Implicitization by Hybrid Methods

Daniel Lichtblau

Wolfram Research, Inc.
100 Trade Centre Dr.
Champaign IL USA, 61820

danl@wolfram.com

ADG 2004, Gainesville, FL
September 2004

Introduction

We investigate a family of trigonometric planar mappings of the form

$$x = \cos(m\,a) + \cos(m\,b) + \cos(m\,(a-b))$$
$$y = \cos(n\,a) + \cos(n\,b) + \cos(n\,(a-b))$$

for fixed integers *m* and *n*. These arose a few years ago in the context of a problem in extremal number theory that in turn gave rise to a lattice packing. The number theory problem was to compute, for given integer *s*,

$$f(s) = \underset{a_j \in \mathbb{R}}{\text{Min Max}}\left[\left|\sum_{j=1}^{s} e^{i\,a_j}\right|, \left|\sum_{j=1}^{s} e^{i\,s\,a_j}\right|\right]$$

The relevant theorem is that this value is 1 for *s* <= 2, and 0 for *s* >= 4. The important case of *s* = 3 was computed by Seon−Hong Kim as part of his Ph.D dissertation work; it used the map above with *m* = 1 and *n* = 3. It is approxi−mately 1.2. At ACA 2003 I showed how one might compute it using symbolic techniques.

Here are some definitions we will need.

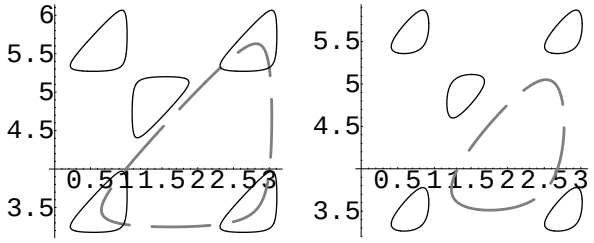
```
trigpoly[n_, a_, b_] =
  Cos[n a] + Cos[n b] + Cos[n (a - b)];
grad[expr_, a_, b_] := {∂a expr, ∂b expr}
trigsubs = {Cos[a] -> ca,
   Sin[a] -> sa, Cos[b] -> cb, Sin[b] -> sb};
```

Theorem [Kim 2003]: We may find $f(3)$ by determining tangential intersec−tions of the curves

$$\cos(a) + \cos(b) + \cos(a - b) = k$$
$$\cos(3a) + \cos(3b) + \cos(3(a - b)) = k$$

for values {*x*, *y*, *k*}. [These new functions are half of (the squares of the modu−li−3).] Moreover at extremal values of $f(3)$ there will be only finitely many such intersection points, up to periodicity.

We illustrate level curves for values of k, −1.1 and −1.3, that bracket the minimizer.



The actual computation is a polynomial elimination from a set of polynomials involving a Lagrange multiplier. We forego the details here (they are in the paper) and just show it.

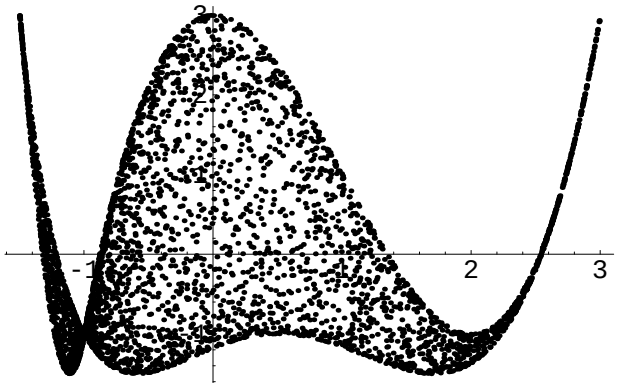
```
{e1, e2} =
  Map[TrigExpand[trigpoly[#, a, b]] &, {1, 3}];
polys = Flatten[{e1 - k, e2 - k, grad[e1, a, b] -
      λ grad[e2, a, b], Cos[a]^2 + Sin[a]^2 - 1,
     Cos[b]^2 + Sin[b]^2 - 1}] /. trigsubs;
Timing[kpoly = First[GroebnerBasis[
    polys, k, {λ, c_a, s_a, c_b, s_b},
    MonomialOrder -> EliminationOrder]]]
```

```
{0.37 Second, 9 + 3 k - 5 k^2 + k^3 - 6 k^4 - 4 k^5 + 2 k^6}
```

```
k /. NSolve[kpoly == 0, k]
```

```
{-1.20409, -1., 0.102047 - 1.11146 i,
 0.102047 + 1.11146 i, 1., 3.}
```

We form a picture of the planar map for the case $m = 1$ and $n = 3$.



The minimal value is the last point in the lower left where this intersects the line $y = x$. The other real roots also have physical significance with respect to this picture. For example, {1, 1} is a local maximizer.

The envelope curves

This brings us to the next question. The map itself looks interesting, and we might wish to know equations to describe the envelope curve. For example, is it indeed (as it appears) the intersection of two quartics of the form $y = f_j(x)$? (No, it is not.) How can we find this envelope curve?

Various possibilities. We might look for where Jacobian vanishes. We might try to extremize one variable as a function of the other.

Moreover, we can work with the trig formulation, or use complex exponen−tials, or use the standard rationalization of trig parametrization.

We will extremize, using the trig formulation.

The idea is to write x and y as functions of trigonometric polynomials, again make substitutions so these become explicitly algebraic, and realize that at the extrema (that is, on the boundary curve(s)), the gradients of the two functions must be parallel. (Reason: on the envelope, for fixed $x = x(a, b)$ we extremize $y = y(a, b)$. This can be set up as a standard Lagrange multiplier problem with $\nabla y = \lambda \nabla x$.)

```
parameters = {a, b}; mainvars = {x, y};
xpoly = x - trigpoly[1, a, b];
ypoly = y - trigpoly[3, a, b];
xypolys = {xpoly, ypoly};
gradientpolys =
  grad[xpoly, a, b] - λ grad[ypoly, a, b];
trigidentities = {c_a^2 + s_a^2 - 1, c_b^2 + s_b^2 - 1};
elimvars = {λ, c_a, c_b, s_a, s_b};
polys = TrigExpand[Join[trigidentities,
    xypolys, gradientpolys]] /. trigsubs
```

$$\{-1 + c_a^2 + s_a^2,\ -1 + c_b^2 + s_b^2,$$
$$x - c_a - c_b - c_a\, c_b - s_a\, s_b,\ y - c_a^3 - c_b^3 - c_a^3\, c_b^3 +$$
$$3\, c_a\, s_a^2 + 3\, c_a\, c_b^3\, s_a^2 - 9\, c_a^2\, c_b^2\, s_a\, s_b + 3\, c_b^2\, s_a^3\, s_b +$$
$$3\, c_b\, s_b^2 + 3\, c_a^3\, c_b\, s_b^2 - 9\, c_a\, c_b\, s_a^2\, s_b^2 + 3\, c_a^2\, s_a\, s_b^3 - s_a^3\, s_b^3,$$
$$s_a - 9\,\lambda\, c_a^2\, s_a + c_b\, s_a - 9\,\lambda\, c_a^2\, c_b^3\, s_a + 3\,\lambda\, s_a^3 +$$
$$3\,\lambda\, c_b^3\, s_a^3 - c_a\, s_b + 9\,\lambda\, c_a^3\, c_b^2\, s_b - 27\,\lambda\, c_a\, c_b^2\, s_a^2\, s_b +$$
$$27\,\lambda\, c_a^2\, c_b\, s_a\, s_b^2 - 9\,\lambda\, c_b\, s_a^3\, s_b^2 - 3\,\lambda\, c_a^3\, s_b^3 + 9\,\lambda\, c_a\, s_a^2\, s_b^3,$$
$$-c_b\, s_a + 9\,\lambda\, c_a^2\, c_b^3\, s_a - 3\,\lambda\, c_b^3\, s_a^3 + s_b + c_a\, s_b - 9\,\lambda\, c_b^2\, s_b -$$
$$9\,\lambda\, c_a^3\, c_b^2\, s_b + 27\,\lambda\, c_a\, c_b^2\, s_a^2\, s_b - 27\,\lambda\, c_a^2\, c_b\, s_a\, s_b^2 +$$
$$9\,\lambda\, c_b\, s_a^3\, s_b^2 + 3\,\lambda\, s_b^3 + 3\,\lambda\, c_a^3\, s_b^3 - 9\,\lambda\, c_a\, s_a^2\, s_b^3\}$$

Now what? We now want to compute the polynomial in {*x*, *y*} in the elimina−tion ideal. For that we can simply form a Gröbner basis with an appropriate term order.

```
Timing[
 gb = GroebnerBasis[polys, mainvars, elimvars,
   Sort → True, MonomialOrder → EliminationOrder]]
```

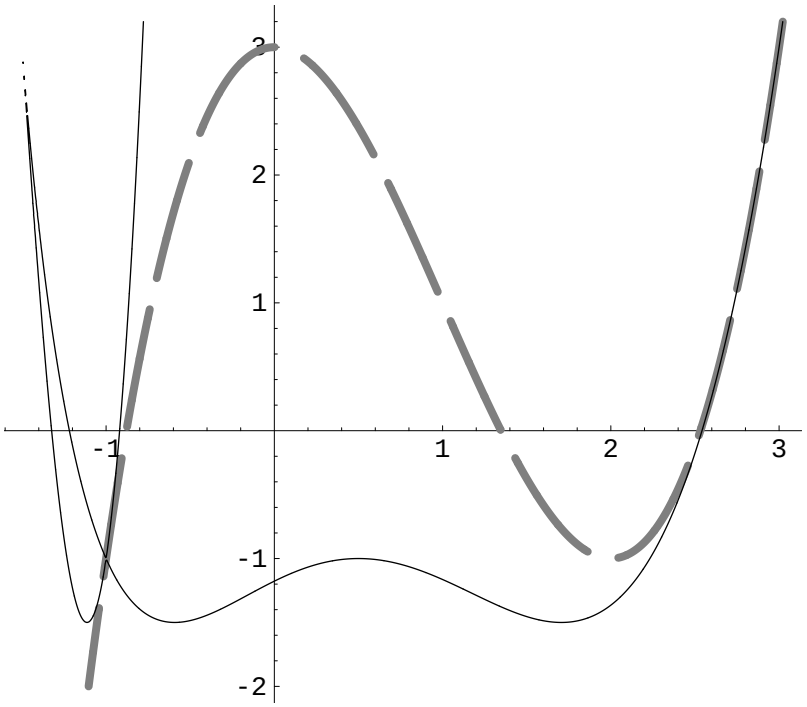
$$\{8.55\ \text{Second},\ \{-216 - 324\,x + 135\,x^2 + 108\,x^3 - 243\,x^4 + 117\,x^5 + 216\,x^6 - 72\,x^7 - 48\,x^8 + 16\,x^9 - 108\,y - 270\,x\,y - 9\,x^2\,y + 342\,x^3\,y + 162\,x^4\,y - 48\,x^5\,y - 24\,x^6\,y + 63\,y^2 + 126\,x\,y^2 + 69\,x^2\,y^2 + 9\,x^3\,y^2 - y^3\}\}$$

```
envelope = Map[#[[1]]^#[[2]] &,
  Drop[FactorList[First[gb]], 1]]
```

$$\{3 - 3\,x^2 + x^3 - y,\ -72 - 108\,x - 27\,x^2 - 48\,x^3 - 72\,x^4 + 16\,x^6 - 60\,y - 126\,x\,y - 72\,x^2\,y - 8\,x^3\,y + y^2\}$$

Let’s have a look at those factors.

```
ImplicitPlot[Evaluate[Thread[envelope == 0]],
  {x, -1.5, 3.2}, {y, -2, 3.2},
  PlotPoints → 700, AspectRatio → 7 / 8,
  PlotStyle → {{Thickness[.01], GrayLevel[.5],
     Dashing[{.15, .04}]}, {Automatic}}];
```



The general problem

On to the general problem. These are interesting planar maps from which one might wish to find envelope curves. We illustrate the $n = 5$ case below using around 4000 points.

```
x[a_, b_] = trigpoly[1, a, b];
y[a_, b_] = trigpoly[5, a, b];
pointlist[m_] :=
 Table[2 * Pi * {Random[], Random[]}, {2^m}]
points2k = pointlist[12];
data[{point__}] := {x[point], y[point]};
datalist2k = Map[data, points2k];
lplot = ListPlot[datalist2k];
```

For $n = 4$ we already cannot find the envelope curves in reasonable time using the method shown. The program Fermat handled a formulation that case in about 15 minutes (while we were at dinner). For $n = 5$ it ran out of memory. So...we need hybrid methods.

Again we set up an appropriate polynomial system.

```
xpoly = x - trigpoly[1, a, b];
ypoly = y - trigpoly[5, a, b];
xypolys = {xpoly, ypoly};
gradientpolys =
  grad[xpoly, a, b] - λ grad[ypoly, a, b];
polys = TrigExpand[Join[trigidentities,
     xypolys, gradientpolys]] /. trigsubs;
```

We'll begin by specifying one variable and solving for the other (we'll need to get many points, but this is a start). For this purpose we use a numeric solver.

```
allbutyvars = {x, λ, c_a, c_b, s_a, s_b};
polyy = polys /. y -> 11 / 10;
Timing[
 soln = NSolve[polyy, allbutyvars, Sort -> True];]
```

```
{51.94 Second, Null}
```

```
Select[Union[x /. soln, SameTest →
   (Abs[#1 - #2] / (Abs[#1] + Abs[#2]) < 10^(-5) &)],
 Im[#] == 0 &]
```

```
{-1.4246, -1.18145, -1.17679, -0.952203,
 -0.895835, -0.684639, -0.191699, 0.382082,
 1.02001, 2.15932, 2.9082, 2.9086}
```

Wasn't very easy. And for higher degrees it will only get worse. We need another method.

What we really need is a way to find roots locally. Alas, for this we need reasonable initial guesses as to values of parameters, which we do not have. We'll instead formulate this step as an optimization problem, with parameters taking values in known ranges. This is why we chose a trig formulation, because the trig variables are known to take on values between −1 and 1. The objective function we "optimize" is a trivial constant function; we are simply performing a constraint satisfaction.

Here is the set of constraints that must be set to zero to solve for *x* when *y* is 11/10.

```
Short[polyy, 8]
```

$$\{-1 + c_a^2 + s_a^2, \ll 4 \gg,$$
$$-c_b s_a + 25 \lambda c_a^4 c_b^5 s_a - 50 \lambda c_a^2 c_b^5 s_a^3 + 5 \lambda c_b^5 s_a^5 + s_b +$$
$$c_a s_b - 25 \lambda c_b^4 s_b - 25 \lambda c_a^5 c_b^4 s_b + 250 \lambda c_a^3 c_b^4 s_a^2 s_b -$$
$$125 \lambda c_a c_b^4 s_a^4 s_b - 250 \lambda c_a^4 c_b^3 s_a s_b^2 +$$
$$500 \lambda \ll 3 \gg s_b^2 - \ll 1 \gg + 50 \lambda c_b^2 s_b^3 + 50 \lambda c_a^5 c_b^2 s_b^3 -$$
$$500 \lambda c_a^3 c_b^2 s_a^2 s_b^3 + 250 \lambda c_a c_b^2 s_a^4 s_b^3 +$$
$$125 \lambda c_a^4 c_b s_a s_b^4 - 250 \lambda c_a^2 c_b s_a^3 s_b^4 + 25 \lambda c_b s_a^5 s_b^4 -$$
$$5 \lambda s_b^5 - 5 \lambda c_a^5 s_b^5 + 50 \lambda c_a^3 s_a^2 s_b^5 - 25 \lambda c_a s_a^4 s_b^5\}$$

We solve and get our value for *x* as well as the parameters. We are looking for a solution with *x* near 1.

```
Timing[{val, root} = NMinimize[{1,
    Flatten[{Thread[polyy == 0], .9 <= x <= 1.2}]},
   {{x, .9, 1.2}, {λ, -10, 10}, {c_a, -1, 1},
    {c_b, -1, 1}, {s_a, -1, 1}, {s_b, -1, 1}},
   MaxIterations -> 200,
   Method -> "DifferentialEvolution"]]
```

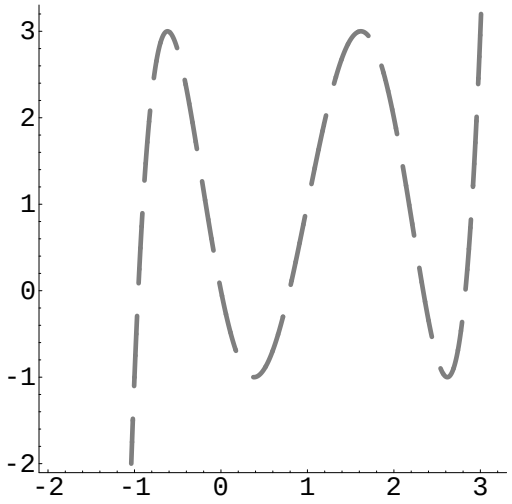
```
{10.78 Second,
 {1., {x → 1.02001, λ → 0.20024, c_a → 0.010004,
   c_b → 1., s_a → -0.99995, s_b → 7.89435 × 10^-18}}}
```

We got more than we anticipated; we see that this part of the envelope curve arises from a segment in parameter space where $b = 0$ (because $\sin(b)$ apparently vanishes). How lucky for us!

```
envelopepiece1 =
 First[GroebnerBasis[Take[polys, 4] /.
    {c_b -> 1, s_b -> 0}, {x, y}, { c_a, s_a}]]
```

$$5\,x - 5\,x^2 - 5\,x^3 + 5\,x^4 - x^5 + y$$

```
envplot1 =
  ImplicitPlot[Evaluate[envelopepiece1 == 0],
   {x, -2, 3.2}, {y, -2, 3.2},
   PlotStyle → {Thickness[.01], GrayLevel[.5],
     Dashing[{.15, .07}]}, PlotPoints → 1000];
```



A similar lucky computation gives us the part of the envelope curve in the lower right.

```
polyy = polys /. y -> -11 / 10;
Timing[{val, root} = NMinimize[{1,
    Flatten[{Thread[polyy == 0], 1.2 <= x <= 2}]},
   {{x, 1.2, 2}, {λ, -10, 10}, {c_a, -1, 1},
    {c_b, -1, 1}, {s_a, -1, 1}, {s_b, -1, 1}},
   MaxIterations -> 200,
   Method -> "DifferentialEvolution"]]
```

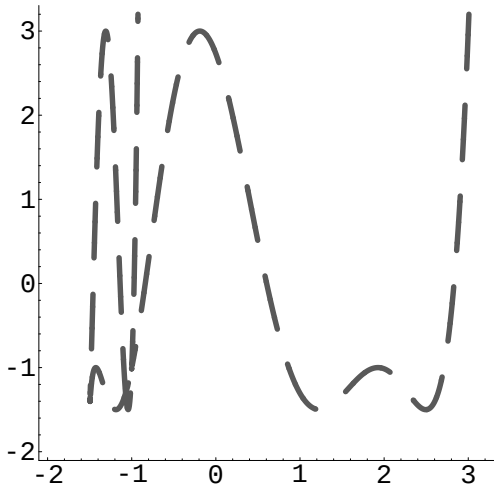
```
{10.85 Second,
 {1., {x → 1.72049, λ → 1.13162, c_a → 0.768955,
   c_b → 0.768955, s_a → 0.639303, s_b → -0.639303}}}
```

Here clearly we have $b = -a$ in the domain (cosines are equal, sines are opposite).

```
envelopepiece2 =
 First[GroebnerBasis[Take[polys, 4] /.
    {c_b -> c_a, s_b -> -s_a}, {x, y}, { c_a, s_a}]]
```

$$-5400 - 11700\,x + 8675\,x^2 + 37800\,x^3 + 20600\,x^4 - 14880\,x^5 - 13680\,x^6 + 1920\,x^7 + 3200\,x^8 - 256\,x^{10} + 1980\,y + 6230\,x\,y + 7280\,x^2\,y + 3800\,x^3\,y + 800\,x^4\,y + 32\,x^5\,y - y^2$$

```
envplot2 =
  ImplicitPlot[Evaluate[envelopepiece2 == 0],
   {x, -2, 3.2}, {y, -2, 3.2},
   PlotStyle → {Thickness[.012], GrayLevel[.35],
     Dashing[{.12, .08}]}, PlotPoints → 1000];
```



The hybrid computation

Now we have to get the last piece of boundary curve. For this we will require real work. We start by grabbing a point near the lower crossing with the *y* axis. I found it expedient to change the optimization slightly so as to better balance the constraints (I now square some of them).

```
polyx1 = polys /. x -> 0;
Timing[{val, root1} = NMinimize[{1, Flatten[
     {Thread[polyx1^2 == 0], -1.6 <= y <= -1.2}]},
   {{y, -1.6, -1.2}, {λ, -10, 10}, {c_a, -1, 1},
    {c_b, -1, 1}, {s_a, -1, 1}, {s_b, -1, 1}},
   MaxIterations -> 200,
   Method -> "DifferentialEvolution"]]
```

```
{9.84 Second,
 {1., {y → -1.25, λ → -0.8, c_a → -0.21616,
   c_b → -0.660006, s_a → 0.976358, s_b → 0.75126}}}
```

Now we must find more envelope points in order to reconstruct the curve therefrom. We begin with the solution point we just found. We refine the result to high precision using a local root finder.

```
allbutxvars = {y, λ, c_a, c_b, s_a, s_b};
rootbetter1 = FindRoot[polyx1 == 0,
  Evaluate[Apply[Sequence, Transpose[
     {allbutxvars, allbutxvars /. root1}]]],
  PrecisionGoal -> 30, AccuracyGoal -> 30,
  WorkingPrecision -> 40]
```

```
{y → -1.250000000000000000000000000000000000000,
 λ → -0.800000000000000000000000000000000000000,
 c_a → -0.216160221673850756457782105666534222001 1,
 c_b → -0.660005866723161169829745025809548792479 4,
 s_a → 0.976357904953870746132758834833973953619 7,
 s_b → 0.751260444779976962667400856474058841541 0}
```

To deduce the actual curve we will require a number of {*x*, *y*} pairs of points lying on it. We will treat this as a homotopy continuation problem wherein the roots found above will be initial values for a differential system. We create the system below but display it in abbreviated form due to excessive size.

```
varsinx = Through[allbutxvars[x]];
parampolys =
  polys /. Thread[allbutxvars → varsinx];
diffpolys = D[parampolys, x];
Short[odesystem1 = Join[Thread[diffpolys == 0],
   Thread[(varsinx /. x → 0) ==
     (allbutxvars /. rootbetter1)]], 12]
```

```
{2 Ca[X] Ca′[X] + 2 Sa[X] Sa′[X] == 0,
 2 Cb[X] Cb′[X] + 2 Sb[X] Sb′[X] == 0,
 1 - Ca′[X] - Cb[X] Ca′[X] - Cb′[X] -
   Ca[X] Cb′[X] - Sb[X] Sa′[X] - Sa[X] Sb′[X] == 0,
 y′[X] - 5 Ca[X]^4 Ca′[X] - 5 Ca[X]^4 Cb[X]^5 Ca′[X] +
   30 Ca[X]^2 Sa[X]^2 Ca′[X] + <<100>> + 50 Ca[X]^2
    Sa[X]^3 Sb[X]^4 Sb′[X] - 5 Sa[X]^5 Sb[X]^4 Sb′[X] == 0,
 <<135>> + <<1>> == 0, <<1>> == 0, <<1>>,
 λ[0] == - <<63>>, Ca[0] == - <<63>>, Cb[0] ==
  -0.660005866723161169829745025809548792 4794,
 Sa[0] ==
  0.976357904953870746132758834833973953 6197,
 Sb[0] ==
  0.751260444779976962667400856474058841 5410}
```

Now we solve it.

```
desoln1 =
  NDSolve[odesystem1, varsinx, {x, 0, 1.2}];
```

```
NDSolve::ndsz :
 At x == 0.3090169939494037‘, step size is effectively zero; singularity or stiff system suspected.
```

The message tells us we were not able to get very far. Most likely we are near a tangential intersection of solution curves. But what we have suffices to give us points on the envelope curve. We refine to high precision.

```
solnfunctions1 = First[varsinx /. desoln1];
valuelists1 = Table[Join[{t → x},
     Thread[List[allbutxvars, solnfunctions1]]],
    {x, 0, 3 / 10, 5 / 1000}] /. t → x;
highprecsolns1 = Table[Flatten[
    {First[valuelists1[[j]]], FindRoot[Evaluate[
       (polys /. First[valuelists1[[j]]]) == 0],
      Evaluate[Apply[Sequence,
        Rest[valuelists1[[j]]]]],
      PrecisionGoal → 100, AccuracyGoal → 100,
      WorkingPrecision → 150]}],
   {j, Length[valuelists1]}];
```

We repeat the process of finding one solution, refining it, approximating a part of the curve through it, and refining individual points thereon.

```
polyx2 = polys /. x -> -2 / 5;
Timing[{val, root2} = NMinimize[{1, Flatten[
     {Thread[polyx2^2 == 0], -1.2 <= y <= -.9}]},
   {{y, -1.2, -.9}, {λ, -10, 10}, {c_a, -1, 1},
    {c_b, -1, 1}, {s_a, -1, 1}, {s_b, -1, 1}},
   MaxIterations -> 200,
   Method -> "DifferentialEvolution"]]
```

```
{10.48 Second,
 {1., {y → -1.02904, λ → 1.5674, c_a → -0.847762,
   c_b → -0.337837, s_a → 0.530377, s_b → 0.941204}}}
```

```
rootbetter2 = FindRoot[polyx2 == 0,
   Evaluate[Apply[Sequence, Transpose[
      {allbutxvars, allbutxvars /. root2}]]],
   PrecisionGoal → 30, AccuracyGoal → 30,
   WorkingPrecision → 40];
odesystem2 = Join[Thread[diffpolys == 0],
   Thread[(varsinx /. x → -2 / 5) ==
     (allbutxvars /. rootbetter2)]];
desoln2 = NDSolve[odesystem2, varsinx,
   {x, -2 / 5, -1}];
```

```
NDSolve::ndsz : At x == -0.809017, step size is effectively zero; singularity or stiff system suspected.
```

```
solnfunctions2 = First[varsinx /. desoln2];
valuelists2 = Table[Join[{t → x},
     Thread[List[allbutxvars, solnfunctions2]]],
    {x, -2 / 5, -4 / 5, -5 / 1000}] /. t → x;
highprecsolns2 = Table[Flatten[
    {First[valuelists2[[j]]], FindRoot[Evaluate[
       (polys /. First[valuelists2[[j]]]) == 0],
      Evaluate[Apply[Sequence,
        Rest[valuelists2[[j]]]]],
      PrecisionGoal → 100, AccuracyGoal → 100,
      WorkingPrecision → 150]}],
   {j, Length[valuelists2]}];
```

With these values we can reconstruct the curve. We guess that it is a polynomial of degree at most 2 in *y*, and 12 in *x*. We form the possible monomials as our "basis" set, evaluate these at the {*x*, *y*} values corresponding to our high precision approximations of points on the envelope, and find the appropriate null space.

```
Short[
 xyvals = Join[highprecsolns1, highprecsolns2] /.
   ((a_ /; ! MatchQ[a, x | y]) → b_) :→ Sequence[], 4]
xypowers = Flatten[Table[x^j * y^k,
    {j, 0, 12}, {k, 0, 2}]];
valuevectors = xypowers /. xyvals;
ns = NullSpace[valuevectors];
Length[ns]
```

```
{{x → 0, y → -<<170>>}, {x → 1/200, y →
   -1.25624937501250000000000000000000000000000000∖
    0000000000000000000000000000000000000000000∖
    0000000000000000000000000000000000000000000∖
    000000000000000000000},
 {x → 1/100, y → -<<170>>}, {x → 3/200, y →
   -1.26873312803750000000000000000000000000000000∖
    0000000000000000000000000000000000000000000∖
    0000000000000000000000000000000000000000000∖
    000000000000000000000}, <<134>>,
 {x → -<<1>>, <<1>>}, {x → -79/100, y → -<<170>>},
 {x → -159/200, y → -<<170>>}, {x → -4/5, y → -<<170>>}}

16
```

16 null vectors means we have too large a basis. We need to "squeeze" out a null vector in terms of lower degree monomials. We do this by Gaussian elimination.

```
ns2 = Chop[ns, 10^(-145)];
ns3 = Map[Reverse, ns2];
rred = RowReduce[ns3];
rred2 = Chop[rred, 10^(-145)];
N[Last[rred2]]
```

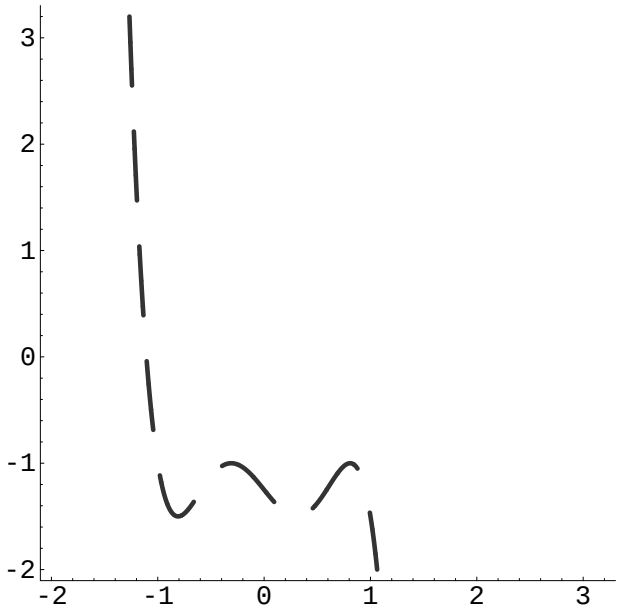
```
{0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 1., 0., 0., 0., 0., 0., -1.25, 0.,
 0., 0., 0., 0., 0.3125, 0., 0.25, 0.3125}
```

We rationalize, find the common denominator, remove it by multiplication, and finally form the curve by associating the resulting vector of coefficients with the monomial basis.

```
coeffs = Rationalize[Last[rred2]];
mult = Apply[LCM, Denominator[coeffs]];
coeffs2 = coeffs * mult;
envelopepiece3 = coeffs2.Reverse[xypowers]
```

$$5 + 5\,x - 20\,x^3 + 16\,x^5 + 4\,y$$

```
envplot3 =
  ImplicitPlot[Evaluate[envelopepiece3 == 0],
   {x, -2, 3.2}, {y, -2, 3.2},
   PlotStyle → {Thickness[.008], GrayLevel[.2],
     Dashing[{.12, .08}]}, PlotPoints → 1000];
```



```
ll = Show[{lplot, envplot1, envplot2, envplot3}];
```

Verification

To verify that our curve is part of the envelope, pick specific exact values and see if they are consistent with the original system. For example, we solve for y when x=−1/2.

```
xycoord = Prepend[
  First[Solve[(envelopepiece3 /. x → -1 / 2) == 0,
    y]], x → -1 / 2]
```

$$\left\{x \to -\frac{1}{2},\ y \to -\frac{9}{8}\right\}$$

Plug into the system, check that we get a nontrivial Gröbner basis.

```
GroebnerBasis[polys /. xycoord, {λ, c_a, c_b, s_a, s_b},
 MonomialOrder -> DegreeReverseLexicographic]
```

$$\{4 - 5\,\lambda,\ 7 - 4\,c_a - 4\,c_b - 8\,s_a^2 - 8\,s_a\,s_b - 8\,s_b^2,$$
$$-1 + c_b^2 + s_b^2,\ -1 - 2\,c_a - 2\,c_b - 2\,c_a\,c_b - 2\,s_a\,s_b,$$
$$1 + 4\,c_a - 8\,c_a^2 + 4\,c_b + 8\,s_a\,s_b + 8\,s_b^2,$$
$$2\,s_a - 2\,c_b\,s_a - 5\,s_b + 2\,c_a\,s_b + 4\,c_b\,s_b + 8\,s_b^3,$$
$$5\,s_a + 4\,c_a\,s_a + 3\,c_b\,s_a - s_b - 7\,c_a\,s_b - 4\,c_b\,s_b - 16\,s_a\,s_b^2,$$
$$-2 - 3\,c_b + 4\,s_b^2 + 8\,c_b\,s_b^2,$$
$$-1 + c_b + 2\,s_a\,s_b - 2\,c_b\,s_a\,s_b + 2\,s_b^2 + 2\,c_a\,s_b^2,$$
$$-3 - 4\,c_a + 4\,s_a\,s_b + 8\,c_a\,s_a\,s_b + 8\,s_b^2 + 8\,c_a\,s_b^2\}$$

If we do this with enough points we can prove that the entire curve satisfies the boundary curve system (standard result involving polynomial interpolation).

Alternative methods

(i) Use integer relation finding to deduce, from one point, the smallest polyno−mial equation it satisfies. A drawback is this will not be useful for approxi−mate cases or cases where the polynomial factorizates nontrivially over an algebraic extension of the rationals.

(ii) Use image processing methods of edge detection to solve for {*x*, *y*} pairs on the envelope curve. Use series "inversion" to formulate a Puiseux series to obtain {*a*, *b*} as (multivalued) functions of {*x*, *y*} on that curve. These could then replace the optimization and ODE solving steps we used above. As those were computationally the most difficult and required the most user interven−tion, anything to make them easier would be advantageous.

Applications

I tend to envision the methods descibed here as useful for finding exact curves and possibly surfaces in cases where these are really needed e.g. problems from pure math. For approximation purposes there are usually better/faster approaches. A possible exception might be in finding relations satisfied by solutions to ODEs, in cases where they lie on algebraic varieties but are not explicitly presented as DAEs (for one, this knowledge can be useful for deducing long term evolution of such solutions).

Needless to say, I would be interested in hearing about other possible uses for this technology.

Summary

Starting with a problem in number theory we were led to consider the problem of finding bounding curves to certain trigonometric planar maps. At modest degree symbolic methods fail to handle this in reasonable time. We instead employed several numeric technologies, to wit, an optimizer (employed as a constraint satisfaction solver), a local root finder, an ODE solver, and numeric linear algebra, to find our curve. We then use a fast symbolic method to verify that it is correct. We illustrated on an example that had not been amenable to tactics from symbolic algebra.

There are several open questions. The general issues are
(i) How to improve performance?
(ii) To what classes of problems to apply these techniques?