Selecting the roots of a small system of polynomial equations by tolerance based matching

H. Bekker*, E. P. Braad*, B. Goldengorin**

{*Department of Mathematics and Computing Science, **Faculty of Economic Sciences}, University of Groningen, P.O.Box 800, 9700 AV Groningen, The Netherlands. Email: bekker@cs.rug.nl, e.p.braad@wing.rug.nl, b.goldengorin@eco.rug.nl

Abstract

The roots of a system of two bivariate polynomial equations are calculated using a two-step method. First all x-roots and y-roots are determined independently. Then tolerance based weighted matching is used to form (x, y)-pairs that together form a minimum-error solution to the system.

Key words: combinatorial optimization, tolerance based bipartite matching, solving polynomial equations

1 Introduction

Consider a system of two polynomial equations

$$f(x, y) = 0 \quad g(x, y) = 0 \tag{1}$$

with symbolic constants and of low degree. By assigning numerical values to the constants we obtain a problem instance. Assuming that it is known that (1) has a finite number of solutions the conventional method to calculate the solutions is as follows. From (1) a univariate polynomial, say $p(x)$, is derived by eliminating y. For every problem instance the symbolic constants in $p(x) = 0$, $f(x, y) = 0$ and $g(x, y) = 0$ are replaced by numerical values and $p(x) = 0$ is solved numerically giving the roots $x_1, \ldots, x_n$. Subsequently, for every root x_i the corresponding root y_i has to be determined. To that end, x_i is backsubstituted in $f(x, y) = 0$ and $g(x, y) = 0$, giving the univariate polynomial equations $f(x_i, y) = 0$ and $g(x_i, y) = 0$. Solving $f(x_i, y) = 0$ for y gives the solutions $y_{f_1}, \ldots, y_{f_l}$, and solving $g(x_i, y) = 0$ for y gives the solutions $y_{g_1}, \ldots, y_{g_m}$. The value y_i occurring both in $y_{f_1}, \ldots, y_{f_m}$ and $y_{g_1}, \ldots, y_{g_m}$ is the desired value, i.e., the pair x_i, y_i is a root of (1). During this process, a number of complications may occur:

1. The equation $f(x_i, y) = 0$ may be degenerate, i.e. may be $0 = 0$, or even worse, may be near degenerate within the noise margin. The case of exact degeneracy is easily detected but it is not trivial to detect near degeneracy. In both cases every solution of the other equation, that is, of $g(x_i, y) = 0$ is

a correct root. Analogously, $g(x_i, y) = 0$ may be degenerate, giving the same problems. As we know that (1) has a finite number of solutions the situation that $f(x_i, y) = 0$ and $g(x_i, y) = 0$ are both degenerate will not occur.

2. It is sometimes hard to select from $y_{f_1}, \ldots, y_{f_l}$ and $y_{g_1}, \ldots, y_{g_m}$ the collective value y_i because, by numerical errors, the actual value of y_i will be different in the two sets.
3. $p(x) = 0$ may have multiple roots, that is, the roots $x_1, \ldots, x_n$ may contain (near) identical values. Let us assume that there is a double root, given by the the identical values x_i and $x_{i'}$. Then there will be two matching roots y_j and $y_{j'}$, not necessarily with the same value. When y_j is matched to x_i, in a later stage $y_{j'}$ should be matched to $x_{i'}$ and not to x_i.

When solving a problem of computational geometry we ran into these problems, first using our own multivariate polynomial solver and later using methods from packages. As a result a small but significant part of the roots, notably multiple roots, were missed or were completely wrong.

2 The CORS method

To avoid the aforementioned complications we propose and test a two-step method, called the CORS method (Combinatorial Optimization Root Selection). First from (1) two univariate polynomials $p(x)$ and $q(y)$ are derived by eliminating y and x respectively. Whether this is done by calculating resultants or a Groebner basis is irrelevant. Now for every problem instance the symbolic constants in $p(x)$, $q(y)$, $f(x, y)$ and $g(x, y)$ are replaced by numerical values and the roots in $\mathbb{C}$ of $p(x)$ and $q(y)$ are calculated numerically. Both $p(x)$ and $q(y)$ have n roots represented by $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$, respectively. These roots are used to calculate n^2 weights, where $w_{i,j}$ is defined as

$$w(i, j) = \sqrt{(f(x_i, y_j))^2 + (g(x_i, y_j))^2}. \tag{2}$$

Subsequently a complete weighted bipartite graph $G(V, E)$ is constructed with $V = X \cup Y$ and $|X| = |Y| = n$. The nodes in X consist of the values $x_1, \ldots, x_n$, and the nodes in Y consist of the values $y_1, \ldots, y_n$. The arc between nodes x_i and y_j is assigned the weight $w(i, j)$. On G the minimum-weight perfect matching π_0 is calculated. The n arcs in π_0 represent the optimal solutions of (1). Here, optimal means that the sum of the errors is minimal. In the following this method of roots selection is called CORS1.

Instead of minimizing the sum of the errors it is more natural to minimize the maximum error. This is done as follows. All n^2 arcs and their weights are stored in a linear list L. Subsequently, L is sorted in increasing order of weights. Now the weight in the first entry in L is set to 1, and the weight of item i is equal to the sum of the weights in previous items plus one, i.e. $weight[i] = (\sum_{j=1}^{i-1} weight[j]) + 1$. Thus the weights are $1, 2, 4, 8, 16, \ldots$. A new graph G' is constructed, identical to G but with the new weights. Of G' the minimum-weight perfect matching π_0 is calculated. The n arcs in π_0 represent the optimal solutions of (1). Here, optimal means that the maximum error of π_0 is minimal. We call this method CORS2.

We here outline the three steps of a proof that this procedure minimizes the maximum weight when no identical weights in G occur, without this assumption the proof is similar but more complex.
1: π_0 is unique because the total weight of π_0 can be constructed only in one way from the weights in G'.
2: In π_0 there is only one element e_m with the maximum weight.
3: There is no matching of G' without e_m, with a lower weight. q.e.d.
The weights in G' become very large causing overflow on standard integer arithmetic. Therefore the infinite precision integer type should be used. The weights in G' have the nice property that none of the weights can be constructed from other weights. This makes G' very suitable for tolerance based matching.

3 Tolerance based matching

A *Feasible Assignment (matching, permutation)* (FA) π on the bipartite graph G' is a mapping π of X *onto* Y with $w(\pi) = \sum_{(i,j)\in\pi} w(i,j) < \infty$ and the set of all FAs is Π. The Linear Assignment Problem (LAP) is the problem of finding a FA $\pi_0 \in \arg\min\{w(\pi) : \pi \in \Pi\}$, and all algorithms are based on shortest paths and the König-Egervary's theorem with $O(n^3)$ time complexity when applied to dense instances [1]. We sketch the idea of algorithms which are based only on tolerances for the Relaxed LAP (RLAP) without using the König-Egervary's theorem. A *Relaxed FA* (RFA) θ is defined on the same graph G' as a mapping θ of X *into* Y with $w(\theta) = \sum_{(i,j)\in\theta} w(i,j) < \infty$. The RLAP is the problem of finding $\min\{w(\theta) : \theta \in \Theta\} = \sum_{i\in X} \min\{w(i,j) : j \in Y\} = w(\theta_0) \leq w(\pi_0)$ on the set of RFA $\Theta \supset \Pi$. A FA π on G' is a set of n arcs (i,j) such that the out-degree $od(i) = 1$ for all $i \in X$ and the in-degree $id(j) = 1$ for all $j \in Y$, and a RFA θ is a set of n arcs (i,j) with $od(i) = 1$ for all $i \in X$ and $\sum_{j\in Y} id(j) = n$. Note that θ is a FA if the $id(j) = 1$ for all $j \in Y$. For each fixed row i of the matrix $W = ||w(i,j)||$ let $w[i,j_1(i)] \leq w[i,j_2(i)] \leq \ldots \leq w[i,j_n(i)]$ be the ordered set of entries in a non-decreasing order. We define the reduced matrix $W^r = ||w^r(i,j)||$ with $w^r(i,j) = w(i,j) - w[i,j_1(i)]$ for all $i \in X$ and $j \in Y$. The *tolerance problem* for the RLAP is the problem of finding for each arc $(i,j) \in X \times Y$ the maximum decrease $l(i,j)$ and the maximum increase $u(i,j)$ of the arc weight $w(i,j)$ preserving the optimality of θ_0 under the assumption that the weights of all other arcs remain unchanged. Now for an arc $[i,j_1(i)] \in \theta_0$ the *upper tolerance* $u[i,j_1(i)] = w[i,j_2(i)]$, and the *lower tolerance* $l[i,j_1(i)] = \infty$. Similarly, for an arc $(i,j) \notin \theta_0$ the *lower tolerance* $l(i,j) = w^r(i,j)$ and the *upper tolerance* $u(i,j) = \infty$. Let us show that the *bottleneck tolerance* $b(\theta_0) = \max\{u(\theta_0),\, l(\theta_0)\}$ is a *tightness measure* between known value of $w(\theta_0)$ and the unknown value of $w(\pi_0)$. For a fixed θ_0 we partition the set Y into three subsets of vertices: the *unassigned set* $Y_0 = \{j \in Y : id(j) = 0\}$, *assigned set* $Y_1 = \{j \in Y : id(j) = 1\}$, and *overassigned set* $Y_2 = \{j \in Y : id(j) > 1\}$. For each fixed $j \in Y_2$ we order the corresponding upper tolerances in non-decreasing order $u[i_1(j),j] \leq u[i_2(j),j] \leq$

... $\leq u[i_{p_j}(j), j]$ and compute $u(\theta_0) = \sum_{j\in Y_2} u(j)$ with $u(j) = \sum_{t=1}^{p_j-1} u[i_t(j), j]$. Similarly, for each fixed $j \in V_0$, $l[i(j), j] = \min\{l(i,j) : i \in X\}$, $l[Y_0(j)] = \max\{l[i(t), t] : t \in Y_0(j)\}$ and $l(\theta_0) = \sum_{j=1}^{k} l[Y_0(j)]$ with $Y_0(j) = \{t \in Y_0 : i(t) = i(j)\}$. Here, $Y_0(1), \ldots, Y_0(k)$ is a partition of Y_0. Further we treat each π, and each θ as the sets of corresponding arcs such that $|\pi| = |\theta| = n$. Note that if either $Y_0 = \emptyset$ or $Y_2 = \emptyset$ then $|Y_1| = n$ and θ_0 is a FA. Hence, for each $\theta_0 \notin \Pi$ we may use the number of unassigned columns $|Y_0| = \sum_{j\in Y_2} |id(j) - 1|$ in the reduced matrix W^r as a *measure of structural infeasibility* of θ_0 to the LAP, for which the bottleneck tolerance $b(\theta_0) \leq w(\pi_0) - w(\theta_0)$. Our algorithm for solving the LAP recursively fixes the arc $(i,j) \in \theta_0$ with the largest tolerance and replaces all other arcs from Y_2 by the arcs representing the tolerances ordered in a non-increasing order, regardless of either upper or lower tolerance will be the next tolerance induced by that order. Therefore, the first obtained $\theta \in \Pi$ is $\theta = \pi_0$, and hence the time complexity of LAP for CORS2 is $O(n^2)$.

4 Implementation, tests and results

Implementation We tested CORS on our computational geometry problem. Of this class of problems it is known that every instance has eight solutions. The univariate polynomials $p(u)$ and $q(w)$ are derived with MAPLE. The numerical calculations are implemented in C++ in double precision. Laguerre's method [2] is used to compute the roots of the polynomials $p(u)$ and $q(w)$. The LEDA [3] implementation of the minimum weight bipartite matching algorithm is used.
Tests We tested the CORS1 and CORS2 method. Every problem instance is solved in two ways: with the CORS method and with SYNAPS, a C++ package for solving polynomial equations [4]. We solved 10^4 problem instances with CORS and SYNAPS, and ≈ 400 with MAPLE. The latter problem instances were solved correctly by CORS and were missed by SYNAPS, i.e. we use MAPLE to decide whether CORS or SYNAPS gave the correct result.
Results In general the results of CORS1 and CORS2 are identical. In the tests approximately 2.4% of the solutions is missed by SYNAPS and are found by CORS. No solutions were missed by CORS. The average error of the solutions found by SYNAPS is 1.3 10^{-10} and of CORS 6.5 10^{-11}. Running 10^5 problem instances with CORS takes 14 *sec.* and with SYNAPS 475 *sec.*

Literature

[1] Burkard, R. E. *Selected topics on assignment problems.* Discrete Applied Mathematics 123, 257–302, 2002.
[2] Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. *Numerical Recipes in C++.* Cambr. Univ. Press, New York.
[3] K. Melhorn, Näher, S. *LEDA A Platform for Combinatorial and Geometric Computing.* Cambridge University press,Cambridge. 1999
[4] Synaps. Available at: http://www.inria.fr/galaad/logiciels/synaps/inex.html