Design Laboratory Geometric Modeling Libraries Reference Manual

—

Volume 9: simp and lex Simultaneous Polynomial Intersection Library and Lexical Library

E. C. Sherbrooke T. Maekawa C.-Y. Hu W. Cho J. Zhou
S. L. Abrams N. M. Patrikalakis

Massachusetts Institute of Technology
Department of Ocean Engineering
77 Massachusetts Avenue
Cambridge, MA 02139-4307, USA

Design Laboratory Memorandum 95-24
Issued: November 11, 1995
Revised: November 9, 2007

1 Purpose

Simultaneous Polynomial Intersection Library.

2 Specification

#include "simpoly.h"
*libfsimp.a (*or *libIsimp.a)*

3 Description

This library is used for solving systems of non-linear polynomial equations. The current version can solve overconstrained, underconstrained, and balanced systems. In other words, for m equations and n unknowns, systems where $m > n$ (overconstrained), $m = n$ (balanced), and $m < n$ (underconstrained) can be solved.

The initial implementation of the solver is described by Sherbrooke and Patrikalakis [9]. This paper develops the basic n-dimensional algorithm (in floating point and rational arithmetic) for finding the real roots of a balanced non-linear polynomial system in the Bernstein basis in a n-dimensional rectangular box. Floating point arithmetic is sufficient for well-conditioned cases, but in general it is not numerically robust for ill-conditioned cases. Rational arithmetic solves the robustness problem; nevertheless, it is ineffective even for moderate size problems [8]. To overcome the lack of numerical robustness in ill-conditioned cases, Maekawa and Patrikalakis [4, 5] introduced rounded interval arithmetic (RIA) for increased robustness without the cost of rational arithmetic. The formulation of equations in rational arithmetic and solution in RIA is appropriate, robust and practically feasible. Maekawa and Patrikalakis [6] applied this algorithm to very high degree multidimensional problems (systems of degree 50 to 68 in dimensions of 1 to 4) with emphasis on manufacturing and fairing. Zhou, Sherbrooke, and Patrikalakis [10] apply the algorithm to distance functions of different geometrical entities.

To improve the efficiency of the rounded interval arithmetic version, Hu [1] developed a method to increase the efficiency of the rounded interval arithmetic operators by a factor of 3–to–1. This method is also mentioned by Patrikalakis *et al.* [7]. Hu [2] and Hu *et al.* [3] extended the algorithm to solve *overconstrained* and *underconstrained* systems. The algorithm has been tested extensively for low dimensional (1 to 6) computer–aided geometric design problems. It should also prove to be effective for higher dimensional overconstrained problems.

The routines and the data structures presented in this manual are written in C++.

This manual describes the following five public functions:

1. *consolidate*() merges multiple intersecting n-dimensional root boxes into a single bounding box, using floating point or interval arithmetic.

2. *convPolyBox*() converts the coefficients of a system of power basis polynomial equations, defined in an arbitrary n-dimensional box, into the unit box, $[0, 1]^n$, using floating point or interval arithmetic.

3. *lexConv*() converts a lexical representation of a system of power basis polynomial equations into Bernstein basis polynomials, using floating point or interval arithmetic.

4. *monoToBern*() converts a system of power basis polynomial equations into Bernstein basis polynomials, using floating point or interval arithmetic.

5. *sim_solve*() solves systems of nonlinear polynomial equations in a $[0,1]^n$ box, using floating point or interval arithmetic.

4 References

[1] C.-Y. Hu, *Robust Algorithms for Sculptured Shape Visualization*, Master's thesis, Massachusetts Institute of Technology, July 1993.

[2] C.-Y. Hu, *Towards Robust Interval Solid Modeling for Curved Objects*, Ph.D. thesis, Massachusetts Institute of Technology, May 1995.

[3] C.-Y. Hu, T. Maekawa, E. C. Sherbrooke, and N. M. Patrikalakis, "Robust Interval Algorithm for Curve Intersections," *Computer Aided Design*, February 1995 (To appear).

[4] T. Maekawa, *Robust Computational Methods for Shape Interrogation*, Ph.D. thesis, Massachusetts Institute of Technology, June 1993.

[5] T. Maekawa and N. M. Patrikalakis, "Computation of singularities and intersections of offsets of planar curves," *Computer Aided Geometric Design* 10(5):407–429, October 1993.

[6] T. Maekawa and N. M. Patrikalakis, "Interrogation of differential geometry properties for design and manufacture," *The Visual Computer* 10(4):216–237, March 1994.

[7] N. M. Patrikalakis, C.-Y. Hu, T. Maekawa, and E. C. Sherbrooke, "Towards robust geometric modellers," in *Proceedings of the 1994 NSF Design and Manufacturing Grantees Conference, January 1994, Cambridge, Massachusetts*, National Science Foundation and Society of Manufacturing Engineers, Dearborn, Michigan, pp. 199-200.

[8] E. C. Sherbrooke, *Computation of the Solutions of Nonlinear Polynomial Systems*, Master's thesis, Massachusetts Institute of Technology, October 1993.

[9] E. C. Sherbrooke, "Computation of the solutions of nonlinear polynomial systems," *Computer Aided Geometric Design* 10(5):379–405, October 1993.

[10] J. Zhou, E. C. Sherbrooke, and N. M. Patrikalakis, "Computation of stationary points of distance functions," *Engineering with Computers* 9(4):231–246, Winter 1993.

5 Datatypes

The solver library can be compiled using (finite precision) floating point arithmetic or (bounded precision) interval arithmetic. The single source code version refers to a `real` datatype, which is specified to be `double` or `Interval` on the basis of a compile-time flag. The correct arithmetic operators (e.g. $+$, $-$, etc.) are automatically invoked by the C++ polymorphism (operator overloading) facility.

1. *typedef double real;*
   Note: Floating point finite precision data type. This is the default type.
2. *typedef Interval real;*
   Note: Interval bounded precision data type. To specify interval arithmetic, define the C++ preprocessor symbol `USE_INTERVAL`, e.g.
   ```
   g++ -DUSE_INTERVAL ...
   ```
   as an option to the C++ compiler.

7 Accuracy

The accuracy of floating point arithmetic is limited to approximately 15 significant digits. RIA provides an explicit error bound on all arithmetic operations.

8 Further Comments

A general n-variate polynomial can be written as follows:

$$\sum_{i=0}^{r}\sum_{j=0}^{s}\cdots\sum_{k=0}^{t} a_{i,j,\ldots,k}x_1^i x_2^j \cdots x_n^k = 0$$

or, in expanded form:

$$\begin{array}{l}
a_{0,0,\ldots,0}x_1^0x_2^0\cdots x_n^0 + a_{0,0,\ldots,1}x_1^0x_2^0\cdots x_n^1 + \cdots + a_{0,0,\ldots,t}x_1^0x_2^0\cdots x_n^t + \\
a_{0,1,\ldots,0}x_1^0x_2^1\cdots x_n^0 + a_{0,1,\ldots,1}x_1^0x_2^1\cdots x_n^1 + \cdots + a_{0,1,\ldots,t}x_1^0x_2^1\cdots x_n^t + \\
\cdots \\
a_{r,s,\ldots,0}x_1^rx_2^s\cdots x_n^0 + a_{r,s,\ldots,1}x_1^rx_2^s\cdots x_n^1 + \cdots + a_{r,s,\ldots,t}x_1^rx_2^s\cdots x_n^t = 0
\end{array}$$

This is a polynomial in n variables, x_1 (with maximum degree r) to x_n (with maximum degree t).

Whenever any of the library functions require the coefficients of a polynomial, they are specified in the sequence given by the previous equation, $a_{0,0,\ldots,0}$ to $a_{r,s,\ldots,t}$.

1 Purpose

Consolidate the roots of a system of nonlinear polynomial equations.

2 Specification

#include "consolidate.h"
*void consolidate(real epsCon, real eps, real **bp, short **ordlists, int numeq, int numvar, real **&roots, int &numroots);*

3 Description

This function collects the intersecting n-dimensional root boxes, whose size is eps^n, and merges them into a single n-dimensional bounding box. After normalizing the new merged bounding box, *sim_solve*() is called, using *epsCon* as the tolerance, to check if the bounding box contains the roots. If the box satisfies the tolerance, we consider it the consolidated root box.

For floating point arithmetic (FPA), the root is considered to be the mid-point of the consolidated box. For rounded interval arithmetic (RIA), the consolidated n-dimensional box is considered the interval root for each direction.

4 References

Not applicable.

5 Parameters

1. *real epsCon*
   On entry: specifies the root tolerance for each normalized bounding box.

2. *real eps*
   On entry: the original root tolerance, in other words, this is the tolerance specified as an argument to the call to *sim_solve*() that produced the initial, non-consolidated roots.

3. *real ** bp*
   On entry: pointer to a two-dimensional array containing the coefficients of the polynomials in the Bernstein basis.

   On exit: pointer to a two-dimensional array containing the coefficients of the polynomials in the power basis.

4. *short ** ordlists*
   On entry: pointer to a two-dimensional array containing the maximum order (degree + 1) of each variable in each equation.

5. *int numeq*
   On entry: the number of equations in the system.

6. *int numvar*
   On entry: the number of variables in the system.

7. *real **& roots*
   On entry: reference to an array of non-consolidated roots.

   On exit: reference to an array of the consolidated roots.

8. *int & numroots*
   On entry: reference to the number of non-consolidated roots.

   On exit: reference to the number of consolidated roots.

6 Return Values, Error Indicators and Warnings

Not applicable.

7 Accuracy

Not applicable.

8 Further Comments

Not applicable.

1 Purpose

Convert polynomials expressed in power basis in an arbitrary box into the unit box $[0,1]^n$.

2 Specification

#include "conv.h"
*void convPolyBox(real **&bp, short **ordlists, real **intv, int numvar, int numeq)*

3 Description

This function calculates new coefficients of a system of polynomials equations, originally defined in an arbitrary n-dimensional box, such that the polynomials are defined in the standard unit box, $[0,1]^n$.

4 References

Not applicable.

5 Parameters

1. *real **& bp*
   On entry: reference to a two dimensional array containing the coefficients of the power basis polynomials, defined in an arbitrary n-dimensional box. For instance, if the array for the j-th equation is n-dimensional with degrees $d_1, d_2, \ldots, d_n$, then $bp[j][(d_n+1)(d_{n-1}+1)\ldots(d_2+1)*i_1+(d_n+1)\ldots(d_3+1)*i_2+\ldots+i_n]$ indexes the coefficient of $x_1^{i_1}x_2^{i_2}\ldots x_n^{i_n}$ of the j-th equation.

   On exit: reference to a two dimensional array containing the coefficients of the power basis polynomials, defined in the unit box $[0,1]^n$.

2. *short ** ordlists*
   On entry: pointer to a two dimensional array containing the orders of each variable in each equation, defined in an arbitrary n-dimensional box.

   On exit: pointer to a two dimensional array containing the orders of each variable in each equation, defined in the unit box $[0,1]^n$.

3. *real ** intv*
   On entry: pointer to a two dimensional array containing the intervals for each variable in the polynomials. The variable intervals are given in the reverse order. In other words, for the *numvar* variables, *intv*[0][*] contains the interval for variable *numvar* and *intv*[*numvar*-1][*] contains the interval for variable 1.

4. *int numeq*
   On entry: the number of equations.

5. *int numvar*
   On entry: the number of variables.

6 Return Values, Error Indicators and Warnings

Not applicable.

7 Accuracy

Not applicable.

8 Further Comments

Not applicable.

1 Purpose

Convert lexical polynomial equations into a system of polynomials expressed in the Bernstein basis.

2 Specification

#include "lexPP.h"
int lexConv(char filename, real **&bp, short **&ordlists, int &numeq, int &numvar);*

3 Description

When the input polynomials are lexical, *lexConv*() can recognize the coefficients of the polynomials and convert the power basis into Bernstein basis.

The current implementation is designed only for a balanced system.

4 References

[1] Steven V. Earhart, *Lex — A Lexical Analyzer Generator Vol. 1–5* of the *UNIX Programmer's Manual.* Holt, Rinehart and Winston, 1986, New York.

5 Parameters

1. *char * filename*
   On entry: pointer to a NULL-terminated character string containing the input filename.

2. *real **& bp*
   On exit: reference to a two-dimensional array, containing the coefficients of input file in the Bernstein basis. For instance, if the array for the j-th equation is n-dimensional with degrees $d_1, d_2, \ldots, d_n$, then $bp[j][(d_n{+}1)(d_{n-1}{+}1)\ldots(d_2{+}1){*}i_1{+}(d_n{+}1)\ldots(d_3{+}1)*i_2+\ldots+i_n]$ indexes the coefficient of $x_1^{i_1}x_2^{i_2}\ldots x_n^{i_n}$ of the j-th equation.

3. *short **& ordlists*
   On exit: reference to a two-dimensional array containing the orders of each variable in each equation.

4. *int & numeq*
   On exit: reference to the number of equations.

5. *int & numvar*
   On exit: reference the number of variables.

6 Return Values, Error Indicators and Warnings

Not applicable.

7 Accuracy

Not applicable.

8 Further Comments

This function is written in C++.

The current implementation is designed only for a balanced system.

The format of the input data file is:

- The first line of the input file is an integer indicating the number of equations.
- Each equation contains two lines: the first specifies the maximum degree of each variable and the second is the body of the equation. Thus, the equation $x_1^2+5x_2-1=0$ is represented by the lines:

  `2 1`
  $\texttt{x1}^2 + \texttt{5x2} - \texttt{1}$
- Variables are written as `x1`, `x2`, ..., `x`i, i.e. `x` followed by an integer.
- Elevation of a variable to a power is expressed by the symbol `^`, for example, x^2 is written `x^2`.
- There should not be any = sign. The right-hand side of all equations is assumed to be equal to zero.
- All coefficients are real numbers and may be written as integers, floating point numbers (e.g. `0.5`), or rational numbers (e.g. `1/5`).
- There should not be any multiplication symbol "*" in the input file. For instance, $x1 * x2$ should be written as `x1x2`.
- No embedded blanks should occur between operators, coefficients, and variable names.
- The last line consist of the sole word `end`, indicating the end of the input file.

1 Purpose

Convert a system of polynomial equations expressed in power basis into a system of polynomials expressed in Bernstein basis.

2 Specification

#include "monoPP.h"
*void monoToBern(real **bp, short **deglist, int numeq, int numvar);*

3 Description

This function converts power basis polynomials into the Bernstein basis.

4 References

[1] E. C. Sherbrooke and N. M. Patrikalakis, "Computation of Solution of Nonlinear Polynomial System," *Computer Aided Geometric Design* 10(5):379-405, October 1993.

5 Parameters

1. *real ** bp*
   On entry: pointer to a two-dimensional array containing the coefficients of polynomials in the power basis. For instance, if the array for the j-th equation is n-dimensional with degrees $d_1, d_2, \ldots, d_n$, then $bp[j][(d_n+1)(d_{n-1}+1)\ldots(d_2+1)*i_1+(d_n+1)\ldots(d_3+1)*i_2+\ldots+i_n]$ indexes the coefficient of $x_1^{i_1}x_2^{i_2}\ldots x_n^{i_n}$ of the j-th equation.

   On exit: pointer to a two-dimensional array containing the coefficients of polynomials in the Bernstein basis.

2. *short ** deglist*
   On entry: pointer to a two-dimensional array containing the maximum *degree* (not order) of each variable in power basis.

   On exit: pointer to a two-dimensional array containing the maximum *order* (not degree) of each variable in Bernstein basis.

3. *int numeq*
   On entry: the number of equations.

4. *int numvar*
   On entry: the number of variables.

6 Return Values, Error Indicators and Warnings

Not applicable.

7 Accuracy

Not applicable.

8 Further Comments

Not applicable.

1 Purpose

Calculate solutions to a system of nonlinear polynomial equations.

2 Specification

#include "pp_solver.h"
*sim_solve(real **bp, short **ordlists, int numeq, int numvar, real eps, real ***roots, int* numroots);*

3 Description

This function uses the projected polyhedron root-finding algorithm to determine all real roots to a system of nonlinear polynomial equations that lie within the n-dimensional box $[0,1]^n$. The polynomial system to be solved should be written in Bernstein form as follows:

$$\begin{gathered} f_1(x_1, x_2, \cdots, x_n) = 0 \\ f_2(x_1, x_2, \cdots, x_n) = 0 \\ \cdots \\ f_m(x_1, x_2, \cdots, x_n) = 0 \end{gathered}$$

where $m = n$ for balanced systems, $m < n$ for underconstrained systems, and $m > n$ for overconstrained systems. The accuracy the roots is determined by the tolerance *eps*.

This function has been implemented for floating point and interval arithmetic.

4 References

[1] C.-Y. Hu, *Robust Algorithms for Sculptured Shape Visualization*, Master's thesis, MIT, Cambridge, Massachusetts, July 1993.

[2] C.-Y. Hu, *Towards Robust Interval Solid Modeling for Curved Objects*, Ph.D. thesis, Massachusetts Institute of Technology, May 1995.

[3] C.-Y. Hu, T. Maekawa, E. C. Sherbrooke, and N. M. Patrikalakis, "Robust Interval Algorithm for Curve Intersections," *Computer Aided Design*, February 1995 (To appear).

[4] C.-Y. Hu, T. Maekawa, E. C. Sherbrooke, and N. M. Patrikalakis, "Robust Interval Algorithm for Curve Intersections," *Computer Aided Design*, February 1995 (To appear).

[5] T. Maekawa, *Robust Computational Methods for Shape Interrogation*, Ph.D. thesis, MIT, Cambridge, Massachusetts, June 1993.

[6] T. Maekawa and N. M. Patrikalakis, "Computation of singularities and intersections of offsets of planar curves", *Computer Aided Geometric Design* 10(5):407-429, October 1993.

[7] T. Maekawa and N. M. Patrikalakis, "Interrogation of differential geometry properties for design and manufacture", *The Visual Computer* 10(4):216-237, March 1994.

[8] E. C. Sherbrooke, *Computation of the Solutions of Nonlinear Polynomial Systems*, Master's Thesis, MIT, Cambridge, Massachusetts, October 1993.

[9] E. C. Sherbrooke and N. M. Patrikalakis, "Computation of Solutions of Nonlinear Polynomial Systems", *Computer Aided Geometric Design* 10(5):379-405, October 1993.

[10] J. Zhou, E. C. Sherbrooke, and N. M. Patrikalakis, "Computation of stationary points of distance functions", *Engineering with Computers* 9(4):231-246, Winter 1993.

5 Parameters

1. *real ** bp*
   On entry: the coefficients of the system. For instance, if the array for the j-th equation is n-dimensional with degrees $d_1, d_2, \ldots, d_n$, then $bp[j][(d_n+1)(d_{n-1}+1)\ldots(d_2+1) * i_1 + (d_n+1)\ldots(d_3+1) * i_2 + \ldots + i_n]$ indexes the coefficient of $x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n}$ of the j-th equation.

   Notice that although we have used the power basis in our example, the multinomial structure is basis-independent. We can just as easily index elements in the tensor-product Bernstein basis as in the power basis.

2. *short ** ordlists*
   On entry: the non-negative integers for the orders (order = degree + 1) of each variable in each equation. For example, to access the orders of the j-th variable of the i-th equation, use $ordlists[i][j]$.

3. *int numeq*
   On entry: the number of equations in the system to be solved.

4. *int numvar*
   On entry: it specifies the number of variables in the system to be solved.

5. *real eps*
   On entry: the tolerance which determines the accuracy of output roots.

6. *real *** roots*
   On entry: pointer to non-initialized double pointer to *real*.

   On exit: the output roots.

7. *int * numroots*
   On exit: pointer to the number of roots found by the solver.

6 Return Values, Error Indicators and Warnings

The routine returns a pointer to an array of roots. If the array is empty (in other words, $roots$ = NULL), then no roots exist within the box $[0, 1]^n$.

7 Accuracy

The resolution of the roots depends on the user-specified tolerances. If the tolerance is set too tight, for example, $eps = 10^{-8}$, this routine may be slow for tangential roots.

8 Further Comments

Not applicable.