

Computing the selfintersection curve of a Bézier bicubic Surface

André Galligo
Nice (France)

To Joos Heintz for his 60th birthday

TERA'05 conference in Buenos Aires (Argentina), October 2005.

* Supported by the EU contract GAIA II(IST-2002-35512)

SUMMARY

The computation of the equation of the selfintersection curve of a surface patch is a major problem in Computer Aided Geometric Design (CAD). It amounts to study a system of 4 equations in 4 variables.

We present two (complementary) approaches to adress this problem for a bicubic Bezier patch.

- First, using a semi-numeric polynomial solver able to deal with a large system of equations with floating point coefficients.
- Second, using a specific sparse bivariate resultant adapted to the corresponding elimination problem. We also discuss the stability of the approximate computation of this resultant.

DEFINITION AND NOTATIONS

A parametric bicubic patch is the image of a map :

$$\Phi : [0, 1] \times [0, 1] \rightarrow \mathbb{R}^3$$

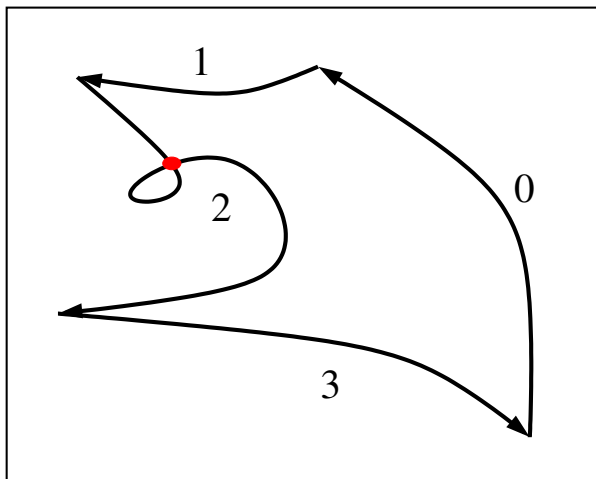
$$(t, u) \mapsto \Phi(u, v) = \left(\frac{\Phi_1(u, v)}{\Phi_0(u, v)}, \frac{\Phi_2(u, v)}{\Phi_0(u, v)}, \frac{\Phi_3(u, v)}{\Phi_0(u, v)} \right)$$

$\Phi_0, \Phi_1, \Phi_2, \Phi_3$ are polynomials of bidegree $(3, 3)$.

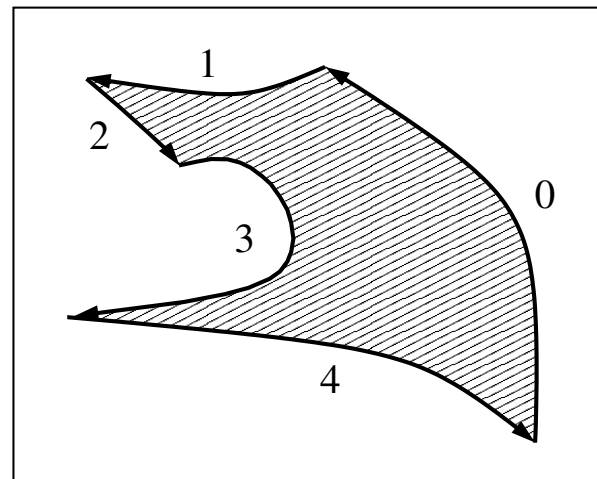
The patch is called Bézier when $\Phi_0(u, v) = 1$. (Our hypothesis).

These patches are glued together to model the surfaces used in Computer Aided Design (C.A.D.).

A domain in the plane is represented in C.A.D. by the list of the curve segments delineating its border. This list represents a well-defined domain if each segment is free of loop.



(a)



(b)

C.A.D. systems use a similar representation for volumes.

When a loop is detected, one has to perform a new parametrization in order to get a so called trimmed surface patch.

This problem is one of the main topics of the European project GAIA II. It has been addressed by various authors and various means. Among many authors, let us quote :

Lassner, via sections of higher dimensional splines,

Homeyer, via numerical computations,

T. Dokken and coworkers, via Approximate implicitization,

A. Galligo and M. Stillman, via geometric and symbolic computations for a subclass of patches,

J.P. Pavone and A. Galligo, via a sampling algorithm.

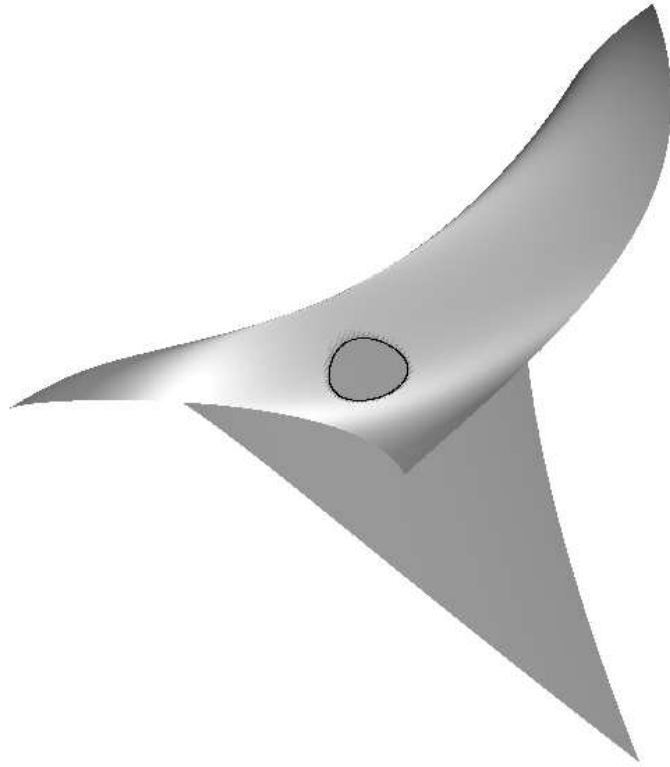
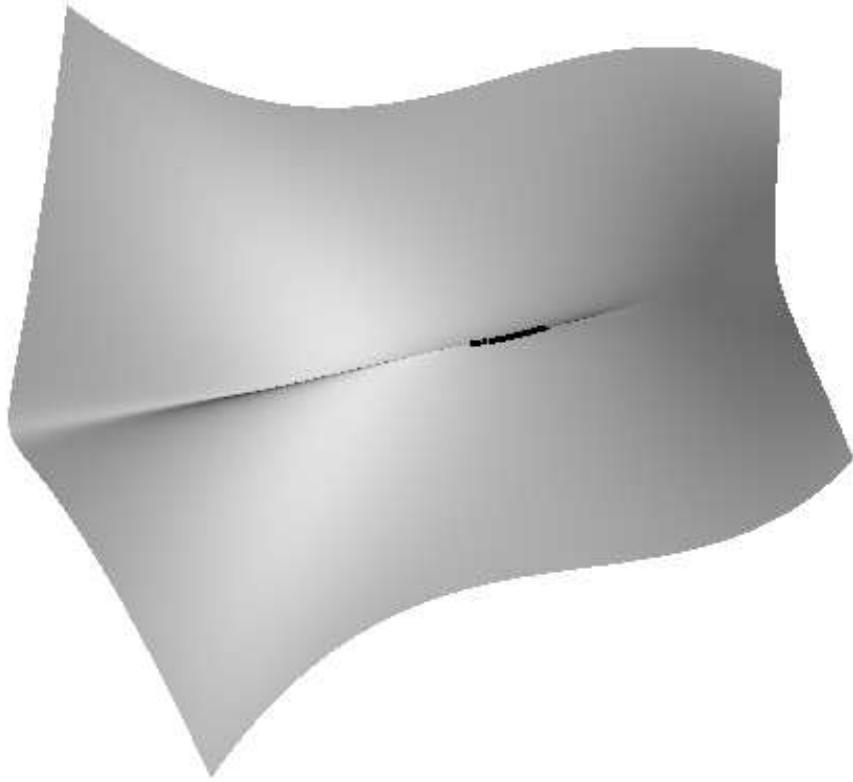


FIG. 1 – A selfintersection loop



SETTING THE EQUATIONS

The set of double points of Φ form a curve, in a 4 dimensional parameter space, characterized by :

$$\mathcal{C} = \{(u, v, u', v') \text{ s.t. } (u, v) \neq (u', v'), \Phi_i(u, v) = \Phi_i(u', v')\}.$$

- We assume $u > u'$ and we let :

$$u = u_1 + l ; v = v_1 ; u' = u_1 ; v' = v_1 + h ;$$

$$h = lk ; m = \frac{1}{l}.$$

- We get 3 equations of multidegree $(3, 3, 2, 3)$:

$$S_i := \frac{\Phi_i(u_1 + l, v_1) - \Phi_i(u_1, v_1 + lk)}{l}$$

that we also write :

$$T_i := m^2 * S_i(u_1, v_1, \frac{1}{m}, k)$$

in order to diminish its total degree.

An enumerative study shows that C is a curve of multi-degree $(44, 44, 44, 44)$.

TWO APPROACHES, TWO CONTRIBUTIONS

- First, via a numerical solver (joint work with J.P. Pavone),
- Second, via an adapted very sparse resultant and a numerical evaluation of a determinant.
- Towards a cooperation of these two approaches.

A NUMERICAL APPROACH (CF. ISSAC'05)

- To compute the critical points, we set :

$$T_4 := \det\left(\left[\frac{\partial T}{\partial v_1}, \frac{\partial T}{\partial k}, \frac{\partial T}{\partial m}\right]\right) = 0.$$

- T_4 is a polynomial of multidegree $(9, 8, 5, 8)$ in (u_1, v_1, m, k) .
- We have to solve (numerically) a system of 4 polynomials in 4 variables : $(T_1, T_2, T_3, T_4) = 0$.
- In his thesis, J. P. Pavone developed a solver, based on multivariate Bernstein representation of polynomials, it is adapted to our setting.

A POLYNOMIAL SOLVER

- The basic principle of this solver, is similar to that of the IPP solver of Sherbrooke and Patrikalakis. They both use control polyhedrons.
- The solvers perform a sequence of carefully chosen subdivisions and reductions to determine a finite number of very small boxes which may contain roots of the system.
- These solvers rely on projections of the control polyhedrons of the graphs of the equations to reduce the domain for each variable involved in the process.

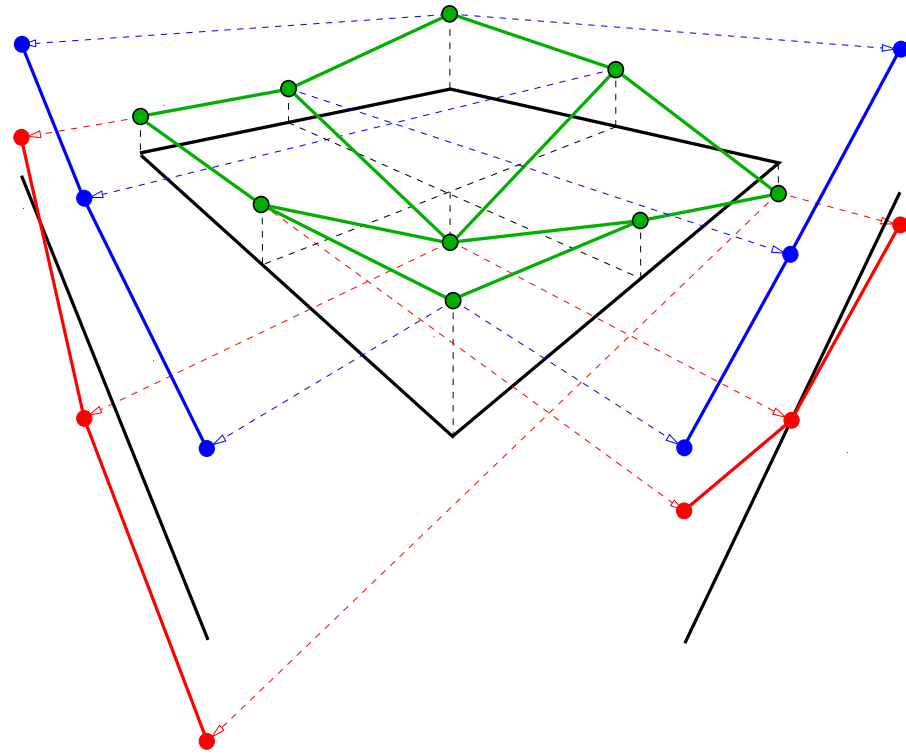


FIG. 2 – Principle of the *IPP* projection

Pavone's solver uses

- a preconditioner inspired by interval analysis,
- proceeds to much less subdivisions, compared to IPP.

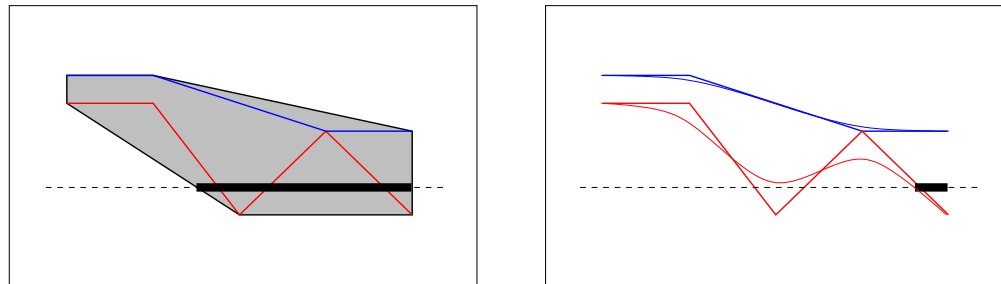


FIG. 3 – *IPP* reduction (a), our solver (b)

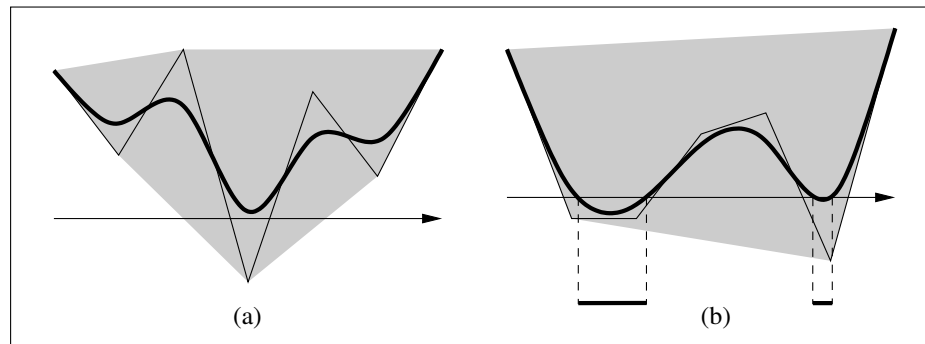


FIG. 4 – exclusion (a) and multiple reductions (b)

EXAMPLES AND TIMINGS

We computed the examples corresponding to the first two presented pictures.

Here are the coordinates of the control points for the first example, they also provide the coefficients of the map $\Phi(u, v)$ in the Bernstein basis of bi-degree $(3, 3)$.

$$\begin{array}{l} x= \\ y= \\ z= \end{array} \begin{bmatrix} 1.718019 & 0.045451 & 1.220627 & -8.437654 \\ 2.245235 & 0.572671 & -0.528670 & -1.472482 \\ -2.706853 & -3.948356 & -0.572671 & -2.245235 \\ -2.565772 & -4.284708 & -3.824920 & -1.718019 \\ -5.030251 & -3.145330 & -2.801745 & -4.636559 \\ -3.561677 & -1.676751 & 0.410812 & 1.075778 \\ -2.077333 & -0.843524 & 1.676751 & 3.561677 \\ -1.994069 & 0.916993 & 3.530333 & 5.030251 \\ 1.935424 & 2.807696 & 0.101320 & -2.755474 \\ -0.227130 & 0.645141 & -0.162758 & 0.015489 \\ 0.790476 & -1.114797 & -0.645141 & 0.227130 \\ -1.779410 & -0.718161 & -2.442728 & -1.935424 \end{bmatrix}$$

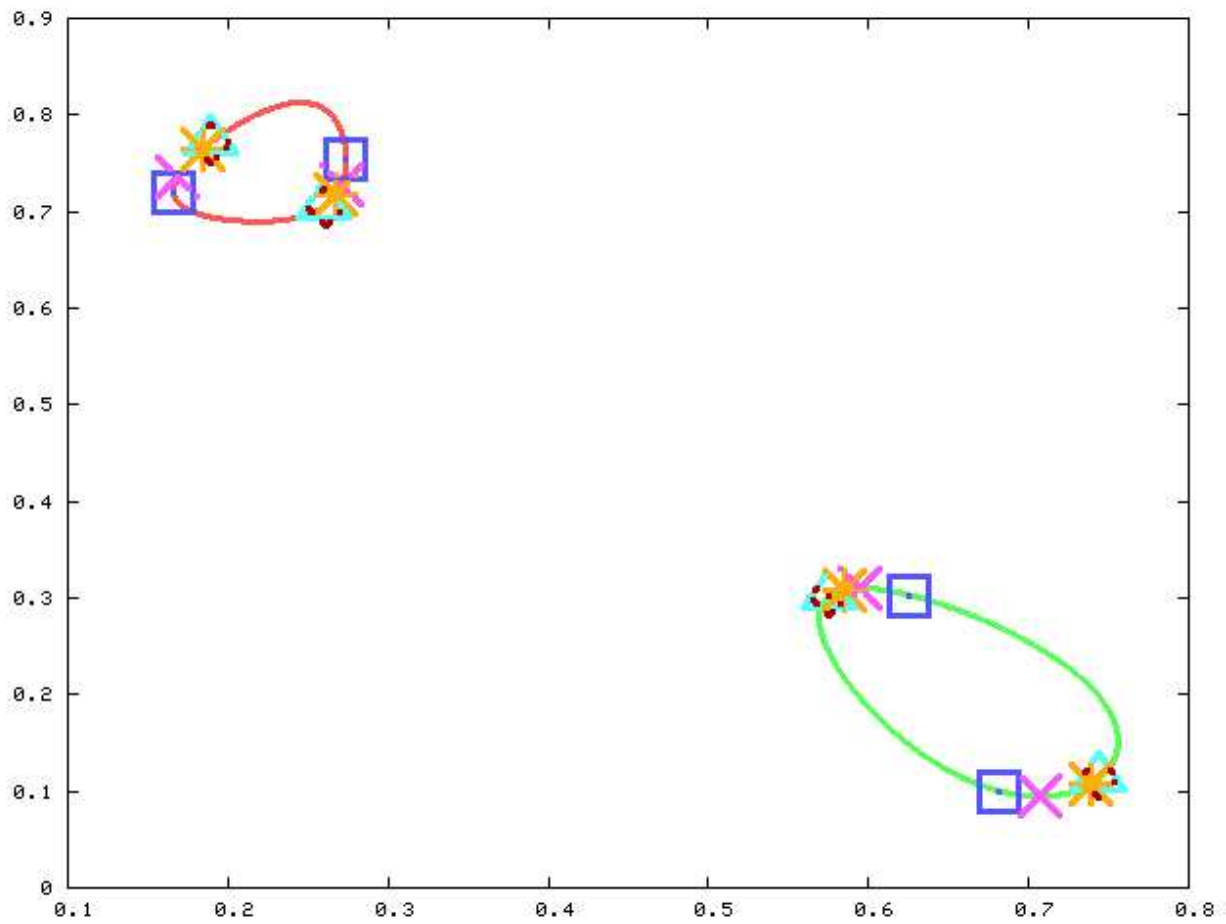


FIG. 5 – Extrema points in the domain 1, in u, v

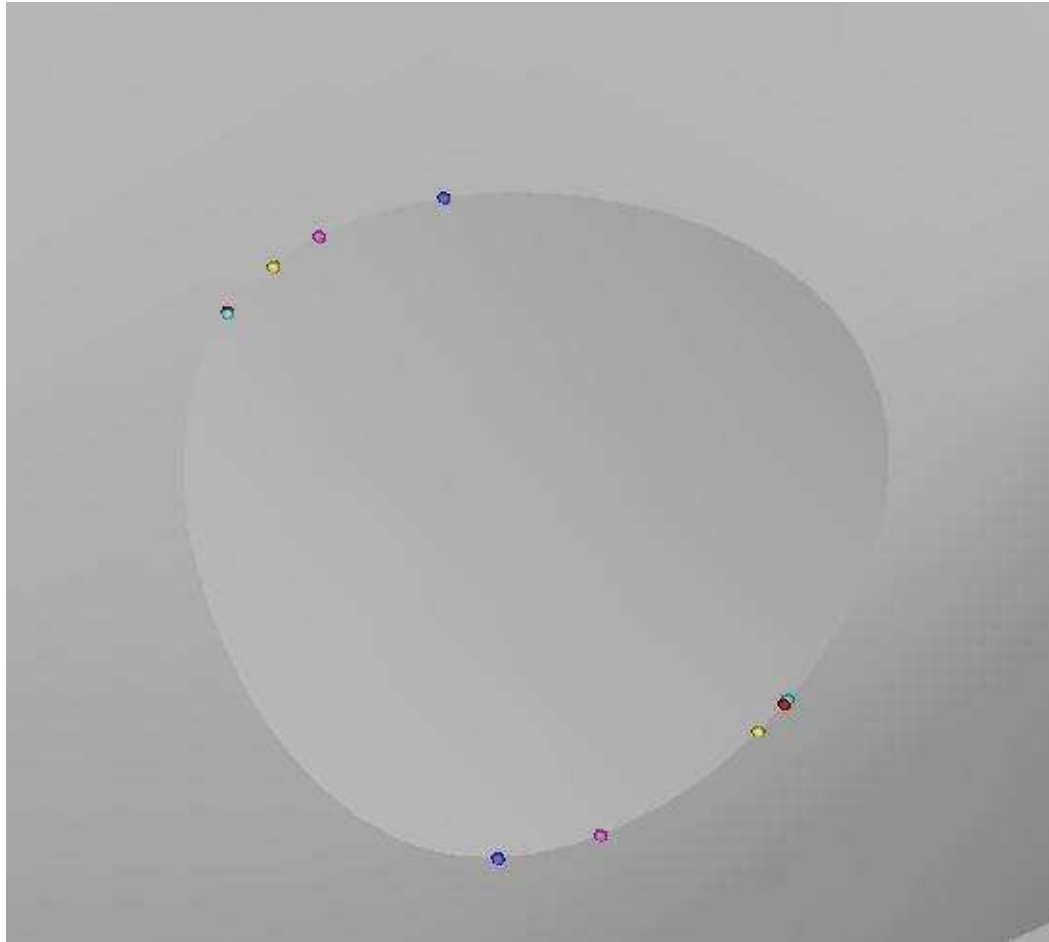


FIG. 6 – Extrema points on the surface 1

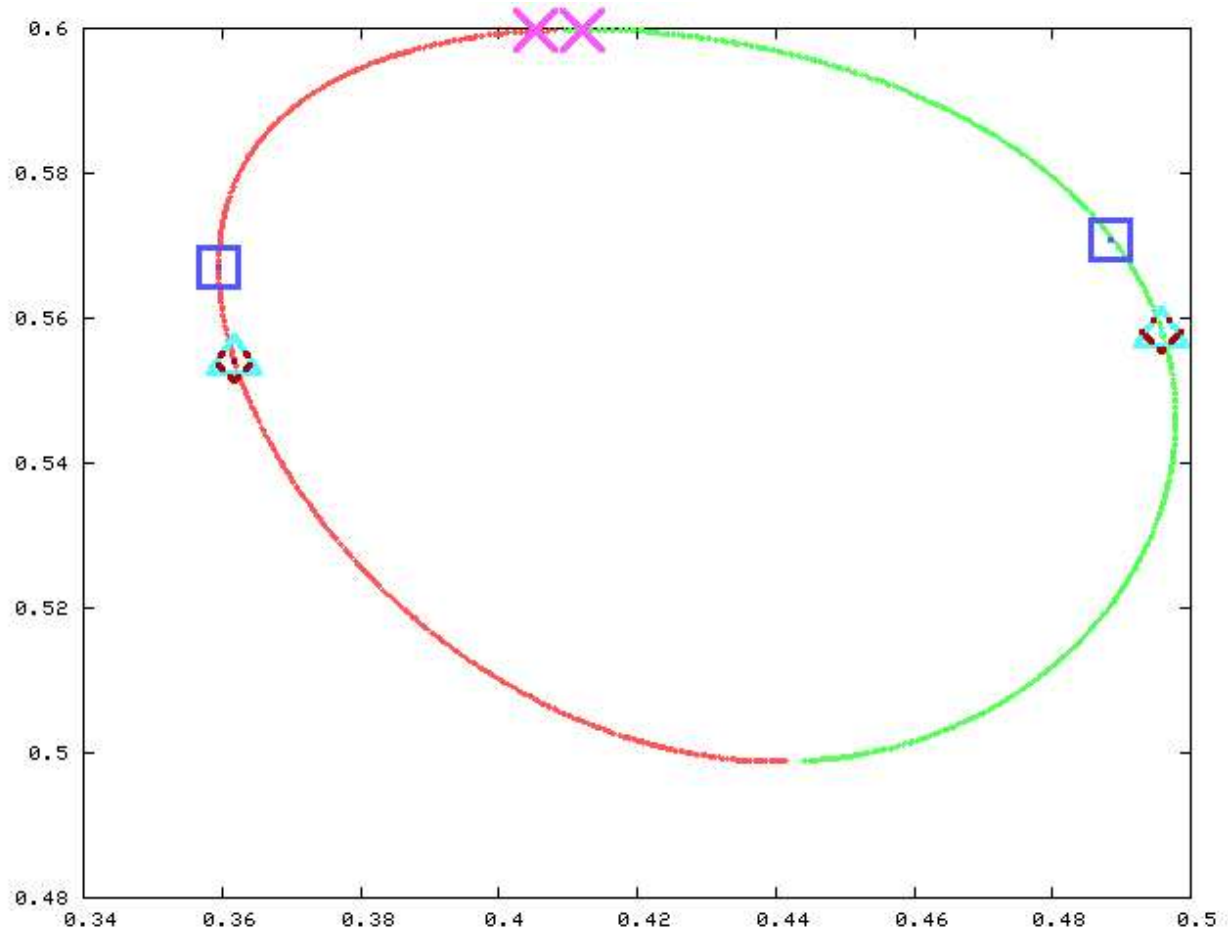


FIG. 7 – Extrema points in the domain 2

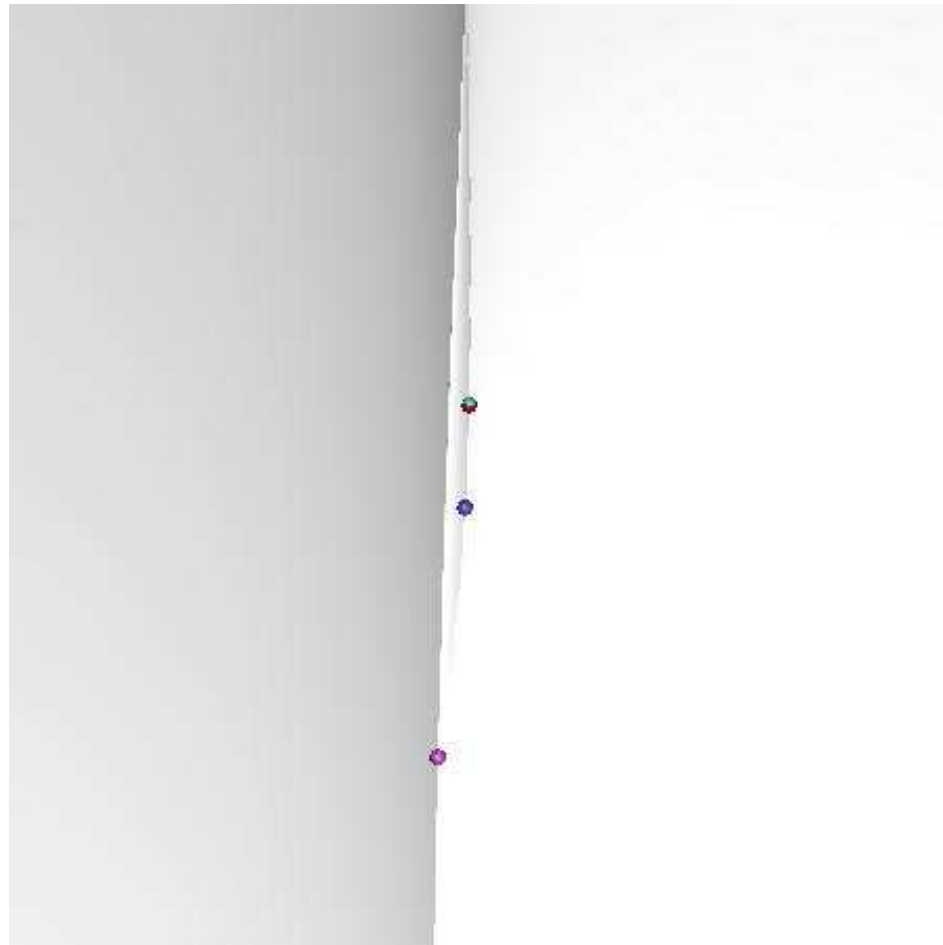


FIG. 8 – Extrema points on the surface 2

time(s)	u_1	v_1	l	k
ex 1	5.1	5.7	5.3	5.5
ex 2	2.5	1.9	2.4	2.2

extrema 1	u	v	u'	v'
u_1	0.625524	0.302895	0.273844	0.754153
	0.681597	0.0998794	0.166012	0.72042
v_1	0.595082	0.310081	0.27086	0.727953
	0.706713	0.0946811	0.168988	0.736224
l	0.574816	0.301324	0.260516	0.706247
	0.744083	0.113855	0.18959	0.771738
k	0.584561	0.308121	0.267303	0.71729
	0.73906	0.107815	0.184838	0.765172

SECOND APPROACH : AN ELIMINATION PROBLEM

In order to characterize the selfintersection curve, the algebraic question amounts to eliminate l and k in the system of 3 equations (S_1, S_2, S_3) .

Each equation S_i have multidegree $(3, 3, 2, 3)$ in (u_1, v_1, l, k) ; but only $k^3l^2, k^2l, l^2, k, l, 1$ do appear. We set :

$$T := m^2 * \text{subs}(l = \frac{1}{m}, S)$$

and get 3 polynomials with the 6 monomials :

$$k^3, k^2m, 1, km^2, m, m^2.$$

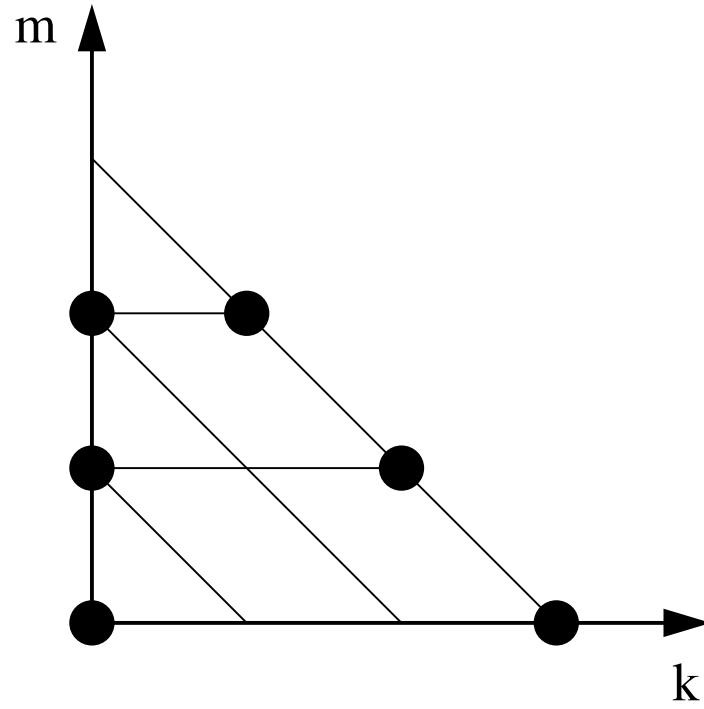


FIG. 9 – A very sparse support

BIVARIATE SPARSE RESULTANTS

- Several families of multivariate resultants exist.
- Some implementations are available as Maple packages
- multires, see the web page of galaad at INRIA.
- bires by A. Khetan, see his web page.
- At ISSAC'05 we played with our specific data and gave a recipe for making a “home-made resultant”.

Now, we turn back to the initial question. We let :

$$u = u_1 + l ; v = v_1 ; u' = u_1 ; v' = v_1 + h;$$

and consider the system of 3 equations :

$$\Phi_i(u_1 + l, v_1) = \Phi_i(u_1, v_1 + h) ; l \neq 0 ; h \neq 0,$$

where we want to eliminate l and h .

These 3 polynomials in l and h are with “separated variables”.

This is a special feature, well suited for expressing their Bezoutian as a sum of two simpler terms.

This leads to a better controlled computations.

We observe that this class of resultants, with “separated variables”, is also well suited for the study of intersections between two parametric varieties.

As far as we know, this has not been considered before.

We started, with L. Busé and M. Elkadi, to analyze it further.

A NICE BEZOUTIAN

Consider the system of 3 polynomials of bi-degree (m, n)

$$f_1(x) + g_1(y) ; f_2(x) + g_2(y) ; f_3(x) + g_3(y).$$

One checks that the Bezoutian $Bez(x, y, x_1, y_1)$ is equal to

$$\det\left(f(x) + g(y), \frac{f(x) - f(x_1)}{x - x_1}, \frac{g(y) - g(y_1)}{y - y_1}\right).$$

It can be written as the sum of two simpler terms.

It is symmetric in (x, y) and (x_1, y_1) and has multi-degree $(m - 1, n - 1, m - 1, n - 1)$.

By multi-linearity, the coefficients of the Bezoutian are sums of $(3, 3)$ minors of the matrix of coefficients of $f + g$.

$$P1 := a_3 * y^3 + a_2 * y^2 + a_1 * y + a_6 * x + a_5 * x^2 + a_4 * x^3;$$

$$P2 := b_3 * y^3 + b_2 * y^2 + b_1 * y + b_6 * x + b_5 * x^2 + b_4 * x^3;$$

$$P3 := c_3 * y^3 + c_2 * y^2 + c_1 * y + c_6 * x + c_5 * x^2 + c_4 * x^3.$$

$$M := matrix(3, 6, [a, b, c]) ; det(submatrix(M, (i, j, k))).$$

- Only 18 such minors do appear in the expression of the Bezoutian.

We denote them by e_i and e_{-i} , $i=1..9$, taking into account the symmetric relation between x and y .

- The Dixon resultant is the determinant of a $(8, 8)$ matrix.

It is a polynomial of total degree 24 in the coefficients of f and g . It is also a polynomial of degree 8 in the 18 variables e_i and e_{-i} ; with 1583 terms.

We get the following matrix, if we choose this ordered support :

$$[x^2, x, x^2 * y, x * y, x^2 * y^2, x * y^2, y, y^2].$$

$$\begin{bmatrix}
 -e_{-7} & -e_{-4} & -e_{-8} & -e_{-5} & -e_{-9} & -e_1 & 0 & 0 \\
 -e_{-4} & -e_{-3} & -e_{-5} & -e_{-2} & -e_1 & -e_{-6} & 0 & 0 \\
 -e_{-8} & -e_{-5} & -e_{-9} & -e_1 & 0 & 0 & -e_6 & -e_{-1} \\
 -e_{-5} & -e_{-2} & -e_1 & -e_{-6} - e_6 & 0 & -e_{-1} & -e_2 & -e_5 \\
 -e_{-9} & -e_1 & 0 & 0 & 0 & 0 & -e_{-1} & -e_9 \\
 -e_1 & -e_{-6} & 0 & -e_{-1} & 0 & -e_9 & -e_5 & -e_8 \\
 0 & 0 & -e_6 & -e_2 & -e_{-1} & -e_5 & -e_3 & -e_4 \\
 0 & 0 & -e_{-1} & -e_5 & -e_9 & -e_8 & -e_4 & -e_7
 \end{bmatrix}$$

(u, v) -STRUCTURES

In our initial problem, we have to substitute to the a_i, b_i, c_i the corresponding polynomials in (u, v) . The bi-degrees in (u, v) of the matrix M are the following :

$$\begin{bmatrix} [3, 2] & [3, 1] & [3, 0] & [0, 3] & [1, 3] & [2, 3] \\ [3, 2] & [3, 1] & [3, 0] & [0, 3] & [1, 3] & [2, 3] \\ [3, 2] & [3, 1] & [3, 0] & [0, 3] & [1, 3] & [2, 3] \end{bmatrix}$$

We also express the degrees in (u, v) of the entries e_i of the Bezoutian matrix.

We observed that, due to the relations between the coefficients of M , the partial and total degrees of the e_i are smaller than the sum of those of the corresponding columns.

The resultant is a polynomial of bi-degree $(44, 44)$ in (u, v) and total degree 64. Moreover we observed a regular stair-like pattern on the monomials.

$$\begin{bmatrix}
 [3, 8] & [4, 8] & [3, 7] & [4, 7] & [3, 6] & [4, 6] & 0 & 0 \\
 [4, 8] & [5, 8] & [4, 7] & [5, 7] & [4, 6] & [5, 6] & 0 & 0 \\
 [3, 7] & [4, 7] & [3, 6] & [4, 6] & 0 & 0 & [6, 5] & [6, 4] \\
 [4, 7] & [5, 7] & [4, 6] & [6, 6] & 0 & [6, 4] & [7, 5] & [7, 4] \\
 [3, 6] & [4, 6] & 0 & 0 & 0 & 0 & [6, 4] & [6, 3] \\
 [4, 6] & [5, 6] & 0 & [6, 4] & 0 & [6, 3] & [7, 4] & [7, 3] \\
 0 & 0 & [6, 5] & [7, 5] & [6, 4] & [7, 4] & [8, 5] & [8, 4] \\
 0 & 0 & [6, 4] & [7, 4] & [6, 3] & [7, 3] & [8, 4] & [8, 3]
 \end{bmatrix}$$

APPROXIMATE COMPUTATIONS

The previous structures are helpful, when dealing with floats.

If one expands (without care) the Bezoutian, the errors quickly propagates in the computation and the obtained result is completely false.

We propose to decompose the computation into several steps and at each step, correct the intermediate result in order to be coherent with the expected qualitative pattern.

The control of the relative sizes of the coefficients is crucial.

EXAMPLES WITH DOUBLE FLOATS

- An example with balanced random coefficients (ranging between 0.1 and 1), performed with double floats, gave a satisfactory answer with a relative error of about 10^{-4} . The coefficients of the obtained equation had a rather large dispersion in sizes : a factor about 10^9 .
- We slightly varied the sizes of the input coefficients (w.r.t. the monomial bases) by a factor $9 = 3 \times 3$: increasing or decreasing the middle terms.
- We also tried with random coefficients w.r.t. the Bernstein bases, and we first got ill conditioned behavior and completely false results.

EXAMPLE 1

Consider Example 1 (already computed via the numerical approach).

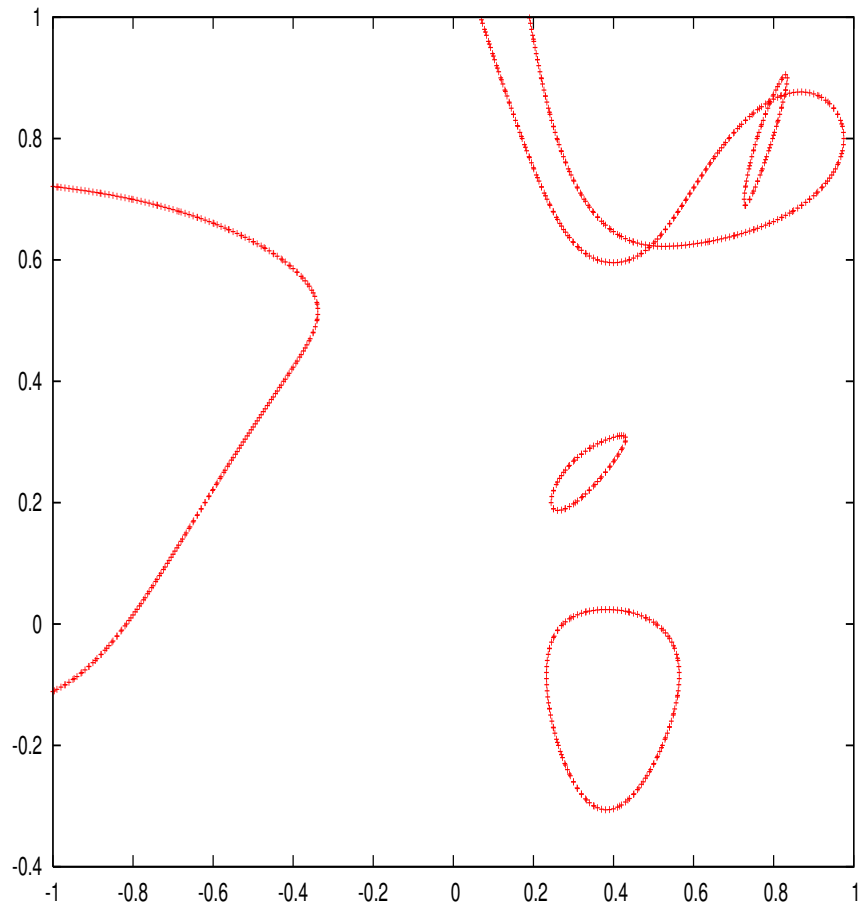
With double floats, I was only able to compute semi-numerically a resultant when a value of u (or v) is fixed. With 2 variables, the result is completely false : it does not commute with specialization.

So, I set the precision to `Digits :=20` (almost quadruple floats). I got a seemingly coherent result : a polynomial of degree 64 and bidegree (44, 44) with a large distribution of sizes ; roughly : 10^7 in the middle, then 10^2 and 1 on the sides and 10^{-7} in the corners.

The `implicitplot` function of Maple was unable to draw the graph. So I sent it to B. Mourrain who uses his library `Synaps/Axel`, he converted the coefficients of the matrix into integers and returned the following curve.

As I wanted the curve from the approximate implicit equation, I computed the traces of the curves on top of 100 values of u via `fsolve`. I did twice, first by specializing the matrix, second by specializing the computed determinant.

The first roughly drawn curve is coherent with my expectations, but not the second one.



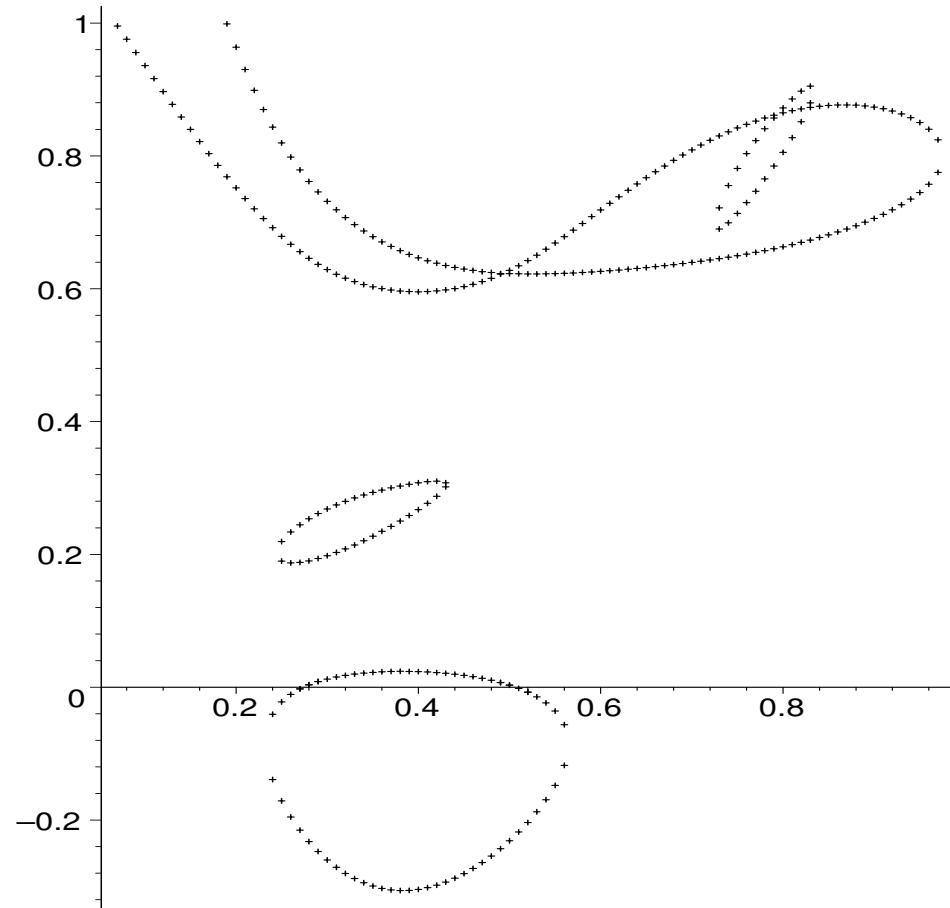


FIG. 10 – Specializing the matrix

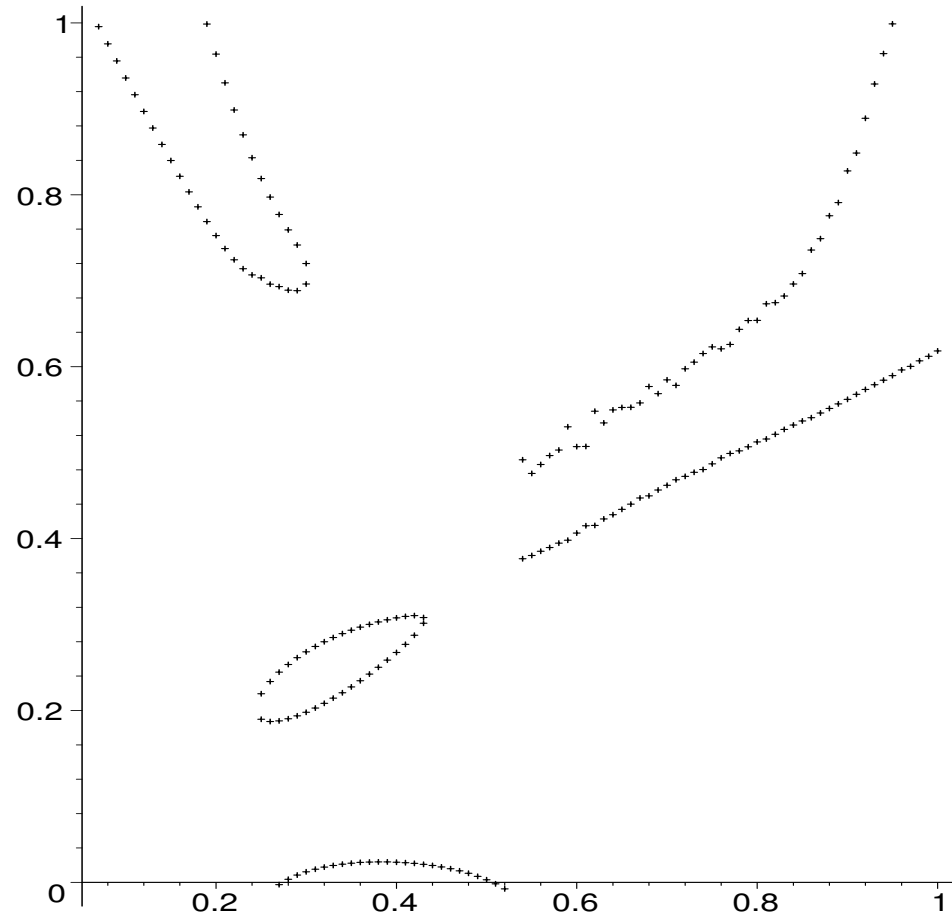


FIG. 11 – Specializing the determinant

I tried to understand where we lost precision. I compared the polynomials and the sets of points on top of $u = 0.5$

The difference between the coefficients of the 2 polynomials was 10^{-15} and the relative difference 10^{-11} , which is small. But the difference between 2 couples of corresponding roots was 0.2 and 0.4 which is big.

Factorizing these two polynomials, I observed that 25 factors were very similar and 2 of them were very different.

This is similar to what happens with the famous Wilkinson's example.

Increasing the precision to `Digits := 30`, the difference between the two families of roots becomes very small, and the two pictures agree.

$$\begin{aligned}
& 1.0094255734235075760 \times 10^{-15} (v + 0.23119152408721904179) \\
& (v - 0.0030587013400951569672) (v - 0.40599691283113654574) \\
& \quad (v - 0.82018335613907483724) \\
& (v^2 + 7.4136480812112394276 v + 14.010695808441447843) \\
& (v^2 + 7.3982764276379055418 v + 29.544367100779314078) \\
& (v^2 + 5.2546133227975999520 v + 8.1168143959950860322) \\
& (v^2 + 4.0729113341523143254 v + 22.750290248686183571) \\
& (v^2 + 3.9596731640113861330 v + 3.9200975737661407691) \\
& (v^2 + 0.58960021388733963644 v + 0.088035167372739133063) \\
& (v^2 - 0.14629194394175138277 v + 0.0053504451131050470583) \dots
\end{aligned}$$

CONCLUSION

- In theory, we should be able to check and improve the quality of the output of the numerical solver, by using our specific resultant.

But there is still work to be done in order to master the stability of the computation with double (or quadruple) floats. This numeric problem is tough but interesting.

- Our first tool is, for now, more satisfactory :
- The timings, we presented in section 8, are good and show that the program can be used in a C.A.D. system.
- It is not only efficient but also very robust as it delivers a solution together with a (small) bound on the possible error.

- These computations were also interesting for me, because they obliged me to look more closely on bivariate resultants.

Bivariate resultants deserve to be studied in more detail, not only from a pure algebraic point of view, but also from algorithmic and numerical points of view.

I think that the work initiated by Bini and co-workers on Bernstein/Bezout matrices for computing univariate resultants and subresultants can and should be extended to the bivariate case.