Resultant computation for polynomials in Bernstein form

L. Gemignani
University of Pisa
gemignan@dm.unipi.it
http://www.dm.unipi.it/~gemignan

Based partly on Joint Works with:
D. A. Bini, J. Winkler
Motivations and Basic Ideas

1. Fast linear solvers for structured matrices can be extended to work over an integral domain:
   (a) Equivalence of fast solvers and Gaussian elimination without pivoting
   (b) Fraction-free variants of Gaussian elimination

2. Computations with polynomials reduce to structured matrix problems and vice versa:
   (a) Equivalence of displacement rank representation and polynomial formulation
   (b) Efficient computation of resultants
Fields of Application

1. Computer Aided Geometric Design (CAGD) and Computer Graphics
2. Geometric and solid modeling
3. Computer Vision
4. Cryptography (algebraic coding and decoding methods)
5. Algebraic Geometry (systems of polynomial equations)
6. Robotics
7. Control and Linear System Theory (partial realizations)
8. Scalar and matrix interpolation problems (computation of matrix normal forms, factorization of linear differential operators, computation of recurrence relations)
Two Examples to Start

Compute the points of intersections of two planar Bézier or algebraic curves in parametric or implicit form

\[ F(x, y) = (x^2 - 2y)^2 + (x - y)^4 - 5 = 0; \]
\[ G(x, y) = x^3 - y^3 - 2x^2 + y^2 - 1 = 0 \]
Intersection of Bézier Curves

\[
\begin{align*}
x &= \frac{x_1(t)}{w_1(t)}, & y &= \frac{y_1(t)}{w_1(t)}, & x &= \frac{x_2(t)}{w_2(t)}, & y &= \frac{y_2(t)}{w_2(t)}\\
x_i(t), & y_i(t), & w_i(t) & \text{polynomials in the Bernstein basis of } \Pi_k\\
\{\beta_0^{(k)}(z), \ldots, \beta_k^{(k)}(z)\}, & \beta_i^{(k)}(z) = \binom{k}{i}(1 - z)^{k-i}z^i,\\
\end{align*}
\]

\[
\begin{align*}
x_1(t) &= 4\beta_0^{(3)}(t) + 10\beta_1^{(3)}(t) + 10\beta_2^{(3)}(t) + 6\beta_3^{(3)}(t), \\
y_1(t) &= \beta_0^{(3)}(t) + 12\beta_1^{(3)}(t) + +4\beta_3^{(3)}(t), \\
w_1(t) &= \beta_0^{(3)}(t) + 2\beta_1^{(3)}(t) + 2\beta_2^{(3)}(t) + \beta_3^{(3)}(t), \\
x_2(t) &= 7\beta_0^{(3)}(t) + 2\beta_1^{(3)}(t) + 18\beta_2^{(3)}(t) + 3\beta_3^{(3)}(t), \\
y_2(t) &= 4\beta_0^{(3)}(t) + 4\beta_1^{(3)}(t) + 4\beta_2^{(3)}(t) + +4\beta_3^{(3)}(t), & w_2(t) &= w_1(t)
\end{align*}
\]
Intersection of Bézier Curves

Two cubic Bézier curves (9 points of intersections at most from Bezout’s theorem)
Resultants

Curve intersection problems reduce to the question of whether two (bivariate) polynomials have a common root.

\[ f(x) = f_1 + f_2 x + f_3 x^2 f_4 x^3, \quad g(x) = g_1 + g_2 x + g_3 x^2 \]

\[
\begin{bmatrix}
p_1 & p_2 & q_1 & q_2 & q_3
\end{bmatrix}
\begin{bmatrix}
f_1 & f_2 & f_3 & f_4 & 0 \\
0 & f_1 & f_2 & f_3 & f_4 \\
g_1 & g_2 & g_3 & 0 & 0 \\
0 & g_1 & g_2 & g_3 & 0 \\
0 & 0 & g_1 & g_2 & g_3
\end{bmatrix} = 0^T
\]

\[ \iff p(x)f(x) + q(x)h(x) = 0 \iff \gcd(f, g) = r, \deg(r) > 0 \]

\[ S = S(f, g) \] Sylvester’s resultant matrix. When \( f_i, g_i \) are polynomials, its determinant \( \text{RES}(f, g) \) (resultant) is a polynomial
Coefficient Growth

Bézier intersection: \( \| \text{RES}(f, g) \|_\infty \approx 1995608380155750 \)

- Typically, problems suffer from *Coefficient Growth*
- For accuracy reasons, computations are performed using
  1. exact environments, like as Maple and Mathematica
  2. variable precision floating point domains (GNU multiprecision software library, ...)
- Even if we are implementing in these settings, coefficient growth is a serious drawback that limits the effectiveness of our algorithms because
  The cost of arithmetic operations depends on the size of components
Subresultant Theory

Thus, in order to compute resultants in these domains one must try for a low complexity algorithm while at the same time keeping the components of the arithmetic operations at a small size in an efficient manner.

For polynomials in power form, this can done by using Subresultant Theory [Collins, Brown & Traub, Loos]. Drawbacks:

1. Derivation is involved since relies upon determinantal relations
2. No easy extension to polynomials expressed in other bases. Explicit conversions to the power form are ill-conditioned and, then a source of coefficient growth.
Bezout(ian) Matrices

\[ f(x) = f_1 + f_2 x + f_3 x^2 f_4 x^3, \quad g(x) = g_1 + g_2 x + g_3 x^2 \]

\[ h(x, y) = \frac{f(x)g(y) - g(x)f(y)}{x - y} = \sum_{i,j=1}^{3} b_{i,j} x^{i-1} y^{j-1}, \]

\[ \mathcal{B}(f, g) = (b_{i,j}) \text{ Bezoutian Matrix} \]

- \( \mathcal{B}(f, g) \) is symmetric \( \Rightarrow \) Inertia computation (Sylvester’s law of inertia) \( \Rightarrow \) Useful for solving Root-counting problems [Kailath, Bistritz & Lev-Ari, Heinig & Rost]

- The definition of \( h(x, y) \) does not depend on the basis used to represent \( f(x) \) and \( g(x) \) \( \Rightarrow \) Extensions to Chebyshev, Bernstein, ..., polynomials \( \Rightarrow \) Vandermonde-like solvers [Gohberg & Olshevsky], Applications to CAGD
Bezout, Toeplitz, Hankel...

Fast linear solvers for Bezoutian matrices translate into fast solvers for several other classes of structured matrices

\[ H = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_2 & h_3 & h_4 \\ h_3 & h_4 & h_5 \end{bmatrix}, \quad JH = T = \begin{bmatrix} h_3 & h_4 & h_5 \\ h_2 & h_3 & h_4 \\ h_1 & h_2 & h_3 \end{bmatrix} \]

Embedding technique [G. LAA, 1997]:

\[ \mathcal{B} = \begin{bmatrix} H & L \\ L^T & 0 \end{bmatrix}, \quad L \text{ triangular and easily computable} \]

\[ -L^T H^{-1} L \text{ is a Schur complement of } \mathcal{B} \]

\[ \text{Computing the sequence of Schur complements provides the inverse of } H \]
Bezout Resultants

The determinant of $\mathcal{B}(f, g)$ is zero if and only if $f(x)$ and $g(x)$ have a common root

Barnett Factorizations:

$$\mathcal{S}(f, g) = \begin{bmatrix} J & 0 \\ -\hat{T}(g) & T(f) \end{bmatrix}^{-1} \begin{bmatrix} I & 0 \\ 0 & \mathcal{B}(f, g) \end{bmatrix} \begin{bmatrix} \hat{T}(f) & T(f) \\ I & 0 \end{bmatrix}$$

$$\mathcal{B}(f, g) = \mathcal{B}(f, 1)g(C_f), \quad C_f \text{ Frobenius}$$

- $\det \mathcal{S}(f, g) \simeq \det \mathcal{B}(f, g)$ (differ from a scalar multiple)

- For computations with univariate polynomials $g(C_f)$ is OK. Difficulties arise for polynomials over integral domains (bivariate polynomials) since $C_f$ has entries over the quotient field.
Schur Complements of Bezoutians

Schur complements of Bezoutian inherit the same structure

**Th. 1** [Bini & G., SICOMP, 1995]:

Assume that \( \mathcal{B}_0 = \mathcal{B}(f, g) \) is strongly nonsingular. For \( 0 \leq k \leq n - 1 \) let \( \mathcal{B}_k = (b_{i,j}^{(k)}) \in \mathbb{F}^{(n-k) \times (n-k)} \) be the Schur complement of the leading principal submatrix of order \( k \) of \( \mathcal{B}_0 \), i.e.,

\[
\mathcal{B}_k = \mathcal{B}_0[k+1:n] - \mathcal{B}_0[k+1:n, 1:k](\mathcal{B}_0[1:k])^{-1}\mathcal{B}_0[1:k, k+1:n].
\]

Moreover, define \( t_k(z) = \sum_{i=1}^{n-k} b_{1,i}^{(k)} z^{i-1} \).

Then we have

\[
\mathcal{B}_k = \mathcal{B}(t_{k-1}, z \cdot t_k), \quad 1 \leq k \leq n - 1.
\]
Manifestations of the Structure

Bezoutian matrices have a displacement structure:

$$ZB(f, g) - B(f, g)Z^T = uv^T - vu^T, \ Z \text{ down} - \text{shift}.$$ 

Schur complements inherit the same displacement structure. Previous Theorem ⇒ easy computation of the generators \((u_k, v_k)\)

It is found that

$$t_{k-1}(z) = (\alpha_k z + \beta_k) t_k(z) + \gamma_k z^2 t_{k+1}(z), \quad 1 \leq k \leq n - 1.$$ 

In a polynomial setting, the computation of the sequence of Schur complements relates to the computation of the Euclidean scheme applied to the reverse polynomials.
Bezoutian over Integral Domains

Fast linear solvers and triangularization methods at the cost of $O(n^2)$ ops using repeated Schur complement operations

\[ F(x, y) = (x^2 - 2y)^2 + (x - y)^4 - 5 = 0; \]
\[ F(x, y) = f_1 + \ldots + f_5 x^4 \]
\[ G(x, y) = x^3 - y^3 - 2x^2 + y^2 - 1 = 0 \]
\[ G(x, y) = g_1 + \ldots + g_4 x^3 \]

\[ \mathcal{B}(F, G) = (b_{i,j}(y)) \in \mathbb{F}^{4 \times 4}, \quad \mathbb{F} = \mathbb{R}[y] \]

- Schur complements have entries over the quotient field (rational functions). To control coefficient growth: GCD computations but they are EXPENSIVE

- Differently, try to perform ring operations and exact divisions only \( \Rightarrow \) Exploit the equivalence between Gaussian Elimination and Schur Complementation and use fraction-free variants of the former algorithm
Fraction-free Gaussian Elimination

\[ \mathcal{B}(F, G) = (b_{i,j}^{(0)}(y)) = (\hat{b}_{i,j}^{(0)}(y)) \in \mathbb{F}^{n \times n} \text{ strongly nonsingular} \]

Bareiss’ variant of Gaussian Elimination:

\[
\hat{b}_{0,0}^{(-1)}(y) = 1 \\
\text{for } k = 1: n - 1 \\
\text{for } i = k + 1: n \\
\text{for } j = k + 1: n \\
\hat{b}_{i,j}^{(k)}(y) = \frac{\hat{b}_{k,k}^{(k-1)}(y)\hat{b}_{i,j}^{(k-1)}(y) - \hat{b}_{i,k}^{(k-1)}(y)\hat{b}_{k,j}^{(k-1)}(y)}{\hat{b}_{k-1,k-1}^{(k-2)}(y)} \\
\text{end} \\
\text{end} \\
\text{end}
Computational Aspects

- The divisions required are always exact so that all the computations remain over the integral domain.

- $\hat{b}^{(n-1)}_{n,n}(y)$ is proportional to $\text{RES}(F, G)$.

- If $\deg(f_i), \deg(g_i) \leq m$, $\hat{b}^{(k)}_{i,j}(y)$ are polynomials of degree at most $2(k + 1)m$, then the cost of Bareiss’ variant is $O((n - k)^2 \mu(2km))$ ops per step, where $\mu(k)$ is an upper bound of the arithmetic cost of multiplying and dividing polynomials in power form of degree at most $k$.

- The overall cost is $O(n^3 \mu(2nm))$ ops.

- Block variants for dealing with singular submatrices (but complexity estimates get worse).
Structured Gaussian Elimination

Combine fraction-free variants with the structured properties of Schur complements

Th. 2 [Bini & G., LAA, 1998]:

For $0 \leq k \leq n - 1$ let $\mathcal{B}_k = (b^{(k)}_{i,j}(y))$ be the Schur complement of the leading principal submatrix of order $k$ of $\mathcal{B}(F, G)$. Moreover, let $\mathcal{G}_k = (\hat{b}^{(k)}_{i+k,j+k}(y))$ the matrix generated at the $k$th step by the fraction-free variant applied to $\mathcal{B}(F, G)$. Finally, denote $\hat{t}_k(y, z) = \sum_{i=1}^{n-k} \hat{b}^{(k)}_{1+k,i+k}(y) z^{i-1}$ the bivariate polynomial formed from the entries in the first row of $\mathcal{G}_k$. Then we find that

$$\hat{b}^{(k-1)}_{k,k}(y) \mathcal{B}_k = \mathcal{G}_k \simeq \frac{1}{\hat{b}^{(k-1)}_{k,k}(y)} \mathcal{B}(\hat{t}_{k-1}, z \cdot \hat{t}_k), \quad 1 \leq k \leq n - 1.$$
Thus, in order to compute resultants in these domains one must try for a low complexity algorithm while at the same time keeping the components of the arithmetic operations at a small size in an efficient manner.

1. **Low complexity factorization algorithm**: Use of the Structure ⇒ Overall cost of $O(n^3 \mu(2nm))$ ops (one order of magnitude less than the customary approach)

2. **Coefficient Growth**: Use the fraction-free algorithm ⇒ The degree of polynomials involved are as small as possible (Cramer’s rule); Under reasonable assumptions, the length of coefficients is bounded by $O(n \log n)$ (Cramer’s rule + Hadamard inequality)
Bernstein-Bezoutian Matrices

\[ f(x) = f_1 \beta_0^{(4)}(x) + \ldots + f_5 \beta_4^{(4)}(x); \quad g(x) = g_1 \beta_0^{(4)}(x) + \ldots \]

\[ f_i = \sum_{j=0}^{m} p_j^{(i)} \beta_j^{(m)}(y); \quad g_i = \sum_{j=0}^{m} q_j^{(i)} \beta_j^{(m)}(y) \]

\[ h(x, y) = \frac{f(x)g(y) - g(x)f(y)}{x - y} = \sum_{i,j=1}^{3} b_{i,j} \beta_{i-1}^{(3)}(x) \beta_{j-1}^{(3)}(y), \]

\[ \mathcal{BB}(f, g) = (b_{i,j}) \text{ Bernstein -- Bezoutian Matrix} \]

\[
\begin{align*}
    b_{i,1} &= \frac{n}{i} (f_{i+1}g_1 - f_1g_{i+1}), \\
    b_{i,j+1} &= \frac{n^2 (f_{i+1}g_{j+1} - f_{j+1}g_{i+1})}{i(n-j)} + \frac{j(n-i)}{i(n-j)} b_{i+1,j}, \\
    b_{n,j+1} &= \frac{n}{n-j} (f_{n+1}g_{j+1} - f_{j+1}g_{n+1}).
\end{align*}
\]
Congruence Relation

\( \mathcal{B} \mathcal{B} \) matrices are congruent to classical \( \mathcal{B} \)

\[
T_{n-1} \begin{bmatrix}
\beta_0^{(n-1)}(\tilde{z}) \\
\vdots \\
\beta_{n-1}^{(n-1)}(\tilde{z})
\end{bmatrix} = \begin{bmatrix}
1 \\
\vdots \\
\tilde{z}^{n-1}
\end{bmatrix}, \quad t_{i,j}^{(n-1)} = \begin{cases} 
0 & \text{if } i > j; \\
\binom{j-1}{i-1}^{-1}(n-1)^{-1} & \text{if } i \leq j,
\end{cases}
\]

\[
\mathcal{B} \mathcal{B}(f, g) = T_{n-1}^T \mathcal{B}(f, g) T_{n-1}
\]

- \( \det \mathcal{B} \mathcal{B}(f, g) \) is proportional to the resultant of \( f(x) \) and \( g(x) \)

- The triangular factorization of \( \mathcal{B} \) provides a triangular factorization of \( \mathcal{B} \mathcal{B} \)

- \( \mathcal{B} \mathcal{B} \) and \( \mathcal{B} \) have the same inertia
Th. 3 [Bini & G., TCS, 2003; Bini & G. & Winkler, 2003]:

For $0 \leq k \leq n - 1$ let $\mathcal{B}_k = (b_{i,j}^{(k)}(y))$ be the Schur complement of the leading principal submatrix of order $k$ of $\mathcal{B}\mathcal{B}(F, G)$. Moreover, let $\mathcal{G}_k = (\hat{b}_{i+k,j+k}^{(k)}(y))$ the matrix generated at the $k$th step by the fraction-free variant applied to $\mathcal{B}\mathcal{B}(F, G)$. Finally, denote $\hat{t}_k(y, z) = \sum_{i=1}^{n-k} \hat{b}_{1+k,i+k}^{(k)}(y) z^{i-1}$ the bivariate polynomial formed from the entries in the first row of $\mathcal{G}_k$. We find that

$$\hat{b}_{k,k}^{(k-1)}(y) \mathcal{B}_k = \mathcal{G}_k \simeq \frac{1}{\hat{b}_{k,k}^{(k-1)}(y)} \mathcal{B}\mathcal{B}(z \cdot t_{k-1}, z \cdot t_k)[k + 1 : n].$$

Shift acts on both polynomials.
Fast Factorization of $BB$ Matrices

% Initialization phase: Given the coefficients of $F$ and $G$,
% compute the entries on the first and second row of $BB(F, G)$

\[
\text{for } k = 1: n - 1
\]
% Compute a vector proportional to the first row
% of $G_k$ by using the fraction-free variant

% Compute the first two rows of a matrix proportional to $BB(z \cdot \hat{t}_{k-1}, z \cdot \hat{t}_k)$

% eliminate the spurious factor $\hat{b}_{k,k}^{(k-1)}(y)$

end

**Output:** an upper triangular matrix $\mathcal{T} = (t_{i,j}(y))$ obtained from $BB(F, G)$ by row operations only
Computational Aspects

- $t_{n,n}(y)$ is a polynomial in Bernstein form that is proportional to the resultant of $F(x, y)$ and $G(x, y)$. A polynomial root-finder should exploit the stability features of Bernstein representation.

- No explicit conversion to the power basis. Factorization Process: Use BB matrices. Inner operations: Use programs working with the genuine Bernstein basis $\Rightarrow$ C++ BPOLY library [Tsai & Farauki]

- The algorithm requires $O(n^2)$ multiplications/divisions between polynomials in Bernstein form of degree at most $O(nm)$, where $m$ is the maximum degree of the input coefficients $f_i$ and $g_i$, $0 \leq i \leq n$. 
Computations with Bernstein polynomials involve binomial coefficients. To avoid overflow problems and minimize roundoff errors: Represent binomial coefficients by their factorization into primes.

1. Arithmetic operations:
   (a) + and – easy to compute but the cost is $O(n(n - m))$
   (b) $\times$ requires $O(nm)$ ops
   (c) $\div$ EXPENSIVE. BPOLY is not optimized for performing exact divisions (the remainder is known to be the zero polynomial). The unknowns are $n + 1$ and the coefficient of the quotient are found by Gaussian elimination with partial pivoting at the cost of $O(n^3)$ ops.

2. Zero finder: Bisection scheme + Newton’s iteration
A Numerical Intersection Algorithm

Bézier intersection problem reduces to

\[
\begin{align*}
x_1(t)w_2(s) - w_1(t)x_2(s) &= p(s, t) = \sum_{i=0}^{n} p_i(s) \beta_i^{(n)}(t) = 0 \\
y_1(t)w_2(s) - w_1(t)y_2(s) &= q(s, t) = \sum_{i=0}^{n} q_i(s) \beta_i^{(n)}(t) = 0,
\end{align*}
\]

1. Given the coefficients \( p_i(s) \) and \( q_i(s) \) in Bernstein form, \( 0 \leq i \leq n \), apply the fast implementation of Bareiss’ variant to compute a polynomial \( f(s) \) in Bernstein form that is proportional to the resultant of \( p(s, t) \) and \( q(s, t) \).

2. Compute the roots of \( f(s) = 0 \) lying in the interval \([0, 1]\). These provide the parameter values of the points of intersection. The \((x, y)\) coordinates of those intersection points can be easily found from the parametric equation of the second curve.
Example I

\[ \| \text{RES}(f, g) \|_\infty \approx 1995608380155750; \quad \| f(s) \|_\infty \approx 3583961 \]

Root 1 (multiplicity 1): 0.938910, Root 2 (multiplicity 1): 0.133017, Root 3 (multiplicity 1): 0.348987, Root 4 (multiplicity 1): 0.594400, Root 5 (multiplicity 1): 0.088811, Root 6 (multiplicity 1): 0.036880, Root 7 (multiplicity 1): 0.533925, Root 8 (multiplicity 1): 0.846324, Root 9 (multiplicity 1): 0.921891, Total of 9 distinct roots are found.

![Graph](image)
Example II

\[ \| \text{RES}(f, g) \|_\infty \approx 2985077007967404; \quad \| f(s) \|_\infty \approx 326024684 \]

Root 1 (multiplicity 1): 0.041840, Root 2 (multiplicity 1): 0.194090, Root 3 (multiplicity 1): 0.123797,
Total of 3 distinct roots are found.
Example III

\[ \| \text{RES}(f, g) \|_\infty \approx 3338740615963116116507680659954176 \]
\[ \| f(s) \|_\infty \approx 2957839441644 \]

Root 1 (multiplicity 1): 0.048012, Root 2 (multiplicity 1): 0.028526,
Total of 2 distinct roots are found.
Further Developments

The design of a multi-precision software package for solving intersection problems for Bernstein-Bézier curves (and surfaces)

1. A numerical multi-precision library for Bernstein polynomial arithmetic: Adapting BPOLY to a multi-precision environment
2. Rounding error analysis for arithmetic operations between Bernstein polynomials
3. Estimates on the conditioning of the intersection problems via reduction to an eigenvalue problem plus standard perturbation theory for eigenvalues
4. A multi-precision polynomial root-finding algorithm for Bernstein polynomials
Conclusion

The exploitation of the structured properties of Bezoutians makes possible the design of fast solvers for other classes of structured matrices in a very transparent and elegant way.

These properties can be combined with fraction-free variants of Gaussian elimination for the solution of structured problems over integral domains.

The interplay between structured matrices and polynomials is made evident. This enables polynomial computations to be reduced to structured matrix problems and vice-versa.

In my opinion, adaptive solution methods for algebraic geometry and computer algebra problems based on the use of a mixed numeric-symbolic approach will become more and more important in the next years.