Beyond the Sato-Tate Conjecture

William Stein and Chris Swierczewski (joint work with Barry Mazur)

24 April, 2007

1 Some Functions

- $X_a^b(T) = \dfrac{\int_a^T \sqrt{1-x^2}dx}{\int_a^b \sqrt{1-x^2}dx} =$ area under arc of semicircle
- $Y_{a,C}^b(T) = \dfrac{\#\{\text{primes } p < C : a < \frac{a_p}{2\sqrt{p}} < T\}}{\#\{\text{primes } p < C : a < \frac{a_p}{2\sqrt{p}} < b\}}$.
- $\Delta_a^b(C) = \displaystyle\int_a^b (X_a^b(T) - Y_{a,C}^b(T))^2 dT =$ the L_2 norm of the difference of X_a^b and $Y_{a,C}^b$.
- $e_a^b(C) = -\log_C(\Delta_a^b(C)) = -\log(\Delta_a^b(C))/\log(C)$ - Measures behavior of $\Delta_a^b(C)$ as a function of C.
- $\epsilon_a^b(E) = \limsup_{C\to+\infty} e_a^b(C)$ - The lim sup of e functions.

2 The Code

We compute the data on the convergence of the $\Delta_a^b(C)$ function using the following code.

```
#auto
from math import asin, log, sqrt

def line1(xmin,xmax):
    return line([(xmin,1),(xmax,1)], rgbcolor=(1,0,0))

def Xab(a,b):
    bb = (asin(b)/2r + b*sqrt(1r-b^2r)/2r)
    aa = (asin(a)/2r + a*sqrt(1r-a^2r)/2r)
    def X(T):
        return (asin(T)/2r + T*sqrt(1r-T^2r)/2r - aa)/(bb - aa)
    return X
```

The Sato-Tate conjecture implies that the values following function, $Y_{a,C}^b(T)$ as C approaches infinity, is equal to $X_a^b(T)$. $X_a^b(T)$ has a closed-form definite integral shown in the code above. In `SAGE`, we define a `SatoTate` class determined by a chosen elliptic curve E.

```
import bisect

class SatoTate:
    def __init__(self, E):
        self._E = E

    def __repr__(self):
        return "Sato-Tate data for ..."
```

```
def anlist(self, n):
    return self._E.anlist(n)

def normalized_aplist(self, n):
    anlist = self.anlist(n)
    two = float(2)
    v = [float(anlist[p])/(two*sqrt(p)) for p in prime_range(n)]
    return v

def sorted_aplist(self, n):
    v = self.normalized_aplist(n)
    v.sort()
    return v

def YCab(self, Cmax, a=-1, b=1):
    v = self.sorted_aplist(Cmax)

    denom = bisect.bisect_right(v, float(b)) - bisect.bisect_left(v, float(a))
    try:
       normalize = float(1)/denom
    except:
       def Y(T):
           return 1.0r
       return Y
    start_pos = bisect.bisect_left(v, float(a))

    def Y(T):
        # find position that T would go in if it were inserted
        # in the sorted list v.
        n = bisect.bisect_right(v, float(T)) - start_pos
        return n * normalize
    return Y
```

In the definition of the function $Y_{a,C}^{b}(T)$, we create a sorted a_p list from the attached elliptic curve where a_p is defined by

$$a_p = p - 1 - \#E(\mathbb{F}_p) \tag{1}$$

where $\#E(\mathbb{F}_p)$ is the number of points on an elliptic curve E over the finite field $\mathbb{F}_p$. After normalization, a function is returned that computes the number of elements in the list between a and T. The `bisect_left` and `bisect_right` commands are a fast way of finding these upper and lower bound elements.

```
def xyplot(self, C, a=-1, b=1):
    """
    Return the quantile-quantile plot for given a,b, up to C.
    """
    Y = self.YCab(C,a=a,b=b)
    X = Xab(a=a,b=b)
    pX = plot(X, a, b, rgbcolor=(1,0,0))
    pY = plot(Y, a, b, rgbcolor=(0,0,1))
    return pX + pY

def qqplot(self, C, a=-1, b=1):
    """
    Return the quantile-quantile plot for given a,b, up to C.
    """
    Y = self.YCab(C,a=a,b=b)
```

```
    X = Xab(a=a,b=b)
    pl = parametric_plot((X, Y), a,b)
    ll = line([(0,0), (1.1,1.1)], rgbcolor=(1,0,0))
    return pl+ll

def Delta(self, C, a, b, max_points=300):
    """
    Delta_{a}^{b} function:
    INPUT: C - cutoff
         a,b - evaluate over the interval (a,b)
         max_points - number of points used in numerical integral
    """
    key = (C,a,b,max_points)
    try:
       return self._delta[key]
    except AttributeError:
       self._delta = {}
    except KeyError:
       pass
    X = Xab(a,b)
    Y = self.YCab(C,a,b)
    def h(T):
        return (X(T) - Y(T))^2r

    val, err = integral_numerical(h, a, b, max_points=max_points, algorithm='qag', rule=1,
    eps_abs=1e-10, eps_rel=1e-10)

    self._delta[key] = (val, err)
    return val, err

def theta(self, C, a=-1, b=1, max_points=300):
    val, err = self.Delta(C, a, b, max_points=max_points)
    return -log(val)/log(C), val, err

def theta_interval(self, C, a=-1, b=1, max_points=300):
    val, err = self.Delta(C, a, b, max_points=max_points)
    return -log(val-abs(err))/log(C), -log(val+abs(err))/log(C)
```

Here we define the $\Delta_a^b(C)$ function as an L_2-norm on the difference of the $X_a^b(T)$ and $Y_{a,C}^b(T)$ functions:

$$\Delta_a^b(C) = \int_a^b (X_a^b(T) - Y_{a,C}^b(T))^2 dT \tag{2}$$

We use an adaptive numerical integration scheme (which is part of the GSL library, and which Josh Kantor made easy to use in SAGE) which allows us to (hopefully) put a bound on the error that is likely to crop up. Another function called `theta` returns the value and error of the `delta` function along with $e_a^b(C) = -\log(\Delta_a^b(C))/\log(C)$. These $e_a^b(C)$ values are what we compute and plot in the following code.

```
def compute_theta(self, Cmax, plot_points=30, a=-1, b=1, max_points=300, verbose=False):
    a,b = (float(a), float(b))
    def f(C):
       z = self.theta(C, a, b, max_points=max_points)
       if verbose: print C, z
       return z[0]
    return [(x,f(x)) for x in range(100, Cmax, int(Cmax/plot_points))]

def compute_theta_interval(self, Cmax, plot_points=30, a=-1, b=1, max_points=300, verbose=False):
```

```
        a,b = (float(a), float(b))
        vmin = []; vmax = []
        for C in range(100, Cmax, int(Cmax/plot_points)):
            zmin,zmax = self.theta_interval(C, a, b, max_points=max_points)
            vmin.append((C, zmin))
            vmax.append((C, zmax))
            if verbose: print C, zmin, zmax
        return vmin, vmax

    def plot_theta_interval(self, Cmax, clr=(0,0,0), *args, **kwds):
        vmin, vmax = self.compute_theta_interval(Cmax, *args, **kwds)
        v = self.compute_theta(Cmax, *args, **kwds)
        grey = (0.7,0.7,0.7)
        return line(vmin,rgbcolor=grey)+line(vmax,rgbcolor=grey)
              + point(v,rgbcolor=clr) + line(v,rgbcolor=clr) + line1(0, Cmax)

    def histogram(self, Cmax, num_bins):
        v = self.normalized_aplist(Cmax)
        d, total_number_of_points = dist(v, num_bins)
        return frequency_histogram(d, num_bins, total_number_of_points) + semicircle

    def x_times_Delta(self, x):
        return x*self.Delta(x, -1,1, max_points=500)[0]
```

The code above handles the plot generation of the `theta` function. Along with the raw data, we plot an "error tube" using the error terms returned by the numerical integration scheme, allowing us to see threshold under where the actual value of $e_a^b(C)$ lies. We also can compute a frequency histogram of the a_p's as computed above.

3 Plots

The following plots are a sample of what we generated in `SAGE` using the above code. We plot the following for various large values of C

- Frequency histograms of the a_p's
- Quantile–Quantile plots of X_a^b and $Y_{a,C}^b$
- $e_a^b(C)$ over various intervals.
- Quantile–Quantile and $e_a^b(C)$ plots for Elkies elliptic curve of rank at least 28

Each figure below contains eight plots. Each plot displays the data for elliptic curves of different rank:

Rank 0	Rank 1	Rank 2
Rank 3	Rank 4	Rank 5
Rank 6	Rank 7	Rank 8

Some observations:

- As rank increases, the histograms skew more to the left. This is due to the larger number of negative a_p's is higher rank curves. This behavior is reflected in the Quantile-Quantile plots as "buldge" diverging from the 45 degree line.
- **Conjecture (Mazur?, Stein, Swierczewski)** $1/2 \leq \epsilon_a^b(E) \leq 1$ for all elliptic curves E.

Figure 1: Frequency Histograms, $C = 10^3, (a, b) = (-1, 1)$

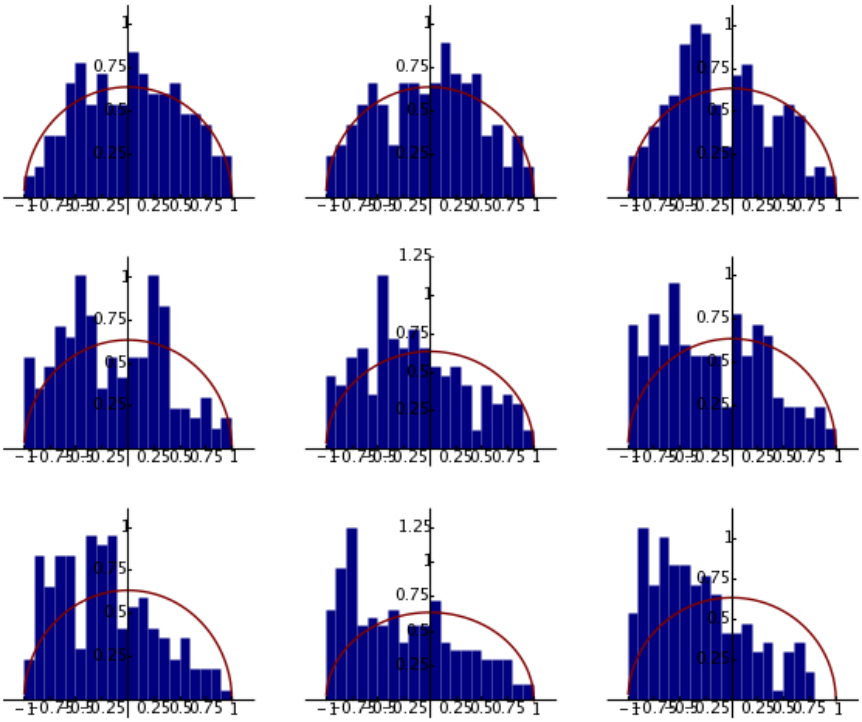

Figure 2: Frequency Histograms, $C = 10^4, (a, b) = (-1, 1)$

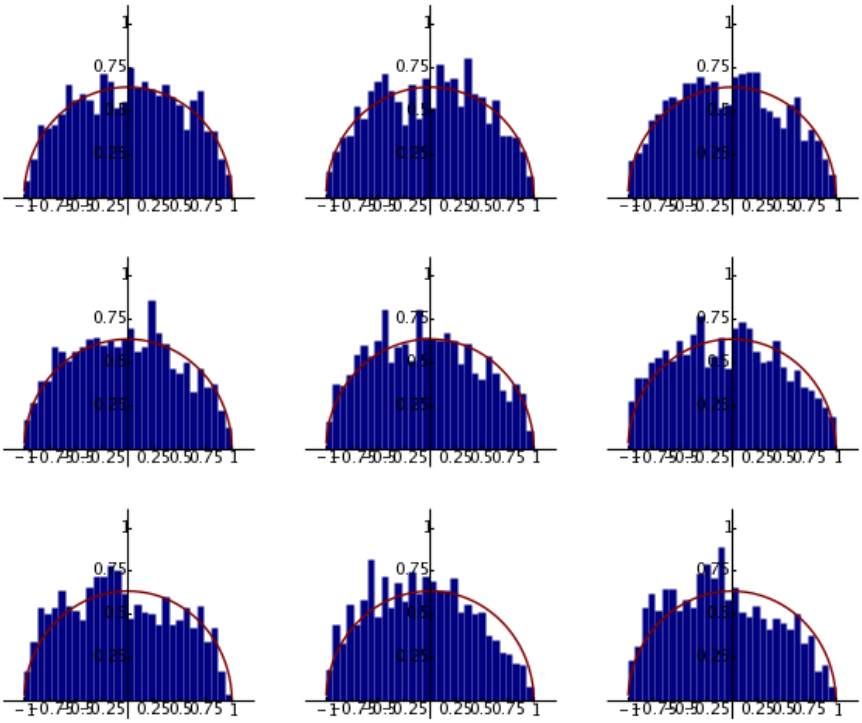

Figure 3: Frequency Histograms, $C = 10^5, (a, b) = (-1, 1)$

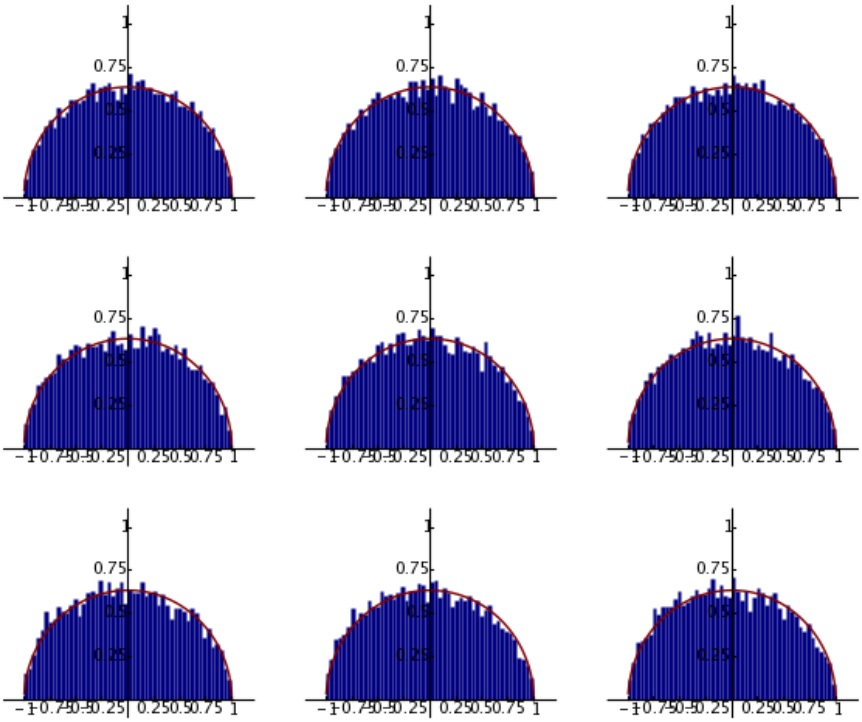

Figure 4: Frequency Histograms, $C = 10^5, (a, b) = (-1, 1)$

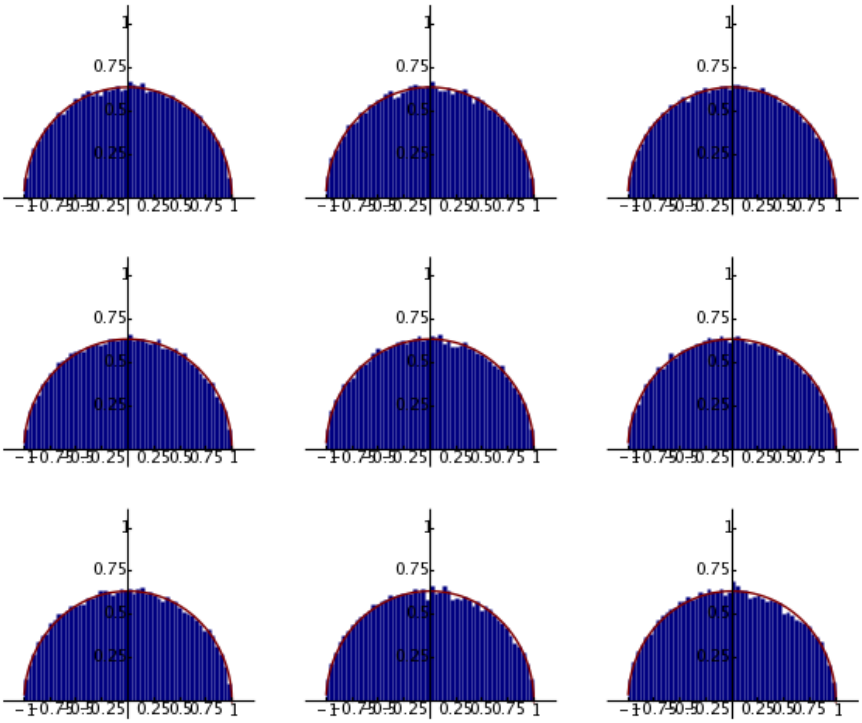

Figure 5: Quantile-Quantile plot for X_a^b and $Y_{a,C}^b$, $C = 10^2, (a, b) = (-1, 1)$

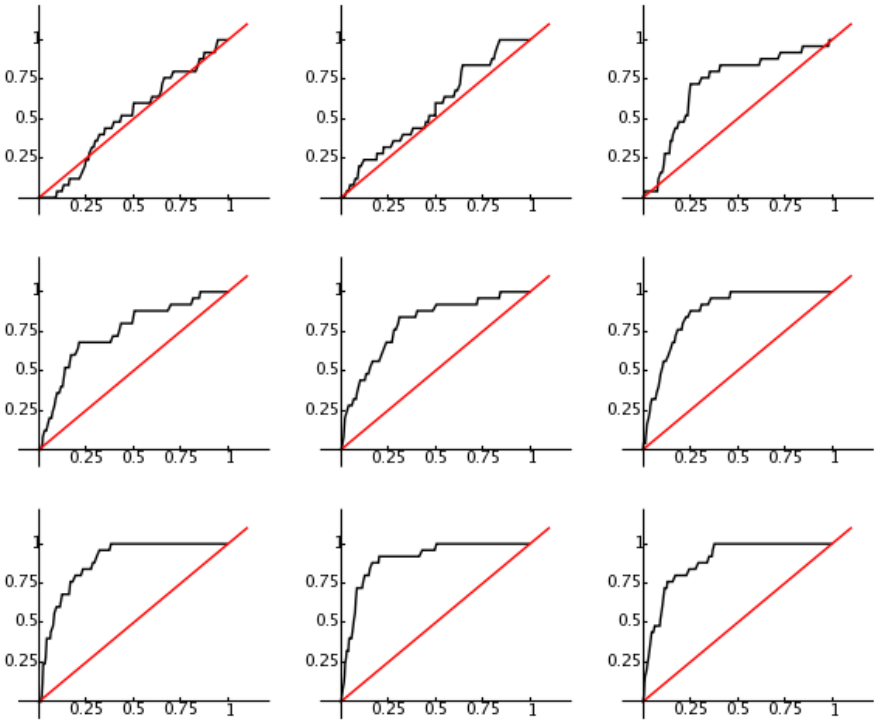

Figure 6: Quantile–Quantile plot for X_a^b and $Y_{a,C}^b$, $C = 10^3, (a, b) = (-1, 1)$

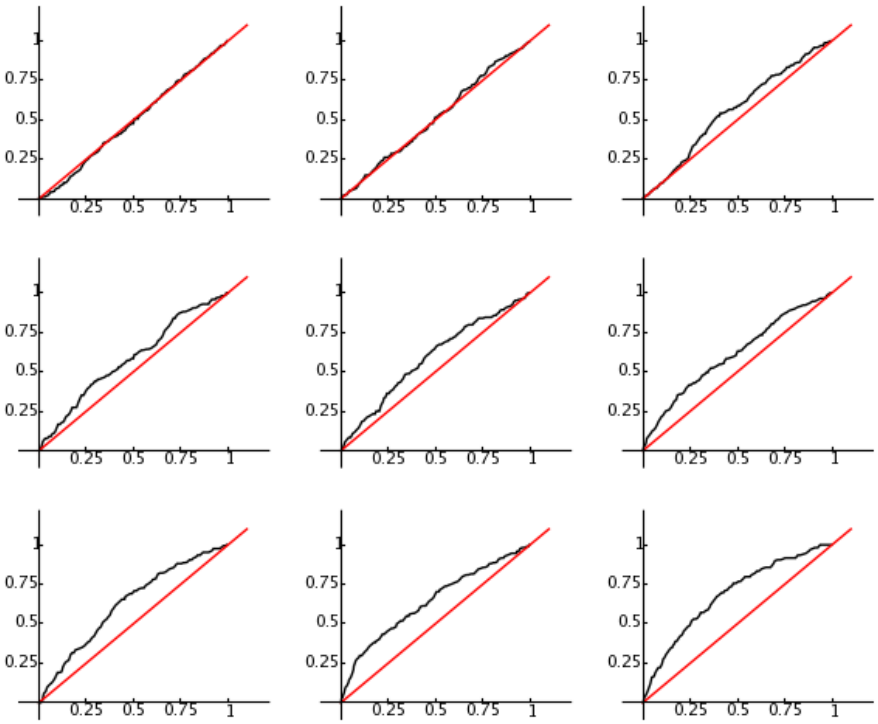

Figure 7: Quantile-Quantile plot for X_a^b and $Y_{a,C}^b$, $C = 10^5, (a, b) = (-1, 1)$

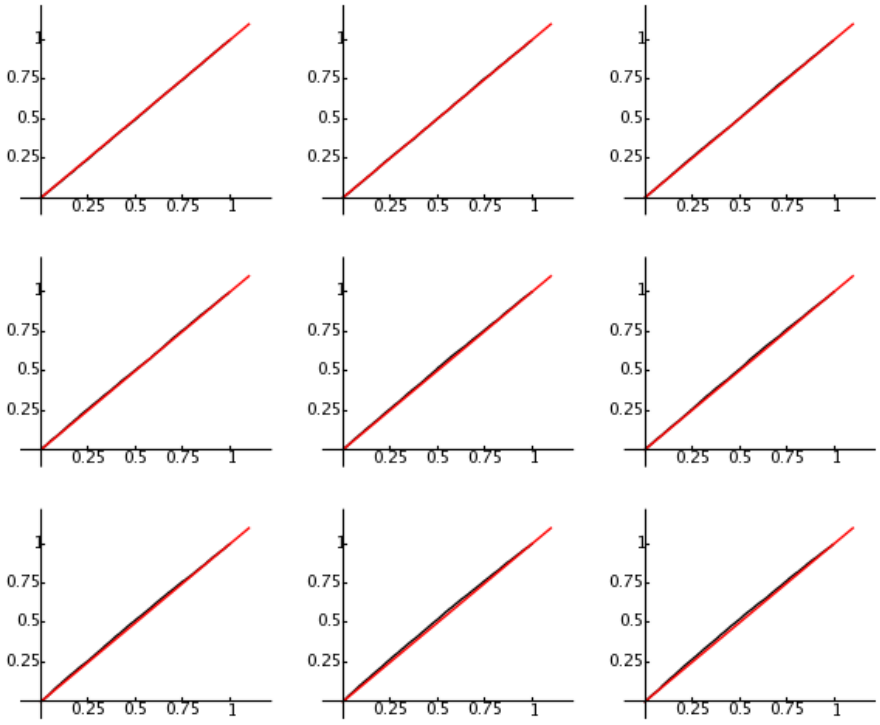

Figure 8: Quantile-Quantile plot for X_a^b and $Y_{a,C}^b$, $C = 10^6, (a, b) = (-1, 1)$

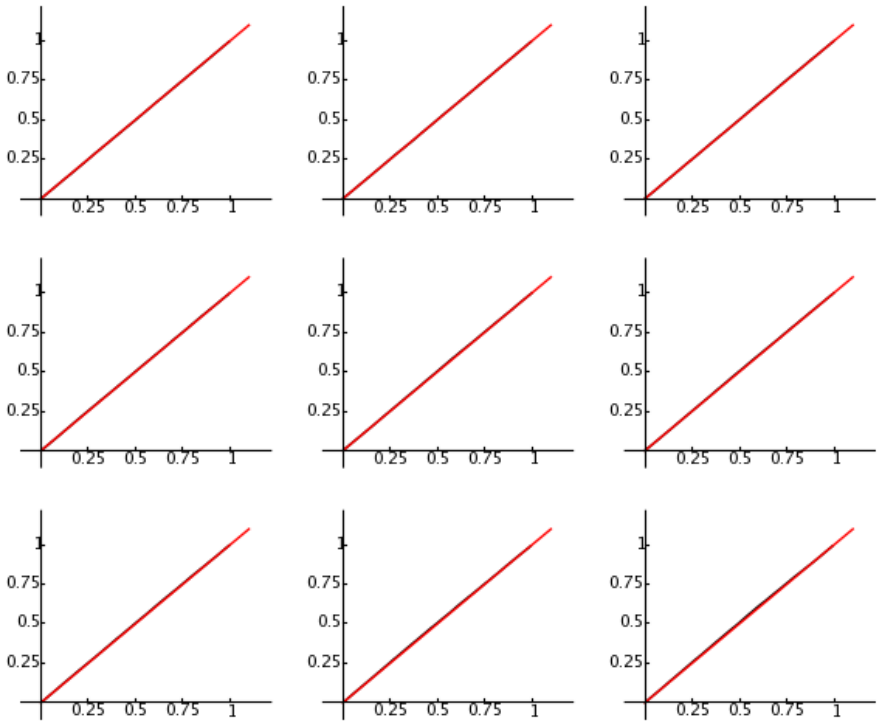

Figure 9: $e_a^b(C) = -\log_C(\Delta_a^b(C))$ plots, $C = 10^3, (a, b) = (-1, 1)$

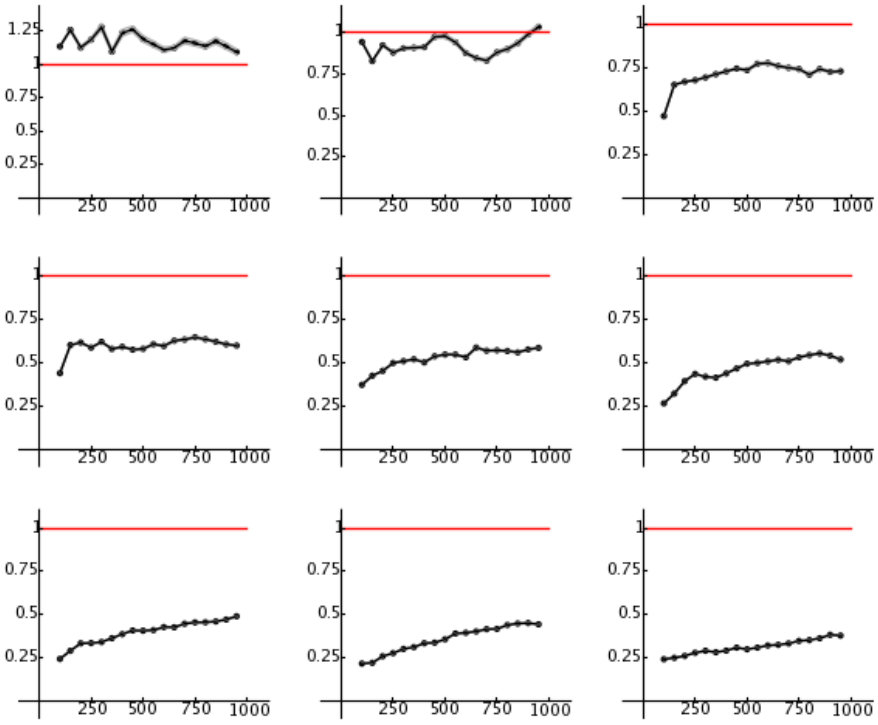

Figure 10: $e_a^b(C) = -\log_C(\Delta_a^b(C))$ plots, $C = 10^5, (a, b) = (-1, 1)$

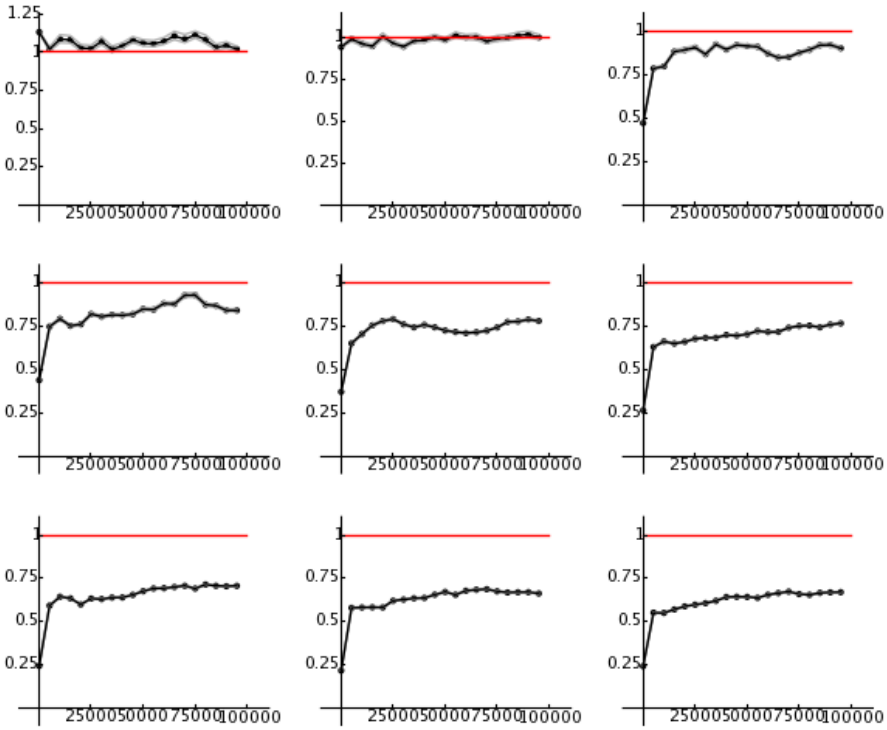

Figure 11: $e_a^b(C) = -\log_C(\Delta_a^b(C))$ plots, $C = 10^6, (a, b) = (-1, 1)$

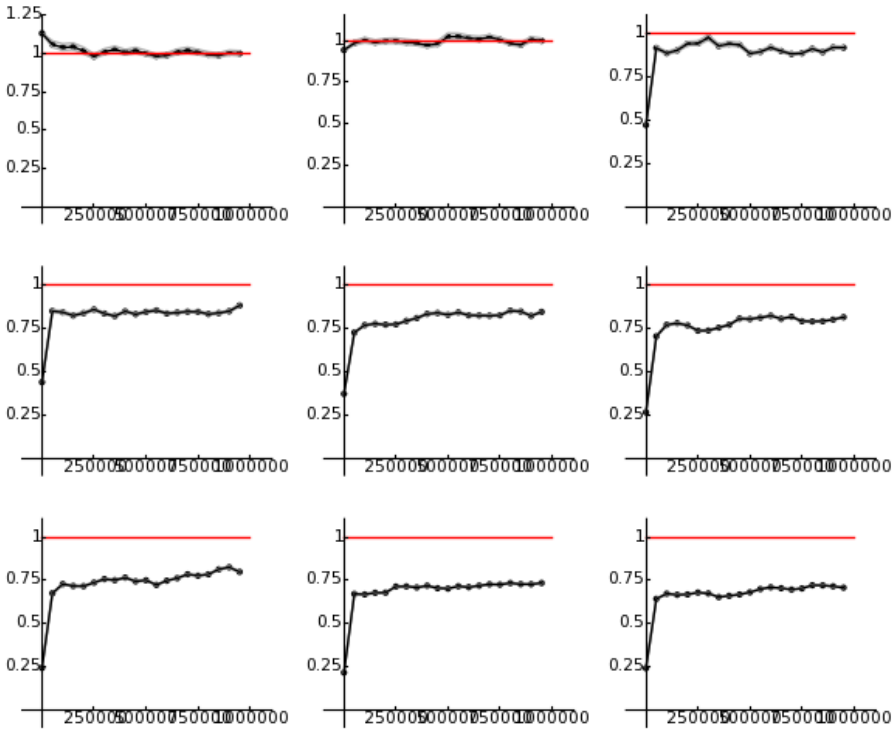

Figure 12: $e_a^b(C) = -\log_C(\Delta_a^b(C))$ plots, $C = 10^5, (a, b) = (-0.01, 0.01)$

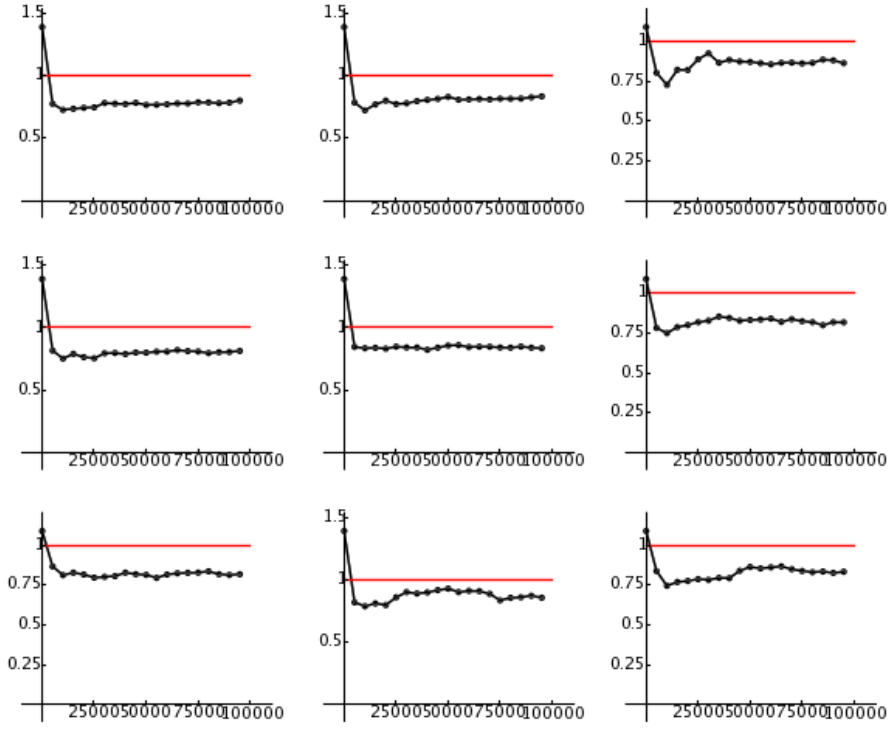

Figure 13: $e_a^b(C) = -\log_C(\Delta_a^b(C))$ plots, $C = 10^5, (a, b) = (-1, -0.5)$

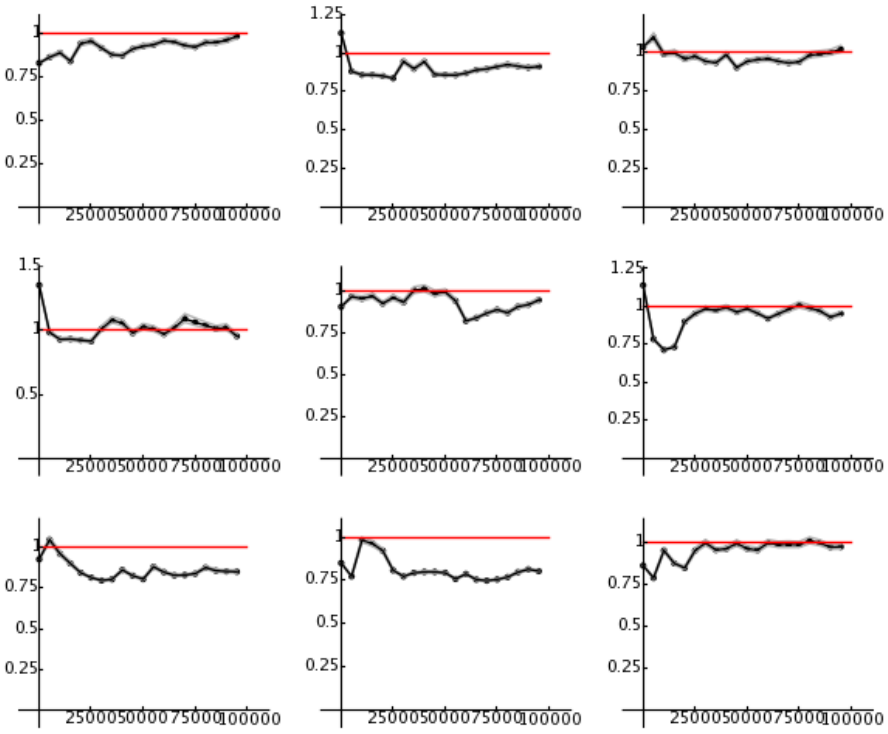

Figure 14: $e_a^b(C) = -\log_C(\Delta_a^b(C))$ plots, $C = 10^5, (a, b) = (0.5, 1)$

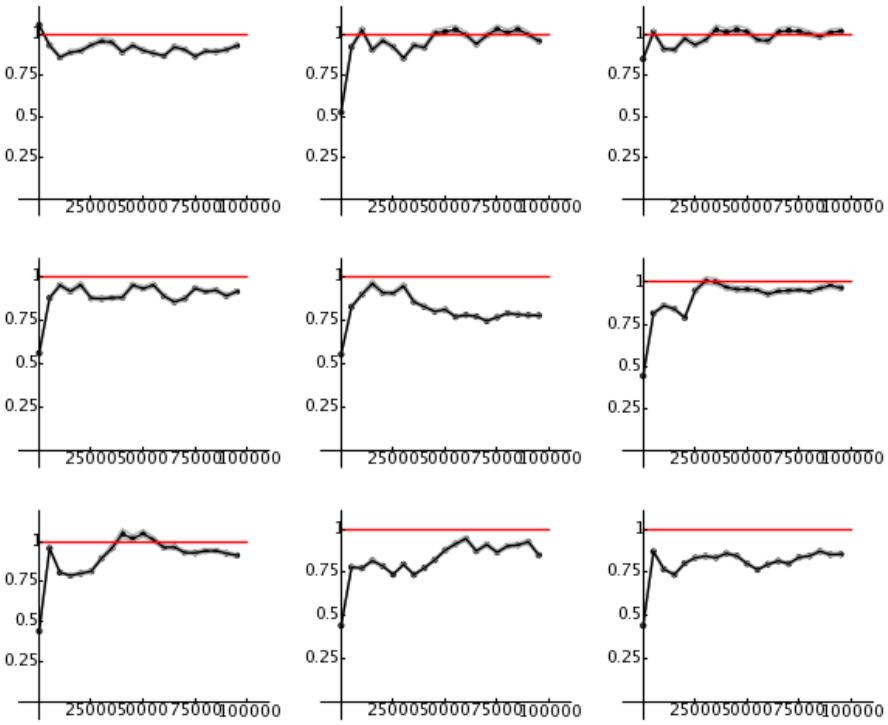

Figure 15: Quantile-Quantile plot for an Elliptic Curve of Rank ≥ 28 for $C = 10^2, ..., 10^6$

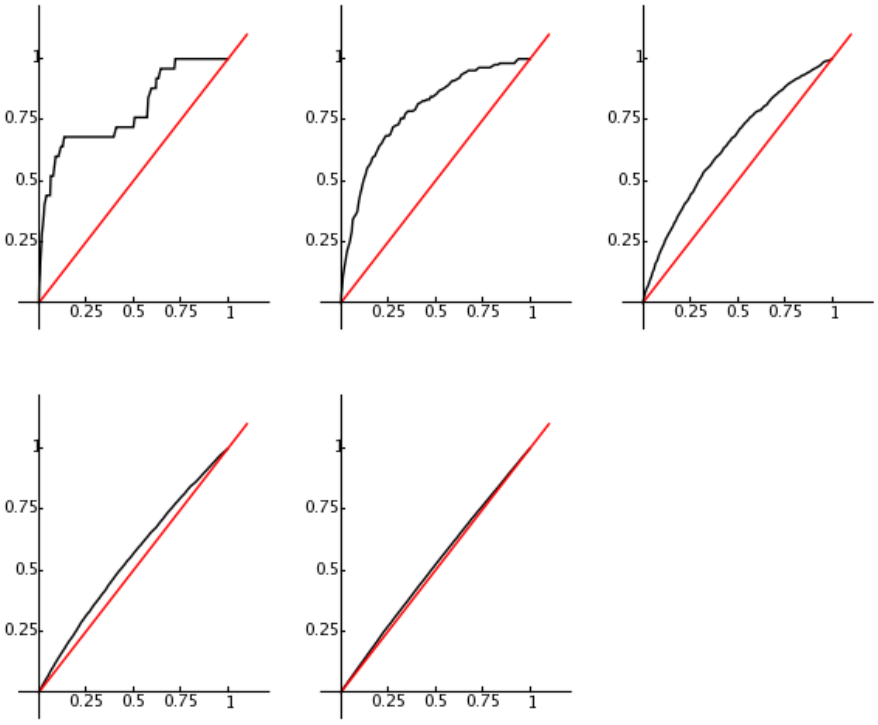

Figure 16: $e_a^b(C)$ plot for an Elliptic Curve of Rank ≥ 28 for $C = 10^3, 10^4, 10^5, 10^6, 2 \cdot 10^6, 3 \cdot 10^6$