Computing with Elliptic Surfaces: GP/PARI Demo

Noam Elkies

December 31, 2011

```
\\-----------------------------------------------------------------\\
\\       introducing some of gp's elliptic-curve functionality      \\
\\-----------------------------------------------------------------\\

\\ Ask gp what the function "ellinit" does
?ellinit

\\ the elliptic curve with a1=a2=a3=0 and prescribed c4,c6
E1(c4, c6) = ellinit([0, 0, 0, -c4/48, -c6/864])

\\ here's the syntax for getting the discriminant and j-invariant
E1(C4,C6).disc
E1(C4,C6).j

\\ Mazur's celebrated torsion theorem also holds -- and is much easier to prove
\\ -- not just for elliptic surfaces over Q(T) [where it's also a consequence
\\ by specialization] but even for *nonconstant* elliptic surfaces over R(T)
\\ where R = field of real numbers, because (once the group is not contained
\\ in the 2-torsion) such a surface is tantamount to a map from P^1 to the
\\ modular curve parametrizing elliptic curves with the given torsion subgroup,
\\ and that curve is rational only for the groups in Mazur's list: cyclic of
\\ order 1,2,...,10 or 12, and 2x2n for n=1,2,3,4.  Over C(T) there are three
\\ further possibilities that can't arise over R, namely 4x4, 3x6, and 5x5.
\\
\\ We next give explicitly for each n=2,3,...,10 the universal
\\ elliptic curve E with an n-torsion point P.  We call these
\\ E2, E3, ..., E10.  In each case P is at [0,0] (<==> a6=0),
\\ and for n>2 the tangent line at P is horizontal (<==> a4=0).
\\ for n=2,3,4 the digits 1,2,3,4 in a1,a2,a3,a4 are weights; thus
\\ for instance E3(a1,a3) \isom E3(c*a1,c^3*a3).  For n>4 the parameters
\\ a,b are homogeneous of the same weight, so we have the universal curve
\\ over the projective (a:b)-line X_1(n).  These formulas, or equivalent ones,
\\ are all "well-known" but I don't know of a standard place that lists
\\ them all.
```

```
E2(a2,a4) = ellinit([0, a2, 0, a4, 0])

E3(a1,a3) = ellinit([a1, 0, a3, 0, 0])

E4(a1,a2) = ellinit([a1, a2, a1*a2, 0, 0])

E5(a,b) = ellinit([a+b, a*b, a*b^2, 0, 0])
E6(a,b) = ellinit([a-b, -a*(a+b), a*b*(a+b), 0, 0])

E7(a,b) = ellinit([a^2+a*b-b^2, a*b^2*(a-b), a^3*b^2*(a-b), 0, 0])
\\ for E7 see [Tate 1966], with b/a being Tate's parameter d

E8(a,b) = ellinit([a^2+2*a*b-b^2, a^2*b*(a-b), a^3*b*(a^2-b^2), 0, 0])

\\ use "\" at the end of a line to continue a long formula ...
E9(a,b) = ellinit([a^3-a^2*b-b^3, \
   a^2*b*(b-a)*(a^2-a*b+b^2), \
   a^2*b^4*(a-b)*(a^2-a*b+b^2), \
   0, 0])

\\ or enclose it with {...}
{
E10(a,b) = ellinit([2*a^3-2*a^2*b-2*a*b^2+b^3,
  a^3*b*(b-a)*(2*a-b),
  a^3*b^2*(b-a)*(2*a-b)*(a^2-3*a*b+b^2),
  0,0])
}

\\ For E9 and E10 see e.g. Lemmermeyer,
\\ www.fen.bilkent.edu.tr/~franz/ta/ta18.pdf,
\\ Prop.6 and 7 with d=a/b

\\ note that these [a1,a2,a3,0,0] are much simpler than the
\\ usual "narrow Weierstrass" form [0,0,0,-c4/48,-c6/864].
\\ For many purposes it is better to work with these extended,
\\ albeit less canonical, formulas; not only are they more appealing,
\\ but they also tend to show more of the relevant structure (we'll see
\\ some examples soon), and to make some computations easier as well.
\\ For example, here's a curve with a 10-torsion point whose conductor
\\ is small enough to be in the original (Tingley/"Antwerp") tables
\\ of modular elliptic curves:
e150 = E10(1,4)

\\ for a curve over Q,  gp knows how to compute the torsion subgroup:
elltors(e150)
```

```
\\ the output "[10, [10], [[0, 480]]]" says the group has size 10,
\\ with a single generator -- this is automatic here, but compare
\\ the outputs of "elltors(ellinit([0,0,0,-1,0]))" and
\\ "elltors(ellinit([0,0,0,4,0]))" -- and one choice of generator is
\\ [0,480] (which is not our P, but has the same x-coordinate, so
\\ must be -P).  Note that the coefficients 26, -24, -480 have factors
\\ 2, 4, 8, so we can obtain an equivalent curve by scaling by 2:
e150 = ellchangecurve(e150,[2,0,0,0])

\\ and now the coefficients are [13, -6, -60, 0, 0].  Compare this
\\ with the standard model
? ellglobalred
e150_red = ellglobalred(e150)
e150_standard = ellchangecurve(e150, e150_red[2])

\\ at the cost of forcing a1, a2, a3 in {0,1,-1} we've made the
\\ coefficients a4, a6 much larger (-828 and 9072).

\\-------------------------------------------------------------------\\
\\         A bit more about the universal curve E10 over X_1(10)     \\
\\-------------------------------------------------------------------\\

\\ for example, to test that P really is a 10-torsion point on E10(a,b):
E = E10(a,b);
\\ ending a line of input with ";" suppresses the output, which for E10(a,b)
\\ covers 30+ lines
P = [0,0]
P_mult = vector(10,n,ellpow(E,P,n))
\\ the last entry is [0], which is gp's notation for the zero point of
\\ an elliptic curve, and is not the same as P=[0,0].

\\ P_mult is long enough that you might suspect there might be another [0]
\\ hiding in the middle (if P were really a 5-torsion point); so let's check:

vector(10,n,P_mult[n]==[0])
\\ the output is [0, 0, 0, 0, 0, 0, 0, 0, 0, 1], as desired; equivalently,

vector(10,n,P_mult[n]==[0]) == vector(10,n,n==10)
\\ outputs 1 (true).

\\ Two warnings:

\\ 1) as in C, be careful about == vs. =

\\ 2) unlike C (and Python and Sage), gp's vectors and matrices
\\ are indexed starting at 1, not 0
\\ Where are the singular fibers of this curve E = E10(a,b)?
\\ We'd like to answer this by telling gp "factor(E.disc)",
\\ but this returns the error message
\\  *** factor: sorry, factor for general polynomials is not yet implemented.
\\ -- and who knows when if ever it will be implemented in gp (Sage has it).
\\ Still, the discriminant is a homogeneous polynomial in (a:b),
\\ so we can dehomogenize by setting (a,b) = (d,1)
\\ [remember that our a/b is Lemmermeyer's d], and then the discriminant
\\ is a polynomial in one variable, which gp does know how to factor:

E_d = E10(d,1);
E_d_sing = factor(E_d.disc)
\\ we find that the discriminant factors as some constant times
\\ d^10 * (d-1)^10 * (2*d-1)^5 * (d^2-3*d+1)^2 * (4*d^2-2*d-1)

\\ but that doesn't account for the zero of disc(E_d) at infinity,
\\ corresponding to b=0.  The multiplicity of this zero can be found
\\ directly as a valuation:
valuation(E.disc,b)

\\ or by subtracting the degree of disc(E_d) from the homogeneous degree
\\ 3*12 = 36 (the factor of 3 is because the coefficients a1,a2,a3,a4,a6
\\ are homogeneous of degree 3*1, 3*2, 3*3, 3*4, 3*6 -- equivalently, the
\\ arithmetic genus of our elliptic surface is 3):
3*12 - poldegree(E_d.disc)

\\ either way we find that the discriminant has a fifth-order zero
\\ at d=infinity (i.e. at b=0).

\\ We claim that the reduction at each of those factors b=0, a=0, a=b, etc.
\\ of disc(E) is multiplicative.  (This is a general fact about the universal
\\ eliiptic curve over X_1(n) for n>2; there's no such curve for n=2
\\ due to quadratic twists.)  Except in characteristics 2 and 3,
\\ this amounts to checking that c4 and c6 are relatively prime,
\\ because additive fibers are characterized by the simultaneous
\\ vanishing of c4 and c6:
gcd(E_d.c4,E_d.c6)

\\ and indeed it's 1.  What about the fiber at infinity?
[poldegree(E_d.c4),poldegree(E_d.c6)]

\\ returns [12, 18] = [4*3, 6*3], so neither c4 nor c6 vanishes there either.
\\ This should remain the case mod p for all p>3 not dividing 10;
\\ we can check this by forming a resultant:
```

```
factor(polresultant(E_d.c4,E_d.c6))

\\ which indeed returns 2^60 3^18 5^3.  We conclude that our surface has
\\ multiplicative reduction of type I_10 at a=0 and a=b, of type I_5 at
\\ b=0 and 2*a=b, of type I_2 at each of the two roots of a^2-3*a*b+b^2 = 0,
\\ and of type I_1 at each of the two roots of 4*a^2-2*a*b-b^2.

\\ (digression: actually "factor" always returns a matrix with 2 columns,
\\ one for factors and one for exponents; here's the syntax for
\\ reconstructing the product.  First we need to know how many
\\ factors we have:
matsize(E_d_sing)

\\ returns [5,2], indicating 5 rows and 2 columns; in particular,
\\ matsize(E_d_sing)[1] = number of rows = number of factors, so:
D = prod(n=1, matsize(E_d_sing)[1], E_d_sing[n,1] ^ E_d_sing[n,2]);
D / E_d.disc
\\ returns 1, so here the constant factor in the polynomial factorization
\\ is trivial.  End of digression.)


\\-------------------------------------------------------------------\\
\\            The canonical height of multiples of P                 \\
\\-------------------------------------------------------------------\\

\\ Since each multiple mP is either the origin or has polynomial
\\ (not merely rational) homogeneous polynomials as its coordinates,
\\ the naive height of mP is always twice the arithmetic genus, which is 6.
\\ But the canonical height should vanish.  gp does not (yet?...) have
\\ built-in caonical heights for elliptic surfaces, so we must compute
\\ them ourselves.  Fortunately for a semistable eliiptic surface
\\ (i.e. one all of whose singular fibers are multiplicative)
\\ this is straightforward: for a point Q, the correction at a fiber
\\ of type I_n is m*(m-n)/n, where m in [0,n) is the index of
\\ the component where the corresponding section s_Q meets the fiber.
\\ This index is nonzero iff Q reduces to the singular point, and then
\\ can be computed (up to the involution m <--> n-m, which does not change
\\ the local correction m*(m-n)/n) as the smaller of n/2 and the valuation
\\  of y(Q) - y(-Q).

\\ Recall we've already computed E_d_sing, but for the dehomogenized
\\ model; first we construct the correspoding homogeneous factorization,
\\ remembering to incorporate the factor b "at infinity":

num_factors = matsize(E_d_sing)[1] + 1;
E_sing = matrix(num_factors, 2);
```

```
{
for(n=1, num_factors-1,
  E_sing[n,1] = b^poldegree(E_d_sing[n,1]) * subst(E_d_sing[n,1], d,a/b);
  E_sing[n,2] = E_d_sing[n,2]
  );
}
E_sing[num_factors,1] = b;
E_sing[num_factors,2] = 5;
E_sing

\\ let's check we got this right:
prod(n=1,num_factors, E_sing[n,1] ^ E_sing[n,2]) / E.disc
\\ indeed it comes to 1

\\ the next program "corr" computes, for a point Q on elliptic surface e
\\ the correction to the canonical height associated to a factor f of disc(e)
\\ occurring to multiplicity n.  The variables x, y, y_diff, and m are
\\ internal to the program.
{
corr(e,Q,f,n, x,y,y_diff,m) =
  x = Q[1];
  y = Q[2];
  if(valuation((e.a1*x + e.a3 - 2*y) , f) == 0, return(0));
  if(valuation((3*x^2+2*e.a2*x + e.a4 - e.a1*y) , f) == 0, return(0));
  y_diff = y - ellpow(e,Q,-1)[2];
  if(y_diff == 0,  return(-poldegree(f)*n/4));  \\ because then m = n/2
  m = min(n/2, valuation(Q[2] - ellpow(e,Q,-1)[2], f));
  return( poldegree(f) * m * (m-n) / n )
}
\\ we need the factor deg(f) because there are really deg(f) Galois-conjugate
\\ factors; that's analogous to the factor log(p) in the  arithmetic setting
\\ that arises for the height correction at a prime p.  Using "poldegree"
\\ without specifying the variable can give unpredictable results, but here
\\ it must work since f is irreducible.

\\ So the following vector should be zero:
vector(9,n, 6 + sum(k=1,num_factors,corr(E,P_mult[n],E_sing[k,1],E_sing[k,2])))

\\ and indeed it is.  Check that the same is true for each of the cases
\\ E5, E6, E7, E8, E9 where both homogeneous parameters have the same weight.

\\-----------------------------------------------------------------\\
\\    The universal curve over X_1(7) in characteristic 3 (and 5)    \\
\\-----------------------------------------------------------------\\

\\ To get examples of nonzero canonical height we need elliptic surfaces
\\ of positive rank.  Computing the rank, let alone the Mordell-Weil group,
\\ of an elliptic surface over C is still an intractable problem in general.
\\ Over a finite field there's still no general algorithm known but we have
\\ some additional tools, as we'll illustrate with the reduction mod 3 of
\\ the universal ellipic curve over X_1(7).  It's known, but probably
\\ not surprising, that in characteristic zero the universal curve
\\ over X_1(N) has no points outside its tautological torsion group Z/NZ,
\\ but remarkably this can fail in positive characteristic!

\\ N=7 is the first case where this universal curve is not rational
\\ *as a surface*; instead it is a K3 elliptic surface (arithmetic genus 2).
\\ We describe its singular fibers as we did for X_1(10):

E = E7(a,b);
E_d = E7(d,1);
E_d_sing = factor(E_d.disc)
\\ This time the factors are  d^7 * (d-1)^7 * (d^3+5*d^2-8*d+1),
\\ and for the factor at infinity:
valuation(E.disc,b)
2*12 - poldegree(E_d.disc)
\\ giving 7 either way.
gcd(E_d.c4,E_d.c6) \\ 1
[poldegree(E_d.c4),poldegree(E_d.c6)] \\ = [8,12] = 2*[4,6]
\\ so again multiplicative reduction everywhere: I_7 at d=0,1,infinity
\\ and I_1 at the conjugate roots of d^3+5*d^2-8*d+1.
factor(polresultant(E_d.c4,E_d.c6))
\\ = 2^24 3^12 7, and it turns out that 2 and 3 are OK too.

\\ Check as before that all nonzero multiples of P have canonical height 0:
num_factors = matsize(E_d_sing)[1] + 1;
E_sing = matrix(num_factors, 2);
{
for(n=1, num_factors-1,
  E_sing[n,1] = b^poldegree(E_d_sing[n,1]) * subst(E_d_sing[n,1], d,a/b);
  E_sing[n,2] = E_d_sing[n,2]
  );
}
E_sing[num_factors,1] = b;
E_sing[num_factors,2] = 7;
E_sing

\\ let's check we got this right:
prod(n=1,num_factors, E_sing[n,1] ^ E_sing[n,2]) / E.disc
\\ It comes to -1, which is still a constant.
P = [0,0]
P_mult = vector(7,n,ellpow(E,P,n))
```

```
vector(6,n, 4 + sum(k=1,num_factors,corr(E,P_mult[n],E_sing[k,1],E_sing[k,2])))

\\ The I_7 fibers contribute 3*6=18 to the Picard number,
\\ and together with the contribution of the fiber and zero-section
\\ we get a total of 6*3 + 2 = 20, so in characteristic zero the
\\ Mordell-Weil rank is zero as expected and we have an "extremal"
\\ elliptic K3 (maximal Picard number, zero rank) of Neron-Severi
\\ discriminant -7^3 / |T|^2 = 7.

\\ This should be reflected in the point-counts modulo a prime p
\\ other than 7.  Each of the I_7 fibers contributes 7p (since the
\\ fiber components are rational -- that's automatic here because
\\ the components are separated by multiples of P); I_1 fibers need
\\ no special treatment because such a fiber's singular point is already
\\ smooth as a point on the surface in (x,y,d) space; so we just try all
\\ (x,d) values other than d=0,1,infty, remembering to include
\\ the point at infinity.  The following works for odd p; we leave
\\ as an exercise (i.e. don't have the patience for) checking the
\\ special case p=2.
cubic(E,x) = poldisc(y^2 + E.a1*x*y + E.a3*y - (x^3 + E.a2*x^2), y);
{
N7(p) = 21*p + sum(n=2,p-1, p+1+sum(x=0,p-1,
  kronecker(subst(cubic(E_d,x),d,n),p)
  )
)
}

\\ When (p/7) = -1, the count is exactly p^2 + 20 p + 1
\\ (i.e. the trace of Frobenius on H^2 is 20p, same as the trace on NS;
\\ equivalently, the trace on the transcendental part of H^2 is zero),
\\ so the following run always prints [p,0]:

forprime(p=3,113,if(kronecker(p,7)==-1,print([p,N7(p)-(p^2+20*p+1)])))

\\ if (p/7) = +1, the number of points is between p^2 + 18 p + 1 and
\\ p^2 + 22 p + 1.  The next run shows, for each such prime up to 113,
\\ how the point-count splits the interval of length 4p between
\\ those two bounds:
{
forprime(p=3,113, if(kronecker(p,7) == +1,
  n = N7(p);
  print([p, n-(p^2+18*p+1), p^2+22*p+1-n])
))
}
\\ What do you observe?
\\ Since the transcendental lattice is only 2-dimensional,
\\ a single point count suffices to determine the L-function.
\\ (In general we'd also have to count over the fields of
\\ p^2, p^3, ..., p^e elements where e is roughly half the
\\ transcendental dimension.)  In particular, when (p/7) = -1,
\\ the L-function has a 21st zero at s=1 over F_p, and a 22nd
\\ over F_{p^2}.  So we expect two independent sections,
\\ one defined over F_p and the other only over F_{p^2}.
\\ More precisely, the BSD / Artin-Tate conjecture, which is
\\ a theorem in this setting of an elliptic K3 surface over F_q(t),
\\ indicates that over F_p(t) the M-W group is generated by [0,0] and
\\ a section P1 of canonical height 2p/7, while over F_{p^2}(t) the
\\ regulator is p^2/7.  This, together with the existence of a
\\ Galois involution and the fact that the canonical height of any
\\ section is in (2/7)Z, implies that there's a section P2 of height 2p
\\ taken to -P2 by the Galois involution, with Q=(P1+P2)/2 and
\\ Q'=(P1-P2)/2 rational of height 4p/7 and switched with each other by
\\ the Galois involution.

\\ In general the canonical height measures not just the complexity of
\\ a section but also the difficulty of finding it, in much the same way
\\ as in the more familiar setting of elliptic curves over Q.  We next
\\ find these sections for p=3, the smallest prime "nonresidue" mod 7,
\\ and thus the one for which those canonical heights are smallest.
\\ as it happens, -1 is a nonsquare mod 3 as well, which means that
\\ we can work with F_9 as F_3[I] where I is [sic] gp's notation for a
\\ square root of -1; in general we could use F_p[w] where w is a
\\ quadratic irrationality (try "?quadgen").  It *is* possible to find
\\ the sections Q and Q' for larger primes p such that (p/7) = -1
\\ in time polynomial in p, via the connection between our K3 surface
\\ and the square of a CM elliptic curve of discriminant -7; but
\\ that's a rather more complicated and extensive computation
\\ which is well beyond the scope of the present notes!

\\ For p=3 we have 2p/7 = 6/7.  There are three ways for a section of E7
\\ to have this canonical height: it could be integral (i.e. with x,y
\\ homogeneous polynomials of degrees 4,6 in (a:b), or equivalently
\\ polynomials of degrees at most 4,6 in d), and with height corrections
\\ (here and later multiplied by -1 to forestall a deluge of minus signs)
\\ 0 + 10/7 + 12/7  or  6/7 + 6/7 + 10/7  adding up to 22/7 = 4 - 6/7
\\ [NB the numerators 0, 6, 10, 12 are 0*7, 1*6, 2*5, 3*4]; or there could be
\\ a pole (i.e. the denominators of x and y are the square and cube of
\\ the same linear polynomial), so the naive height is 6, and at each of
\\ the three I_7 fibers the height correction is the maximal 12/7,
\\ adding to 36/7 = 6 - 6/7.
\\ Now given an elliptic K3 surface E, a point of naive height h=4,6,8,10,...
\\ is specified by (3h/2)-1 = 5,8,11,... parameters: the h+1 coefficients
\\ of the numerator of x, and the (h/2)-2 non-leading coefficients of
\\ monic polynomial of whose square and cube are the demoniators of x and y.
\\ For such x to work, a polynomial of degree 3h occurring in the numerator
\\ of  (2y + a1 x + a3)^2  must be a square, which is 3h/2 conditions;
\\ this suggests that the condition that such x exist is a union of
\\ hypersurfaces in the moduli space of K3 surface, which over C is confirmed
\\ by the Torelli theorem for K3's.  But finding such a section when it exists
\\ is still a challenge, and even five variables is a lot if we want to
\\ solve a system of nonlinear equations.  When the height is reduced by
\\ a correction m(n-m)/n, it's also easier to find the point, because
\\ the condition that the section (x,y) go through the m-th or (n-m)th
\\ component of an I_n fiber above d = d_0 anmounts to min(m,n-m)
\\ *linear* conditions on the coefficients of x, namely a choice of
\\ the first min(m,n-m) coefficients in the Taylor expansion of x about d_0.
\\ (This description is true for a general multiplicative fiber, not just
\\ in the K3 setting).  For example, the first of these coefficients must be
\\ the x-coordinate of the singular point of the Weierstrass equation, because
\\ all the non-identity components map to that singularity.  The later
\\ coefficients are determined by the condition that (2y + a1 x + a3)^2
\\ vanish to order at least 2, 4, 6, ..., 2*min(m,n-m) at d=d_0.

\\ We illustrate this for our surface E7.  Remember that cubic(x) computes
\\ (2y + a1 x + a3)^2.  Setting d=0:

subst(cubic(E_d,x),d,0)

\\ yields  4*x^3 + x^2,  so clearly the singularity is at x=0.  To find
\\ the next term, start from cubic7(x1*d), which automatically has
\\ a factor of d^2, and divide by that:

subst(cubic(E_d,x1*d)/d^2,d,0)

\\ to get x1^2.  So, the next condition is that x is 0 mod d^2.  Thus,
\\ the third condition:

subst(cubic(E_d,x2*d^2)/d^4,d,0)

\\ gives x2^2 so (x,y) is on component 3 or 4 at d=0 iff x = O(d^3).
\\ [This is basically retracing Tate's algorithm for our surface;
\\ the simplicity of the answer illustrates our earlier remarks on
\\ extended vs. narrow Weierstrass form.]  Likewise at d=1:

subst(cubic(E_d,x),d,1)
subst(cubic(E_d,x1*(d-1))/(d-1)^2,d,1)

\\ the first step proceeded as before, but the second gives
\\ x1^2 + 2*x1 + 1 = (x1+1)^2, so:

subst(cubic(E_d,x2*(d-1)^2 - (d-1)) / (d-1)^4, d, 1)

\\ and this gives x2^2 + 4*x2 + 4 = (x2+2)^2.

\\ For d=infinity, we look at leading coefficients:

pollead(cubic(E_d,x4*d^4),d)

\\ is 4*x4^3 + x4^2, with a double root at x4=0, so the singularity is
\\ at x4=0; thus projectively x vanishes (as a homogeneous function of
\\ degree 4 in (a:b)) at d=infinity (i.e. at b=0).

pollead(cubic(E_d,x3*d^3),d)

\\ is x3^2, so the next step is x=O(d^2), with a double zero at b=0; and then

pollead(cubic(E_d,x2*d^2),d)

\\ gives (x2+1)^2, so components 3 and 4 have x = -d^2 + O(d) at infinity.

\\ Thus: if we want to get height 6/7 with corrections 0/7, 10/7, and 12/7
\\ we can put the 0/7 at infinity (there's an automorphism that cyclically
\\ permutes the cusps and thus the I_7 fibers), and the 10/7 and 12/7
\\ are at d=0 and d=1 in some order.  We put the 12/7 at d=0, because the
\\ other choice gives the 7-torsion point with x = d^2-d^3 that we know
\\ already.  So, we want x to be a quartic in d congruent to 0 mod d^3
\\ and to (d-1) modulo (d-1)^2.  This is easy enough to calculate by hand,
\\ but gp's "chinese" program does it automatically:

lift(chinese(Mod(0,d^3), Mod(-(d-1), (d-1)^2)))

\\ we get -d^4 + d^3, and then

factor(cubic(E_d, -d^4 + d^3))

\\ reveals a constant multiple of  d^6 (d-1)^4 (3*d^2-4); so indeed
\\ this works in characteristic 3, and the constant is -1 so we
\\ we actually get a point over Z/3Z, not just the 9-element field.
\\ Solving for y we find that one solution is -(d-1)^2 d^4:

P1_d = [-d^4+d^3, -(d-1)^2*d^4];
```

```
\\ or in homogeneous form:

P1 = subst(P1_d,d,a/b);
P1 = Mod(1,3) * [P1[1]*b^4, P1[2]*b^6]

ellisoncurve(E,P1)

\\ The other choices for getting 6/7 come from translates by
\\ multiples of the 7-torsion point [0,0]:

P1_trans = vector(7,n,elladd(E,P1,ellpow(E,P,n)));

\\ It will be seen that translate by 5P is the one non-integral section
\\ of these 7.  Check that all these have the same height 6/7:

H_naive(P) = poldegree(subst(numerator(P[1]),a,d*b),b)
H(P) = H_naive(P) + sum(k=1,num_factors,corr(E,P,E_sing[k,1],E_sing[k,2]))

vector(7,n, H(P1_trans[n]))

\\ [For p=5 we can find the section P1 similarly.  Here we can get 10/7 as
\\ 4 - (0 + 6/7 + 12/7), so this time we make x a multiple of d^3 (to get
\\ the 12/7 at d=0) and require that the d^4 coefficient of x vanish
\\ to get the 6/7 correction from the reducible fiber at d=infinity.

yy = cubic(E_d, x1*d^3) / d^6

\\ that’s a quartic in d that we want to make square; what’s the square root?

yy2 = truncate(sqrt(yy+O(d^3)))
yydiff = (yy2^2 - yy) / d^3
yydiff1 = yydiff / content(yydiff)

\\ the "content" was 4*x1^5 / (x1+1)^6, and x1 cannot vanish because then
\\ we’re back to a 7-torsion point.  This gives us the linear polynomial
\\ (2*x1^3+3*x1^2+3*x1+1)*d - (x1^4+2*x1^3+2*x1^2+2*x1+1) in d, and we want
\\ to choose x1 to make that polynomial vanish identically.  This is not
\\ possible in characteristic zero, because these coefficients have
\\ no common root; but:

polresultant(polcoeff(yydiff1,1,d), polcoeff(yydiff1,0,d), x1)

\\ returns -5, so modulo 5 (and no other prime) we get the desired section
\\ by making x1 a common root:
```

```
gcd(Mod(1,5)*polcoeff(yydiff1,1,d), Mod(1,5)*polcoeff(yydiff1,0,d))

\\ gives x1+3 so we must take x1=2.  This gives:

P1_5_d = [2*d^3, -d^4-2*d^3];
P1_5 = subst(P1_5_d,d,a/b)
P1_5 = Mod(1,5) * [P1_5[1]*b^4, P1_5[2]*b^6]

\\ but back to p=3.]

\\ There are three routes to 12/7: either naive height 4 and corrections
\\ 0, 6/7, 10/7,  or naive height 6 and corrections 6/7, 12/7, 12/7 or
\\ 10/7, 10/7, 10/7.  We choose the last of these, which leaves only
\\ two undetermined coefficients in x.  Thus x = sextic(d) / (d-d0)^2
\\ for some d0 other than 0, 1, and infinity, with x = O(d^2) at d=0,
\\ x = -(d-1) + O(d-1)^2  at d=1, and  x = O(d^2) at d=infinity.
\\ Thus  x = (c*d - (c+(d0-1)^2)) * (d-1) * d^2 / (d-d0)^2)
\\ for some c and d0, and after eliminating square factors
\\ we find a sextic in d that must be a square:

{
yy = subst(cubic(E_d,X), X, (c*d - (c+(d0-1)^2)) * (d-1) * d^2 / (d-d0)^2)
  * (d-d0)^6 / (d^4 * (d-1)^4);
}
\\ I'm not sure why this circumlocution is necessary instead of just
\\ yy = cubic(E_d, (c*d - (c+(d0-1)^2)) * (d-1) * d^2 / (d-d0)^2) etc. ...
yy2 = truncate(sqrt(yy+O(d^4)));
yydiff = (yy2^2 - yy) / d^4;
yydiff1 = yydiff / content(yydiff);

\\ there's still a common factor that "content" doesn't pick up but
\\ "gcd" does:
yydiff1 /= (c + (d0-1)^2) * d0

\\ We now have three simultaneous nonlinear equations (the vanishing of
\\ the coefficients of 1, d, and d^2 in yydiff) to solve for the
\\ two variables c, d0.  We solve the first two of these, using
\\ a resultant to eliminate c, then check whether each of the solutions
\\ is feasible for the third equation:
R = polresultant(polcoeff(yydiff1,1,d), polcoeff(yydiff1,0,d), c)
centerlift(factormod(R,3))

\\ we find that there are spurious roots d0=0 and d0=1 of multiplicity
\\ 32 and 2 (they must be spurious beause x must have zeros there,
\\ not poles); a double root at d0 = -1; and two further irreducible
\\ sextic factors d0^6-d0^5-d0^4+d0^2+d0+1, d0^6-d0^5-d0^4-d0^3+d0^2+d0-1
\\ that cannot be right because we know our solution will be defined
\\ over the 9-element field.  So we set d0=1 and solve for c:

yydiff1 = subst(yydiff1, d0, Mod(-1,3));
yydiff1 /= content(yydiff1)
\\ for some reason gp recognizes the common denominator of c+1
\\ but not the common factor of the numerator; so we ask for it directly:

gcd(gcd(polcoeff(yydiff1,0,d),polcoeff(yydiff1,1,d)),polcoeff(yydiff1,2,d))

\\ and we find a common factor -c^2-c+1, i.e. c=1+I or c=1-I.





Q1 = Mod(1,3) * [I*a^2*b^2 + (-1+I)*a*b^3, -a^2*b^4]
Q2 = conj(Q1)
\\ The same technique works for p=5 to find the section of height 2p/7

\\--------------------------------------------------------------------\\
\\         An example of rank 4 with a 4-torsion point over Q         \\
\\--------------------------------------------------------------------\\
```

Kiran Kedlaya's worksheet -- Introduction to Python and Sage

AMS Short Course: Computing with Elliptic Curves using Sage

January 2-3, 2012

Lecture 1: Introduction to Python and Sage

Kiran S. Kedlaya (MIT/UC San Diego)

This lecture is an introduction to using the computer algebra system Sage. In the process, we will also introduce the Python programming language, on which Sage is built; however, we'll keep the programming discussion to a minimum because it is easy to find Python tutorials and other documentation online. (For example, try the Beginner's Guide to Python.)

All of the lectures in this minicourse are meant to be interactive, so you are strongly encouraged to follow along on your own computer as we go along. To make this possible, we begin by explaining how to run Sage on your own computer. There are two natural ways to do this.

- Since Sage is free for everyone, you may install Sage on your own computer and run it locally using the command line. Go to http://www.sagemath.org and follow the links to download and install Sage. (Warning: this is a bit tricky if you run Windows. Ask one of us for help if you get stuck.)
- Besides the command line, Sage also offers a graphical interface, the *Sage Notebook*, in which you use your web browser as a client to access a Sage server on some possibly different machine. For example, right now, I am running Sage on my local computer as a server, and using this browser to connect to that server in order to display this presentation. However, there also exists a public notebook server on which *anyone* can create an account and run Sage! This is by far the easiest way to try out Sage right now. To do so, point your browser to http://www.sagenb.org.

To simplify the exposition, I'm mostly going to stick to explaining the notebook interface. We will come back to the command line later.

Getting started with the notebook

If you have successfully started a notebook session, you should see something that looks much like this window, except that instead of all of this text, you should just see an empty box like the one below.

This box is called a *cell*, and can be used to evaluate any Sage command. For example, try clicking on the cell, typing 2+2, and clicking "evaluate". (A shortcut for "evaluate" is Shift+Enter.)

```
2+2
```

```
    4
```

Note that the answer appears immediately below, and then a new cell is generated so that you can type another command. You can also go back and edit the previous cell, but the answer won't change until you evaluate again. You can even insert new cells between existing cells: if you click on the blue bar that shows up when you mouse between two existing cells, a new cell will pop up in between.

If you Shift-click instead, you get a text box like this, which supports some formatting and even basic TeX commands. This is one way to add annotations to a notebook; a more direct one is simply to insert comments in cells themselves. The # character denotes a comment, and forces everything to the end of the line to be ignored.

```
2+2 # should equal 4
```

```
    4
```

Besides individual calculator-style expressions, a cell can contain a sequence of instructions; hit Enter to separate instructions. (This is why Shift-Enter is the shortcut for evaluation, not Enter.) The last expression is evaluated and printed; you can also add "print" commands to force additional printing.

```
x = 5
y = 7
print x+y
x*y
```

```
    12
    35
```

Each cell evaluation remembers any definitions from cells that were evaluated previously, including variables and functions (more on which later).

```
x+y
```

```
    12
```

Warning: Sage will let you evaluate cells out of order. It is the order of evaluation that counts, not the order of appearance in the worksheet.

If you try to evaluate a cell which contains a mistake, such as trying to use a variable which is not yet defined, Sage will give you a (hopefully) useful error message. You can click on the output to see a more verbose error message, which may help you isolate the problem if it occurs in a large cell.

```
z
```

```
    Traceback (click to the left of this block for traceback)
    ...
    NameError: name 'z' is not defined
```

Basic Python/Sage objects

Let's get familiar with some basic types of Python/Sage objects. These will all be things that can be assigned to variables, which we have already seen how to do in the case of integers.

```
x = 2+2
print x
```

```
    4
```

Note that variables do not need to be declared before being used; in particular, there is no need to specify in advance what type of object is going to be assigned to a given variable.

So far our variable names have only been one character long. However, this is not required: you may use any sequence of letters, numbers, and the underscore _ as a variable name as long as it starts with a letter and does not coincide with a Python reserved wood (such as "print"). Considered choice of variable names may make your code easier for a human to understand.

```
square_root_of_2 = sqrt(2)
print square_root_of_2^4
```

```
    4
```

Besides integers, some other basic types of objects include rational numbers, real numbers, and complex numbers (where capital I denotes the chosen square root of -1). These support the usual arithmetic operations.

```
print 7 % 3 ## modular reduction
print 7 // 3 ## integer division
print 10/6 ## returns a rational number
print exp(2.7)
print (2+I)*(3+I)
```

Another basic object type is strings, which are enclosed in either single or double quotes (there is a subtle distinction which is not relevant here). One common operation on strings is concatenation.

```
'3' + '4'
```

Arrays

Besides simple types like numbers and strings, one also has compound types like arrays. Arrays can be specified using square brackets (the entries need not be all of one type).

```
print [3, 4] + [5, 'six'] ## plus sign denotes composition
```

One also uses square brackets to access the individual entries of an array, or even to assign them. Two important things to note:

- Python indexes starting from 0, not 1.
- You may also use negative indices: -n denotes the n-th term from the end.

```
u = [2, 3, 4]
print u[0] ## first element
print u[-1] ## last element
u[1] = 1 ## modify an existing entry
print u[-2] ## should equal u[1]
u[3] = 5 ## this won't work
```

```
    [2, 3]
    [0, 1, 2, 3, 4]
    [3, 5, 7, 9, 11]
```

Some other ways to construct arrays are the following.

- One can concatenate existing arrays using the + operator (see above).
- One can modify in place entries of an existing array (see above).
- One can form *slices* of an existing array by picking out a range of values. This is the safest way to make a copy of an array (see below).
- Some functions return arrays, such as the "range" function.
- One can use *list comprehensions* to apply a function to each term of one array to make a new array. There is also the option to put in a conditional statement to pick out only some of the terms of the original array (using == for equality, rather than = which means assignment). This is a lot like how mathematicians make sets!

```
u = [2,3,4,5]
print u[1:3] ## left index is included, right index is not
print u[:4] ## left index defaults to 0
print u[1:] ## right index defaults to the length of the array
print range(5) ## range(n) includes 0, excludes n
print range(3,13) ## specify left and right endpoints as for
slices
print [x^2 for x in u if x%2==1]
```

```
[3, 4]
[2, 3, 4, 5]
[3, 4, 5]
[0, 1, 2, 3, 4]
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
[9, 25]
```

Warning: arrays are copied by reference, not by value. This is in fact true for all Python/Sage objects, but the fact that arrays can be modified after they are created (that is, they are *mutable*) creates a real danger.

```
a = [1,2,3]
b = a ## does not create a new array!
b[1] = -1
print "a = ", a ## has now been modified
c = a[:] ## this does create a new array
c[-1] = 4
print "a = ", a
print "c = ", c
```

Conditional evaluation

Like most programming languages, Python supports conditional evaluation via such mechanisms as for loops, while loops, and if-then-else constructions. Here is an example.

```
for i in range(5):
    if i%2 == 0:
        print i, "is an even",
    else:
        print i, "is an odd",
    print "number"
print "loop complete"
```

Let's step through this example in detail.

- The first line creates the for loop. The range is created using the "range" function described above.
- Everything within the for loop is indented. This is not optional! Python uses this indentation to figure out where the loop starts and ends. The notebook will help you: after you type the "for" line (ending with the colon), the next line will be indented appropriately.
- The "if" command takes a Boolean expression. As we saw before, == denotes equality, as opposed to a single = which denotes an assignment.
- The "else" command is optional. Again, the indentation tells Python where the if and else clauses begin and end.
- The "print" command can take multiple arguments, and adds a line break unless you end it with a

comma.

Here is another example, this time of a while loop. Note the effect of the "continue" and "break" statements.

```
t = 4
while t < 10:
    t += 1 ## has the same effect as t = t + 1
    if t%2 == 0:
        continue ## jump to the next iteration of the loop
    print t
    if t > 8:
        break ## jump out of the loop
```

```
    5
    7
    9
```

Functions

By now, you have probably noticed that much Python functionality is provided via callable functions, such as "range". Sage extends this functionality by specifying many new advanced mathematical functions.

```
prime_pi(1000) ## The number of primes up to 1000
```

```
    168
```

Sage includes a couple of handy techniques for inquiring about functions. One of these is *tab completion*: if you type part of the name of a function and hit Tab, Sage will attempt to complete the name of the function. On one hand, this can save a lot of typing.

```
charac
```

Perhaps more usefully, if more than one completion is possible, Sage will show you all possible completions; this can be a useful way to find out about what is available in Sage. (This technique is even more possible when applied to object methods; see below.)

```
ran
```

Another important inquiry technique is *introspection*: if one evaluates the name of a function followed by a ?, Python will show you a bit of documentation about that function. For instance, if you have forgotten how to specify a range other than 0, ..., n-1, we can use introspection on the range function.

```
range?
```

Python and Sage also allow for user-defined functions. As with variables, these may be defined in one cell and then used in later cell evaluations. (These will not admit introspection unless you do some extra work, but they will admit tab completion.)

```
def strange_sum(a, b):
    c = a+b
    return(a+b+c)
```

```
strange_sum(2, 3)
```

```
    10
```

```
strange_sum
```

File: /tmp/tmpDTc7pM/___code___.py

Type: <type 'function'>

Definition: strange_sum(a, b)

Docstring:

```
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
```

Mathematical objects in Sage

One important difference between Python and some lower-level programming languages like C is that Python is *object-oriented*. That is, besides the variables and functions that one can define globally, one can also create *objects* with their own variables and functions attached to them. In Sage, this framework is used to implement the construction of mathematical objects in a rigorous fashion which corresponds pretty well to the way mathematicians are used to thinking about these objects. For example, one can create such objects as the ring of integers and the field of rational numbers.

```
Z = IntegerRing()
Q = RationalField()
```

One can then feed these into more complicated constructions, like the ring of polynomials in a single variable.

```
R.<T> = PolynomialRing(Q)
```

In this example, we have created a polynomial ring over the rationals with a distinguished generator, to which we have assigned the name T. We can now generate elements of the ring R simply by writing down expressions in T.

```
poly = T^3 + 1
print poly.parent()
print poly.parent() == R
```

```
    Univariate Polynomial Ring in T over Rational Field
    True
```

The "parent" command here is an example of a *method* of the object poly.

Now that we can make polynomials over the rationals, what can we do with them? An excellent way to find out is to use *tab completion*: if one types "poly." and then hits Tab, one is presented a list of all of the methods associated to poly.

```
poly.factor() ## can also be invoked as factor(poly)
```

```
    (T + 1) * (T^2 - T + 1)
```

In the previous example, the "factor" method did not require any additional arguments. In other examples, one or more arguments may be required.

```
poly.xgcd(T-2) ## extended GCD; can also be invoked as
xgcd(poly, T-2)
```

```
    (1, 1/9, -1/9*T^2 - 2/9*T - 4/9)
```

This syntax might seem a bit strange: the extended GCD operation is essentially symmetric in its variables, so calling it in an asymmetric fashion may be counterintuitive. There is an extremely good reason for using the object-oriented syntax, namely tab completion. One can for instance type "poly." and his Tab to see the list of all of the methods associated to poly!

The ease of looking up the operations available on a given type of mathematical object is one of my favorite features of Sage. Even Magma, which has both object orientation and tab completion, fails on this point because functions are not viewed as object methods, but intsead all inhabit a single namespace.

```
poly.
```

As for bare functions, one can also type a few characters and get only the methods that start with the characters you type. For example, if you can't remember whether the method for factoring a polynomial is called "factor", "factorize", "factorise", "factorization", or "factorisation", you can check by doing a tab completion:

```
poly.factor
```

There can sometimes be some ambiguity about what the correct parent of a mathematical object should be. For instance, 3/4 is a rational number, but it can also be viewed as a polynomial over the rationals, e.g., when one wants to add or multiply it with another such polynomial. For the most part, Sage will take care of this for you automatically.

```
poly2 = poly + 3/4
print poly2.parent()
```

```
    Univariate Polynomial Ring in T over Rational Field
```

Sometimes, however, one needs to make this change of parent explicit, e.g., in case one wants to call a method which exists for polynomials but not for rationals. For example, this fails:

```
poly2 = 3/4
poly2.coefficients()
```

```
    Traceback (click to the left of this block for traceback)
    ...
    AttributeError: 'sage.rings.rational.Rational' object has no
    attribute 'coefficients'
```

This change of parent (called *coercion*) is usually accomplished by the following syntax, which looks like plugging the element into the new parent viewed as a function.

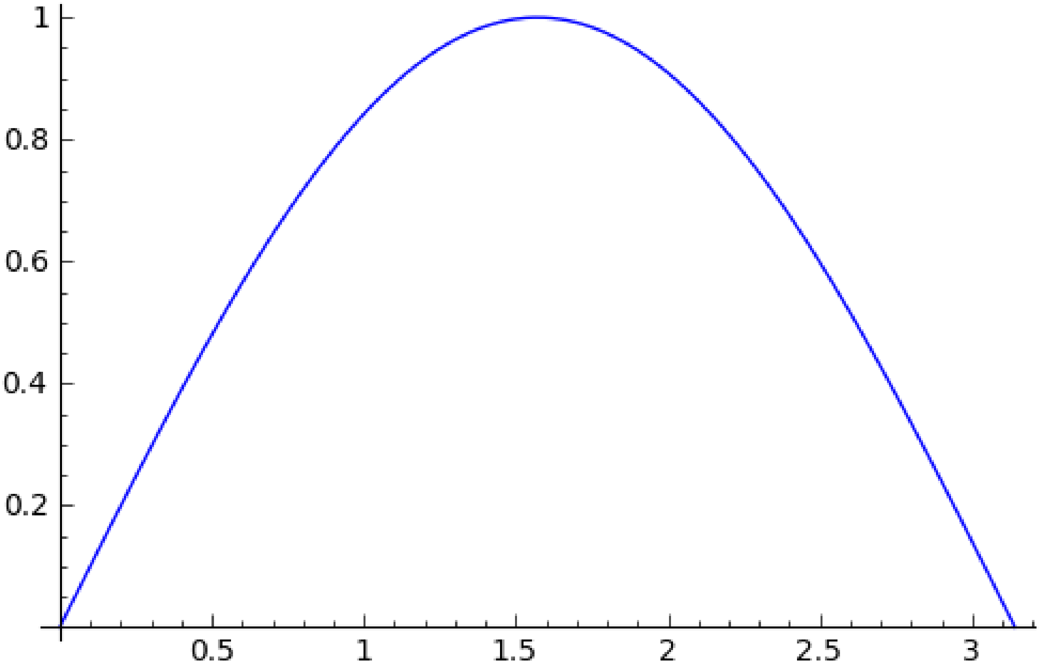
```
poly2 = R(3/4)
poly2.coefficients()
```

```
    [3/4]
```

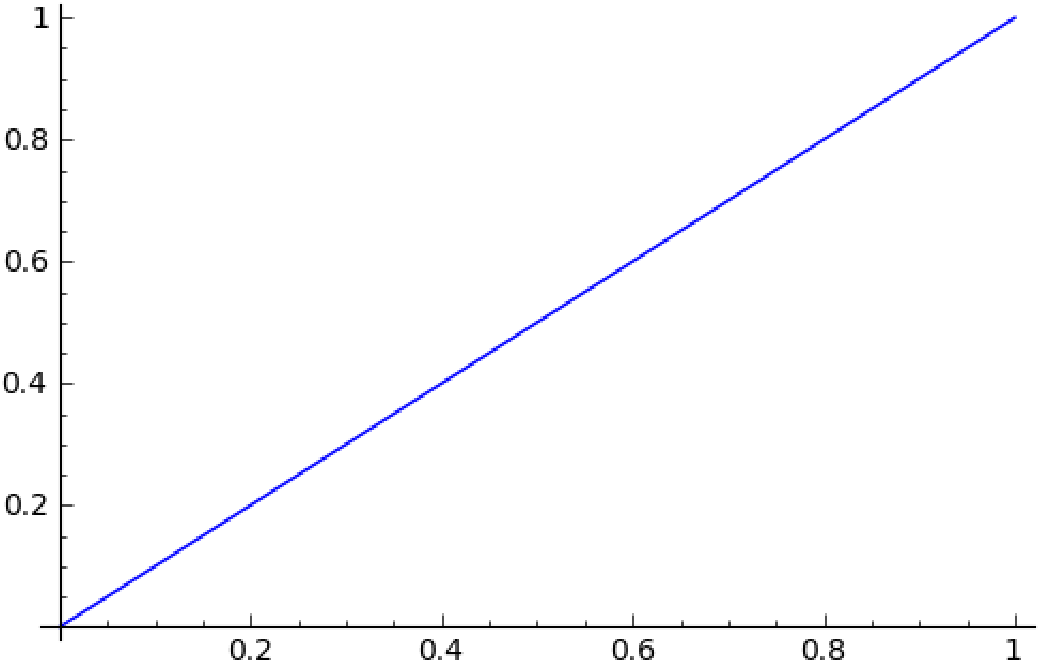
Plotting

Sage has many more mathematical features than we can introduce here (but which are well-documented). One feature which will be used later is plotting of various kinds. (We only need 2D plotting her; there are also extremely cool 3D plotting capabilities, but you can discover those on your own.)

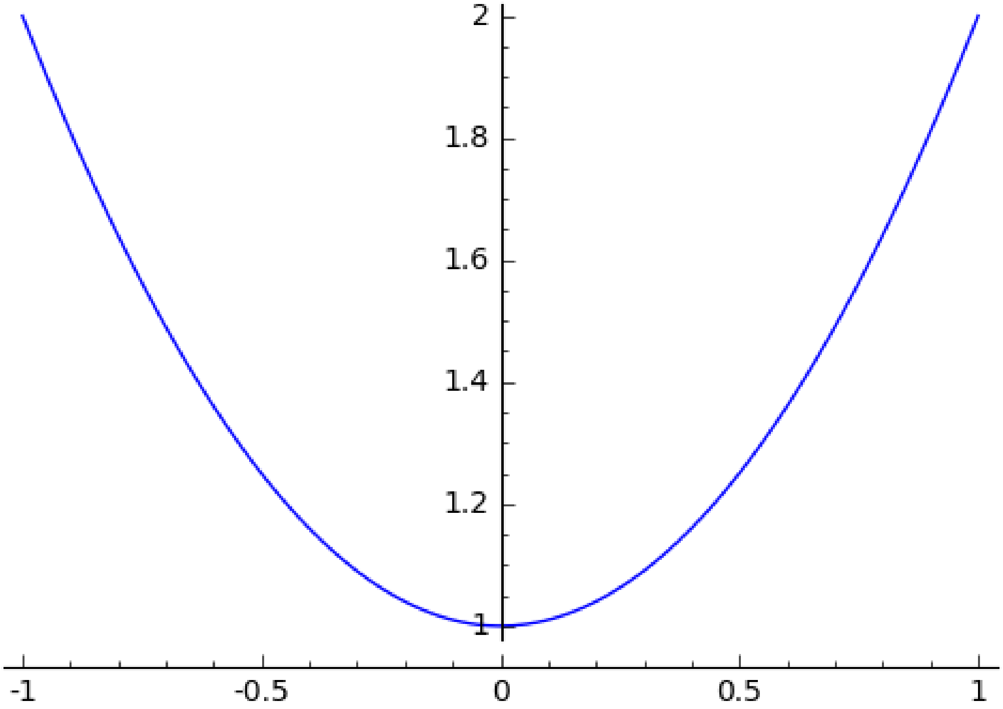
```
plot(sin, 0, pi) ## Plotting a built-in function
```



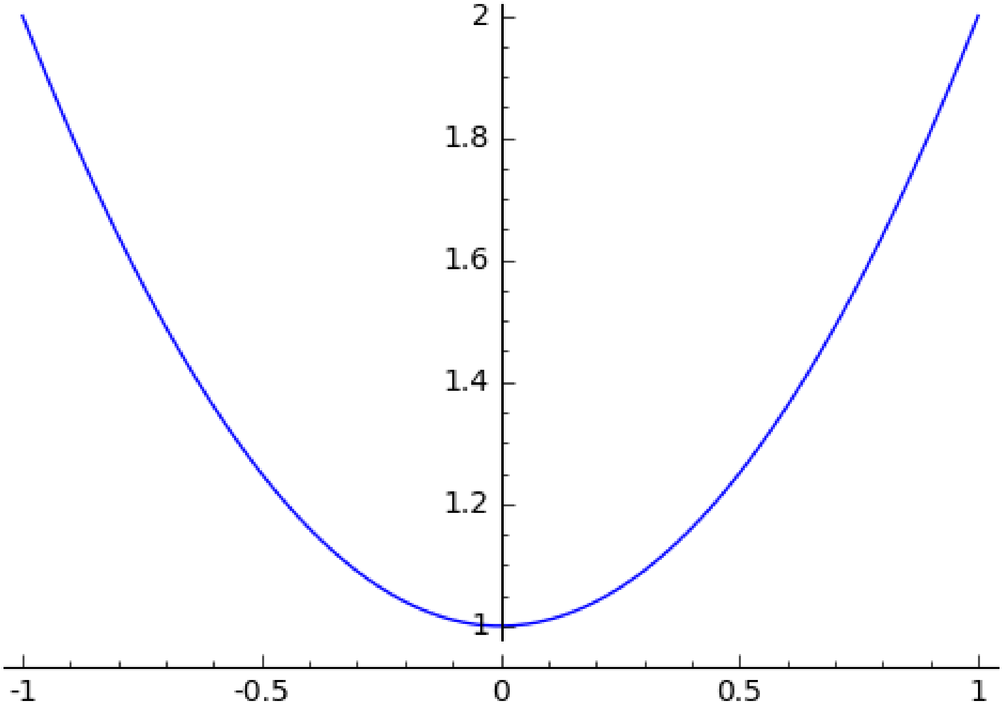
```
var('x') ## Define a symbolic variable named x
plot(x, (x, 0, 1)) ## Plot a function specified in terms of a
symbolic variable
```



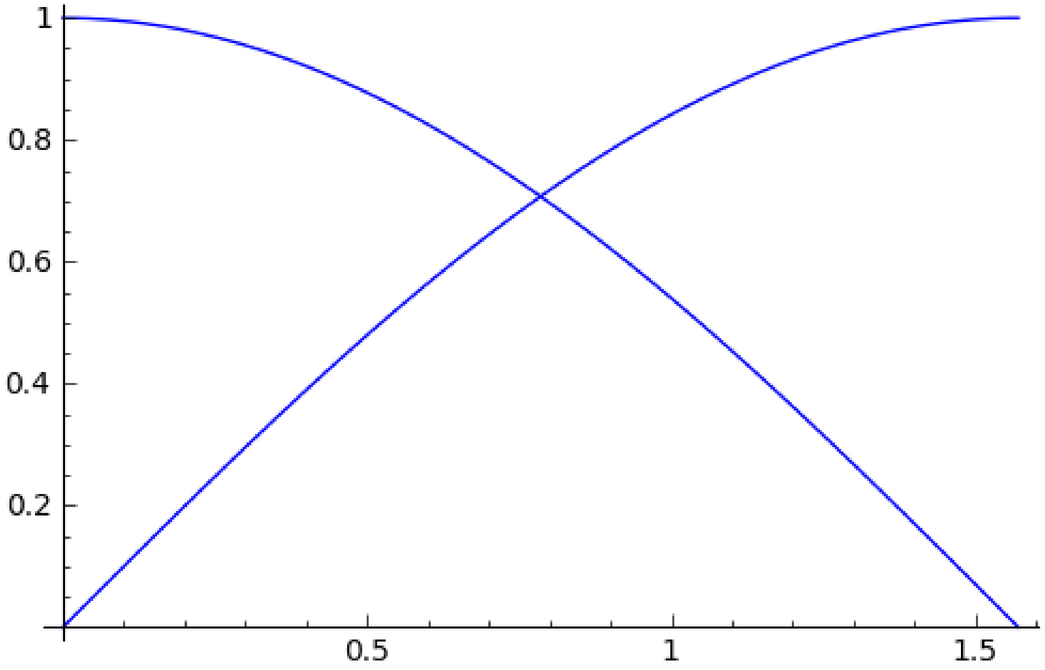
```
def f(x):
    return(x^2+1)

plot(f, -1, 1) ## Plot a user-defined function
```



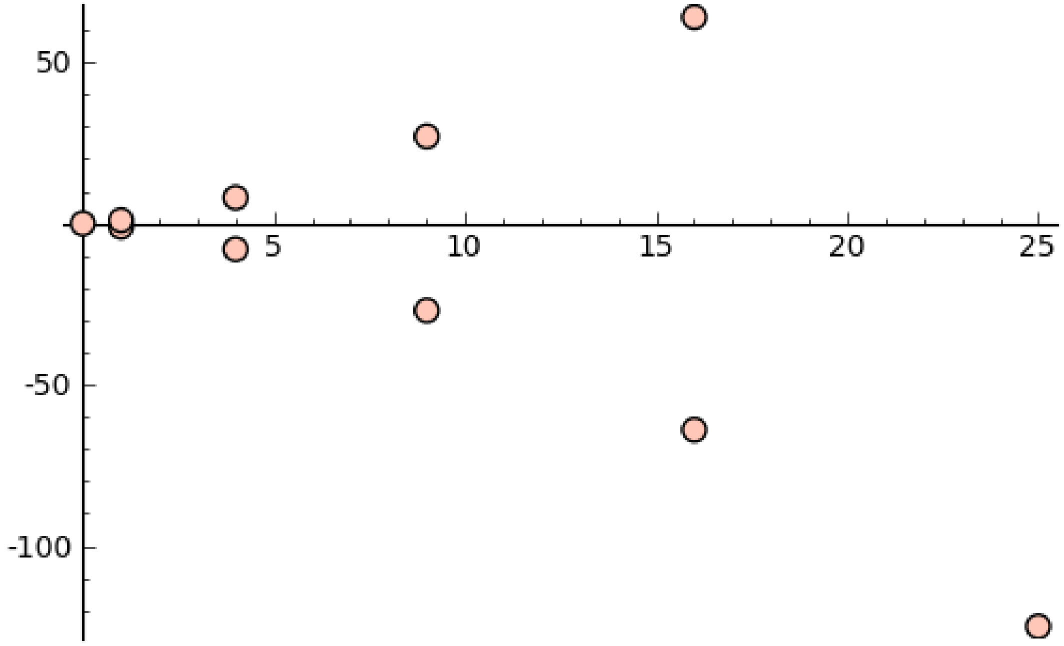
```
plot(lambda x: x^2+1, -1, 1) ## Same thing using an inline
function
```



```
plot([sin(x), cos(x)], (x, 0, pi/2)) ## Plot multiple functions
on the same axes
```



```
scatter_plot([(x^2,x^3) for x in range(-5,5)]) ## plot a
collection of points
```



Exception handling

This last point is a bit technical, but it will arise in the later lectures. As we saw earlier, if you make a mistake, Sage normally interrupts the computation to return an error message explaining what went wrong.

```
print (3 / (1-1))
```

```
    Traceback (click to the left of this block for traceback)
    ...
    ZeroDivisionError: Rational division by zero
```

One can explicitly create these errors as a debugging tool.

```
def f(n):
    if n < 0:
        raise ValueError, "Argument of f must be nonnegative"
    return(sqrt(n))

print f(1)
print f(-1)
```

```
    1
    Traceback (click to the left of this block for traceback)
    ...
    ValueError: Argument of f must be nonnegative
```

One can also catch these errors to a limited extent, using the try and except commands.

```
def f(x,y):
    try:
        n = 1/x
        print "inversion complete"
        return(n + y[0])
    except ZeroDivisionError:
        return 0

print f(1, [2,3]) ## normal evaluation
print f(0, [4,5]) ## exception encountered, skipping the
internal print statement
print f(2, []) ## this exception is not caught, so causes an
error
print f(3, [3, 4]) ## we never get this far
```

```
    inversion complete
    3
    0
    inversion complete
    Traceback (click to the left of this block for traceback)
    ...
    IndexError: list index out of range
```

Conclusion: what else can Sage do?

To conclude, we point out a few more advanced things that can be accomplished using Sage, just to give the flavor of what the possibilities are. These are mostly complementary to the advanced functionality for elliptic curves that will be highlighted in the subsequent lectures.

- Communicate with other mathematical software. Sage itself ships with many well-regarded free software packages, such as Pari (number theory), Gap (group theory), Singular (commutative algebra), R (statistics), etc. Sage can also communicate with nonfree packages such as Mathematica, Maple, Matlab, and Magma if you have them installed. This means that you can access the power of many software packages while (mostly) only having to deal with one well-designed programming language!
- Communicate with TeX at various levels. For instance, any Sage object has a method that returns its representation in TeX. An even more spectacular example is SageTeX, a TeX package that allows your TeX document to pass inputs to Sage and retrieve the answers!
- Annotate notebooks by inserting formatted text and even TeX code between the cells, as in this presentation.

AMS Short Course -- K. Ribet

AMS Short Course: Computing with Elliptic Curves using Sage

January 2-3, 2012

Lecture 2: Elliptic Curves over Finite Fields

Kenneth A. Ribet, UC Berkeley

Some references: http://www.sagemath.org/doc/reference/sage/schemes/elliptic_curves/ell_finite_field.html and http://www.sagemath.org/doc/reference/sage/schemes/elliptic_curves/formal_group.html

This lecture is about elliptic curves over finite fields, so we should pick one. We'll start with $p = 23$ and consider the fields with p elements and with p^2 elements:

```
p=23; k=GF(p); K.<a> = GF(p^2)
```

```
p; k ; K
```

```
    23
    Finite Field of size 23
    Finite Field in a of size 23^2
```

```
a.minimal_polynomial()
```

```
    x^2 + 21*x + 5
```

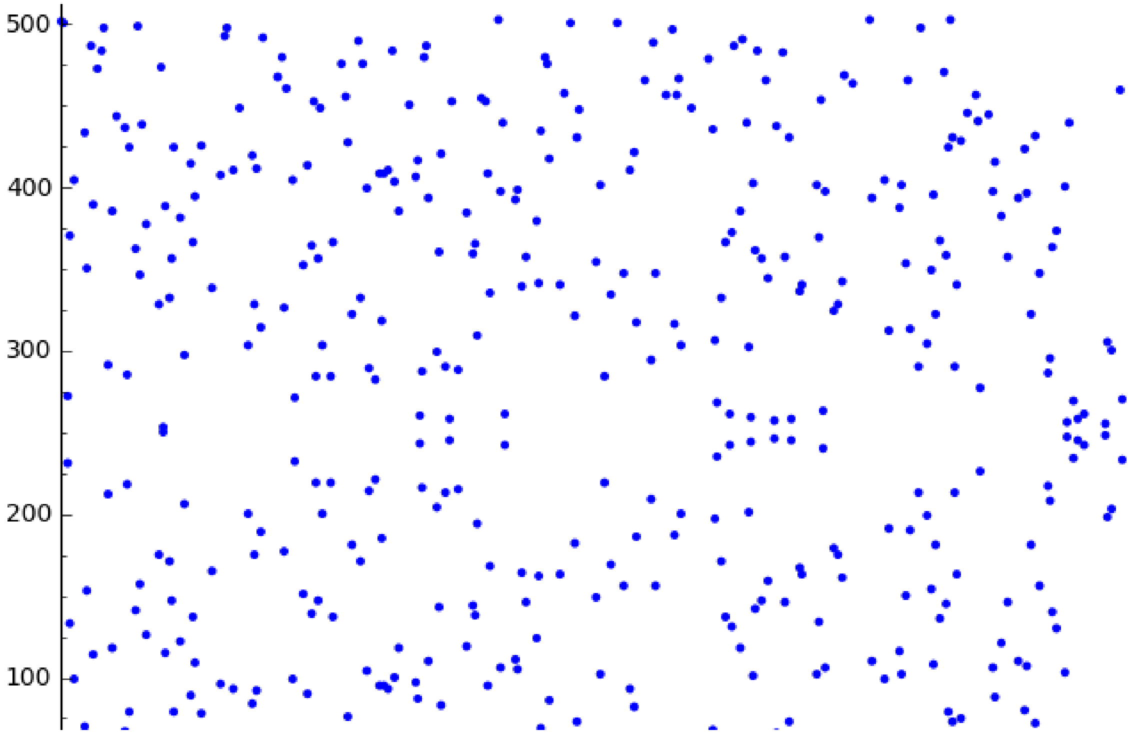
There are at least three different ways to define an elliptic curve over k or K: (1) we can specify the two coefficients of a short-form Weierstrass equation; (2) we can specify the five coefficients of a long-form Weierstrass equation; (3) we can specify a j-invariant and let sage pick a curve with that j-invariant.

```
E1 = EllipticCurve(k, [1,2]); E2 = EllipticCurve(k,[1,2,3,4,5]); E3 =
EllipticCurve(j=k(2904)) ; E1; E2; E3
```

```
    Elliptic Curve defined by y^2 = x^3 + x + 2 over Finite Field of
    size 23
    Elliptic Curve defined by y^2 + x*y + 3*y = x^3 + 2*x^2 + 4*x + 5
    over Finite Field of size 23
    Elliptic Curve defined by y^2 = x^3 + 15*x + 7 over Finite Field of
    size 23
```

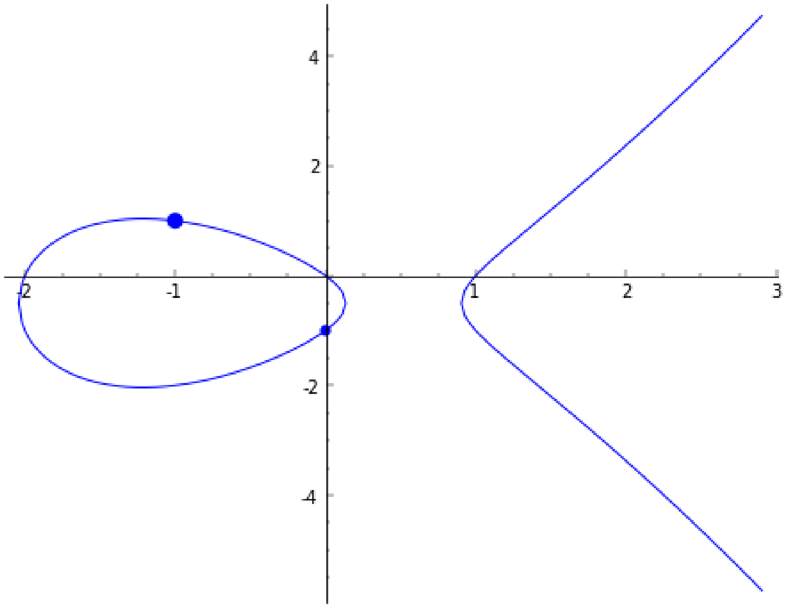
Sage will "graph" elliptic curves over finite fields, but the resulting image may not add much to your day.

```
EllipticCurve(GF(503),[4,4]).plot()
```



Even when working over finite fields, our mental image of an elliptic curve will tend to be like the picture below:

```
E=EllipticCurve('389a');
G=plot(E, dpi=70);
G += sum([plot(p,pointsize=100*p.height()) for p in E.gens()]);
show(G)
```



This is the curve that I am wearing.

I will return to $E1$, $E2$ and $E3$ in a couple of minutes, but first I want to look at a fourth curve, which I'll call E, and find a single non-zero point on this curve "by hand."

```
E=EllipticCurve(GF(144169),[-1,3]); E
```

```
    Elliptic Curve defined by y^2 = x^3 + 144168*x + 3 over Finite Field
    of size 144169
```

I am fond of the prime number 144169 for two reasons. First, it's the discriminant of the Hecke algebra associated to the space of cusp forms of weight 24 on $\mathbf{SL}(2,\mathbf{Z})$. Second, it's the concatenation of 12^2 and 13^2.

```
E.abelian_group()
```

```
    Additive abelian group isomorphic to Z/143944 embedded in Abelian
    group of points on Elliptic Curve defined by y^2 = x^3 + 144168*x +
    3 over Finite Field of size 144169
```

The group of rational points on this curve is cyclic of order $2^3 \cdot 19 \cdot 947$.

```
Mod(1^3 + 144168*1 + 3, 144169); kronecker(1^3 + 144168*1 + 3, 144169)
```

```
    3
    1
```

If $x = 1$, the RHS of the definining equation for E is 3, which is a square mod 144169. Can we find a square root of 3 mod 144169? Cipolla's algorithm, which I will explain at the end of the talk, outputs 79896 as a square root of 3.

```
P=E([1,79896]); P; P.order()
```

```
    (1 : 79896 : 1)
    143944
```

We were able to find a generator for the group of rational points of E simply by messing around. The a priori probability for success was a bit larger than 0.47:

```
euler_phi(143944)/143944.
```

```
    0.473184016006225
```

End of interlude. Now let's go back to the three original curves and compute some of their invariants.

```
E1.j_invariant(); E2.j_invariant(); E3.j_invariant()
```

```
    19
    22
    6
```

Note that $19 \equiv -4$ happens to be the unique supersingular j-invariant in characteristic 23, other than $j = 0$ and $j = 1728$, which are supersingular because 23 is -1 mod 3 and mod 4, respectively. It's serendipitous that we stumbled on the j-invariant 19 by entering 1 and 2 as the coefiicients of our short Weierstrass equation.

```
E1.is_supersingular(); E2.is_supersingular(); E3.is_supersingular()
```

```
    True
    False
    False
```

```
E1.hasse_invariant(); E2.hasse_invariant(); E3.hasse_invariant()
```

```
0
17
8
```

```
E1.cardinality(); E2.cardinality(); E3.cardinality()
```

```
24
30
16
```

```
E1.trace_of_frobenius(); E2.trace_of_frobenius(); E3.trace_of_frobenius()
```

```
0
-6
8
```

If t is the trace of Frobenius of a curve E, the number of points of E over k is $1+p-t$. Once we know t, we can compute the number of points of E over any finite extension of K. In particular, the number of points of E over K is $(1+p)^2 - t^2 = ($

.

```
(1+p-E1.trace_of_frobenius())*(1+p+E1.trace_of_frobenius()); (1+p-
E2.trace_of_frobenius())*(1+p+E2.trace_of_frobenius()); (1+p-
E3.trace_of_frobenius())*(1+p+E3.trace_of_frobenius())
```

```
576
540
512
```

We can check this in all three cases, but let's just choose one case---say the middle one.

```
E2.cardinality(extension_degree=2)
```

```
540
```

Sage takes great pleasure in computing the abelian group structure of groups of points on an elliptic curve over a finite field.

```
E1.abelian_group(); E2.abelian_group(); E3.abelian_group()
```

```
Additive abelian group isomorphic to Z/2 + Z/12 embedded in Abelian
group of points on Elliptic Curve defined by y^2 = x^3 + x + 2 over
Finite Field of size 23
Additive abelian group isomorphic to Z/30 embedded in Abelian group
of points on Elliptic Curve defined by y^2 + x*y + 3*y = x^3 + 2*x^2
+ 4*x + 5 over Finite Field of size 23
Additive abelian group isomorphic to Z/2 + Z/8 embedded in Abelian
group of points on Elliptic Curve defined by y^2 = x^3 + 15*x + 7
over Finite Field of size 23
```

What happens over K?

```
E1.change_ring(K).abelian_group(); E2.change_ring(K).abelian_group();
E3.change_ring(K).abelian_group()
```

```
Additive abelian group isomorphic to Z/24 + Z/24 embedded in Abelian
group of points on Elliptic Curve defined by y^2 = x^3 + x + 2 over
Finite Field in a of size 23^2
Additive abelian group isomorphic to Z/6 + Z/90 embedded in Abelian
group of points on Elliptic Curve defined by y^2 + x*y + 3*y = x^3 +
2*x^2 + 4*x + 5 over Finite Field in a of size 23^2
Additive abelian group isomorphic to Z/16 + Z/32 embedded in Abelian
group of points on Elliptic Curve defined by y^2 = x^3 + 15*x + 7
```

```
    group of points on Elliptic Curve defined by y^2 = x^3 + 15*x + 7
    over Finite Field in a of size 23^2
```

So $E3$ has two "independent" points of order 16; we can take the Weil pairing of these two points to get a 16th root of unity in K :

```
P,Q = E3.change_ring(K).gens(); P; Q; P.order(); Q.order()
```

```
    (9*a : 6*a + 2 : 1)
    (4*a + 19 : 13*a + 6 : 1)
    32
    16
```

```
(2*P).weil_pairing(Q,16)
```

```
    4*a + 17
```

To check that this element is indeed a 16th root of unity, it suffices to check that its eighth power is $-1 \equiv 22$.

```
(_)^8
```

```
    22
```

If we exchange P and Q, the value of the Weil pairing is inverted:

```
((2*P).weil_pairing(Q,16))*(Q.weil_pairing(2*P,16))
```

```
    1
```

Isogenies

We'll divide $E3$ by a cyclic subgroup of order 32 defined over K:

```
E3overK = E3.change_ring(K)
```

```
phi = E3overK.isogeny(P); phi
```

```
    Isogeny of degree 32 from Elliptic Curve defined by y^2 = x^3 + 15*x
    + 7 over Finite Field in a of size 23^2 to Elliptic Curve defined by
    y^2 = x^3 + (4*a+9)*x + (16*a+7) over Finite Field in a of size 23^2
```

```
Eprime = phi.codomain(); Eprime
```

```
    Elliptic Curve defined by y^2 = x^3 + (4*a+9)*x + (16*a+7) over
    Finite Field in a of size 23^2
```

```
Eprime.is_isogenous(E3)
```

```
    True
```

Over finite fields, isogenous curves have the same number of elements.

```
Eprime.cardinality()
```

```
    512
```

Automorphisms

A "generic" curve has only $\{\pm 1\}$ as its group of automorphisms. The curves with $j = 0$ and $j = 1728$ have actions of μ_3 and μ_4, respectively.

```
E1.change_ring(K).automorphisms()
```

```
[Generic endomorphism of Abelian group of points on Elliptic Curve
defined by y^2 = x^3 + x + 2 over Finite Field in a of size 23^2
  Via:  (u,r,s,t) = (1, 0, 0, 0), Generic endomorphism of Abelian
group of points on Elliptic Curve defined by y^2 = x^3 + x + 2 over
Finite Field in a of size 23^2
  Via:  (u,r,s,t) = (22, 0, 0, 0)]
```

```
EllipticCurve(j=k(0)).automorphisms()
```

```
[Generic endomorphism of Abelian group of points on Elliptic Curve
defined by y^2 = x^3 + 1 over Finite Field of size 23
  Via:  (u,r,s,t) = (1, 0, 0, 0), Generic endomorphism of Abelian
group of points on Elliptic Curve defined by y^2 = x^3 + 1 over
Finite Field of size 23
  Via:  (u,r,s,t) = (22, 0, 0, 0)]
```

```
EllipticCurve(j=K(0)).automorphisms()
```

```
[Generic endomorphism of Abelian group of points on Elliptic Curve
defined by y^2 = x^3 + 1 over Finite Field in a of size 23^2
  Via:  (u,r,s,t) = (1, 0, 0, 0), Generic endomorphism of Abelian
group of points on Elliptic Curve defined by y^2 = x^3 + 1 over
Finite Field in a of size 23^2
  Via:  (u,r,s,t) = (22, 0, 0, 0), Generic endomorphism of Abelian
group of points on Elliptic Curve defined by y^2 = x^3 + 1 over
Finite Field in a of size 23^2
  Via:  (u,r,s,t) = (4*a + 7, 0, 0, 0), Generic endomorphism of
Abelian group of points on Elliptic Curve defined by y^2 = x^3 + 1
over Finite Field in a of size 23^2
  Via:  (u,r,s,t) = (4*a + 8, 0, 0, 0), Generic endomorphism of
Abelian group of points on Elliptic Curve defined by y^2 = x^3 + 1
over Finite Field in a of size 23^2
  Via:  (u,r,s,t) = (19*a + 15, 0, 0, 0), Generic endomorphism of
Abelian group of points on Elliptic Curve defined by y^2 = x^3 + 1
over Finite Field in a of size 23^2
  Via:  (u,r,s,t) = (19*a + 16, 0, 0, 0)]
```

```
(19*a+16)^3
```

```
22
```

```
EllipticCurve(j=K(1728)).automorphisms()
```

```
[Generic endomorphism of Abelian group of points on Elliptic Curve
defined by y^2 = x^3 + x over Finite Field in a of size 23^2
  Via:  (u,r,s,t) = (1, 0, 0, 0), Generic endomorphism of Abelian
group of points on Elliptic Curve defined by y^2 = x^3 + x over
Finite Field in a of size 23^2
  Via:  (u,r,s,t) = (22, 0, 0, 0), Generic endomorphism of Abelian
group of points on Elliptic Curve defined by y^2 = x^3 + x over
Finite Field in a of size 23^2
  Via:  (u,r,s,t) = (11*a + 12, 0, 0, 0), Generic endomorphism of
Abelian group of points on Elliptic Curve defined by y^2 = x^3 + x
over Finite Field in a of size 23^2
  Via:  (u,r,s,t) = (12*a + 11, 0, 0, 0)]
```

```
(12*a + 11)^2
```

```
22
```

```
F.<b>= GF(4); len(EllipticCurve(j=F(0)).automorphisms())
```

```
24
```

Embedding degree

At this point, we can fool around a bit with some large primes, just to show that sage does not choke on large numbers. The example here is from an article by David Freeman, a mathematical cryptographer who is now at Stanford.

```
q=6462310997348816962203124910505252082673338846966431201635262694402825461643;
 factor(q)
```

In other words, q is a large prime, in fact a 252-bit prime:

```
len(q.str(2))
```

```
E=EllipticCurve(GF(q),[-
3,4946538166640251374274628820269694144249181776013154863288086212076808528141])
```

```
time n=E.order(); n; is_prime(n)
```

In other words, this curve has a group of rational points that is (cyclic) of prime order; we're calling n the order. For Freeman, the striking fact about this example is that you can get the full group $E[n] \approx (\mathbf{Z}/n\mathbf{Z})^2$ to be rational by passing to a relatively small extension of the base field (the field with q elements):

```
Mod(q,n).multiplicative_order()
```

In other words, if we pass to a degree-10 extension of the base field, we force all the n-division points on the curve to become rational.

```
time E.cardinality(extension_degree=10)/n^2
```

The fact that the ratio is an integer means that n^2 does divide the group of points on E with coordinates in the large field.

The fact that the ratio is an integer means that n^2 does divide the group of points on E with coordinates in the large field. As you are about to see, it is easy to find one point of order n on E. Finding a second independent point is a computational challenge that I haven't attempted.

```
time E.gens()
```

Trying to run the following command will get us into a computation for which I can't estimate the "arrival time." Therefore, we won't go there!

```
#  K.<a>= GF(q^10); E.change_ring(K).gens()
```

Supersingular j-invariants in characteristic 103

```
from sage.schemes.elliptic_curves.ell_finite_field import
supersingular_j_polynomial
```

```
f=supersingular_j_polynomial(103); type(f)
```

```
<type
'sage.rings.polynomial.polynomial_zmod_flint.Polynomial_zmod_flint'&\
gt;
```

```
supersingular_j_polynomial(103).factor()
```

```
(j + 34) * (j + 69) * (j + 79) * (j + 80) * (j^2 + 63*j + 69) * (j^2
+ 84*j + 73)
```

```
R.<x>=L[]
```

```
R
```

```
Univariate Polynomial Ring in x over Finite Field in c of size 103^2
```

```
f(x)
```

```
x^8 + 100*x^7 + 84*x^6 + 83*x^5 + 70*x^4 + 58*x^3 + 24*x^2 + 15*x +
64
```

```
factor(_)
```

```
(x + 34) * (x + 69) * (x + 79) * (x + 80) * (x + 40*c + 63) * (x +
46*c + 19) * (x + 57*c + 65) * (x + 63*c)
```

```
supersingular_j_polynomial(103).degree()
```

```
8
```

```
floor((103-1)/12)
```

```
8
```

```
E=EllipticCurve(j=L(-63*c))
```

```
E.is_supersingular()
```

```
True
```

Formal groups

```
G1=E1.formal_group(); G3 = E3.formal_group()
```

The group law attached to a formal group is a power series in two variables, called $t1$ and $t2$ by sage.

```
G1.group_law(15); G3.group_law(15)
```

```
t1 + O(t1^15) + (1 + 21*t1^4 + 17*t1^6 + 21*t1^8 + 7*t1^10 +
18*t1^12 + 19*t1^14 + O(t1^15))*t2 + (19*t1^3 + 5*t1^5 + 3*t1^9 +
10*t1^11 + 4*t1^13 + O(t1^15))*t2^2 + (19*t1^2 + 16*t1^4 + 8*t1^6 +
t1^8 + 10*t1^10 + 16*t1^12 + 3*t1^14 + O(t1^15))*t2^3 + (21*t1 +
16*t1^3 + 16*t1^5 + 5*t1^7 + t1^9 + 18*t1^11 + 12*t1^13 +
O(t1^15))*t2^4 + (5*t1^2 + 16*t1^4 + 20*t1^6 + 18*t1^8 + 7*t1^10 +
7*t1^12 + 21*t1^14 + O(t1^15))*t2^5 + (17*t1 + 8*t1^3 + 20*t1^5 +
8*t1^7 + 6*t1^9 + 4*t1^11 + 10*t1^13 + O(t1^15))*t2^6 + (5*t1^4 +
8*t1^6 + 12*t1^8 + 18*t1^10 + 15*t1^12 + 5*t1^14 + O(t1^15))*t2^7 +
(21*t1 + t1^3 + 18*t1^5 + 12*t1^7 + t1^9 + 22*t1^11 + 21*t1^13 +

O(t1^15))*t2^8 + (3*t1^2 + t1^4 + 6*t1^6 + t1^8 + 20*t1^10 + 6*t1^12
O(t1^15))*t2^8 + (3*t1^2 + t1^4 + 6*t1^6 + t1^8 + 20*t1^10 + 6*t1^12
+ 22*t1^14 + O(t1^15))*t2^9 + (7*t1 + 10*t1^3 + 7*t1^5 + 18*t1^7 +
20*t1^9 + 17*t1^11 + t1^13 + O(t1^15))*t2^10 + (10*t1^2 + 18*t1^4 +
4*t1^6 + 22*t1^8 + 17*t1^10 + 13*t1^12 + O(t1^15))*t2^11 + (18*t1 +
16*t1^3 + 7*t1^5 + 15*t1^7 + 6*t1^9 + 13*t1^11 + 8*t1^13 +
O(t1^15))*t2^12 + (4*t1^2 + 12*t1^4 + 10*t1^6 + 21*t1^8 + t1^10 +
8*t1^12 + 14*t1^14 + O(t1^15))*t2^13 + (19*t1 + 3*t1^3 + 21*t1^5 +
5*t1^7 + 22*t1^9 + 14*t1^13 + O(t1^15))*t2^14 + O(t2^15)
t1 + O(t1^15) + (1 + 16*t1^4 + 2*t1^6 + 10*t1^8 + 11*t1^10 + 6*t1^12
+ t1^14 + O(t1^15))*t2 + (9*t1^3 + 6*t1^5 + 8*t1^9 + 19*t1^11 +
22*t1^13 + O(t1^15))*t2^2 + (9*t1^2 + 10*t1^4 + 6*t1^6 + 18*t1^8 +
11*t1^10 + 19*t1^12 + 18*t1^14 + O(t1^15))*t2^3 + (16*t1 + 10*t1^3 +
12*t1^5 + 21*t1^7 + 18*t1^9 + 7*t1^11 + 20*t1^13 + O(t1^15))*t2^4 +
(6*t1^2 + 12*t1^4 + 15*t1^6 + 11*t1^8 + 4*t1^10 + t1^12 + 10*t1^14 +
O(t1^15))*t2^5 + (2*t1 + 6*t1^3 + 15*t1^5 + t1^7 + 10*t1^9 +
22*t1^11 + O(t1^15))*t2^6 + (21*t1^4 + t1^6 + 20*t1^8 + 19*t1^10 +
12*t1^12 + 16*t1^14 + O(t1^15))*t2^7 + (10*t1 + 18*t1^3 + 11*t1^5 +
20*t1^7 + 18*t1^9 + 5*t1^11 + 6*t1^13 + O(t1^15))*t2^8 + (8*t1^2 +
18*t1^4 + 10*t1^6 + 18*t1^8 + 21*t1^10 + 15*t1^12 + O(t1^15))*t2^9 +
(11*t1 + 11*t1^3 + 4*t1^5 + 19*t1^7 + 21*t1^9 + 7*t1^11 + t1^13 +
O(t1^15))*t2^10 + (19*t1^2 + 7*t1^4 + 22*t1^6 + 5*t1^8 + 7*t1^10 +
21*t1^12 + 2*t1^14 + O(t1^15))*t2^11 + (6*t1 + 19*t1^3 + t1^5 +
12*t1^7 + 15*t1^9 + 21*t1^11 + 4*t1^13 + O(t1^15))*t2^12 + (22*t1^2
+ 20*t1^4 + 6*t1^8 + t1^10 + 4*t1^12 + 13*t1^14 + O(t1^15))*t2^13 +
(t1 + 18*t1^3 + 10*t1^5 + 16*t1^7 + 2*t1^11 + 13*t1^13 +
O(t1^15))*t2^14 + O(t2^15)
```

```
G0=EllipticCurve(j=F(0)).formal_group(); G0
```

```
Formal Group associated to the Elliptic Curve defined by y^2 + y =
x^3 over Finite Field in b of size 2^2
```

```
G0.mult_by_n(2)
```

```
t^4 + O(t^10)
```

```
G1.mult_by_n(23,prec=30)
```

```
O(t^30)
```

```
G3.mult_by_n(23, prec=30)
```

```
8*t^23 + O(t^30)
```

Square roots mod p

The problem is this: if a is a non-zero integer mod p, it's easy to see whether a is a square mod p by computing $a^{(p-1)/2}$ mod p: the result is $+1$ if and only if a is a square. Suppose it is? How do we find a square root of a? There's an interesting algorithm, Cipolla's algorithm http://en.wikipedia.org/wiki/Cipolla's_algorithm, that solves the problem. I learned about it from my colleague Matt Baker. It's easy to implment in sage.

The method is as follows: take random values of t until you find a t such that $t^2 - a$ is a non-square. To $\mathbf{F}_p$, adjoin a square root of $t^2 - a$; call it ω. Then $(a + \omega)^{(p+1)/2}$ is a square root of a.

```
p=1234567891; a= 11; kronecker(a,p)
```

```
1
```

Thus 11 is a square mod $p = 1234567891$. Can we find its square root?

```
t=7; kronecker(t^2-a,p)
```

```
-1
```

The valiue $t=7$ wasn't hard to find: I tried $t=1,\ldots$ until I got to 7.

```
R.<x> = PolynomialRing(GF(p)); R
```

```
    Univariate Polynomial Ring in x over Finite Field of size 1234567891
```

```
t=7; b=(t^2-a)%p; S.<omega> = R.quo((x^2-b)); S
```

```
    Univariate Quotient Polynomial Ring in omega over Finite Field of
    size 1234567891 with modulus x^2 + 1234567853
```

```
squareroot= (t+omega)^((p+1)/2); squareroot
```

```
    77590393
```

In other words, the algorithm outputs 77590393 as a square root of 11 mod p. We should check the result!

```
77590393^2%p
```

```
    11
```

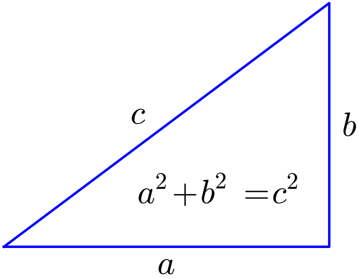
Fin

Solving Cubic Equations

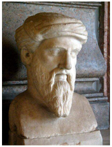
Benedict Gross and William Stein

January, 2012

Algebraic equations



$$a^2+b^2 = c^2$$

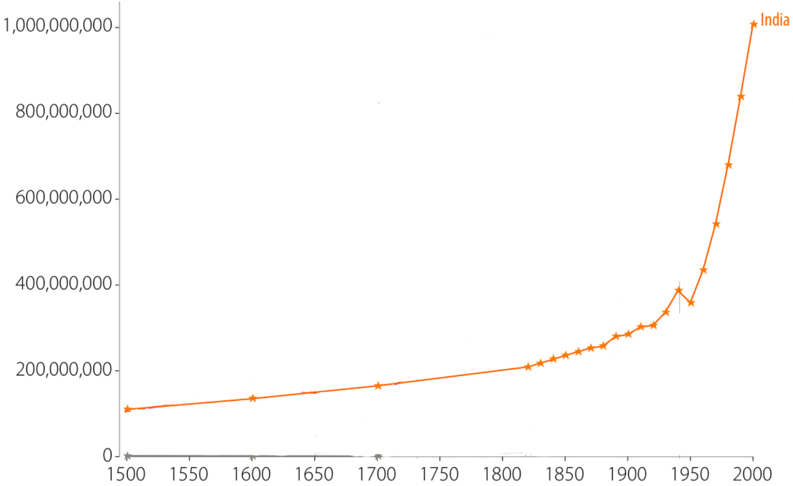


Pythagoras (600 BCE)   Baudhāyana (800 BCE)

Differential equations

$$F'(T) = F(T) \qquad dF/dT = F \qquad F(0) = 1$$

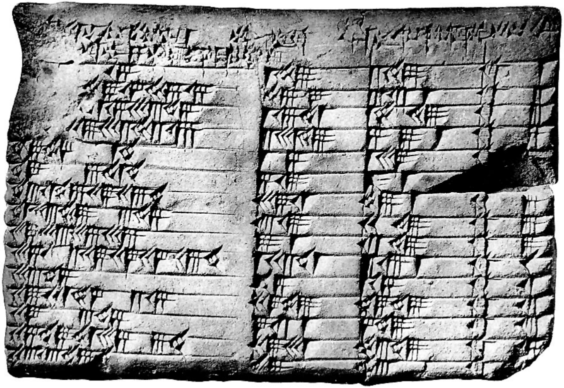
$$F(T) = exp(T) = 1 + T + T^2/2 + T^3/6 + T^4/24 + T^5/120 + ...$$



Pythagorean triples

$a^2 + b^2 = c^2$ has solutions $(3, 4, 5), (5, 12, 13), (7, 24, 25), \ldots$

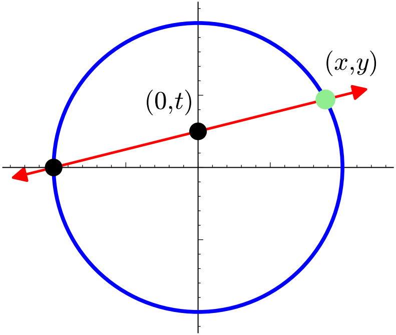
There are more solutions on a Babylonian tablet (1800 BCE):



$(3, 4, 5)$
$(5, 12, 13)$
$(7, 24, 25)$
$(9, 40, 41)$
$(11, 60, 61)$
$(13, 84, 85)$
$(15, 8, 17)$
$(21, 20, 29)$
$(33, 56, 65)$
$(35, 12, 37)$
$(39, 80, 89)$
$(45, 28, 53)$
$(55, 48, 73)$
$(63, 16, 65)$
$(65, 72, 97)$

The general solution of $a^2 + b^2 = c^2$

$x = a/c$ and $y = b/c$ satisfy the equation $x^2 + y^2 = 1$



$$t = \frac{y}{1+x} \qquad x = \frac{1-t^2}{1+t^2} \qquad y = \frac{2t}{1+t^2}$$

Write $t = p/q$. Then

$$x = \frac{q^2 - p^2}{q^2 + p^2} \qquad y = \frac{2qp}{q^2 + p^2}$$

$$a = q^2 - p^2 \qquad b = 2qp \qquad c = q^2 + p^2$$

$$t = 1/2 \longrightarrow (a, b, c) = (3, 4, 5)$$

$$t = 2/3 \longrightarrow (a, b, c) = (5, 12, 13)$$

$$t = 3/4 \longrightarrow (a, b, c) = (7, 24, 25)$$

Cubic equations

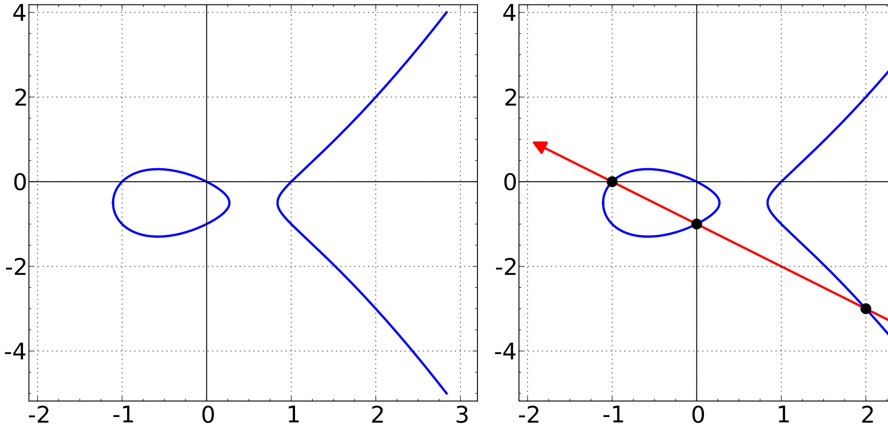
After linear and quadratic equations come cubic equations, like

$$x^3 + y^3 = 1 \qquad\qquad y^2 + y = x^3 - x$$

Here there may be either a finite or an infinite number of rational solutions.

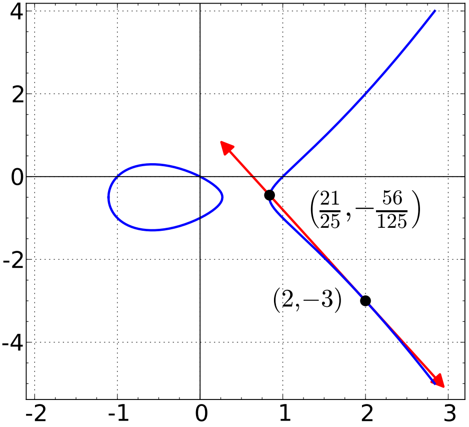
The graph

$$y^2 + y = x^3 - x$$



The limit of a secant line is a tangent

$$y^2 + y = x^3 - x$$



Large solutions

If the number of solutions is infinite, they quickly become large.

$$y^2 + y = x^3 - x$$

```
(0, 0)
(1, 0)
(-1, -1)
(2, -3)
(1/4, -5/8)
(6, 14)
(-5/9, 8/27)
(21/25, -69/125)
(-20/49, -435/343)
(161/16, -2065/64)
(116/529, -3612/12167)
(1357/841, 28888/24389)
(-3741/3481, -43355/205379)
(18526/16641, -2616119/2146689)
(8385/98596, -28076979/30959144)
(480106/4225, 332513754/274625)
(-239785/2337841, 331948240/3574558889)
(12551561/13608721, -8280062505/50202571769)
(-59997896/67387681, -641260644409/553185473329)
(683916417/264517696, -18784454671297/4302115807744)
(1849037896/6941055969, -318128427505160/578280195945297)
(51678803961/12925188721, 10663732503571536/1469451780501769)
(-270896443865/384768368209, 6631634575107447/238670664494938073)
```

Even the simplest solution can be large

$$y^2 + y = x^3 - 5115523309x - 140826120488927$$

Numerator of x-coordinate of smallest solution (5454 digits):

```
36777053718667750661400564234182717008793226949228558472621877006165354634927101580536513437032674306114130646450005288670465199839976647884079191530786174150
72739338026281573250924797082687602171017553858718167805487654785022844156276828471927526818990949626599378706300367603592935770218062374839710749312284163465
07852381696883227650072039964481597215995993299744934117106289850389364006552497835877740257534533113775202882210048356163645919345794812074571029660897173224
37033770105616573500859064029709029870912150626669726646199320182539736999955086814229431275632217741073053282806475960497536924235099356803072693704991160726
41097827468479512837941192989412144907943309029865829912295694015235199387427463761071907702040105138183490127866378892547110594555551738109049119276198990318
55149292325338589831979737026402711049742594116000380601480839982975557506035851728035645241044229165029649347049289119188596869401159325131363345962579503132
33984727542244009455382470518922565367745951286311791172183855293430912450813449336643740809392436203974991190741697350414232211175705858420072502263211616472
01649986417295226774605259994990779421258204288795260637356926859910185168629387960475973239865371541712483169437963732171919939969937146546295368843960579247
90938647656663281596178145722116098216500930333824321806726937018190136190556573208807048355335567078793126656928657859036779350593274598717379730880724034301
86773944374984180945671588419372032890146155265988262840584220975675716781666213994508186464210853359598997571625925924015283405094065447961714768592250085694
44498220453860921224090969785448172188478976405134778065983291776042463808123777390491844755507773416209859765703930378802827649670195524084007307548226764414
81715385344001979832232652414888335865567377214360456003296961668177481944809066257442596772347829664126972931904101685281128944780074646796760942430959617022
25747987408940356496503888537981786692004892981452026849367750705907376590267163808736648849670283632626857459312324510742034887810176312389334765702027559124
88242478005942708620520821859733932900091898676772594580806760650987034535395255769756395437005076256407298723407894063143944684005844559206833619762001218344
30751233901473228497490561998078486251074993528871318797403348087370426900997556442577081254910572185107856605139877331015042842121106080690743578173268489400
49905689831262195394796701235841454775289708109709179574420369768404606625566320124229276012675987126600451637743296191727204021714708356339998761242059527579
20338556769918233682548621595584500438080514815332972700352873822470382792932239463850701180823069589872686033969240544031038574440588486055874154005176700326
31121206127732481340391088277796488544415738156553014768406246154666005139690428085145098272500791416214774673484501826722500527091164944262537169595848931680
75409677471286049057274622409403118704320452610723920107960346829752289510659856743701508334879787536416279769396881980413954888575152826871522370782603587052 3
02844262030644936842506142828799181077337962070672500038239594129356776240932360470386373655773263995890088045077860119731559277310730347065365574614438066227
07622411087809371872157210456836892493613836792026761820382217165481998924123604782787923229739171920575447007099501678380795077013113325989801385729993920818
30165442425133956460687682012192837224621339985921328279251116804395344383979390113997419447930029756609766453919938465190843618873242881837330238304638885942
79378938418880142666851776166056447837041357949318307502656863359340665652409440494482130055919971289855607602603992142786359126343515867623548693540215307461
89992899582554597632108309638569296964800046983072736238483149014714600896056552029642747991419063454749142059564274298254654925893866404955146903330024475746
16354371499624965242017117105423172633649354158697143177894405148105963373839941141857432381177094972972684361267292500063135565983416420055444131545100343345
24662047071238116636236628372968629480617587599286317636619851856158018862057707210320063041448677873470583163922956715800916558720872094859132869301288586404
42589125454268580397484571921012318872311624898317615607628176460097441336323549031828235965636277950827328087547939511112374216436584203379248450122647406094
03517113074066372354767593988595936388113589303510201838944421274614625032834824261067352402237899497839202009881472197450206269281573668922975906582209394279
53187053452755989894263352359355056053114113015603211922694308617337435444029085864973053536009094312149332025225287171092144929593300160658102876231441792884
66664888540622702346704213752456372574449563979215782406566937885352945871994541770838871930542220307771671498466518108722622109421676741544945695403509866953
16727762828023246483921500347404889696803754466002975574006558127013908324990321257223041794224979546710070039394431032500967717918210997094334680733501444683
96122825088243240736795841228512083604591663154848919522994493400258965092989359393577217235439331087432419973874470183959253201676376403284079570698454395013
81234605867495003402016724626400855369636521155009147176245904149069225438646928549072337653348704931901764847439772432025275648964681387210234070849306330191
79038041239611544624083258348136637213230084906083526213683231531105290336750385743792050893130528314337942393060136915457253067727886206663888425022179164712
3563828956462530983567929499493346622977494903591722345188975062941907415400740881
```

Denominator:

```
42550442729747398883181472438884631491027966762628698441290370019390690825369783206694892509489159845392479060050865239326960543711865223880415469150998005445
00632766713011953565768977996873028383821265775526344346357123541112589188755147040471997532333827695072210398747506143870850088480631610375575105629402969923
37990669688363242672664077782923842468765734764552501489904789995181590663825882826884437614702086601072912460205046718987111919759741772924082107906614768188
25928778559738389606781437786937703940490529701550198752240522047885413980183556155907460046868073063343753429797282371426595980125377117083031923094847501187
81539765280348857873798901748823243463804440411715478920014959236515352089221657498291013202443206177483221913021678256235579893414436481218449503035888524081
15740619547664383566612706438064220965934420574805698187965300040168887347070624862281490478227347630906482652560940554823360522517964582349779334663133777009
32389603015388462267191386514299244523629967196928070338170422172904291954458588735665717672690912582485100989972418433868929605913487922691271558721181323613
70012052090648575376001600422257728323075912369868818631162455043825487235531941123620115037512323584244014680824673299882453235229917216799756372612183353210
23686195578854195273414555800682053521914530044820400116481027789228810525723047625080568425427769667815771926774952742389468234144023574739674218399717724794
64490794909903308233044977729321377913661153832551845135705059041770761773002335204714530513728518042273513609336859761902134080074259874956489916483283558415
18790233089971640826391421865446416159176533305374315501526687536310287865258225341831409297024404559487986223470061428509852910119956281813557240650373612303
80612421682647619158275971206216911075102110620022250292103973758280786545321293000346117555930108145915118154377991059233962745297863017035414522428825832481
10185696903019214515170137956322244850592956107013529355675348962971488951184208519602258834025155300581173742601529286315229193789791848640470464974617150056
62259748200507982418704000034189585130679648460399502892795419432308411176961777683169393789440445404586765622160449971094877274592261820143359041089932670055
51298147417813688141747600315791793741837036656460760259084286414504016916928653373373749351673138873697355678073568445462500597124999569428768348290694470131
39695758671845345053150692524956190311186650121760981782607840168999098928254294766906595754090551526543788821211073870016226804688310128261364457467501381181
99895452299627549452761944871112397182275542812517516928333071853473496479471177474588594469094399278566301150327219929702835948186371102871019478295556041049
94569790530090215161267032801442076978425884340578290684944705663842203873434946895313693410268205549042641332729214043530781277783400932521461065846050516266
06369309829333522153142406896441472229879201250507247335030253750414919268086142811991181135615102063450273874178955396056033628717223631646597545299321971795
72083803798469762841343018748973041067954703588990548236248494723651739045289521882710298950201026345472967707812057733059495444795253020394607206147343184720
12891382027947113716845979329657679391121796342422193808825527319590678236462421470515380610223708802325911166125101655807805544810736511736112015725788207983
04346277884275334856562425654767711062664505338976464447359505881649914260980555651233453080870411547161166201727321046199497573313805291689786242134493722493
65401328137208190879603847948651527305255792412098085899417962189727321732456666040832323196614881188508752637063922635721727888588861006697067295082662856621
42789555098127098307594733822740958886080483268977873098575694622813361281769384832759774254005221673936338256702877699726892622342424638164907882484814569895
78820841326437902307074825587297298227228464352952356348415245051280414970934534815003363838566586486090914244124413024752619262170944161605020754084937204942
16372326712489077723971624589353187301664632745253191939259669507578605710768387023661969449344902879751178806889386035847044662250758843856626244205371560027
87345086683247957409590801131490445553949472979089821198999392392205812011786324196553409374966862790769096287105922007091926687147181741236537543028535996580
45947455276513698860733871942939615110361783065380776837197220049720205456646520373809780940384932684877526623946791294274354580233914301360048309031821084423
72425264218315757261837807638185861100198403815128178367563386992036547922694358498639362771798041618046013662235878191246173024597377861825966407677879013966
44826035500413324947183241852334621567713018309711089099896390758911615128829646468875441800478905861321031022715201049054765075321382013759982613256651365903
08677310205945594774669134858545209082084155986593443637589125910770359979110742764732875552438147802286623083638506353523937829379144982925749510228726599819
64923226850485187083410037527178126285256712196330538305579411947408618370648081044119314858092966789985880558618838749736465935723939056195292855739824197002
85747166387929856818643218648078765165327952717275940366217333921605018810173490426678520993395921957402807842113084883601771499771585515616148588419257652541
10965702365800067847782401155042150831554731143366545320353097125730540763006480842903826240579636224118544527127667643337427417482571764186299386690494751101
45831487629518887622732479751575475237265720522099995921350373526902203987716.
```

The rank

The rank of E is essentially the number of independent solutions.

- rank $(E) = 0$ means there are finitely many solutions.
- rank $(E) > 0$ means there are infinitely many solutions.
- The curve $E(a)$ with equation

  $$y(y+1) = x(x-1)(x+a)$$

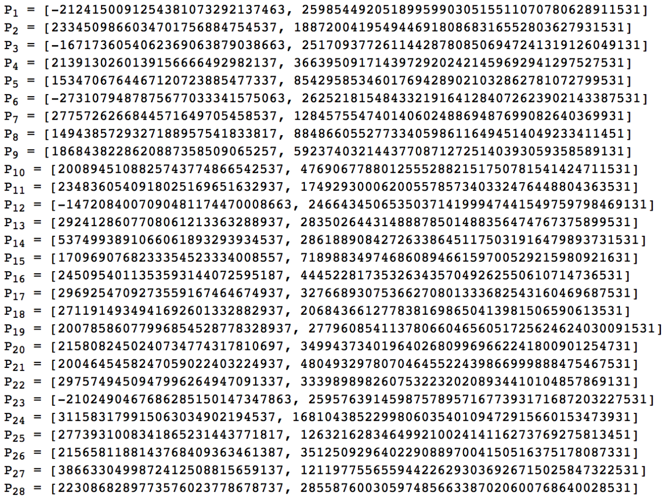
  has rank $= 0, 1, 2, 3, 4$ for $a = 0, 1, 2, 4, 16$.

The rank is finite

Can it be arbitrarily large?

The current record is rank(E) = 28

$$y^2 + xy + y = x^3 - x^2 - 20067762415575526585033208209338542750930230312178956502x + 3448161179503055646703298569039072037485594435931918036126600829629193\ 9448732243429$$



P_1 = [-2124150091254381073292137463, 259854492051899599030515511070780628911531]
P_2 = [2334509866034701756884754537, 18872004195494469180868316552803627931531]
P_3 = [-1671736054062369063879038663, 251709377261144287808506947241319126049131]
P_4 = [2139130260139156666492982137, 36639509171439729202421459692941297527531]
P_5 = [1534706764467120723885477337, 85429585346017694289021032862781072799531]
P_6 = [-2731079487875677033341575063, 262521815484332191641284072623902143387531]
P_7 = [2775726266844571649705458537, 12845755474014060248869487699082640369931]
P_8 = [1494385729327188957541833817, 88486605527733405986116494514049233411451]
P_9 = [1868438228620887358509065257, 59237403214437708712725140393059358589131]
P_{10} = [2008945108825743774866542537, 47690677880125552882151750781541424711531]
P_{11} = [2348360540918025169651632937, 17492930006200557857340332476448804363531]
P_{12} = [-1472084007090481174470008663, 246643450653503714199947441549759798469131]
P_{13} = [2924128607708061213363288937, 28350264431488878501488356474767375899531]
P_{14} = [5374993891066061893293934537, 286188908427263386451175031916479893731531]
P_{15} = [1709690768233354523334008557, 71898834974686089466159700529215980921631]
P_{16} = [2450954011353593144072595187, 4445228173532634357049262550610714736531]
P_{17} = [2969254709273559167464674937, 32766893075366270801333682543160469687531]
P_{18} = [2711914934941692601332882937, 2068436612778381698650413981506590613531]
P_{19} = [20078586077996854528778328937, 2779608541137806604656051725624624030091531]
P_{20} = [2158082450240734774317810697, 34994373401964026809969662241800901254731]
P_{21} = [2004645458247059022403224937, 48049329780704645522439866999888475467531]
P_{22} = [2975749450947996264947091337, 33398989826075322320208934410104857869131]
P_{23} = [-2102490467686285150147347863, 259576391459875789571677393171687203227531]
P_{24} = [311583179915063034902194537, 168104385229980603540109472915660153473931]
P_{25} = [2773931008341865231443771817, 12632162834649921002414116273769275813451]
P_{26} = [2156581188143768409363461387, 35125092964022908897004150516375178087331]
P_{27} = [3866330499872412508815659137, 121197755655944226293036926715025847322531]
P_{28} = [2230868289773576023778678737, 28558760030597485663387020600768640028531]

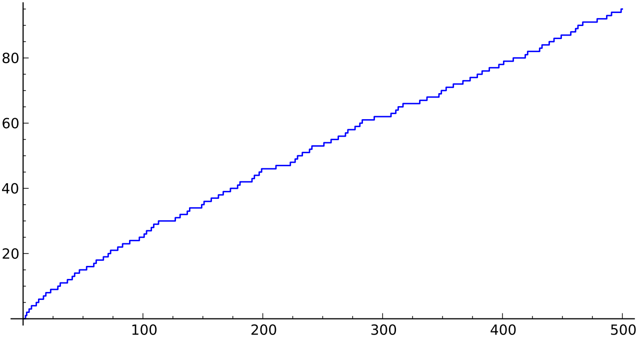
Bryan Birch and Peter Swinnerton-Dyer made a prediction for the rank, based on the average number of solutions at prime numbers *p*.

Primes

A prime p is a number greater than 1 that is not divisible by any smaller number.

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, . . .

There are infinitely many primes. The largest explicit prime known is $2^{43112609} - 1$ with 12,978,189 digits.



The Prime Number $2^{9689} - 1 =$

478220278805461202952839298660005909741497172402236500851334510991837895094266297027892768
611270789458682472098152425631930658505267683408748083442943326479742589324762368833102163
320895484735480579994334130982598901374380618710958104314868081377832153049671560156328262
441404039814320762203627219040859079053720347525610556407157926386787524098557335652265610
854212857732105787905232886503535587361567936365588992571157442015383209175242284304691881
142740066213555930351685370397681268638575037622778794958058208183126172570100349820651232
987267723348951095346937568303703837399969677158578890563911552261340549570718452415821920
822376644205901459333065700972215396237685342377048613857808977562130116781129916640736174
660669780818675796691467124607371290420058840892318638773788767529288695379706698096740605
353012285353903696549022478492464900795489867850331465554647550450168618735486696437455261
412064078294962245202778896213860266593314768769632208950427879162465151931232783175655377
937719452467339581928148666857638401959072017941334958297031939388438881049454604034208753
656362833215207318161430072176937142623851754052084521466531330118355196259184955893849902
534878037671647707393063443684008446825593744345169031599934913766463896897261419901530490
654781905622717122494707073971630095377574344130792050186353223446654564569577433188504497
825014866346737213039209989485214519099823287877248665051301081676990289251871925006694721
570653621624869624056925686555429622155221156042777866254593699880107018616260147647429345
983018365127336346273267588306070141035925482914977433929717368076561095959991130918978823
835013163567266143596921823997719693387439540399662367558052821120713639637085805605116078
177098545257698803233381293927275210194462952749031383555198519709592888523641530178921867
514101454120309619127093436903952209828031766894206132557234964363840305648734929088422378
629288747223121903238528103409182430661894774072726552428489330447486145494207679904173944
716583828167141043583120679050191452732628737033997470720601688256282740427017032260672798
034347932642573009183981307771932245539476396060658821432660315614149074055769805516626304
444758375671151649018119344223685942415184379538933576543212994405485534515585927342456182
514681371472060628778102124092370802149229834963517952727030296297015692768651163505008040
728267425236264469571076976886613730278931360967438271901738550848466337347612084356798306
505955807293511063754424080735066708298723377976887493898358452309563899612061631863439196
711208646438464947096323007272920091258614726799976249670985276950353573392441620265772074
124868359220282898331114083392330243391779797699031142584361935093675448381119440881276338
808420445180491245438388418080094527562666805762895476338464130510775377324708249580453335
571748196502507081973046642282610569751056428979895118219288597635222905389894873761464213
9910911535864505818992696826225754111

Primality testing

Determining that $n > 1$ is a prime can be done quickly.

”PRIMES is in P”

AKS: Manindra Agrawal, Neeraj Kayal, and Nitin Saxena (2002)

If n fails the primality test, it is more difficult to factor it.

1230186684530117755130494958384962720772853569595334
7921973224521517264005072636575187452021997864693899
5647494277406384592519255732630345373154826850791702
6122142913461670429214311602221240479274737794080665
351419597459856902143413 $=$ RSA-768 $=$

3347807169895689878604416984821269081770479498371376
856891243138898288379387800228761471165253174308773
7814467999489

$\times$

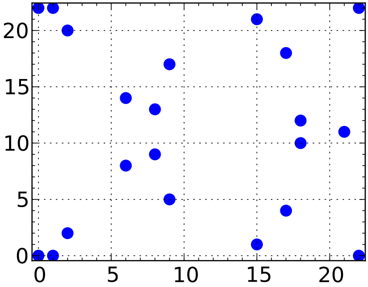
3674604366679959042824463379962795263227915816434308
764267603228381573966651127923337341714339681027009
2798736308917

What do we mean by a solution of the cubic equation at the prime number p?

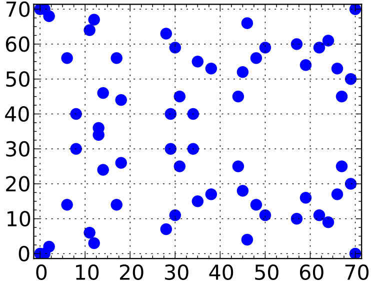
$$y^2 + y = x^3 - x$$

$(x, y) \equiv (3, 1)$ is a solution at $p = 11$

There are finitely many solutions $A(p)$ at each prime p.



$p = 23, \quad A(23) = 22$



$p = 71, \quad A(71) = 63$

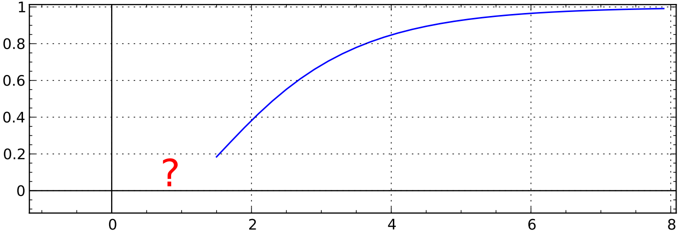
It is common to write

$$A(p) = p + 1 - a(p)$$

We define the *L*-function of *E* by the infinite product

$$L(E, s) = \prod_p (1 - a(p)p^{-s} + p^{1-2s})^{-1} = \sum a(n)n^{-s}$$

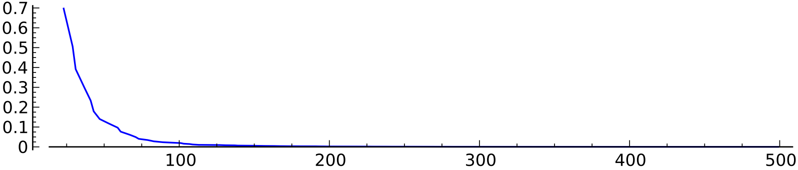
This definition only works in the region $s > 3/2$, where the infinite product converges.



If we formally set $s = 1$ in the product, we get

$$\prod_p (1 - a(p)p^{-1} + p^{-1})^{-1} = \prod_p p/A(p)$$

If $A(p)$ is large on average compared with p, this will approach 0. The larger $A(p)$ is on average, the faster it will tend to 0.

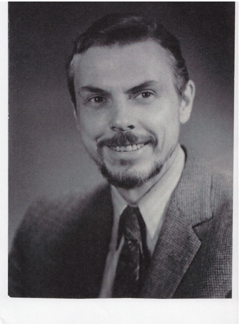


The conjecture of Birch and Swinnerton-Dyer

1. The function $L(E, s)$ has a natural (analytic) continuation to a neighborhood of $s = 1$.
2. The order of vanishing of $L(E, s)$ at $s = 1$ is equal to the rank of E.
3. The leading term in the Taylor expansion of $L(E, s)$ at $s = 1$ is given by certain arithmetic invariants of E.

$$L(E, s) = c(E)(s - 1)^{\mathrm{rank}(E)} + \dots$$

The most mysterious arithmetic invariant was studied by John Tate and Igor Shafarevich, who conjectured that it is finite. Tate called this invariant Ш.



The Birch and Swinnerton-Dyer Conjecture

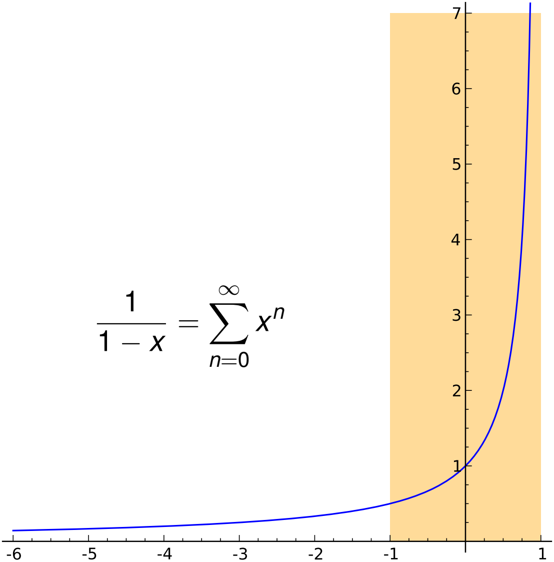
$$L(E,s) = c(E)(s-1)^{\mathrm{rank}(E)} + \cdots$$

$$c(E) = \frac{\Omega_E \cdot \mathrm{Reg}_E \cdot \#\text{Ш}_E \cdot \prod c_p}{\#E(\mathbb{Q})_{\mathrm{tor}}^2}$$

Each quantity on the right measures the size of an abelian group attached to E.

Natural (analytic) continuation

The infinite sum $\sum_{n=0}^{\infty} x^n$ converges when $-1 < x < 1$.



$$\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n$$

The natural (analytic) continuation of $L(E, s) = \sum a(n)n^{-s}$ was obtained by Andrew Wiles and Richard Taylor (1995). They proved that the function defined by the infinite series

$$F(\tau) = \sum a(n)e^{2\pi in\tau}$$

is a modular form.

Combining a limit formula I proved with Don Zagier (1983) with work of Victor Kolyvagin (1986) we can now show the following.

If $L(E, 1) \neq 0$ the rank is zero, so there are finitely many solutions.

If $L(E, 1) = 0$ and $L'(E, 1) \neq 0$ the rank is one, so there are infinitely many solutions.

In both cases, we can also show that $Ш$ is finite.

When the order of $L(E, s)$ at $s = 1$ is greater than one we cannot prove anything in general…

But the computer has been a great guide.

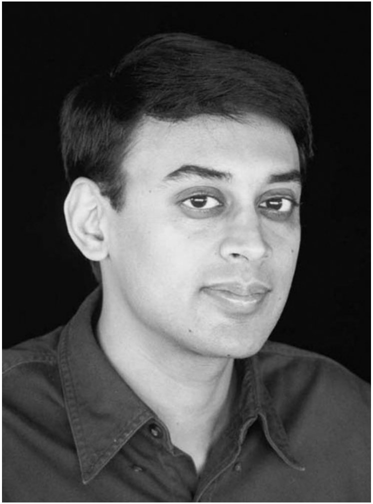
Here is a summary of the evidence for the simplest rank 2 curve

$$y(y+1) = x(x-1)(x+2)$$

- the order of vanishing is equal to 2
- most primes up to 50,000 do not divide the order of Ш

The average rank

Manjul Bhargava has recently made progress on the study of the average rank, for ALL cubic curves with rational coefficients.



Enumerating the curves

- Every such curve has a unique equation of the form $y^2 = x^3 + Ax + B$ where A and B are integers (not divisible by p^4 and p^6, for any prime p).

- Define the height $H(E)$ as the maximum of the positive integers $|A|^3$ and $|B|^2$.

- For any positive real number X, there are only finitely many curves with $H(E) \leq X$.

- Call this number $N(X)$. It grows at the same rate as $(X)^{1/2}(X)^{1/3} = X^{5/6}$.

- Define the average rank by the limit as $X \to \infty$ of

$$\frac{1}{N(X)} \sum_{H(E) \leq X} rank(E)$$

- We suspect that this limit exists, and is equal to $1/2$.
- In fact, we think that on average half the curves have rank zero and half have rank one.
- Bhargava and Shankar have shown why there is an upper bound on the limit, and have obtained a specific upper bound which is less than 1.

Thank you

JMM -- Solving Cubic Equations

Solving Cubic Equations (JMM 2012 Short Course)

William Stein

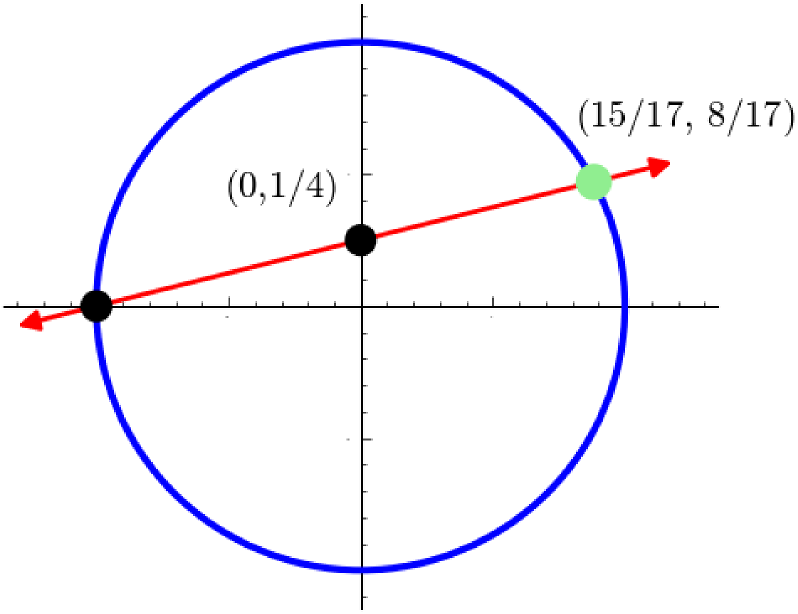
This worksheet accompanies these slides.

Pythagorean triples

```
@interact
def _(t=(1/4,(1/16,1/8,..,1))):
    t0 = t
    x,y,t=var('x,y,t')
    show([x==(1-t^2)/(1+t^2), y==2*t/(1+t^2)])
    t = t0
    (x,y) = ((1-t^2)/(1+t^2), 2*t/(1+t^2))
    a = 1/3
    G = circle((0,0), 1, color='blue', thickness=3)
    G += arrow((-1-a,-t*a), (x+a,y+t*a), head=2, color='red')
    G += point((0,t), pointsize=150, color='black', zorder=100)
    G += point((-1,0), pointsize=150, color='black', zorder=100)
    G += point((x,y), pointsize=190, color='lightgreen', zorder=100)
    G += text("$(0, %s)$"%t, (-.3, t+.2), fontsize=16, color='black')
    G += text(r"$(%s,\,%s)$"%(x,y), (x+.35, y+.25), fontsize=16, color='black')
    G.show(aspect_ratio=1, ymin=-1.1, ymax=1.1, xmax=1.3, xmin=-1.3, fontsize=0,
figsize=6)
```

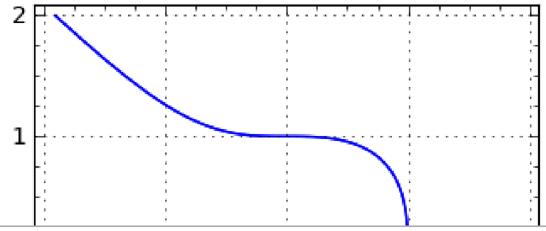
t 1/4

$$\left[x = -\frac{t^{2} - 1}{t^{2} + 1}, y = \frac{2\,t}{t^{2} + 1}\right]$$

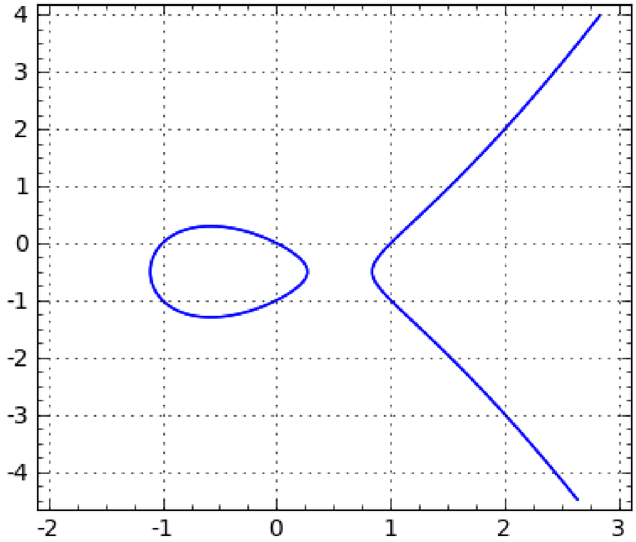


Cubic equations

```
var('x,y')
implicit_plot(x^3 + y^3 == 1, (x,-2,2), (y,-2,2), aspect_ratio=1, figsize=5,
gridlines=True)
```



```
var('x,y')
implicit_plot(y^2 + y == x^3 - x, (x,-2,3), (y,-4.5,4), aspect_ratio=1/2,
figsize=5, gridlines=True)
```



```
@interact
def _(equation='x^3 + y^3 == 1',
      xmin=-2, xmax=2, ymin=-2, ymax=2, aspect_ratio=1, gridlines=True):
    x,y = var('x,y')
    eqn = sage_eval(equation, {'x':x, 'y':y})
    print eqn
    G = implicit_plot(eqn, (x,xmin,xmax), (y,ymin,ymax), aspect_ratio=aspect_ratio,
gridlines=gridlines)
    G.show(figsize=4)
```

equation x^3 + y^3 == 1

xmin -2

xmax 2

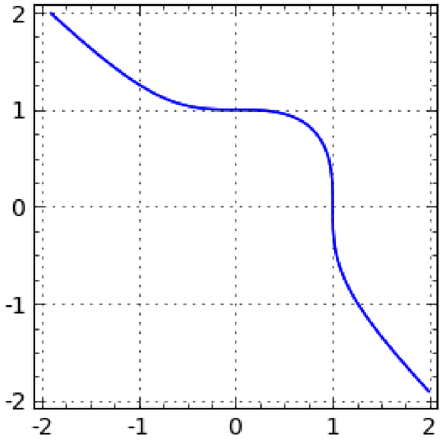
ymin -2

ymax 2

aspect_ratio 1

gridlines ☑

`x^3 + y^3 == 1`

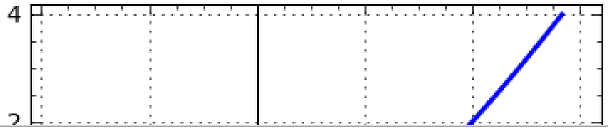


The Group Law

How the figure in the slide was made:

```
E = EllipticCurve([0,0,1,-1,0]); print E
G = E.plot(plot_points=600, thickness=2)
G += arrow((-2,1), (3,-4), head=2, color='red', width=2)
G += points([(-1,0), (0,-1), (2,-3)], color='black', pointsize=70, zorder=50)
G += text("$(2,-3)$", (1.3,-3.1), fontsize=18, color='black')
G += text("$(-1,0)$", (-.9,1), fontsize=18, color='black')
G += text("$(0,-1)$", (-.7,-1.85), fontsize=18, color='black')
G.show(gridlines=True, frame=True, aspect_ratio=1/2, xmax=3.1, xmin=-2, figsize=5)
```

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



```
E = EllipticCurve([0,0,1,-1,0])
P = E([-1,0]); Q = E([0,-1]); R = E([2,-3])
print P + Q + R
print -(P+Q)
print 7*P
```

```
    (0 : 1 : 0)
    (2 : -3 : 1)
    (1849037896/6941055969 : -260151768440137/578280195945297 : 1)
```

```
for n in range(10):
    print n, n*P
```

```
    0 (0 : 1 : 0)
    1 (-1 : 0 : 1)
    2 (6 : -15 : 1)
    3 (-20/49 : 92/343 : 1)
    4 (1357/841 : -53277/24389 : 1)
    5 (8385/98596 : -2882165/30959144 : 1)
    6 (12551561/13608721 : -41922509264/50202571769 : 1)
    7 (1849037896/6941055969 : -260151768440137/578280195945297 : 1)
    8 (4881674119706/5677664356225 :
    -4590618167456560854/13528653463047586625 : 1)
    9 (2786836257692691/16063784753682169 :
    -1600059682932627475385835/2035972062206737347698803 : 1)
```

The Rank

```
E = EllipticCurve([0,0,1,-1,0])
E.rank()
```

```
    1
```

```
E.rank?
```

File: /home/wstein/sage/sage-4.8.alpha5/local/lib/python2.6/site-packages/sage/schemes/elliptic_curves/ell_rational_field.py

Type: <type ‘instancemethod’>

Definition: E.rank(use_database=False, verbose=False, only_use_mwrank=True, algorithm=’mwrank_lib’, proof=None)

Docstring:

Return the rank of this elliptic curve, assuming no conjectures.

If we fail to provably compute the rank, raises a RuntimeError exception.

INPUT:

- `use_database (bool)` - (default: False), if True, try to look up the regulator in the Cremona database.
- `verbose` - (default: None), if specified changes the verbosity of mwrank computations. algorithm -
- `- 'mwrank_shell'` - call mwrank shell command
- `- 'mwrank_lib'` - call mwrank c library
- `only_use_mwrank` - (default: True) if False try using analytic rank methods first.
- `proof` - bool or None (default: None, see proof.elliptic_curve or sage.structure.proof). Note that results obtained from databa considered proof = True

OUTPUT:

- `rank (int)` - the rank of the elliptic curve.

IMPLEMENTATION: Uses L-functions, mwrank, and databases.

EXAMPLES:

```
sage: EllipticCurve('11a').rank()
0
sage: EllipticCurve('37a').rank()
1
sage: EllipticCurve('389a').rank()
2
sage: EllipticCurve('5077a').rank()
3
sage: EllipticCurve([1, -1, 0, -79, 289]).rank()   # This will use the default proof behavi
4
sage: EllipticCurve([0, 0, 1, -79, 342]).rank(proof=False)
5
sage: EllipticCurve([0, 0, 1, -79, 342]).simon_two_descent()[0]
5
```

Examples with denominators in defining equations:

```
sage: E = EllipticCurve([0, 0, 0, 0, -675/4])
sage: E.rank()
0
sage: E = EllipticCurve([0, 0, 1/2, 0, -1/5])
sage: E.rank()
1
sage: E.minimal_model().rank()
1
```

A large example where mwrank doesn’t determine the result with certainty:

```
sage: EllipticCurve([1,0,0,0,37455]).rank(proof=False)
0
sage: EllipticCurve([1,0,0,0,37455]).rank(proof=True)
Traceback (click to the left of this block for traceback)
...
```

Try a *random curve (if you try a different one it could take a long time -- press "escape" with the cursor in the box to interrupt)*:

```
E = EllipticCurve([2012,3])
print E.rank()
print E.gens()
```

```
    1
    [(7753/19044 : 75356155/2628072 : 1)]
```

A family

```
def F(a):
    return EllipticCurve([0,(a-1),1,-a,0])

for a in [0..20]:
    print a, F(a).rank()
```

```
    0 0
    1 1
    2 2
    3 2
    4 3
    5 2
    6 2
    7 3
```

```
8 3
9 3
10 2
11 3
12 3
13 3
14 3
15 2
16 4
17 3
18 2
19 3
20 3
```

Exercise: Find the first a such that $F(a)$ has rank 5. Rank 6.

Elkies Curve of Rank (at least) 28

```
E = EllipticCurve([1,-1,1,-
20067762415575526585033208209338542750930230312178956502,
34481611795030556467032985690390720374855944359319180361266008296291939448732243429])
```

That the first few good $a_p = p + 1 - \#E(F_p)$ are negative is evidence that E has high rank:

```
D = E.discriminant(); [p for p in primes(1000) if D%p==0]
```

```
[2, 3, 5, 7, 11, 13, 17, 19]
```

```
for p in primes(20,200):
    print E.ap(p),
```

```
-9 -10 -8 -11 -10 -12 -12 -9 -12 -15 -16 -16 -15 -13 -18 -16 -13 -6
-20 -12 -20 -19 -11 -16 -10 -22 -17 -9 -24 -12 -23 -22 -7 -10 -7 -22
-22 -25
```

Exercise: What is the smallest good prime p such that $a_p > 0$?

```
P = [E([-2124150091254381073292137463,
259854492051899599030515511070780628911531]),
     E([2334509866034701756884754537, 18872004195494469180868316552803627931531]),
     E([-1671736054062369063879038663,
251709377261144287808506947241319126049131]),
     E([2139130260139156666492982137, 36639509171439729202421459692941297527531]),
     E([1534706764467120723885477337, 85429585346017694289021032862781072799531]),
     E([-2731079487875677033341575063,
262521815484332191641284072623902143387531]),
     E([2775726266844571649705458537, 12845755474014060248869487699082640369931]),
     E([1494385729327188957541833817, 88486605527733405986116494514049233411451]),
     E([1868438228620887358509065257, 59237403214437708712725140393059358589131]),
     E([2008945108825743774866542537, 47690677880125552882151750781541424711531]),
     E([2348360540918025169651632937, 17492930006200557857340332476448804363531]),
     E([-1472084007090481174470008663,
246643450653503714199947441549759798469131]),
     E([2924128607708061213363288937, 28350264431488878501488356474767375899531]),
     E([5374993891066061893293934537, 28618890842726338645117503191647989373153 1]),
     E([1709690768233354523334008557, 7189883497468608946615970052921598092163 1]),
     E([2450954011353593144072595187, 444522817353263435704926255061071473653 1]),
     E([2969254709273559167464674937, 3276689307536627080133368254316046968753 1]),
     E([2711914934941692601332882937, 206843661277838169865041398150659061353 1]),
     E([2007858607799685452877832893 7,
27796085411378066046560517256246240300915 31]),
     E([2158082450240734774317810697, 34994373401964026809969662241800901254731]),
     E([2004645458247059022403224937, 48049329780704645522439866999888475467531]),
     E([2975749450947996264947091337, 33398989826075322320208934410104857869131]),
     E([-2102490467686285150147347863,
25957639145987578957167739317168720322 7531]),
     E([3115831799150630349021945 37, 168104385229980603540109472915660153473931]),
     E([2773931008341865231443771817, 12632162834649921002414116273769275813451]),
     E([2156581188143768409363461387, 35125092964022908897004150516375178087331]),
     E([3866330499872412508815659137, 121197755655944226293036926715025847322531]),
     E([2230868289773576023778678737, 28558760030597485663387020600768640028531])]
```

```
P[0] + P[1]
```

```
(310801760282037317127091226854726337713781455351 8653/11465117276447\
98490358769 :
180255809065592657084558925414149675357657278492965377853843866 12174\
56501493/122763073305337604770264342023541 0103 : 1)
```

```
time E.regulator_of_points(P[:7])
```

```
3.04313979267944e11
Time: CPU 3.18 s, Wall: 3.18 s
```

```
time E.regulator_of_points(P[:15])
```

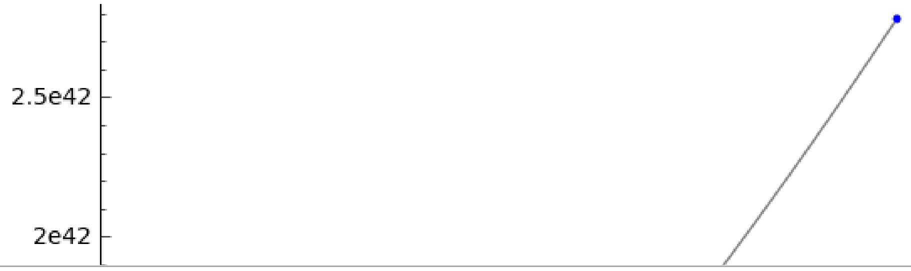
```
1.97964758730350e23
Time: CPU 15.72 s, Wall: 15.71 s
```

The following takes about 60 seconds (on my laptop), and shows that the 28 points are independent:

```
time E.regulator_of_points(P)
```

```
Traceback (click to the left of this block for traceback)
...
__SAGE__
```

```
points([(x,y) for x,y,_ in P]) + plot(E, color='grey', xmax=2e28, ymin=-50)
```



Primes

```
prime_range(50)
```

```
    [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

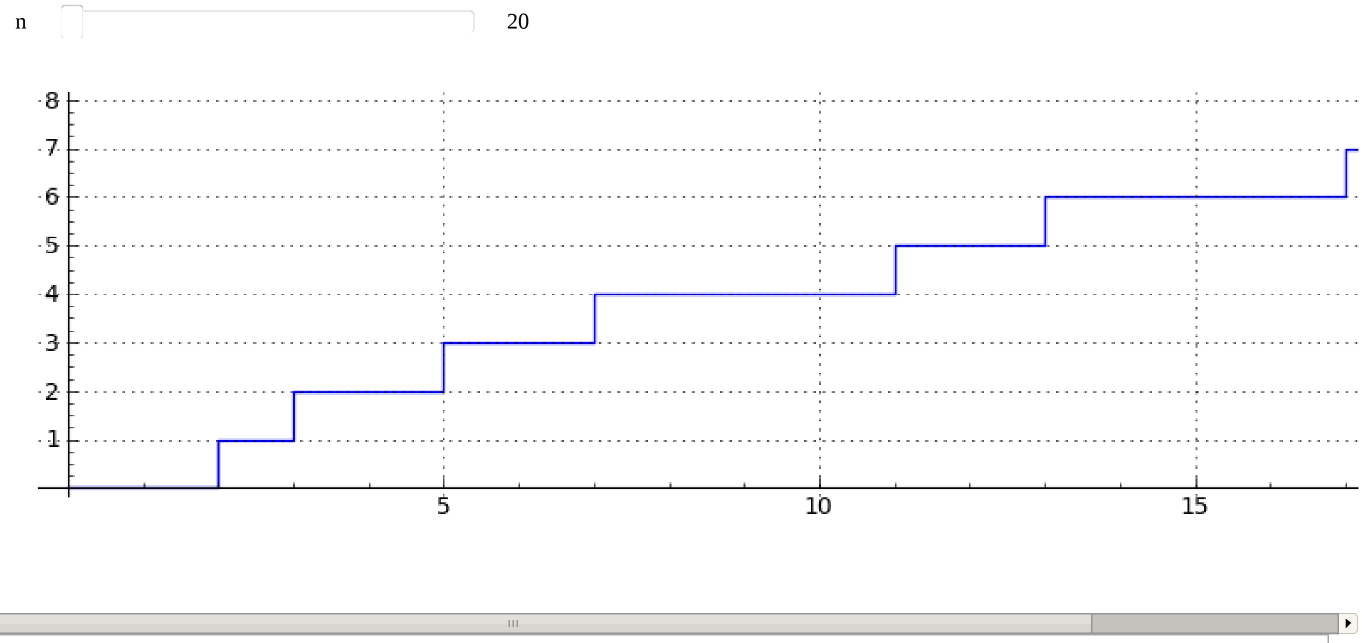
```
primes(50)
```

```
    <generator object primes at 0x5c53d20>
```

```
for p in primes(50):
    print p,
```

```
    2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
```

```
@interact
def _(n=(20,30,..,2000)):
    prime_pi.plot(0, n).show(figsize=[12,3],gridlines=True)
```



```
time p = 2^43112609 - 1
```

```
    Time: CPU 0.00 s, Wall: 0.01 s
```

It takes a while to compute the string representation of p.

```
time s_bigp = str(2^43112609 - 1)
len(s_bigp)
```

```
    Time: CPU 14.81 s, Wall: 15.08 s
    Time: CPU 14.81 s, Wall: 15.02 s
    12978189
```

```
@interact
def _(digits = (5,20,..,10000)):
    print "Showing %.5f percent of the digits"%(100*2.0*digits/len(s_bigp))
    print "p = " + s_bigp[:digits] + ' ... ' + s_bigp[-digits:]
```

digits 5

```
Showing 0.00008 percent of the digits
p = 31647 ... 52511
```

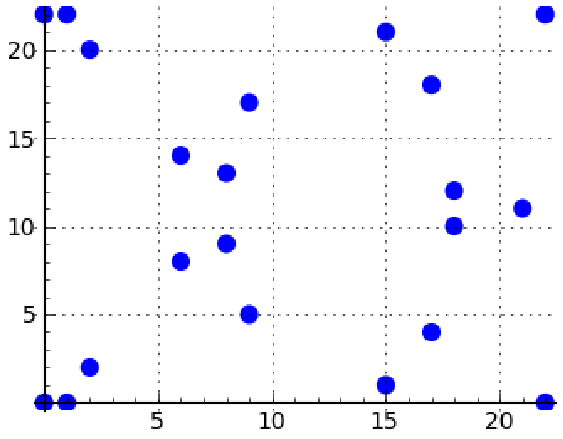
```
E = EllipticCurve([0,0,1,-1,0]); E
```

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

```
E23 = E.change_ring(GF(23)); E23
```

```
    Elliptic Curve defined by y^2 + y = x^3 + 22*x over Finite Field of
    size 23
```

```
E23.plot(pointsize=50, figsize=4, gridlines=True)
```



Exercise: Make an interact that has a slider letting you select a prime, which plots the graph of E modulo that prime.

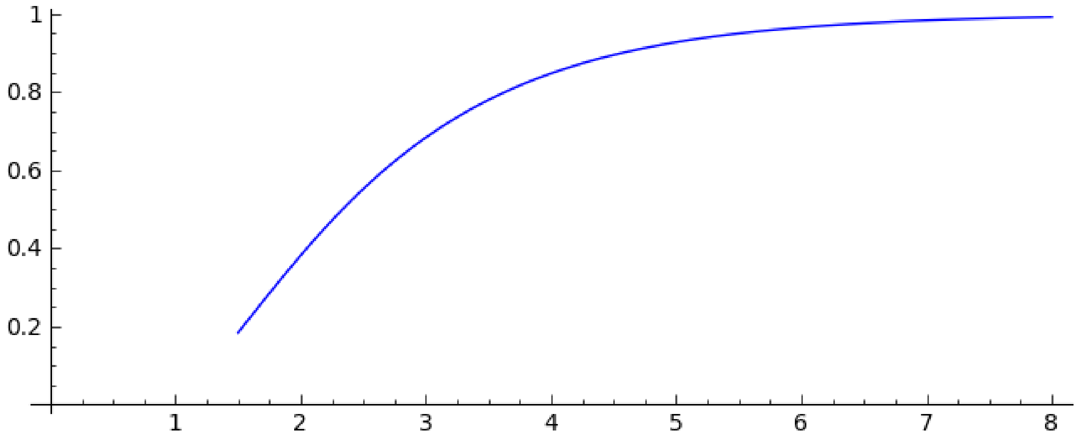
The *L*-Series

```
E = EllipticCurve([0,0,1,-1,0])
L = E.lseries(); L
```

```
L = E.lseries(); L
```

```
    Complex L-series of the Elliptic Curve defined by y^2 + y = x^3 - x
    over Rational Field
```

```
show(line([(x,L(x)) for x in [1.5,1.6, .., 8]]), figsize=[8,3], xmin=0, ymin=0)
```



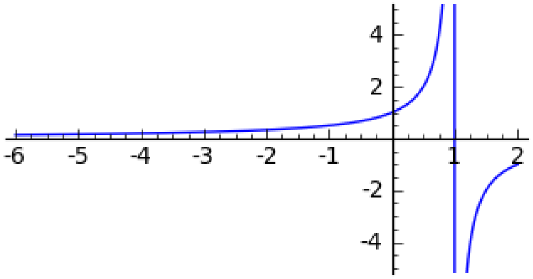
```
@interact
def _(E = ['y^2 + y = x^3 - x^2', 'y^2 + y = x^3 - x',
            'a rank 4 curve', 'elkies rank>=28 curve', '2012'],
      B = (30..1000)):
    if E == 'y^2 + y = x^3 - x^2':
        E = EllipticCurve([0,-1,1,0,0])
        r = E.rank()
    elif E == 'y^2 + y = x^3 - x':
        E = EllipticCurve([0,0,1,-1,0])
        r = E.rank()
    elif E == 'a rank 4 curve':
        E = EllipticCurve([1, -1, 0, -79, 289])
        r = 4
    elif E == 'elkies rank>=28 curve':
        E = EllipticCurve([1,-1,1,
    -20067762415575526585033208209338542750930230312178956502,

34481611795030556467032985690390720374855944359319180361266008296291939448732243429])

        r = ">=28"
    elif E == '2012':
        E = EllipticCurve([0,2012])
        r = "?"

    L_approx = 1
    print '%4s%6s%5s%9s%20s'%('p', 'A(p)', 'p/Ap', '  prod p/Ap', 'Rank = %s'%r)
    v = []
    t = ''
    for p in primes(B):
        if E.discriminant()%p:
            Ap = p+1-E.ap(p)
            L_approx *= float(p/Ap)
            t += '%4s%4s%8.3f%8.3f\n'%(p, Ap, float(p/Ap), L_approx)
            v.append((p, L_approx))
    (line(v) + points(v,color='black')).show(figsize=[8,2])

    print t
    print t
```

E [y^2 + y = x^3 - x^2] [y^2 + y = x^3 - x] [a rank 4 curve] [elkies rank>=28 curve] [2012]

B 30

Natural (analytic) continuation

```
var('x')
f = 1/(1-x)
plot(f, -6, 2, figsize=[4,2], ymax=5, ymin=-5)
```



```
f.taylor(x,0,5)
```

```
    x^5 + x^4 + x^3 + x^2 + x + 1
```

The Birch and Swinnerton-Dyer Conjecture:

A Rank 1 Curve

```
E = EllipticCurve([0,0,1,-1,0])
L = E.lseries()
Lser = L.taylor_series(); Lser
```

```
    0.305999773834052*z + 0.186547797268162*z^2 - 0.136791463097188*z^3
    + 0.0161066468496401*z^4 + 0.0185955175398802*z^5 + O(z^6)
```

```
c = Lser[1]; c
```

```
    0.305999773834052
```

```
Omega_E = E.period_lattice().omega(); Omega_E
```

```
    5.98691729246392
```

```
Reg_E = E.regulator(); Reg_E
```

```
    0.0511114082399688
```

```
# this *uses* the formula; but we do know in this case that Sha_E 1
# this ^uses^ the formula; but we do know in this case that Sha_E=1.
Sha_E = E.sha().an(); Sha_E
```

```
    1
```

```
prod_cp = E.tamagawa_product_bsd(); prod_cp
```

```
    1
```

```
T = E.torsion_order()^2; T
```

```
    1
```

```
Omega_E * Reg_E * Sha_E * prod_cp / T^2
```

```
    0.305999773834052
```

A Rank 2 Curve

```
E = EllipticCurve([0,1,1,-2,0])
L = E.lseries()
Lser = L.taylor_series(); Lser
```

```
    -2.69129566562797e-23 + (1.52514901968783e-23)*z +
    0.759316500288427*z^2 - 0.430302337583362*z^3 -
    0.193509313829981*z^4 + 0.459971558373642*z^5 + O(z^6)
```

```
E.rank()
```

```
    2
```

If you solve for the order of the Shafarevich-Tate group in the conjecture:

```
E.sha().an()
```

```
    1.00000000000000
```

```
S = E.sha(); S
```

```
    Tate-Shafarevich group for the Elliptic Curve defined by y^2 + y =
    x^3 + x^2 - 2*x over Rational Field
```

The following proves that $p = 5, 7$ do not divide the order of this group:

```
S.p_primary_bound(5)
```

```
    0
```

```
S.p_primary_bound(7)
```

```
    0
```

Open Problem: Prove that the Shafarevich-Tate group of E is *finite*.

A Rank 4 Curve

```
E = EllipticCurve([0,15,1,-16,0]); E
```

```
E = EllipticCurve([0,15,1,-16,0]); E
```

```
    Elliptic Curve defined by y^2 + y = x^3 + 15*x^2 - 16*x over
    Rational Field
```

```
E.rank()
```

```
    4
```

```
E.gens()
```

```
    [(-15 : 15 : 1), (-14 : 20 : 1), (-51/4 : 187/8 : 1), (22 : 132 :
    1)]
```

```
L = E.lseries()
Lser = L.taylor_series(); Lser
```

```
    4.32638791417839e-24 + (-1.96674959799307e-23)*z +
    (2.05660099586894e-22)*z^2 + (-7.97704812013524e-22)*z^3 +
    10.8463853245874*z^4 - 49.3070071384507*z^5 + O(z^6)
```

Open Problem: *Prove* that $L(E, s)$ vanishes to order 4 at $s = 1$.

Elliptic Curves over the Rational Numbers

Elliptic curves over $\mathbb{Q}$: Creation

```
Given a vector $[a_1,a_2,a_3,a_4,a_6]$ of rationals, it is easy
to create the elliptic curve with Weierstrass equation
$$y^2+a_1xy+a_3y=x^3+a_2x^2+a_4x+a_6.$$
```

Given a vector $[a_1, a_2, a_3, a_4, a_6]$ of rationals, it is easy to create the ellipt equation

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6.$$

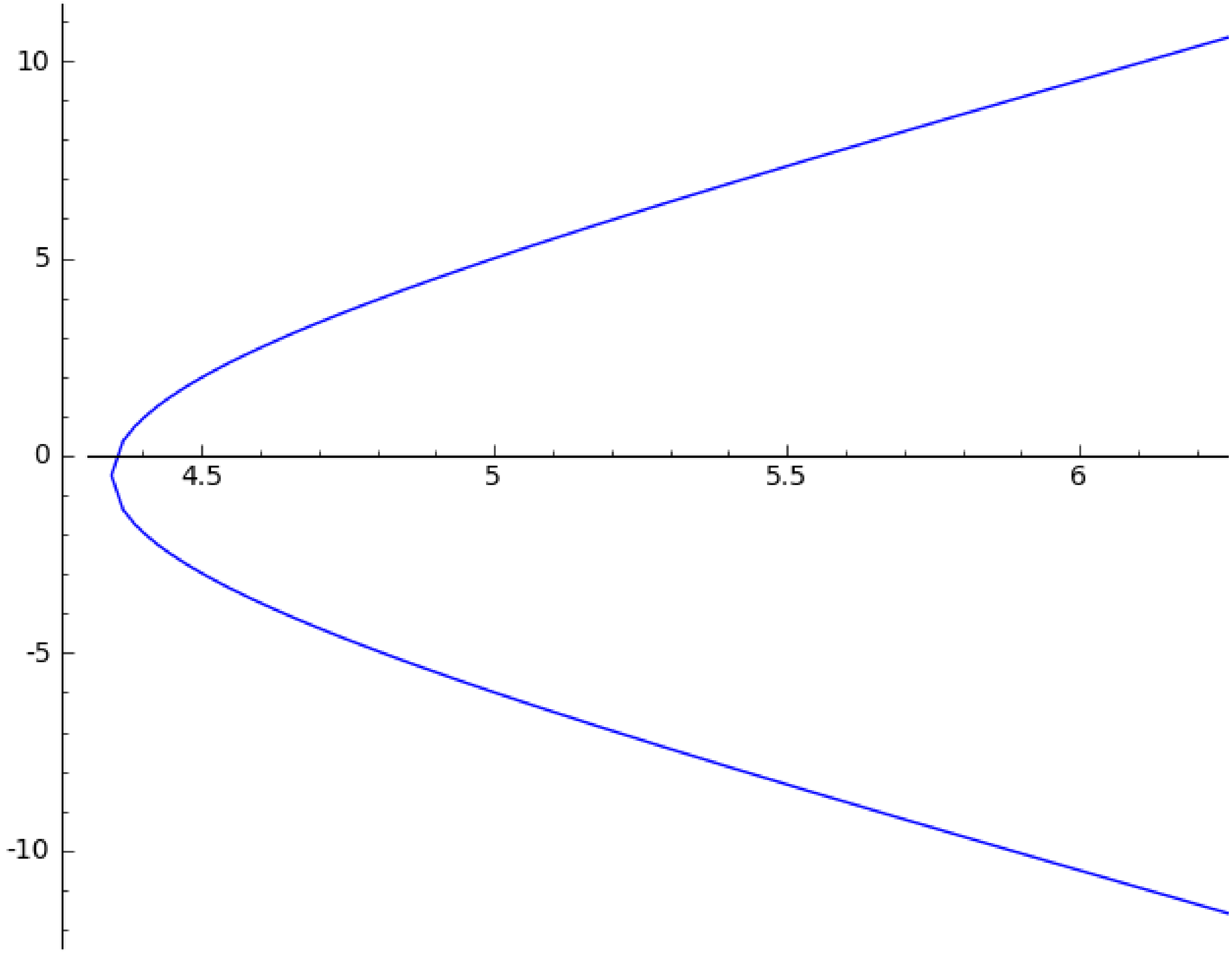
```
E=EllipticCurve([0,-1,1,-10,-20]);
E
```

$y^2 + y = x^3 - x^2 - 10x - 20$

```
#Basic Invariants
print E.discriminant().factor();
print E.conductor();
print E.j_invariant().factor()
```

```
-1 * 11^5
11
-1 * 2^12 * 11^-5 * 31^3
```

```
plot(E)
```



```
E;  EllipticCurve('11a')
```

$$y^2 + y = x^3 - x^2 - 10x - 20$$
$$y^2 + y = x^3 - x^2 - 10x - 20$$

```
EllipticCurve('11a'); EllipticCurve('101a');
EllipticCurve('1001a');
```

$$y^2 + y = x^3 - x^2 - 10x - 20$$
$$y^2 + y = x^3 + x^2 - x - 1$$
$$y^2 + y = x^3 - x^2 - 15881x + 778423$$

```
E.an(103); #This is the coefficient of 103^{-s} in the L-series
of E.
```

-16

```
E103=E.base_extend(GF(103))
E103.order()
```

120

```
103+1-120
```

-16

```
E.anlist(20)
```

$[0, 1, -2, -1, 2, 1, 2, -2, 0, -2, -2, 1, -2, 4, 4, -1, -4, -2, 4, 0, 2]$

```
E.modular_form()
```

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

```
#The Mordell-Weil group
```

```
The most interesting object attached to $E$ is the Mordell-Weil
group $E(\mathbf{Q})$.  This has a torsion part
$E(\mathbf{Q})_{\text{tors}}$ and a free part $\mathbf{Z}^r$.
First we'll discuss $E(\mathbf{Q})_{\text{tors}}$.  Computing
$E(\mathbf{Q})_{\text{tors}}$ is generally done quickly using
the Nagell-Lutz theorem.
```

```
Traceback (click to the left of this block for traceback)
...
SyntaxError: invalid syntax
```

```
E.torsion_subgroup()
```

$\frac{\mathbf{Z}}{5\mathbf{Z}}$

```
E.torsion_points()
```

$[(0 : 1 : 0), (5 : -6 : 1), (5 : 5 : 1), (16 : -61 : 1), (16 : 60 : 1)]$

```
#To list elliptic curves with given torsion structure:
for Ell in cremona_curves(range(1,1000)):
    G = Ell.torsion_subgroup()
    if G.order() == 7:
        print Ell,G.0
```

```
Elliptic Curve defined by y^2 + x*y + y = x^3 - x^2 - 3*x + 3 o
Rational Field (1 : 0 : 1)
Elliptic Curve defined by y^2 + x*y = x^3 - x + 137 over Ration
Field (2 : 11 : 1)
Elliptic Curve defined by y^2 + x*y = x^3 + 159*x + 1737 over
Rational Field (6 : 51 : 1)
Elliptic Curve defined by y^2 + x*y = x^3 - 141*x + 657 over
Rational Field (6 : 3 : 1)
Elliptic Curve defined by y^2 + x*y + y = x^3 - x^2 + 918*x + 5
over Rational Field (-3 : 51 : 1)
Elliptic Curve defined by y^2 + x*y = x^3 + 714*x - 82908 over
Rational Field (42 : 126 : 1)
Elliptic Curve defined by y^2 + x*y + y = x^3 - x^2 - 19353*x +
958713 over Rational Field (103 : 172 : 1)
Elliptic Curve defined by y^2 + x*y = x^3 - 1661*x + 26097 over
```

```
Rational Field (-14 : 223 : 1)
Elliptic Curve defined by y^2 + x*y = x^3 - 101946*x + 12401892
Rational Field (204 : 222 : 1)
Elliptic Curve defined by y^2 + x*y = x^3 - 5774401*x + 5346023
over Rational Field (-436 : 88427 : 1)
Elliptic Curve defined by y^2 + x*y = x^3 + 2305*x - 15975 over
Rational Field (10 : 85 : 1)
Traceback (click to the left of this block for traceback)
...
__SAGE__
```

```
Sage has a built-in function to compute $E(\mathbb{Q})$:
```

Sage has a built-in function to compute $E(\mathbb{Q})$:

```
E.rank()
```

0

```
E37=EllipticCurve('37a');  E37
```

$y^2 + y = x^3 - x$

```
E37.rank()
```

1

```
E37.gens()
```

$[(0 : 0 : 1)]$

```
E389=EllipticCurve('389a'); E389
```

$y^2 + y = x^3 + x^2 - 2x$

```
E389.gens()
```

$[(-1 : 1 : 1), (0 : 0 : 1)]$

```
E5077=EllipticCurve('5077a'); E5077
```

$y^2 + y = x^3 - 7x + 6$

```
E5077.gens()
```

$[(-2 : 3 : 1), (-1 : 3 : 1), (0 : 2 : 1)]$

```
E389
```

$y^2 + y = x^3 + x^2 - 2x$

```
E389.integral_points()
```

$[(-2 : 0 : 1), (-1 : 1 : 1), (0 : 0 : 1), (1 : 0 : 1), (3 : 5 : 1), (4 : 8 : 1), (6 : 15 : 1), (3$

```
E389.S_integral_points([2])
```

```
E389
```

$$y^2 + y = x^3 + x^2 - 2x$$

```
#Elliptic Curves with Complex Multiplication
```

```
E=EllipticCurve([0,0,0,1,0]); E
```

$$y^2 = x^3 + x$$

```
E.has_cm()
```

True

```
E.cm_discriminant() #E has CM by Z[i]
```

$$-4$$

```
f=E.division_polynomial(3); f
```

$$3x^4 + 6x^2 - 1$$

```
#The roots of this polynomial are the xcoordinates of all 2-
torsion points of E.
```

```
g=3^3*f(x/3); g
```

$$(x^2 + 18)x^2 - 27$$

```
K=NumberField(x^2+1,'i'); K
```

$$\mathbf{Q}[i]/(i^2 + 1)$$

```
L=K.extension(g,'z')
```

```
L
```

$$(\mathbf{Q}[i]/(i^2 + 1))[z]/(z^4 + 18z^2 - 27)$$

```
L.base_field()
```

$$\mathbf{Q}[i]/(i^2 + 1)$$

```
L.relative_discriminant().factor()
```

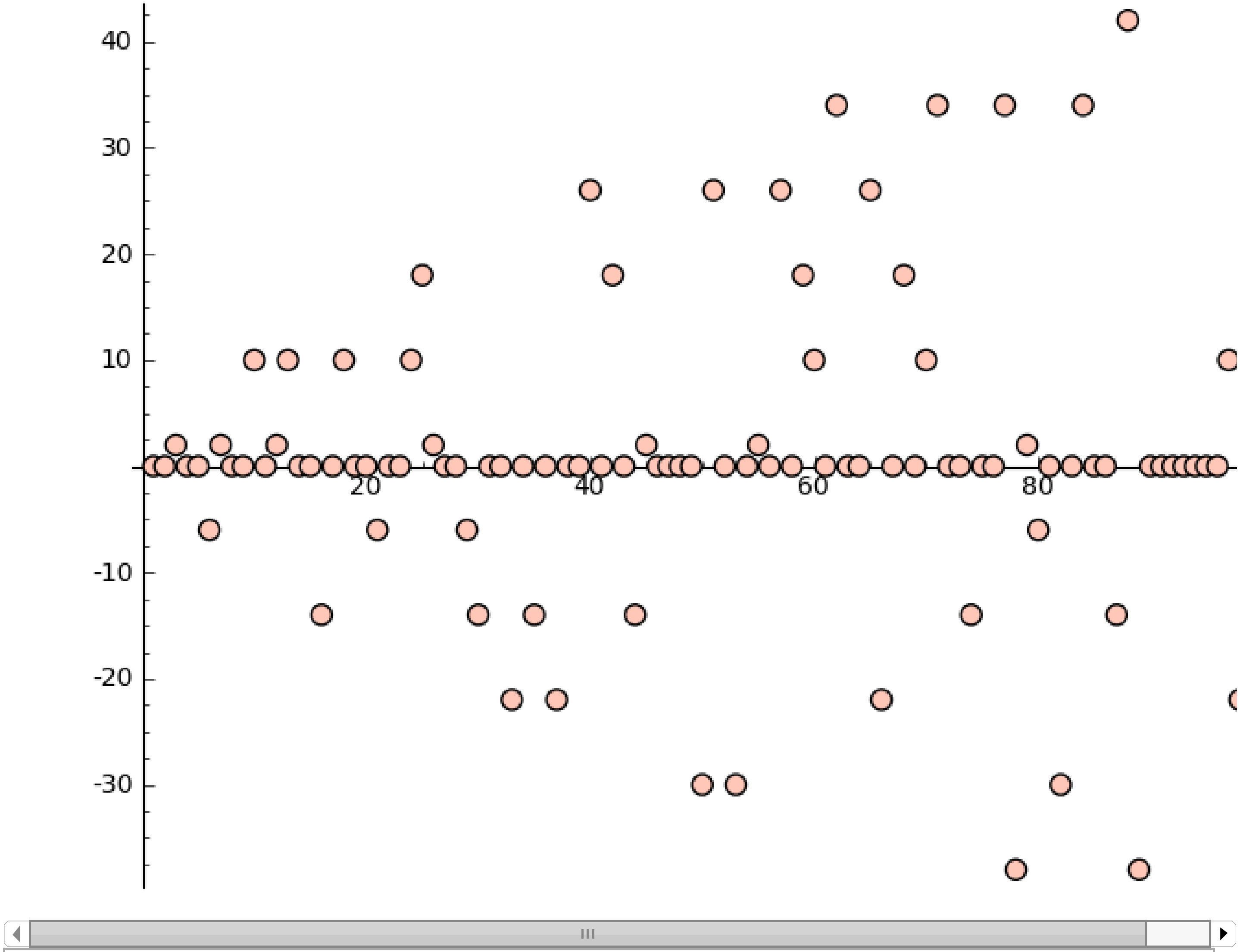
$$((i + 1))^4 \cdot (3)^3$$

```
L.galois_group()
```

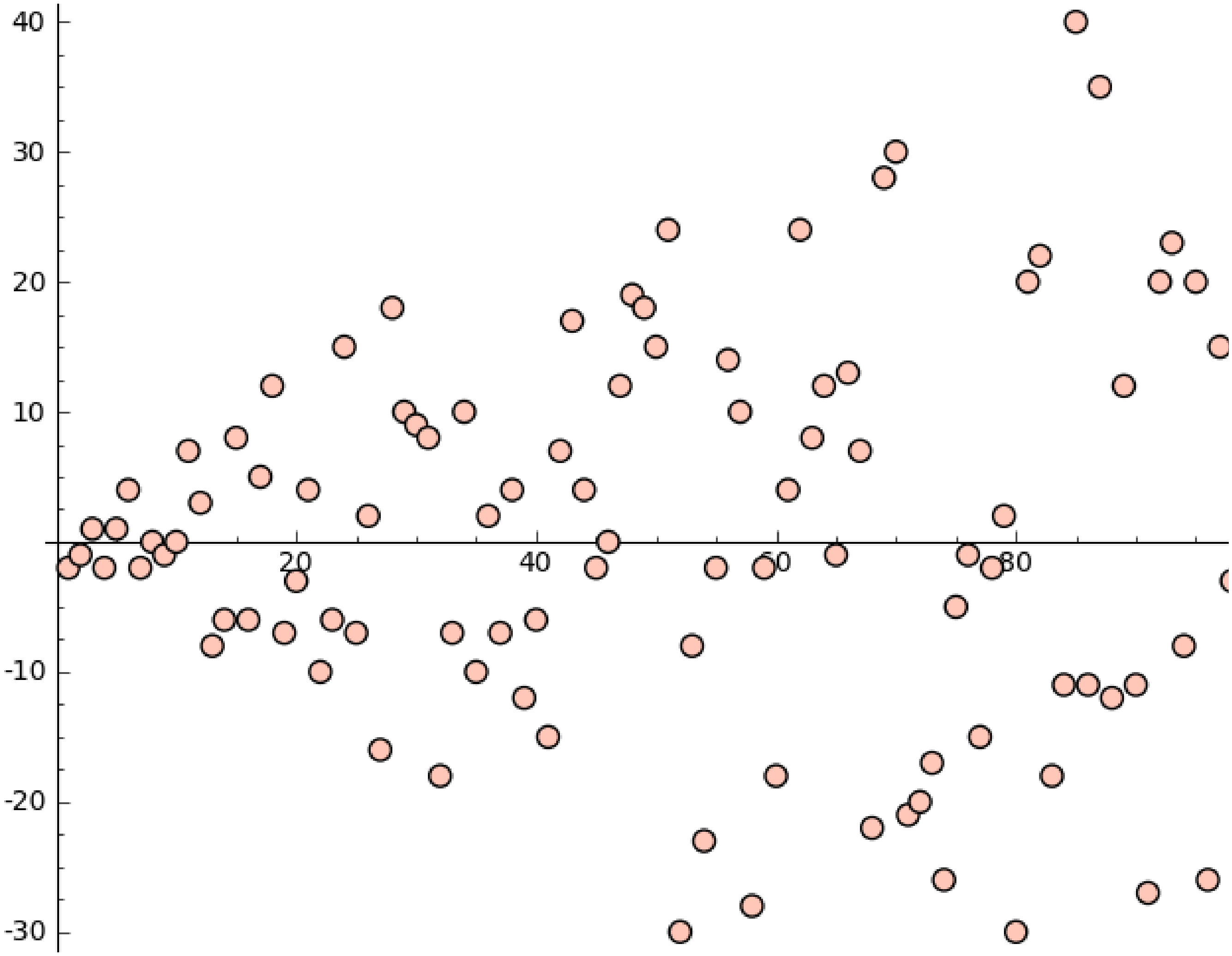
```
Galois group PARI group [8, 1, 4, "D_8(8)=[4]2"] of degre
```

```
$L$ is abelian over $K=\mathbb{Q}(i)$ and unramified outside of
the primes above 2 and 3, in accord with the theory of CM
elliptic curves.
```

```
scatter_plot([(t,E.an(nth_prime(t))) for t in range(1,100)])
```



```
E11=EllipticCurve('11a');
scatter_plot([(t,E11.an(nth_prime(t))) for t in range(1,100)])
```



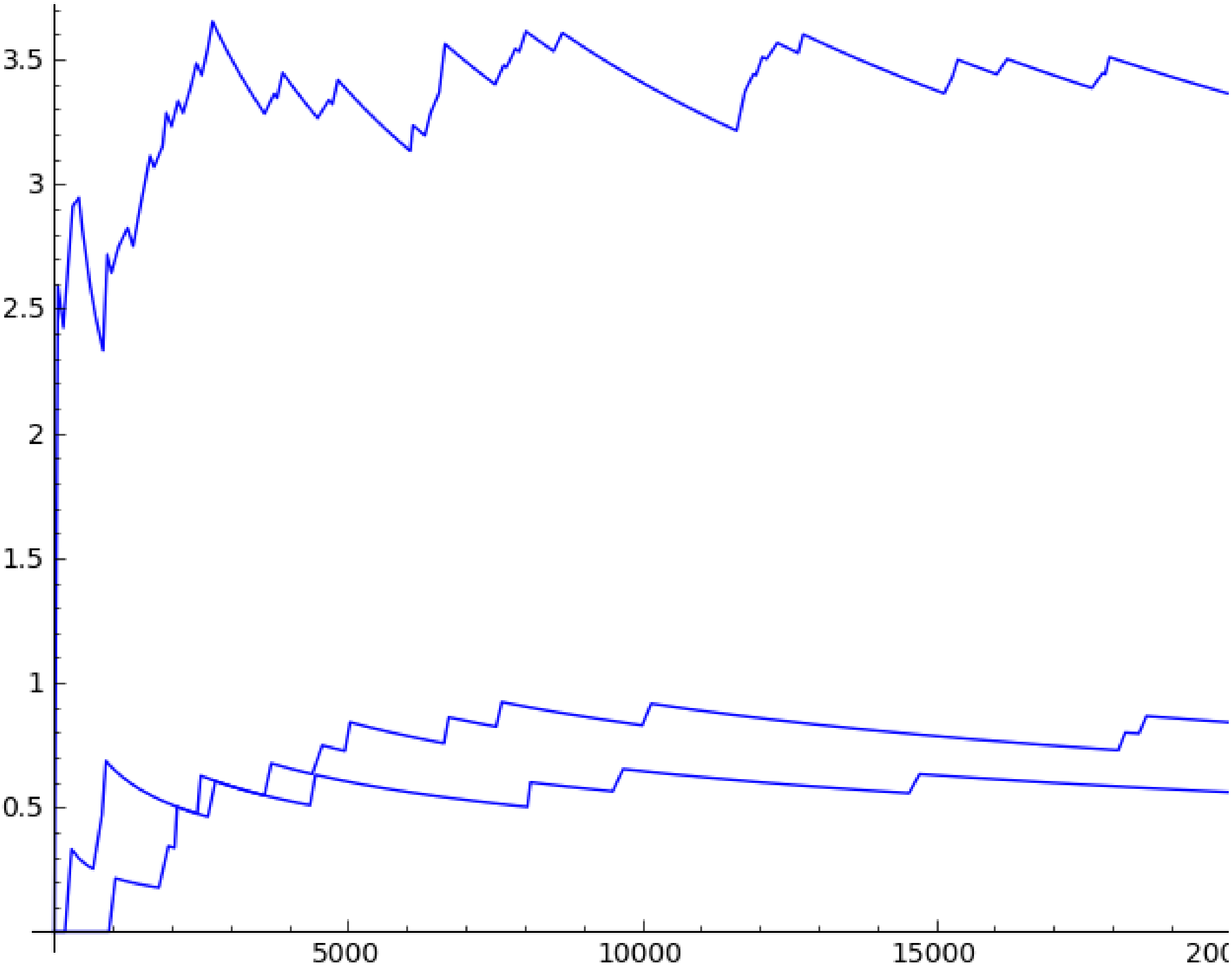
```
The Lang-Trotter conjecture:  if $E$ is non-CM, $K$ is an
imaginary quadratic field and $\pi_E(x,K)$ is the number of
primes below $x$ for which
$\text{End}^0(E\otimes\mathbb{F}_p)=K$, then $\pi_E(x,K)\sim
C\sqrt{x}/\log(x)$.
```

The Lang-Trotter conjecture: if E is non-CM, K is an imaginary quadratic
number of primes below x for which $\text{End}^0(E\otimes\mathbb{F}_p)=K$, then $\pi_E(x,K)\sim C\sqrt{x}$

```
def LTcount(x,D,E):
    sum=0
    for p in prime_range(x):
        a = E.an(p)
        d = a^2 - 4*p
        if QQ(d/D).is_square():
            sum=sum+1
    return sum
```

```
def g4(x):
    return LTcount(x,-4,E11)*log(x)/sqrt(x)
def g5(x):
    return LTcount(x,-5,E11)*log(x)/sqrt(x)
def g7(x):
    return LTcount(x,-7,E11)*log(x)/sqrt(x)
```

```
plot([g4,g5,g7],(2,20000))
```



```
A modular parametrization of a (modular) elliptic curve is a
nonconstant holomorphic map
$\phi\colon\Gamma_0(N)\backslash\mathcal{H}\to E(\mathbb{C})$,
where $N$ is the conductor of $E$.
```

A modular parametrization of a (modular) elliptic curve is a nonconstant holon

$E(\mathbb{C})$, where N is the conductor of E.

```
E=EllipticCurve('11a')
```

```
phi=E.modular_parametrization()
```

```
X=phi.power_series()[0]; Y=phi.power_series()[1]; X; Y #q=exp(2
pi i z)
```

$\frac{1}{q^2} + \frac{2}{q} + 4 + 5q + 8q^2 + q^3 + 7q^4 - 11q^5 + 10q^6 - 12q^7 - 18q^8 - 22q^9 + 26q^{10} - 11q^{11}$

$\frac{-1}{q^3} + \frac{-3}{q^2} + \frac{-7}{q} - 13 - 17q - 26q^2 - 19q^3 - 37q^4 + 15q^5 + 16q^6 + 67q^7 + 6q^8 + 144q^9$

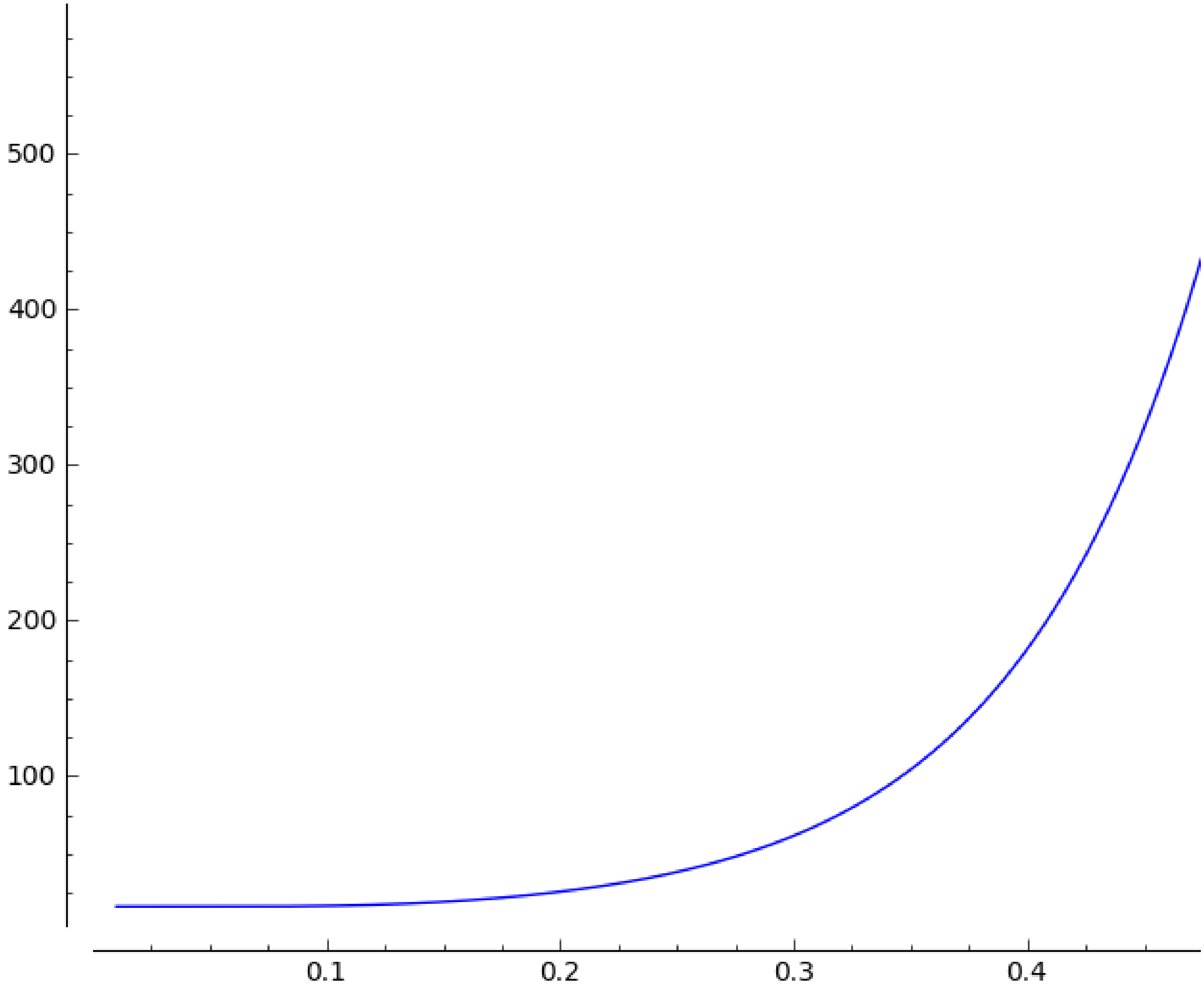
```
z1=I/7;
z2=(2*z1-1)/(11*z1-5);
phi(z1); phi(z2)
```

$(18.4384772424687 : -76.1607833071425 : 1.00000000000000)$

$(18.4384772424687 + 2.52001331151533 \times 10^{-14} i : -76.1607833071423 - 1.620457$

```
def f(x):
    return phi(I*x)[0]
```

```
plot(f,(.01,.5))
```



```
phi(.001*I)
```

```
(16.0000000000000 : −60.9999999999999 : 1.00000000000000)
```

```
P=E(16,-61,1); P
```

```
(16 : −61 : 1)
```

```
E(0); P; 2*P; 3*P; 4*P; 5*P;
```

```
(0 : 1 : 0)
(16 : −61 : 1)
(5 : −6 : 1)
(5 : 5 : 1)
(16 : 60 : 1)
(0 : 1 : 0)
```

```
L=E.lseries()
```

```
Lambda=E.period_lattice()
```

```
Lambda.real_period()/5
```

0.253841860855911

```
L(1)
```

0.253841860855911

```
def f(x,y):
    return phi(x+I*y)[0].abs()
```

```
phi(2*I)[0].abs()
```

$8.22268890922212 \times 10^{10}$

```
plot3d(lambda x, y: f(x,y), (-1,1),(.1,.2))
```

Get Image

```
#Heegner points
P=phi((2+sqrt(-7))/11)
```

```
P
```

$(-5.50000000000000 - 22.4888861440490i : 103.000000000000 + 44.97777228809$

```
b=P[0]; b
```

$-5.50000000000000 - 22.4888861440490i$

```
b.algdep(2)
```

$x^2 + 11x + 536$

```
_.discriminant().factor()
```

$-1 \cdot 7 \cdot 17^2$

```
Pbar = phi((-2+sqrt(-7))/11); Pbar
```

$(-5.49999999999998 + 22.4888861440490i : 103.000000000000 - 44.97777228809$

```
K=QuadraticField(-7)
```

```
K(11).factor()
```

$(-1) \cdot (\sqrt{-7} - 2) \cdot (\sqrt{-7} + 2)$

```
P=E.heegner_point(-7).point_exact(); P
```

$(a : 4a - 4 : 1)$

```
a=P[0]; K=a.parent(); K
```

$\mathbf{Q}[a]/(a^2 - a + 2)$

```
EK=E.base_extend(K)
```

```
Pbar=EK((1-a,-4*a)); Pbar
```

$(-a + 1 : -4a : 1)$

```
Q=P + Pbar; Q
```

$(16 : -61 : 1)$

```
Q.has_finite_order()
```

True

```
Hlist=E.heegner_discriminants_list(30); Hlist #A list of
discriminants which satisfy the Heegner hypothesis with respect
to E.
```

$[-7, -8, -19, -24, -35, -39, -40, -43, -51, -52, -68, -79, -83, -84, -87, -95,$

```
P=E.heegner_point(-35).point_exact(); P
```

$$\left(a : \frac{1}{201}a^3 - \frac{12}{67}a^2 + \frac{586}{201}a + \frac{262}{67} : 1\right)$$

```
a=P[0]; H=a.parent(); H #It turns out that H is the Hilbert
class field of Q(sqrt(-35)).
```

$$\mathbf{Q}[a]/(a^4 - 12a^3 + 1330a^2 + 3996a + 3789)$$

```
K=H.subfields()[1][0]
```

```
sigma=K.embeddings(H)[0]
```

```
Hrel=H.relativize(sigma(K.gens()[0]),'z')
```

```
Hrel.relative_discriminant()
```

$$(1)$$