a : 1)
x - 2
```

This tells us that $(2, (16)^{1/3})$ is a 9-torsion point on $y^3 - f$. To compute the order of a torsion point, `algebraic_test` calls `torsion_order`, which can be called on its own:

```
> f:=x^4 + 12*x^3 - 18*x^2 + 8*x - 1;
> L<a>:=NumberField(x^3-2);
> R:=[1,a,1];
> torsion_order(f,R,L,10);
4 Place at (1 : a : 1)
```

Most usefuly, one can batch analyze the output of `batch_extras`, like so:

```
> batch_analyze("outputwithbvals.txt","all.txt","intersting.txt","oldfailures.txt",
"newfailures.txt":bvals:=true);
```

will create four output files which run the `tests.m` suite of functions. (Similarly one can run the functions on the output without b-values. The optional parameter `bvals` is by default false.)

This will append three lists to the previous extras data, first one with Weierstrass points, then one with torsion points, and finally one with non-torsion points that have linear relations to rational points in $J(\mathbb{Q})$. If the tests fail, for example, if the integrals in torsion tests fail or if searching for linear relations in a box does not yield any, it will output "failure". Here is an example of the output for a file without b-values.

```
4897760256:x^4 + 12*x^3 + 15*x^2 - 36*x - 54:x^2 + 7*x + 9:[[ 0, 1, 0 ]]:
[[ -337806862225525081, 17, 15 ],[ 337806862225525069, 17, 15 ]]:[ 17, 15, 140 ]:
[[ -337806862225525081, 17, 15 ],[ 337806862225525069, 17, 15 ]]:[* *]:[* *]
4920750000:x^4 + x^3 - 3*x^2 - 2*x:x + 1:[[ 0, 1, 0 ],[ 0, 0, 1 ],[ 2, 2, 1 ],[ -2, 0, 1 ],
[ -1, -1, 1 ]]:[[ 58207, 7, 6 ]]:[ 7, 15, 40 ]:[]:[* *]:[*[ 58207, 7, 6 ],[ -3, -5, 3 ],
[Place at (2 : 2 : 1),Place at (-1 : -1 : 1),Place at (a : a^2 + 2*a : 1)],x^3 + 3*x^2 + 1*]
4920750000:x^4 + x^3 + x - 1:x:[[ 0, 1, 0 ],[ 0, -1, 1 ]]:[[ -631831049, 7, 11 ]]:
[ 7, 15, 40 ]:[]:[* *]:[*[ -631831049, 7, 11 ],[ 1, 3 ],[Place at (0 : -1 : 1),
Place at (a : -a^2 - 1 : 1)],x^3 + 3*x + 1*]
```

The "all" output file has all of the data. The "interesting" output file has all curves which have recognizable extra points which are non-Weierstrass. The "oldfailures" file compiles curves which failed during the `batch_extras` phase and the "newfailures" file compiles curves which failed during the `batch_analyze` phase so that the user can run them again manually.

Finally, there is a parsing file parsing.m which has some parsing functionality for colon-separated strings used without the code. All functions and procedures have been documented throughout the code, and contain short explanations of their use.