TENTH WORLD CONGRESS ON THE THEORY OF
MACHINES AND MECHANISMS
Oulu, Finland, June 20–24, 1999

APPLICATION OF A FAST NUMERICAL GRÖBNER BASES IMPLEMENTATION IN KINEMATIC SYNTHESIS

U. Hirn, P. Dietmaier
Technische Universität Graz, Institut für Allgemeine Mechanik
Kopernikusgasse 24, A–8010 Graz
e-mail: dietmaier@mech.tu–graz.ac.at

Keywords: *Gröbner Bases, floating point arithmetic, Kinematic synthesis, Algebraic equations*

1 Introduction

In many fields of engineering, for instance kinematic analysis, it is necessary to solve systems of polynomial equations. Mathematics software mostly uses the Buchberger Algorithm to find a lexicographic Gröbner Basis and subsequently compute the solutions. Although in theory the Buchberger Algorithm eventually terminates, the effort to compute Gröbner Bases from equations of higher degree or several unknowns grows rapidly. In fact for many problems it consumes so much performance, that powerful computers fail to produce results.
This paper introduces the free software *gfloat* which numerically computes the Gröbner Basis and the solutions. It is suited for problems which have zero dimensional solutions i.e. there exists a finite set of solution points. Two techniques are used to reduce the demand of computer memory and processor time. These are (1) the use of floating point/modular arithmetic and (2) computing a lexicographic Gröbner Basis indirectly using a conversion algorithm.
Section 6 demonstrates an application with a problem from kinematic synthesis.

2 Solving algebraic equations using Gröbner Bases and floating point arithmetic

For a detailed description of the Buchberger Algorithm see [Cox, Little, O'Shea 1992]. During the algorithm a large number of polynomials has to be stored, each having thousands of monomials. Using integer arithmetic the coefficients of the monomials gradually grow, they have more and more digits; this consumes memory and processor performance.
The use of floating point coefficients limits the storage space for the monomial coefficients, but it raises several severe problems concerning the accuracy of the floating point arithmetic. The only way to produce truly reliable error estimates is by using interval arithmetic. Interval arithmetic is very pessimistic, because it usually does not consider interdependencies between intermediate results [Knuth 1981] and uses strictly worse case criteria. Consequently one would need huge mantissas for the large number of operations performed during the Buchberger Algorithm, which leaves no advantage over integer arithmetic.

Looking at the computational side of floating point arithmetic two main reasons for inaccuracy can be identified. Subtractive cancellation comes into effect, when two nearly equal floating point numbers are subtracted. Some of the leading bits in the mantissa are cancelled to zero. After that the mantissa is normalized i.e. it is shifted left until the highest bit is nonzero [Knuth 1981]. During the shifting inaccurate bits enter at the right side of the mantissa reducing the number of accurate bits by the number of shifts (i.e. the number of leading bits that were cancelled).
Additionally, rounding errors take place, being at most 1/2 bit per operation. It is important to see that rounding errors tend to even out over a large number of operations, whereas subtractive cancellation is an irreversible loss of accuracy.
During the Buchberger Algorithm subtractive cancellation is a much bigger problem than rounding errors, because polynomial coefficients of nearly the same size are subtracted over and over again. We want to stress that this assumption is determined by the nature of the algorithm. In algorithms that do not perform subtractions of nearly equal terms, the rounding error is the dominant source of inaccuracy.
Following this idea the floating point arithmetic in *gfloat* performs cancellation control, therefore it tracks the loss of mantissa bits due to subtractions. Additionally, a certain percentage of bits in the mantissa are reserved for rounding errors. The remaining bits (significant bits) are used as an estimate for the precision of the floating point number.

3 Replacing integers with a pair of modular coefficient / floating point coefficient

The usage of floating point coefficients makes it impossible to decide if the result of a subtraction *a*–*b* yields exactly zero. This would be necessary, because during the Buchberger Algorithm it is checked if monoms are cancelled, i.e. the coefficient becomes zero. Incorrect cancellation of a monomial has unpredictable effects in the algorithm, from minor changes in the result to disastrous consequences like the wrong number of solutions or totally incorrect solutions.
Using floating point arithmetic one usually defines a threshold value ε and takes $|a-b|<\varepsilon$ equivalent to $a-b=0$. This is not applicable here because the difference of two integers can be nonzero although the difference of the floating point representations of these integers is zero.
For example take *a*=451614, *b*=451638 represented by a decimal float with a 4 digit mantissa.

Float: 0.4516e6 – 0.4516e6 = 0.0e0 Integer: 451638 – 451614 = –24

All digits in the mantissa have been cancelled, but the integer result still is not zero.
In order to detect exact zeroes an additional modular coefficient is introduced where the modular coefficient is the residue class of the integer coefficient modulo a prime modulus [Lazard 1992]. Consequently all polynomial operations are performed for the floating point coefficient and the modular coefficient. The integer is zero if, and only if the modular and the floating point coefficient are zero.
Taking the above example with modulus = 31991 then *amod* = 3740 and *bmod* = 3764.

Modular: 3740–3764 = 31967; Float: 0.4516e6–0.4516e6 = 0.0e0; Int: 451638–451614 =–24

The modular operation shows, that this is not a real zero. But this operation also has another, more important result: all significant digits in the mantissa were cancelled. As we showed in the previous section that means that the content of the mantissa has no meaning whatsoever. In order to find useful results we have to restart the computation with a longer mantissa in the floats.
This can be summarized in the coefficient test [Lösch 1996].

if amod- bmod = 0
    if afloat - bfloat = 0.0e0 *
        The result is a real zero.
    else if afloat - bfloat ≠ 0.0e0 *
        a - b is a multiple of the modulus. Choose different modulus and restart.
else if amod- bmod ≠ 0
    if afloat - bfloat = 0.0e0 *
        All digits of the mantissa have been cancelled. Raise mantissa length and restart.

* afloat - bfloat = 0.0e0 means that all significant bits of the mantissa yield zero.

There only remains the very unlikely occurrence where the integer result is a multiple of the modulus and the floating point operation yields zero. Then the monomial would be cancelled falsely.
Repeating the calculations using a different modulus would further lower the probability for false cancellation; the integer coefficient would have to be a multiple of the product of the two moduli.

4 Transformation of Gröbner Bases

For polynomials with more than one variable there exist different monomial orderings, that is sets of rules how to sort the monomials within the polynomial [Cox, Little, O'Shea 1992]. Fixing different monomial orderings leads to different Gröbner Bases; for our purposes we need to find a *lexicographic* Gröbner Basis. For equations with a zero dimensional solution it is possible to transform a Gröbner Basis with respect to a certain monomial ordering to a Gröbner Basis with respect to a different ordering. However it is much better in terms of number and size of the polynomials involved to use *total degree reverse lexicographic* ordering to compute a Gröbner Basis than using *lexicographic* ordering.
It is appropriate to produce a *lexicographic* Gröbner Basis indirectly by first computing a *total degree reverse lexicographic* (tdegrev) Gröbner Basis and transforming it to a *lexicographic* Gröbner Basis. In *gfloat* the transformation algorithm CONVGRÖBNER [Becker, Weispfennig 1993] is used.

5 The software *gfloat*

The floating point arithmetic uses multiple precision software floats with error estimation. It is basically the free software library GNU gmp-2.0.2 where we added mantissa cancellation control (see section 2). If the precision of the result is not sufficient it is possible to increase the mantissa length of the floats to get the desired precision.
The solution of the input equation consists of four steps (Figure 1). First a total degree reverse lexicographic Gröbner Basis with only modular coefficients is computed. During the computation all operations necessary for the tdegrev Gröbner Basis are recorded in the trace file [Traverso 1988)]. The point about the trace file is, that all superfluous operations (i.e. zero reductions of S-Polynomials) are not recorded and thus will not be performed during the TRACE. The modular Gröbner Basis already allows to determine the dimension of the solution and, if the solution is zero dimensional, the number of solution points including complex and multiple solutions [Becker T., Weispfennig V. 1993]. The second step, TRACE, performs

exactly the operations recorded in the trace file using floating point arithmetic. Subsequently it produces a total degree reverse lexicographic ordering Gröbner Basis with modular/floating point coefficients. Step three, CONVERT, converts the tdegrev Gröbner Basis to a lexicographic Gröbner Basis, which is finally used in step four to find the actual solutions of the equations. The roots are found with Laguerres method [Press, et al. 1990]. For a detailed description of the algorithms, implementation of the software as well as examples see [Hirn 1999] and [Lösch 1996].

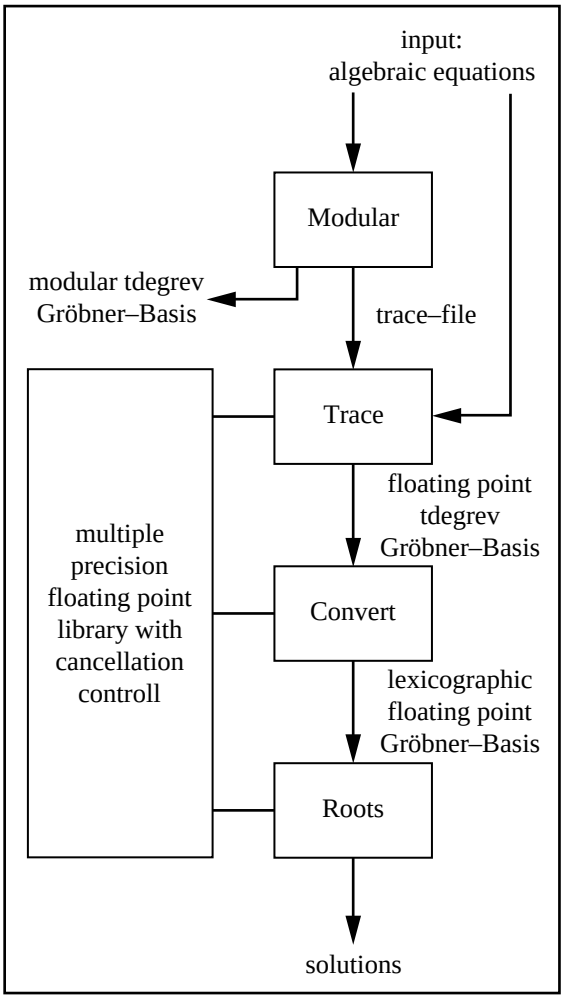

Figure 1. Flow–chart of the *gfloat* software.

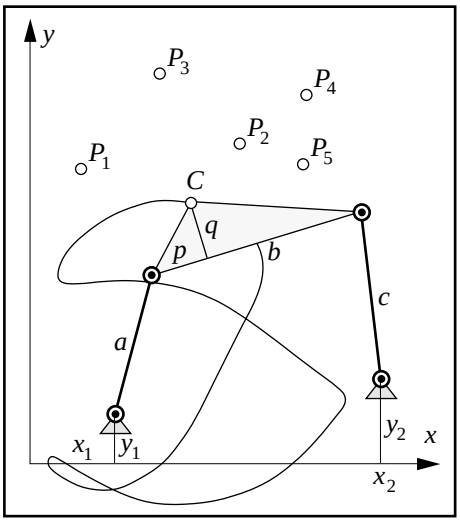

Figure 2. A planar four–bar mechanism and its design parameters together with 5 given points of the desired coupler–curve.

6 Example

In order to demonstrate the capabilities of the *gfloat* software we took an example from mechanism synthesis. Figure 2 shows a planar four–bar mechanism with its nine design parameters (x_1, y_1, x_2, y_2, a, b, c, p, and q) and the coupler–curve traced by the coupler–point C during the motion of the mechanism. For this example we consider the specific synthesis problem where the positions of the fixed pivots are prescribed (for the example at hand $x_1 = -0.5$, $y_1 = 0$, $x_2 = 0.5$, and $y_2 = 0$) and the remaining five parameters (a, b, c, p, and q) have to be computed such that the coupler–curve of the corresponding mechanism passes through a set of 5 specified points. The coordinates of the 5 chosen points are listed in Table 1. Substituting successively the coordinates of the 5 given points into the equation of the coupler–curve and making use of the given values for x_1, y_1, x_2, and y_2 we obtain 5 algebraic equations for the five unknown design parameters. These equations possess a total 36 solutions and half of them turn out to be real. The real sets are listed in Table 2 and the mechanism with the corresponding coupler curves are depicted in Figure 3.

The input equations for *gfloat* are of total degree 8 and consist of 78 monomials each. The solutions were computed using floating point arithmetic with a mantissa length of 5700 bit. The total computation time to obtain the lexicographic Gröbner Basis was 1067s on a DEC Alpha 3000.

i	P_i (x, y)	
1	-0.25	0.25
2	-0.25	0.5
3	-0.25	0.75
4	0.25	0.25
5	0.25	0.75

Table 1. Coordinates of the 5 prescribed points of the coupler–curve.

k	a_k	b_k	c_k	p_k	q_k
1	0.447487	0.914329	0.530446	0.629811	0.629811
2	0.452147	0.915731	0.533073	0.634554	0.634554
3	0.353823	1.00999	0.353823	0.504996	0.504996
4	0.40334	0.933735	0.40334	0.466867	0.466867
5	0.461822	0.934294	0.461822	0.467147	0.467147
6	0.664327	0.412791	0.463808	-0.236283	-0.236283
7	0.981242	0.982689	0.981242	0.491345	0.491345
8	6.57234	7.83864	6.57234	3.91932	3.91932
9	1.20093	1.41933	1.20093	0.709667	0.709667
10	1.58689	2.23007	1.58689	1.11504	1.11504
11	0.821306	0.977939	1.14441	0.356661	0.356661
12	0.44775	1.04078	0.379797	0.490022	0.490022
13	2.13756	3.31435	2.13756	1.65717	1.65717
14	0.82896	1.68817	1.30241	0.445045	0.445045
15	2.68846	2.42892	0.714357	2.23457	2.23457
16	1.88011	2.37829	1.45393	1.42876	1.42876
17	2.18498	2.55381	0.710428	2.32537	2.32537
18	24.7342	24.7385	1.00406	24.4161	24.4161

Table 2. The 18 sets of design parameters for the 18 different four–bar mechanisms.

7 Conclusions

The concepts used in the implementation of *gfloat* reduce the computational effort to obtain a lexicographic Gröbner Basis. This makes it possible to find the solutions of more difficult (i.e. higher degree or more variables) algebraic equations than other implementations of Gröbner Bases, which are using integer arithmetic and computing the lexicographic Gröbner Basis directly.

Gfloat delivers approximate values for all the solutions of a system of polynomial equations including multiple and complex solutions. It is very likely that the correct number of solutions is found (see section 3), this can be strengthened further by repeating the whole procedure using a different modulus.

Although the precision of the solution points cannot be guaranteed, some verification is provided via automatic substitution of the solutions in the input equations. The mantissa length of the floating point arithmetic can be increased arbitrarily, so the precision of the result can be raised if necessary.

We are perfectly aware of the fact that *gfloat* cannot produce results that guarantee a certain precision - but the software is often able to deliver results where programs using integer or interval arithmetic fail because of their demand of computer performance.

Hence, *gfloat* is a useful tool for engineering applications, when it is sufficient to find results with a limited precision rather than exact solutions.

The source code of the software written in C is available on the internet via our home page.

References:

Cox D., Little J., O'Shea D. (1992), Ideals Varieties and Algorithms, An Introduction to Computational Algebraic Geometry and Commutative Algebra, Springer.

Becker T., Weispfennig V., (1993) Gröbner Bases, A Computational Approach to Commutative Algebra, Graduate Texts in Mathematics, Vol 141, Springer.

Knuth D.E. (1981). The Art of Computer Programming, Vol. 2, Seminumerical Algorithms,

Addition Wesley Publishing Co., Reading, Massachusetts, 2nd edition.

Lazard D. (1992), Steward platforms and Gröbner Bases. Proceedings of the 3rd International Workshop on advances in Robot Kinematics (3ARK, Ferrara, Italy, Sept. 1992).

Lösch S., Gröbner Basen in Fließkommazahlen, Anwendungen in der Kinematik, Dissertation, Institute of Mechanics, University of Technology, Graz.

Press W. et al. (1990), Numerical Recipes in C, The Art of Scientific Computing. Cambridge University Press, Cambridge.

Traverso C., (1988), Gröbner Trace Algorithms. ISSAC 88, Lecture Notes in Computer Science, Springer.

Hirn U. (1999), An Implementation of Gröbner Bases using Floating Point Arithmetic and Change of Ordering, Diploma Thesis, University of Technology Graz.

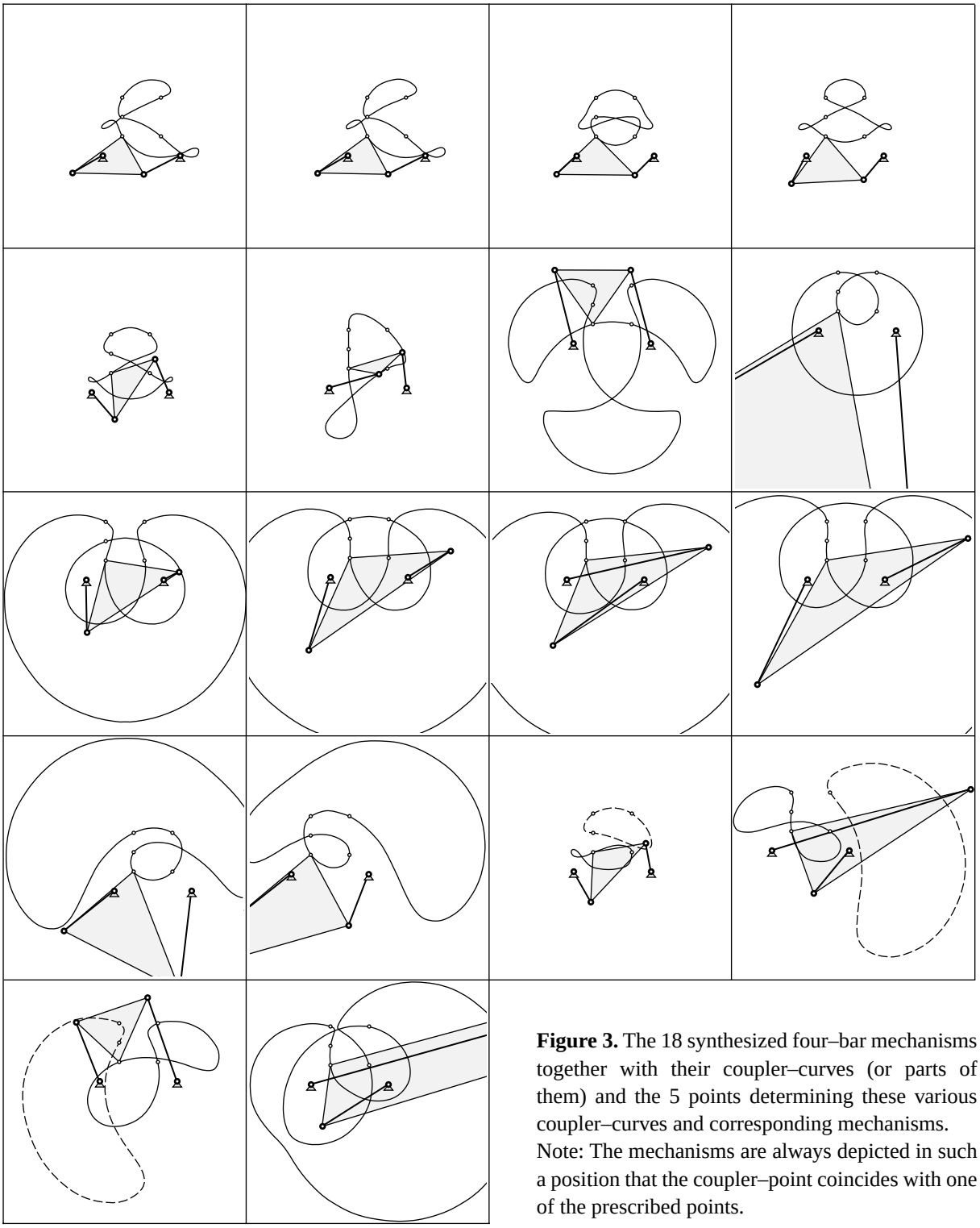

Figure 3. The 18 synthesized four–bar mechanisms together with their coupler–curves (or parts of them) and the 5 points determining these various coupler–curves and corresponding mechanisms.
Note: The mechanisms are always depicted in such a position that the coupler–point coincides with one of the prescribed points.