Solving Systems of Algebraic Equations, or the Interface between Software and Mathematics

Gaston H. Gonnet
Michael B. Monagan

Department of Computer Science
University of Waterloo

Research Report CS-89-13

May, 1989

Solving Systems of Algebraic Equations, or the Interface between Software and Mathematics

Extended Abstract

by

Gaston H. Gonnet

and

Michael B. Monagan

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

This paper describes two fundamental aspects of the solution of systems of algebraic equations: the use of *substitution* and the use of a *complexity function* to determine, at each step, which of several methods / equations / unknowns to use. These techniques, which have received little attention in the past, can be viewed as in the interface between mathematics and software. An implementation of these techniques in Maple[Cha83] is described. Timing results comparing the Maple, Macsyma[Mos74] and Reduce[Hea71] symbolic algebra systems on a range of sample problems are presented. The proposed method proves to be superior to other techniques.

1. Introduction

Solving large systems of algebraic equations is, in general, an almost hopeless task for most symbolic algebra systems. Even modest system of equations, which can be expressed in a very few lines, simply cannot be handled by most systems.[Rim84, Gon83, Ste84]

Most annoying is the fact, that in some cases, manipulation by a human produces a solution with little difficulty. What is going wrong? Without suggesting that we plunge into artificial intelligence, what we can say is that most present day algorithms are not *poly-algorithms*[Cha84] but somewhat general algorithms that will attack any type of problem with the same technique.

Let us use as an example the system of equations[Gon83]

$$a_4b_4 = 0, \quad a_5b_5 = 0, \quad a_2b_2 = 0$$

$$a_5b_1+b_5+a_4b_3+a_3b_4 = 0,$$

$$a_0b_2+b_2+a_4b_2+a_2b_4+c_2+a_2b_0+a_2b_1 = 0$$

$$a_0b_0+a_0b_1+a_0b_4+a_3b_2+b_0+b_1+b_4+a_4b_0+a_4b_1+a_2b_5+a_4b_4+c_1+c_4+a_5b_2+a_2b_3+c_0 = 0$$

$$a_3b_0+a_0b_3+a_0b_5+a_5b_0+b_3+b_5+a_5b_4+a_4b_3+a_4b_5+a_3b_4+a_5b_1+a_3b_1+c_3+c_5 = 1$$

$$a_5b_3+a_5b_5+a_3b_5+a_3b_3 = 0$$

$$a_5b_3+2a_5b_5+a_3b_5 = 0$$

$$a_0b_5+a_5b_0+a_3b_4+2a_5b_4+a_5b_1+b_5+a_4b_3+2a_4b_5+c_5 = 0$$
$$a_4b_0+2a_4b_4+a_2b_5+b_4+a_4b_1+a_5b_2+a_0b_4+c_4 = 0$$
$$a_2b_4+a_4b_2 = 0, \quad a_4b_5+a_5b_4 = 0, \quad 2a_3b_3+a_5b_3+a_3b_5 = 0$$
$$c_3+a_0b_3+2b_3+b_5+a_4b_3+a_3b_0+2a_3b_1+a_5b_1+a_3b_4 = 0$$
$$c_1+a_0b_1+2b_1+a_4b_1+a_2b_3+b_0+a_3b_2+b_4 = 0$$
$$a_2b_1+b_2 = 0, \quad a_5b_3+a_3b_5 = 0, \quad b_4+a_4b_1 = 0$$
$$b_1 = 0, \quad a_3b_3 = 0, \quad b_3+a_3b_1 = 0$$

This system[†] is not linear, and it is overdetermined. Applying the standard resultant method is hopeless (Macsyma[Mos74] had not finished after 10 hours CPU). Yet, by inspection, we find the sequence of simple substitutions $b_1=0$, $b_2=0$, $b_3=0$, $b_4=0$, $b_5=0$, $c_1=-b_0$ which lead to the system:

$$a_4b_0+c_4 = 0$$
$$a_3b_0+a_5b_0+c_3+c_5 = 1$$
$$a_5b_0+c_5 = 0, \quad c_3+a_3b_0 = 0$$
$$a_0b_0+a_4b_0+c_4+c_0 = 0$$
$$c_2+a_2b_0 = 0$$

which can now be solved directly, or still more substitutions can be tried. Even better would be to observe that the 2nd, 3rd and 4th equation cannot be solved simultaneously, and hence the system has no solution. One of the goals of this paper is to exploit the method suggested by this example, namely solving systems of equations by *substitution*. The challenging part is of course to decide what substitutions to make, and in which order to make them.

The following two equations come from of a polynomial system (Allaway) with 21 equations in 18 unknowns and two free parameters x and y.

$$\frac{t_{22}}{2t_{33}} - \frac{x}{2t_{22}} + \frac{1}{6x} - \frac{t_{33}}{6t_{44}} = 0$$
$$\frac{7}{2x} - \frac{3y}{2x} - \frac{8x}{t_{22}} + \frac{3t_{21}}{2t_{22}} + \frac{9t_{22}}{2t_{33}} - \frac{t_{32}}{2t_{33}} = 0$$

The remaining equations are defined in Appendix 2. They are similar in structure to the above but range in size up to 110 terms. This system is much too difficult to solve by hand, yet our method of substitution can be employed to solve it relatively quickly. We believe that no other system is able to handle this particular problem at this time.

† These kinds of polynomial systems arise in numerous places. This example is part of the proof that there is no method for approximating the inverse of a number that uses only two multiplications and achieves third degree or higher. Many complexity theorems of the form "There is no formula using ... to do ..." give rise to similar systems.

2. A Repertoire of Actions.

The first proposal in this paper is to attack a given system of equations with a large repertoire of solving techniques, not just a single method. The second proposal is to perform the selection of the technique to apply with the aid of a complexity function and an ordering of all the applicable techniques. The complexity function will, *at each step*, select the simplest equation/unknown to attack. Whilst further considerations about the complexity function itself, about the size of the repertoire or about the relative ordering of the techniques may be made, we are completely convinced about the usefulness of this general approach.

The main repertoire of actions is:

(a) solving an equation for an unknown then eliminating that unknown by substitution into the remaining equations.

(b) solving systems of linear equations.

(c) factoring the equations to split the system into one or more simpler subsystems.

(d) solving systems of polynomial equations.

(e) global considerations for invalid or unwanted solutions.

This classification can be subdivided further.

Solving one equation in a single unknown and substituting the solution for that unknown into the remaining equations.

(a1) solving the equation $x = 0$.

(a2) solving the equation $x = k$, where k does not contain any unknowns being solved for, that is, for the solving function k is to be considered a "constant" expression.

(a3) solving the equation $a\,x = expr$, where a is a rational and $expr$ is any expression (possibly involving unknowns).

(a4) solving the equation $k\,x = expr$, where k is a constant expression and $expr$ is any expression (possibly involving unknowns).

(a5) solving the equation $expr_1\ x = expr_2$, where $expr_1$ and $expr_2$ are any expression (possibly involving unknowns).

(a6) solving other non-linear equations

For systems of linear equations (possibly sparse) we consider three cases. These are discussed in detail in a later section:

(b1) solving a system of linear equations with coefficients polynomials over the rationals. A *primitive* fraction-free method is used here.

(b2) solving a system of linear equations with floating point coefficients†.

(b3) solving a general system of linear equations by Gaussian elimination.

Equations may be factored by:

(c1) being factored on input

† Linear systems with floating point coefficients necessitate a separate algorithm because of numerical stability considerations. This is not discussed further here.

(c2) by an explicit factorization

(c3) by the (implicit) content computation in x for case (a5)

(c4) when eliminating common gcds with the inequalities

The purpose of factoring equations is to split the system of equations into one or more simpler subsystems. Although we may generate an exponentially large number of subsystems, this is still desirable for two reasons: the total number of solutions remains the same, but the subsystems will be considerably simpler to solve and secondly, the user may only be interested in obtaining only one (or only a few) solution, and after obtaining the desired solutions, we can stop.

For systems of polynomial equations we currently use

(d1) a Gröbner basis algorithm using the pure lexicographical ordering.[Buc84, Cza86] The pure lexicographical ordering is used because generally it separates the variables. This greatly simplifies the back-substitution process.

(d2) the elimination method based on resultants.[Knu81] We use the resultant method for systems of two equations and when we can guarantee that it will not introduce spurious solutions.

3. A Complexity Function.

The job of the complexity function is to select at each step the best technique to apply. We compute for each unknown in each equation a complexity value and, select the equation / unknown with lowest complexity. Of the techniques mentioned above, there are three general criteria that yield a good ordering for the complexity function:

(1) a linear dependence is preferred over higher degree dependences.

(2) the number of other unknowns involved, number of other parameters involved and total size of the expression. The higher these quantities the more "complex" is likely to be the substitution. The relative weight of these should be as given. For example the current complexity of expressions for many types of equations is computed as

$$500\times \#\text{unknowns} + 50\times \#\text{parameters} + \text{Size}$$

For example, the equation

$$x = a\,y + c$$

will be preferred over

$$x = a\,y + b\,z + c$$

Note that the number of other unknowns involved is kept as low as possible to preserve sparsity (when the system is sparse).

(3) a non-splitting substitution is preferred over a *splitting* substitution. A *splitting* substitution is one that generates two or more systems to solve. For example, $expr_1 x = expr_2$ generates a split, that is, $x = expr_2/expr_1$, $expr_1 \neq 0$ and $expr_1 = 0$, $expr_2 = 0$. Equations which are products of terms involving unknowns will also generate a split. Splitting may cause an exponential explosion on the number of systems of equations to be solved.

The splitting of systems of equations poses two problems: firstly, repeated solutions could be generated; if the solutions are identical there is no difficulty, but typically the duplication is more subtle or it becomes a duplication of a special case of a more general solution. Secondly, how do we order splits according to their type and branching factor?

The first problem is solved by the introduction of inequalities together with the equations. In other words, the objective of the solver becomes: determine an assignment for the unknowns that satisfies all the equations but none of the inequalities. The use of inequalities provided an additional feature at the external user interface, namely the ability to pose systems of equations and inequalities. The user may pose inequalities to reflect constraints on the problem, to avoid singularities or to reduce the complexity of solving by eliminating trivial solutions. Internally the solver generates inequalities to avoid repeated computation for equations that split and for denominators that contain unknowns.

Splits are ordered by the complexity of their simplest factor for products of equations. For example, the equation $x\,y\,z = 0$ is preferred over $(x+z)(x+y) = 0$.

The first counterintuitive situation occurs with factors which have similar complexities but different branching factors. For example, which equation is preferred between $x\,y\,z = 0$ and $v\,w = 0$? Assuming some uniformity, in the sense that substituting any of the variables for 0 will give equally-complex systems, it can be shown that the highest order branching factor is preferred over the lower order branching. This is mostly a consequence of the use of inequalities.

4. The Maple Solve Function.

The function *solve* in Maple solves systems of algebraic equations. Other solvers exist such as *dsolve* for solving differential equations, *rsolve* for solving recurrences, *msolve* for solving equations over finite fields, etc. Here we discuss the syntax, code organization, crucial implementation considerations and some global considerations with regard to dealing with unwanted and invalid solutions of the *solve* function.

A call to *solve* consists of the system of equations followed optionally by the unknowns to be solved for. The system of equations is specified as a set of equations, inequalities or expressions. If an expression is specified, it is assumed that the expression must be equated to zero. If no unknowns are specified, it is assumed that all indeterminates occurring in the equations are the unknowns. A further optional third argument of the form $MaxSols = n$ may be used to limit the number of solutions computed.

For example, a typical call might be

```
solve( {x^2 - y^2 = a,  x + y + a = 3*b,  x <> 3*b},  {x, y} );
```

which yields:

$$\{ y = -\frac{a^2-6ab+9b^2-a}{2a-6b},\ x = -\frac{a^2-6ab+9b^2+a}{2a-6b} \}$$

The three main components of *solve* are, *scalar, linear* and *gensys*. A single equation in a single unknown is solved by the *scalar* solver (includes solutions to irreducible polynomials up to degree four). For systems of linear equations, the *linear* solver is called. For systems of non-linear equations, the *gensys* (*gen*eral *sys*tems) routine is called. The *gensys* routine embodies all of the techniques mentioned in the previous section, and makes use of the function *complexity* to decide which of the techniques to apply at each step. Internally, the *notzero* set contains expressions which must not be zero. Inequalities are initially placed in this set by *solve*.

The *gensys* routine performs the following steps (in the order given).

(1) If any expression in the *notzero* set is zero, then the NULL expression is returned (indicating that there are no solutions).

(2) The routine *cleanup* is used to put the equations in normal form, and make a modest attempt to simplify them. For example, factors independent of the unknowns are discarded, etc.

(3) Equations of the form $x = 0$ are eliminated by simultaneous substitution.

(4) Complexities are computed for each equation/unknown by the routine *complexity*. The equation/unknown with the minimum complexity is selected.

(5) If the complexity indicates an unknown with a linear occurrence, that unknown is eliminated by substitution into the equations which are then solved recursively.

(6) If the complexity indicates an equation in a single unknown, the *scalar* solver is invoked. If it succeeds, the solution(s) are substituted into the system which is solved recursively. Note: a non-linear equation in a single unknown is solved at this point.

(7) If the complexity indicates a product, the system is split into two or more simpler sub-systems, which are then solved recursively.

(8) The equations are factored in an attempt to split the system. If any of the equations factor, the new system is solved recursively.

(9) If the equations are all polynomials, a Gröbner basis or resulant is computed using the pure lexicographical ordering. The Gröbner basis is solved recursively.

(10) The *scalar* solver is invoked on the simplest of the equations/unknowns, and if it succeeds, a substitution is made.

(11) At this point, the *gensys* routine gives up. The user is informed that some solutions may have been lost and the NULL expression is returned.

A crucial aspect of this organization is in realizing that computing complexity values, cleaning up the equations and factoring the equations are not cheap operations. In order to avoid redoing these calculations, we make use of Maple's *remember* facility. Briefly, if the remember option is specified in a Maple procedure, a table of function results indexed by the function's arguments is maintained. Any function with option remember is computed only once for each set of arguments.

How does this method compare to other methods, like resultants, or purely Gröbner bases? Note that any correct system solver, whether by substitutions, by resultants, etc. should produce the same answers, so the only issue here is an issue of efficiency. There are three important advantages in this method with respect to the others, even in their optimal coded forms:

(i) The solution of a system by substitutions allows to eliminate, at each step, the most convenient unknown. At every step, it is easy to predict which is the best unknown to substitute for next. This is not possible (at least at present) with Gröbner bases where the ordering is defined at the beginning of the computation. Bad orderings are known to be disastrous for some problems.

(ii) The substitution method allows to split the system into two or more sub-systems whenever an equation can be factored (at any stage) or for linear equations when the leading coefficient depends on other unknowns. This is a fundamental advantage, given that the complexity of most methods is at least as large as the size of the final answer, and that the final answer will contain a polynomial of degree equal to the number of solutions and that this polynomial alone can be (and often is) of exponential size in its degree.

(iii) Our implementation of the substitution method does not generate spurious solutions, as the resultant method may generate. These spurious solutions are very difficult to identify. Their presence can greatly increase the time taken to obtain the solutions.

5. Linear Systems.

Although solving linear systems is better understood than polynomial systems, it has nevertheless been our experience that computer algebra systems are poor at dealing with sparse linear systems. Yet in our experience, almost all the larger systems of linear equations that we have encountered in practice are sparse. Furthermore, for systems with multivariate polynomial coefficients, the coefficients are also sparse. Thus in designing an algorithm for solving linear equations, we believe that the methods used should definitely contemplate such classes of systems. In this section, we present a "primitive" version of the basic fraction-free method of Gaussian Elimination which is geared toward solving sparse systems. The basic underlying idea is the same as that presented in the previous section: use a complexity criteria to select the best equation / unknown to eliminate at each step. To justify our choice of algorithm, we briefly mention previous work done in the area, identifying the known methods, and the classes of problems for which they do well. Next we describe our algorithm in broad detail, and discuss the advantages of this approach over the other methods. Finally, we present some example problems comparing our implementation in Maple with that in Reduce and Macsyma, illustrating the improvements achieved.

Early work by Bareiss[Bar68] and Lipson[Lip68] saw the development of fraction-free algorithms that avoid the quotient field, thus require no greatest common divisor computations, except to reduce the final solutions to lowest terms. The coefficient growth that occurs is reduced by inclusion of divisions that are known to be exact. Later work by McClellan [McC73] exploited modular and evaluation homomorphisms to map the problem domain down to solving linear systems over GF(q). This significantly improved the theoretical running time over the fraction-free algorithm. However, this approach assumes that the coefficients are uniformly dense, which as we have observed, is rarely the case in practice. The running time (which is exponential in the number of variables and their degrees) is essentially the same regardless of the sparsity of the coefficients. Gentleman & Johnson[Gen76] and Horowitz and Sahni[Hor75] considered computing the determinant of the coefficient matrix using the method of minor expansion (this approach can be applied to solving systems of linear equations by using Cramer's rule). Although minor expansion is inherently exponential, they demonstrated that for matrices with sparse polynomial coefficients, that minor expansion can be much faster than the fraction free algorithm. More recently, Kaltofen[Kal85] has presented a probabilistic approach which runs in polynomial time in the total degree and number of non-zero terms of the equations. However, early experience with this approach indicates that the intermediate expressions that arise can be quite large. Of all the algorithms available, it is fair to say that no particular algorithm is a clear winner. Which algorithm will do best depends on the domain of the computation, the structure of the system being solved and also, the particular strengths and weaknesses of the system in which the algorithm in being implemented.

5.1. The Primitive Fraction-Free Method.

From a general point of view, the major problem with the fraction-free algorithm is that is suffers from coefficient growth. In the primitive fraction-free method, we minimize this coefficient growth as follows. Consider the two equations $a = a_1 x_1 + \cdots + a_n x_n$ and $b = b_1 x_1 + \cdots + b_m x_m$ where a and b are linear in x_k and suppose we wish to eliminate x_1. In the fraction-free algorithm of Bareiss, x_1 is eliminated by computing $c = (b_1 a - a_1 b)/d$ where d is the previous *pivot*. In the primitive fraction-free algorithm, we first compute $g = \gcd(a_1, b_1)$ then eliminate x_1 by computing $c = (b_1/g)a - (a_1/g)b$. Next we compute and divide out the content of c with respect to the unknowns. By dividing by the content, we minimize the coefficient growth. A number of further improvements are possible. We include a trial division by the previous pivot. This heuristic works well in the dense case, where the algorithm effectively reduces to the fraction-free algorithm of Bareiss. The content (the gcd of the coefficients of a polynomial), can be computed as one gcd with high probability by computing the gcd of two random linear combinations of the coefficients. We note that the gcd's computed are often 1, which is the best case for the modular based gcd algorithms. On the other hand, if a gcd is non-trivial, then this represents a gain because we have succeeded in reducing the size of the coefficients.

5.2. Improvements for Sparse systems.

A system of equations is represented and manipulated as a set of polynomials (a sparse representation is used) which are linear in the unknowns. Our main goal here is to reduce the coefficient growth that occurs by taking advantage of the sparsity present. We can do this in two ways. First, at each step we can select an unknown to eliminate whose coefficient is smallest in "size". Alternatively, we can try to minimize the "fill-in" that occurs. The purpose of minimizing fill-in is that later elimination steps will involve fewer equations. We have tried two strategies. The simpler is, at each step, choose the equation which is smallest in size, and from that equation, eliminate the unknown whose coefficient is smallest in size. We have found, however, that preserving sparsity in large systems seems to be more effective in limiting the coefficient growth. The second strategy we tried is to apply the "minimal degree algorithm" a well known heuristic from Numerical Analysis. In the minimal degree algorithm, at each step the unknown which occurs in the fewest number of equations is eliminated. Where there is a choice, we select the unknown whose coefficient is smallest in size. As a final consideration, before we use the primitive fraction-free algorithm, trivial equations (of the form $x+k$ where k is independent of the unknowns being solved for), are eliminated in one step by simultaneous substitution.

An analysis of the running time of the primitive fraction free algorithm thus presented would show that it is no better in general than the fraction free algorithm of Bareiss. In fact, it will be somewhat worse because of the extra work done in computing gcd's. However, for systems of equations that arise in practice, the extra work done in trying to minimize the coefficient growth will generally pay off by greatly reducing the running time. Clearly, the extent to which this approach succeeds can only be determined from experience. Roughly speaking, we find that whereas sometimes the gain, if anything, is insignificant, quite often there is a huge improvement. An important advantage of the primitive fraction-free algorithm over the fraction free algorithm of Bareiss, McClellan's modular algorithm, or the algorithms based on minor expansion and Cramer's rule, is that the determinant of the coefficient matrix need not explicitly be computed. The determinant may be too big to be computed, and yet because of the structure of the system, the solutions can be computed.

6. Sample Problems.

In this section, we compare the Maple, Macsyma and Reduce solvers on some sample problems. A description of these problems is given in the Appendix. Times reported (in CPU seconds) were obtained from a Vax 11/785 running the Berkeley Unix 4.2 BSD operating system.

Examples of Linear Systems.

Problem	Dimension	Domain	Maple 4.2	Macsyma 308	Reduce 3.1
1	20 by 20	Z	150	157	219
2	81 by 81	Z	25	2,045	344
3	147 by 147	Q	282	>10,000	>10,000
4	289 by 289	Q	654	>10,000	(1)
5	5 by 5	Z[a1,a2,a3,a4,a5]	77.0	79.2	22.5
6	147 by 49	Q(a,b,c)	21.0	1,596	(2)
7	21 by 21	$Z[\beta_1,\beta_2,\beta_3,\beta_4,\beta_5]$	38.4	6,281	(3)

Examples of Polynomial Systems.

For the sample problems in our second table, we note that Reduce is currently not programmed to handle non-linear systems. Also, we found that Macsyma† was unable to solve any of the problems given 10,000 seconds of CPU time.

Problem	Description	Domain	Time
8	3 by 3 with 8 solutions	Z	23.4
9	19 by 17 with 3 solutions	Z	72.7
10	22 by 17 with no solutions	Z	9.1
11	21 by 18 with 3 solutions	Z[x,y]	815
12	52 by 16 with 79 solutions	Z	880
13	104 by 36 with 2 solutions	Z	287

7. Conclusion.

Maple, even in its present state, can solve general systems of algebraic equations which are far beyond the grasp of other systems. This alone should be a strong indication of power of the presented scheme. Furthermore, we believe that this approach is similar to the approach taken by a mathematician when solving difficult problems: looking for the easiest part and attacking it with the most suitable tool. These problems are all exponential space hard in their most general form, so an exponential behaviour for the general case is unavoidable. Yet, sometimes we can either afford the exponential behaviour or be confident that the solver will be "clever" enough to find a shortcut.

(1) Reduce Crashed with an "Illegal Instruction" error.
(2) Reduce is not programmed to handle non-square linear systems.
(3) Reduce ran out of heap space.
† Macsyma obtained floating point approximations to the solutions for problem 8 after 138 seconds. Macsyma does not attempt to find the exact symbolic solutions when irreducible polynomials which are not linear, quadratic or bi-quadratic are encountered.

8. Appendix 1.

The following are descriptions of the sample problems. A complete listing of the problems, which are too lengthy for publication here, will be made available at the conference and via electronic mail.

1 A dense 20 by 20 linear system with random 10 digit integer coefficients.

2 A moderately sparse linear system of 81 equations in 81 unknowns with integer coefficients. The solution defines the degree 8 uniformly spaced knots B-Spline.

3 A sparse system of 147 equations in 147 unknowns with rational coefficients[Ziv82] from the third order analysis of 2-3 trees.

4 A sparse system of 289 equations in 289 unknowns with rational coefficients.[Fla86] This system is extremely ill-conditioned.

5 A dense linear system of 5 equations in 5 unknowns with polynomial coefficients in five parameters.[Dev85]

6 A very sparse linear system of 147 equations in 49 unknowns with small rational function coefficients in three parameters.[Ste84] This solves a truncated power series solution of a particular differential equation.

7 A sparse 21 by 21 linear system in 5 parameters. The solution defines a general cubic Beta-Spline.[Dev85a]

8 A homogeneous polynomial system of 3 equations in 3 unknowns[Pav85] which has two real and six complex solutions.

9 A sparse polynomial system of 19 equations in 17 unknowns[Gon83] with three solutions. This problem and the next one arise in the study of iterative formulas to compute the inverse of a number.

10 A sparse polynomial system of 22 equations in 17 unknowns[Gon83] with no solution.

11 A sparse polynomial system of 21 equations in 18 unknowns and two free parameters.[All85] See Appendix 2.

12 A sparse polynomial system of 52 equations in 16 unknowns[Gon86] with 79 solutions. Note: two inequalities were specified to avoid symmetric solutions.

13 A very sparse polynomial system of 104 equations in 36 unknowns.[Sha86]

Appendix 2.

The system is defined as follows. Let $m(k,a)$, a real, be defined recursively by:

$$m(k,a) = \frac{(1+a)^{\overline{k}}}{t_{kk}} - \sum_{i=0}^{k-1} \frac{t_{ki} m(i,a)}{t_{kk}} \qquad (1)$$

where $m(0,a) = 1$ and $(1+a)^{\overline{k}} = (1+a)(2+a) \cdots (k+a)$, is the ascending factorial. Thus $m(k,a)$ has the following properties:

a) it is a polynomial of degree k in a,
b) it is linear in t_{ij} for $0 \leq j < i \leq k$,
c) it is linear in $1/t_{ii}$ for $1 \leq i \leq k$.

Also we know that

$$\sum_{k=0}^{n} \frac{(-1)^k (1+a)^{\bar{n}}}{k!(n-k)!(1+a)^{\bar{k}}} \sum_{i=0}^{k} t_{ki} m(i+j,a) \equiv 0, \tag{2}$$

for $n=2,3,4,\ldots$, $1 \leq j < n$.

Equation (1) can be used to eliminate $m(i+j,a)$ from equation (2) thus obtaining a polynomial in a of degree $n+j$, which is identically equal to zero. By setting the coefficients of the left-hand-side of (2) equal to zero for $n=2,j=1$; $n=3,j=1,2$; $n=4,j=1$; we obtain 21 non-linear equations in the 21 unknowns $t_{ij,}$ $0 \leq j \leq i \leq 5$. Now let $t_{00}=1$, $t_{10}=y$ and $t_{11}=x$.

Solve the 21 equations in the 18 remaining unknowns t_{ij} in terms of x and y.

9. Appendix 3.

Appendix 3 contains a listing of examples 1 through 10, 12 and 13.

```
# Example 1 : a dense system of 20 linear equations
c := rand(-10^10..10^10):
for i to 20 do eqns[i] := sum('c()*x.k-c()',k=1..20) od:
eqns := {eqns[k] $ k=1..20};
vars := {'x.k' $ k=1..20};


# Example 2 : 81 linear equations in 81 unknowns
eqns := {5040*c47+40320*c48-5040*c57, 5040*c87+40320*c88-5040*c97, 5040*c77+
40320*c78-5040*c87, 720*c16, 5040*c67+40320*c68-5040*c77, 720*c96+5040*c97+
20160*c98, c10, c11, 5040*c57+40320*c58-5040*c67, c10+c11+c12+c13+c14+c15+c16+
c17+c18-c20, c11+2*c12+3*c13+4*c14+5*c15+6*c16+7*c17+8*c18-c21, 2*c12+6*c13+
12*c14+20*c15+30*c16+42*c17+56*c18-2*c22, 5040*c17+40320*c18-5040*c27, 6*c13+
24*c14+60*c15+120*c16+210*c17+336*c18-6*c23, 24*c14+120*c15+360*c16+840*c17+
1680*c18-24*c24, 120*c15+720*c16+2520*c17+6720*c18-120*c25, 720*c16+5040*c17+
20160*c18-720*c26, c20+c21+c22+c23+c24+c25+c26+c27+c28-c30, c21+2*c22+3*c23+
4*c24+5*c25+6*c26+7*c27+8*c28-c31, 2*c22+6*c23+12*c24+20*c25+30*c26+42*c27+
56*c28-2*c32, 6*c23+24*c24+60*c25+120*c26+210*c27+336*c28-6*c33, 24*c24+120*c25+
360*c26+840*c27+1680*c28-24*c34, 120*c25+720*c26+2520*c27+6720*c28-120*c35,
5040*c27+40320*c28-5040*c37, 5040*c37+40320*c38-5040*c47, 720*c26+5040*c27+
20160*c28-720*c36, c30+c31+c32+c33+c34+c35+c36+c37+c38-c40, c31+2*c32+3*c33+
4*c34+5*c35+6*c36+7*c37+8*c38-c41, 2*c32+6*c33+12*c34+20*c35+30*c36+42*c37+
56*c38-2*c42, 6*c33+24*c34+60*c35+120*c36+210*c37+336*c38-6*c43, 24*c34+120*c35+
360*c36+840*c37+1680*c38-24*c44, 120*c35+720*c36+2520*c37+6720*c38-120*c45,
720*c36+5040*c37+20160*c38-720*c46, c40+c41+c42+c43+c44+c45+c46+c47+c48-c50,
c41+2*c42+3*c43+4*c44+5*c45+6*c46+7*c47+8*c48-c51, 2*c42+6*c43+12*c44+20*c45+
30*c46+42*c47+56*c48-2*c52, 6*c43+24*c44+60*c45+120*c46+210*c47+336*c48-6*c53,
24*c44+120*c45+360*c46+840*c47+1680*c48-24*c54, 120*c45+720*c46+2520*c47+
6720*c48-120*c55, 720*c46+5040*c47+20160*c48-720*c56, c50+c51+c52+c53+c54+c55+
c56+c57+c58-c60, c51+2*c52+3*c53+4*c54+5*c55+6*c56+7*c57+8*c58-c61, 2*c52+6*c53+
12*c54+20*c55+30*c56+42*c57+56*c58-2*c62, 6*c53+24*c54+60*c55+120*c56+210*c57+
336*c58-6*c63, 24*c54+120*c55+360*c56+840*c57+1680*c58-24*c64, 120*c55+720*c56+
2520*c57+6720*c58-120*c65, 720*c56+5040*c57+20160*c58-720*c66, c60+c61+c62+c63+
c64+c65+c66+c67+c68-c70, c61+2*c62+3*c63+4*c64+5*c65+6*c66+7*c67+8*c68-c71,
2*c62+6*c63+12*c64+20*c65+30*c66+42*c67+56*c68-2*c72, 6*c63+24*c64+60*c65+
120*c66+210*c67+336*c68-6*c73, 24*c64+120*c65+360*c66+840*c67+1680*c68-24*c74,
120*c65+720*c66+2520*c67+6720*c68-120*c75, 720*c66+5040*c67+20160*c68-720*c76,
c70+c71+c72+c73+c74+c75+c76+c77+c78-c80, c71+2*c72+3*c73+4*c74+5*c75+6*c76+
7*c77+8*c78-c81, 2*c72+6*c73+12*c74+20*c75+30*c76+42*c77+56*c78-2*c82, 6*c73+
24*c74+60*c75+120*c76+210*c77+336*c78-6*c83, 24*c74+120*c75+360*c76+840*c77+
1680*c78-24*c84, 120*c75+720*c76+2520*c77+6720*c78-120*c85, 720*c76+5040*c77+
20160*c78-720*c86, c80+c81+c82+c83+c84+c85+c86+c87+c88-c90, c81+2*c82+3*c83+
4*c84+5*c85+6*c86+7*c87+8*c88-c91, 2*c82+6*c83+12*c84+20*c85+30*c86+42*c87+
56*c88-2*c92, 6*c83+24*c84+60*c85+120*c86+210*c87+336*c88-6*c93, 24*c84+120*c85+
360*c86+840*c87+1680*c88-24*c94, 120*c85+720*c86+2520*c87+6720*c88-120*c95,
720*c86+5040*c87+20160*c88-720*c96, 24*c14, 5040*c97+40320*c98, 2*c12, 120*c15,
6*c13, 5040*c17, c91+2*c92+3*c93+4*c94+5*c95+6*c96+7*c97+8*c98, 2*c92+6*c93+
12*c94+20*c95+30*c96+42*c97+56*c98, 6*c93+24*c94+60*c95+120*c96+210*c97+336*c98,
 24*c94+120*c95+360*c96+840*c97+1680*c98, 120*c95+720*c96+2520*c97+6720*c98,
c90+c91+c92+c93+c94+c95+c96+c97+c98, c10+c20+c30+c40+c50+c60+c70+c80+c90-1}:
```

```
# Example 3 : 147 linear equations in 148 unknowns.
# The solutions represent probabilities and hence should sum to 1
for i from 4 to 9 do for j from i to 9 do
        eq.j.i := 0;
        eq.i.j := - (i+j+1) * p.i.j od od;
for i from 4 to 9 do
        for j from 4 to 9 do
                for k from i to 9 do
                        eq.k.j.i := 0;
                        eq.i.j.k := - (i+j+k+1) * p.i.j.k od od od;
nn[4] := [[[5],4]]:
nn[5] := [[[6],5]]:
nn[6] := [[[7],6]]:
nn[7] := [[[4,4],3],[[8],4]]:
nn[8] := [[[4,5],3],[[5,4],3],[[9],2]]:
nn[9] := [[[5,5],3],[[4,6],3],[[6,4],3]]:
for i from 4 to 9 do for j from i to 9 do
                n := i+j;
                # leftmost component
                for l to nops(nn[i]) do
                        transf := nn[i][l];
                        p := p.i.j * transf[2]/n;
                        if nops(transf[1])=1 then
                                eq.(i+1).j := eq.(i+1).j + p*(i+j+1)
                        else    i1 := transf[1][1];
                                i2 := transf[1][2];
                                eq.i1.i2.j := eq.i1.i2.j + p*(i+j+1);
                                fi
                        od;
                # rightmost component
                for l to nops(nn[j]) do
                        transf := nn[j][l];
                        p := p.i.j * transf[2]/n;
                        if nops(transf[1])=1 then
                                eq.i.(j+1) := eq.i.(j+1) + p*(i+j+1)
                        else    j1 := transf[1][1];
                                j2 := transf[1][2];
                                eq.i.j1.j2 := eq.i.j1.j2 + p*(i+j+1);
                                fi
                        od;
        od od;
for i from 4 to 9 do
        for j from 4 to 9 do
                for k from i to 9 do
                        n := i+j+k;
                        # leftmost component
                        for l to nops(nn[i]) do
                                transf := nn[i][l];
                                p := p.i.j.k * transf[2]/n;
                                if nops(transf[1])=1 then
                                   eq.(i+1).j.k := eq.(i+1).j.k + p*(i+j+k+1)
                                else    i1 := transf[1][1];
                                        i2 := transf[1][2];
                                        eq.i1.i2 := eq.i1.i2 + p*(i1+i2);
                                        eq.j.k := eq.j.k + p*(j+k)
                                        fi
                od;
            # central component
            for l to nops(nn[j]) do
                transf := nn[j][l];
                p := p.i.j.k * transf[2]/n;
                if nops(transf[1])=1 then
                  eq.i.(j+1).k := eq.i.(j+1).k + p*(i+j+k+1)
                else  j1 := transf[1][1];
                      j2 := transf[1][2];
                      eq.i.j1 := eq.i.j1 + p*(i+j1);
                      eq.j2.k := eq.j2.k + p*(j2+k)
                      fi
                od;
            # rightmost component
            for l to nops(nn[k]) do
                transf := nn[k][l];
                p := p.i.j.k * transf[2]/n;
                if nops(transf[1])=1 then
                  eq.i.j.(k+1) := eq.i.j.(k+1) + p*(i+j+k+1)
                else  k1 := transf[1][1];
                      k2 := transf[1][2];
                      eq.i.j := eq.i.j + p*(i+j);
                      eq.k1.k2 := eq.k1.k2 + p*(k1+k2)
                      fi
                od;
        od od od;
for i from 4 to 9 do for j from i+1 to 9 do eq.i.j := eq.i.j + eq.j.i od od:
for i from 4 to 9 do
    for j from 4 to 9 do
        for k from i+1 to 9 do
            eq.i.j.k := eq.i.j.k + eq.k.j.i od od od;
tot := 0:
eqns := {}:
for i from 4 to 9 do for j from i to 9 do
    tot := tot + p.i.j;
    eqns := eqns union {eq.i.j} od od;
for i from 4 to 9 do
    for j from 4 to 9 do
        for k from i to 9 do
        tot := tot + p.i.j.k;
            eqns := eqns union {eq.i.j.k} od od od;
vars := map(op,map(indets,eqns)):
```

```
# Example 4 : 289 linear equations in 289 unknowns, sparse
# m is the number of memory cells
m := 32;
p := 1/6;
q := 1/2-p;
for i from 0 to m do
        for j from 0 to m-i do
                eq.i.ww.j := -p.i.ww.j od od;
#       Add p contributions
for i to m do
        for j from 0 to m-i do
                eq.i.ww.j := eq.i.ww.j + p * p.(i-1).ww.j;
                eq.j.ww.i := eq.j.ww.i + p * p.j.ww.(i-1);
                od od;
#       Add q contributions
for i from 0 to m-2 do
        for j from 0 to m-2-i do
                eq.i.ww.j := eq.i.ww.j + q*p.(i+1).ww.j + q*p.i.ww.(j+1) od od;
#       Do side contributions
for i from 0 to m-1 do
        eq.i.ww.0 := eq.i.ww.0 + q*p.i.ww.0;
        eq.0.ww.i := eq.0.ww.i + q*p.0.ww.i;
        od;
#       Equation for sink/link
eqs := -ps;
for i from 0 to m do eqs := eqs + p.(m-i).ww.i od;
eq.0.ww.0 := eq.0.ww.0 + ps;
#       Equate symmetric variables
for i to m do
        for j from 0 to i-1 do
                p.j.ww.i := p.i.ww.j od od;
#       Set of all indeterminates and equations
eqns := {}:
vars := {};
for i from 0 to m do
        eqns := eqns union {eq.i.ww.(0..m-i)};
        vars  := vars  union {p.i.ww.(0..m-i)};
        od;
ps := 1;
```

```
# Example 5 : a dense linear system with free parameters a1,a2,a3,a4,a5
eqn1 := (a4^4 -4*a4^2*a3 + 4*a4*a2 + 2*a3^2 -4*a1) *x4 + (-a4^3 + 3*a4*a3
  - 3*a2)*x3  + (a4^2 - 2*a3)*x2 - a4*x1 + 5*x0 = 0:
eqn2 := (a4^3*a3 - a4^2*a2 - 3*a4*a3^2 + 5*a3*a2 + a4*a1 - 5*a0) *x4 +
  (-a4^2*a3 + 2*a3^2 + a4*a2 - 4*a1) *x3 + (a4*a3 -3*a2)*x2 - 2*a3*x1 +
  4*a4*x0 = 0:
eqn3 := (a4^3*a2 -a4^2*a1 -3*a4*a3*a2 +3*a2^2 + 2*a3*a1 + a4*a0) *x4 +
  (-a4^2*a2 + 2*a3*a2 + a4*a1 - 5*a0)*x3 + (a4*a2 - 4*a1)*x2 - 3*a2*x1
  + 3*a3*x0 = 0:
eqn4 := (a4^3*a1 - a4^2*a0 - 3*a4*a3*a1 + 3*a2*a1 + 2*a3*a0)*x4 +
  (-a4^2*a1 + 2*a3*a1 + a4*a0)*x3 + (a4*a1 -5*a0)*x2 - 4*a1*x1 + 2*a2*x0 = 0:
eqn5 := (a4^3*a0 - 3*a4*a3*a0 + 3*a2*a0)*x4 + (-a4^2*a0 + 2*a3*a0)*x3 +
  a4*a0*x2 - 5*a0*x1 + a1*x0 = 1:
eqns := {eqn.(1..5)};
vars := {x.(0..4)};


# Example 6 : A very sparse linear system of 147 equations in 49 unknowns
eqns:={
 -b*k8/a+c*k8/a, -b*k11/a+c*k11/a, -b*k10/a+c*k10/a+k2,
 -k3-b*k9/a+c*k9/a, -b*k14/a+c*k14/a, -b*k15/a+c*k15/a,
 -b*k18/a+c*k18/a-k2, -b*k17/a+c*k17/a, -b*k16/a+c*k16/a+k4,
 -b*k13/a+c*k13/a-b*k21/a+c*k21/a+b*k5/a-c*k5/a,
  b*k44/a-c*k44/a, -b*k45/a+c*k45/a, -b*k20/a+c*k20/a,
 -b*k44/a+c*k44/a, b*k46/a-c*k46/a,
  b^2*k47/a^2-2*b*c*k47/a^2+c^2*k47/a^2,
  k3, -k4, -b*k12/a+c*k12/a-a*k6/b+c*k6/b,
 -b*k19/a+c*k19/a+a*k7/c-b*k7/c, b*k45/a-c*k45/a,
 -b*k46/a+c*k46/a, -k48+c*k48/a+c*k48/b-c^2*k48/(a*b),
 -k49+b*k49/a+b*k49/c-b^2*k49/(a*c), a*k1/b-c*k1/b,
  a*k4/b-c*k4/b, a*k3/b-c*k3/b+k9, -k10+a*k2/b-c*k2/b,
  a*k7/b-c*k7/b, -k9, k11, b*k12/a-c*k12/a+a*k6/b-c*k6/b,
  a*k15/b-c*k15/b, k10+a*k18/b-c*k18/b,
 -k11+a*k17/b-c*k17/b, a*k16/b-c*k16/b,
 -a*k13/b+c*k13/b+a*k21/b-c*k21/b+a*k5/b-c*k5/b,
 -a*k44/b+c*k44/b, a*k45/b-c*k45/b,
  a*k14/c-b*k14/c+a*k20/b-c*k20/b, a*k44/b-c*k44/b,
 -a*k46/b+c*k46/b, -k47+c*k47/a+c*k47/b-c^2*k47/(a*b),
  a*k19/b-c*k19/b, -a*k45/b+c*k45/b, a*k46/b-c*k46/b,
  a^2*k48/b^2-2*a*c*k48/b^2+c^2*k48/b^2,
 -k49+a*k49/b+a*k49/c-a^2*k49/(b*c), k16, -k17,
 -a*k1/c+b*k1/c, -k16-a*k4/c+b*k4/c, -a*k3/c+b*k3/c,
  k18-a*k2/c+b*k2/c, b*k19/a-c*k19/a-a*k7/c+b*k7/c,
 -a*k6/c+b*k6/c, -a*k8/c+b*k8/c, -a*k11/c+b*k11/c+k17,
 -a*k10/c+b*k10/c-k18, -a*k9/c+b*k9/c,
 -a*k14/c+b*k14/c-a*k20/b+c*k20/b,
 -a*k13/c+b*k13/c+a*k21/c-b*k21/c-a*k5/c+b*k5/c,
  a*k44/c-b*k44/c, -a*k45/c+b*k45/c, -a*k44/c+b*k44/c,
  a*k46/c-b*k46/c, -k47+b*k47/a+b*k47/c-b^2*k47/(a*c),
 -a*k12/c+b*k12/c, a*k45/c-b*k45/c, -a*k46/c+b*k46/c,
 -k48+a*k48/b+a*k48/c-a^2*k48/(b*c),
  a^2*k49/c^2-2*a*b*k49/c^2+b^2*k49/c^2, k8, k11, -k15,
  k10-k18, -k17, k9, -k16, -k29, k14-k32, -k21+k23-k31,
 -k24-k30, -k35, k44, -k45, k36, k13-k23+k39, -k20+k38,
  k25+k37, b*k26/a-c*k26/a-k34+k42, -2*k44, k45, k46,
  b*k47/a-c*k47/a, k41, k44, -k46, -b*k47/a+c*k47/a,
 k12+k24, -k19-k25, -a*k27/b+c*k27/b-k33, k45, -k46,
-a*k48/b+c*k48/b, a*k28/c-b*k28/c+k40, -k45, k46,
 a*k48/b-c*k48/b, a*k49/c-b*k49/c, -a*k49/c+b*k49/c,
-k1, -k4, -k3, k15, k18-k2, k17, k16, k22, k25-k7,
 k24+k30, k21+k23-k31, k28, -k44, k45, -k30-k6, k20+k32,
 k27+b*k33/a-c*k33/a, k44, -k46, -b*k47/a+c*k47/a, -k36,
 k31-k39-k5, -k32-k38, k19-k37, k26-a*k34/b+c*k34/b-k42,
 k44, -2*k45, k46, a*k48/b-c*k48/b, a*k35/c-b*k35/c-k41,
-k44, k46, b*k47/a-c*k47/a, -a*k49/c+b*k49/c, -k40, k45,
-k46, -a*k48/b+c*k48/b, a*k49/c-b*k49/c, k1, k4, k3, -k8,
-k11, -k10+k2, -k9, k37+k7, -k14-k38, -k22, -k25-k37, -k24+k6,
-k13-k23+k39, -k28+b*k40/a-c*k40/a, k44, -k45, -k27, -k44,
 k46, b*k47/a-c*k47/a, k29, k32+k38, k31-k39+k5, -k12+k30,
 k35-a*k41/b+c*k41/b, -k44, k45, -k26+k34+a*k42/c-b*k42/c,
 k44, k45, -2*k46, -b*k47/a+c*k47/a, -a*k48/b+c*k48/b,
 a*k49/c-b*k49/c, k33, -k45, k46, a*k48/b-c*k48/b,
-a*k49/c+b*k49/c }:

vars := {k1, k2, k3, k4, k5, k6, k7, k8, k9, k10, k11, k12, k13, k14, k15,
  k16, k17, k18, k19, k20, k21, k22, k23, k24, k25, k26, k27, k28, k29,
  k30, k31, k32, k33, k34, k35, k36, k37, k38, k39, k40, k41, k42, k43,
  k44, k45, k46, k47, k48, k49}:
```

```
# Example 7 : A system of 21 linear equations which arising in the study
# of modelling curves using cubic Beta-Splines in Computer Graphics.
eqns := {C15+C22-1, C14+C21-1, C06+C13+C20-1, 2*b5^2*C24-4*b5^2*C25+
2*b5^2*C26-2*C26, b2*C10+C10-C11, 2*b3^2*C20-2*C20+4*C21-2*C22, b3*C20+C20-
C21, -b2*C02+b2*C03+C03-C04, b1*C00+C00-C01, 2*b2^2*C10-2*C10+4*C11-2*C12,
2*b1^2*C00-2*C00+4*C01-2*C02, -b4*C15+b4*C16+C16, 2*b3^2*C04-4*b3^2*C05+
2*b3^2*C06-2*C06, b5*C25-b5*C26-C26, -b4*C22+b4*C23+C23-C24, 4*C24-2*C25-2*C23+
2*b4^2*C21+2*b4^2*C23-4*b4^2*C22, -b3*C12+b3*C13+C13-C14, 2*b4^2*C14-
4*b4^2*C15+2*b4^2*C16-2*C16, -b3*C05+b3*C06+C06, 4*C14-2*C13-2*C15+2*b3^2*C11+
2*b3^2*C13-4*b3^2*C12, -2*C03+4*C04-2*C05+2*b2^2*C01-4*b2^2*C02+2*b2^2*C03};
vars := {C14, C21, C24, C25, C26, C06, C22, C13, C20, C10, C11, C03, C04, C00,
C01, C12, C05, C15, C02, C23, C16};


# Example 8 : A system of 3 equations in the unknowns a,b and c
eqns := {b^2+(a-1)^2=3, c^2+(b-1)^2=3, a^2+(c-1)^2=3};
vars := {a,b,c};


# Example 9, 19 equations in 17 unknowns with three solutions
eqns:={2*a3*b3+a5*b3+a3*b5,a5*b3+2*a5*b5+a3*b5,a5*b5,a2*b2,a4*b4,a5*b1+b5
+a4*b3+a3*b4,a5*b3+a5*b5+a3*b5+a3*b3,a0*b2+b2+a4*b2+a2*b4+c2+a2*b0+a2*b1,
a0*b0+a0*b1+a0*b4+a3*b2+b0+b1+b4+a4*b0+a4*b1+a2*b5+a4*b4+c1+c4+a5*b2+a2*b3+c0,
-1+a3*b0+a0*b3+a0*b5+a5*b0+b3+b5+a5*b4+a4*b3+a4*b5+a3*b4+a5*b1+a3*b1+c3+c5,
b4+a4*b1,a5*b3+a3*b5,a2*b1+b2,a4*b5+a5*b4,a2*b4+a4*b2,a0*b5+a5*b0+a3*b4+
2*a5*b4+a5*b1+b5+a4*b3+2*a4*b5+c5,a4*b0+2*a4*b4+a2*b5+b4+a4*b1+a5*b2+a0*b4+c4,
c3+a0*b3+2*b3+b5+a4*b3+a3*b0+2*a3*b1+a5*b1+a3*b4,c1+a0*b1+2*b1+a4*b1+
a2*b3+b0+a3*b2+b4};


# Example 10 : 22 equations in 17 unknowns with no solutions
eqns := {2*a3*b3+a5*b3+a3*b5,a5*b3+2*a5*b5+a3*b5,a4*b4,a5*b3+a5*b5+a3*b5+a3*b3,
b1,a3*b3,a2*b2,a5*b5,a5*b1+b5+a4*b3+a3*b4,a0*b2+b2+a4*b2+a2*b4+c2+a2*b0+a2*b1,
b4+a4*b1,b3+a3*b1,a5*b3+a3*b5,a2*b1+b2,a4*b5+a5*b4,a2*b4+a4*b2,
a0*b0+a0*b1+a0*b4+a3*b2+b0+b1+b4+a4*b0+a4*b1+a2*b5+a4*b4+c1+c4+a5*b2+a2*b3+c0,
-1+a3*b0+a0*b3+a0*b5+a5*b0+b3+b5+a5*b4+a4*b3+a4*b5+a3*b4+a5*b1+a3*b1+c3+c5,
a0*b5+a5*b0+a3*b4+2*a5*b4+a5*b1+b5+a4*b3+2*a4*b5+c5,
a4*b0+2*a4*b4+a2*b5+b4+a4*b1+a5*b2+a0*b4+c4,
c3+a0*b3+2*b3+b5+a4*b3+a3*b0+2*a3*b1+a5*b1+a3*b4,
c1+a0*b1+2*b1+a4*b1+a2*b3+b0+a3*b2+b4};
```

```
# Example 12 : 52 equations in 16 unknowns with 79 solutions
eqns:={-a7+a6*a7+a7*b7,-a6+a6^2+a7*b6,a7*b4-a5*b6,a7*b1-a3*b6,a7*b0-
a2*b6,a2*a3-a1*a3,a3*b4-a5*b2,a3*b1-a3*b2,a3*b0-a2*b2,a1*b4-a5*b0,a1*b1-
a3*b0,a1*b0-a2*b0,a5*b6-a7*b4,a3*b6-a7*b1,a2*b6-a7*b0,a6*a5+a7*b5-a4*a7-
a5*b7,a6*a3+a7*b3-a1*a7-a3*b7,a5-a4*a5-a5*b5,a6*a2+a7*b2-a0*a7-a2*b7,-a1*a5+
a3*b7+a2*a7-a3*b5,a6*a2+a3*b6-a0*a5-a2*b5,a2*a5+a3*b5-a4*a3-a5*b3,a2^2+
a3*b2-a2*b3-a0*a3,a4-a5*b4-a4^2,-b7+b7^2+a7*b6,-b6+b6*a6+b7*b6,-a1^2+
a0*a3+a1*b3-a3*b1,a0*a6+a1*b6-a0*a4-a2*b4,a0*a2+a1*b2-a0*a1-a2*b1,a0*a7+
a1*b7-a1*a4-a3*b4,a1*b5-a5*b1+a0*a5-a1*a4,-b4*a6+b6*a4+b7*b4-b5*b6,a1*b6+
b7*b1-b1*a6-b3*b6,b6*a0+b7*b0-b0*a6-b2*b6,a7*b2-a5*b1+b3*b7-b3*b5,b5-a5*b4-
b5^2,b4-b5*b4-b4*a4,a5*b2-a3*b4,b2*a6+b3*b6-a5*b0-b2*b5,-a3*b1+a3*b2,
a2*b2-a3*b0,a5*b0-a1*b4,a3*b0-a1*b1,a2*b0-a1*b0,b1*b0-b2*b0,b2*a4+b3*b4-
a2*b4-b2*b5,a1*b2+b3*b1-a2*b1-b3*b2,-b2^2-a2*b0+b2*a0+b3*b0,b1*b6-b2*b4+
b0*a6-b0*a4,-b1*a4+b7*b1+a7*b0-b3*b4,b1*b4-b5*b0+b0*a4-b4*a0,b1^2+a1*b0-
b1*a0-b3*b0,a1*aa1*bb2+a2*aa2*bb1+a4*aa1+a5*aa2+a6*bb1+a7*bb2-a1*bb1*aa2-
a2*bb2*aa1-a4*bb1-a5*bb2-a6*aa1-a7*aa2<>0,a0*aa1*bb1+a1*aa1*bb2+a2*aa2*bb1+
a3*aa2*bb2+a4*aa1+a5*aa2+a6*bb1+a7*bb2-b0*aa1*bb1-b1*aa1*bb2-b2*aa2*bb1-
b3*aa2*bb2-b4*aa1-b5*aa2-b6*bb1-b7*bb2<>0};
vars:={a0,a1,a2,a3,a4,a5,a6,a7,b0,b1,b2,b3,b4,b5,b6,b7};


# Example 13 : 104 equation in 36 unknowns with two solutions
vars:={a2,a0,ep,a9,a3,a12,a6,a18,a33,a34,a5,a16,a32,a11,a31,
a30,a7,a1,a19,a8,a28,a17,a29,a15,a27,a26,a13,a23,a14,a25,a24,
a21,a10,a22,a20,a4};
eqns:={-1/2*a2+1/2*ep-3*a0*ep=0,-1/2*a9-1/2*a3*ep=0,2*a6*ep-1/2*a18=0,
9/2*a12*ep-1/2*a33=0,7*a34*ep=0,-4-5*ep^2-a5-a2*ep=0,3/2*a9*ep-a16=0,
-a32+4*a18*ep=0,13/2*a33*ep=0,10*ep+a5*ep-3/2*a11=0,-3/2*a31+7/
2*a16*ep=0,6*a32*ep=0,-2*a30+3*a11*ep=0,11/2*a31*ep=0,5*a30*ep=0,
4*a3-1/2*a7-3/2*a1*ep=0,-1/2*a19+a8*ep+8*a6+15*ep=0,12*a12-1/2*a28+7/
2*a17*ep=0,16*a34+6*a29*ep=0,-a15-10*a3*ep+1/2*a7*ep+4*a9=0,8*a18-a27+
3*a19*ep-20*a6*ep=0,-30*a12*ep+12*a33+11/2*a28*ep=0,-40*a34*ep=0,8+
4*a8-1/2*a13+12*ep^2-3*a2*ep=0,5/2*a15*ep-3/2*a26+4*a16-10*a9*ep=0,
-20*a18*ep+8*a32+5*a27*ep=0,-30*a33*ep=0,9/2*a26*ep-10*a16*ep+4*a31=0,
-20*a32*ep=0,-10*a31*ep=0,-3*a9*ep+5/2*a14*ep+8*a17-1/2*a23=0,5*a25*ep+
12*a29-3*a18*ep=0,-3*a33*ep=0,2*a13*ep-10*a8*ep-6*a5*ep-38*ep+4*a19-a24=
0,9/2*a23*ep+8*a28-20*a17*ep-6*a16*ep=0,-30*a29*ep-6*a32*ep=0,4*a27-
10*a19*ep-9*a11*ep+4*a24*ep=0,-9*a31*ep-20*a28*ep=0,-10*a27*ep-12*a30*ep
=0,4*a14-1/2*a21+3/2*a10*ep+6*a3*ep-3*a7*ep=0,-3*a19*ep+12*a6*ep+4*a22*ep+
8*a25=0,-3*a28*ep+18*a12*ep=0,24*a34*ep=0,6*a9*ep-6*a15*ep-10*a14*ep+
4*a23+7/2*a21*ep=0,-6*a27*ep-20*a25*ep+12*a18*ep=0,18*a33*ep=0,
-9*a26*ep-10*a23*ep+6*a16*ep=0,12*a32*ep=0,6*a31*ep=0,-3*a13*ep+
3*a20*ep+12*ep+4*a22+6*a8*ep=0,12*a17*ep-3*a23*ep=0,18*a29*ep=0,
6*a19*ep-10*a22*ep-6*a24*ep=0,12*a28*ep=0,6*a27*ep=0,-3*a21*ep+
6*a14*ep=0,12*a25*ep=0,6*a23*ep=0,ep^2=1,6*a22*ep=0,-1/2*a3-1/
2*a1*ep=0,-a6-1/2*a8*ep-15/2*ep=0,-3/2*a12-1/2*a17*ep=0,-1/2*a29*ep-
2*a34=0,5*a3*ep-1/2*a9-1/2*a7*ep=0,-1/2*a19*ep+10*a6*ep-a18=0,-3/2*a33+
15*a12*ep-1/2*a28*ep=0,20*a34*ep=0,-1/2*a16-1/2*a15*ep+5*a9*ep=0,-a32-
1/2*a27*ep+10*a18*ep=0,15*a33*ep=0,-1/2*a26*ep-1/2*a31+5*a16*ep=0,
10*a32*ep=0,5*a31*ep=0,-1/2*a8-11/2-a4*ep+3*a2*ep=0,-a17+3*a9*ep-
a14*ep=0,3*a18*ep-a25*ep-3/2*a29=0,3*a33*ep=0,49*ep+6*a5*ep-1/2*a19+
5*a8*ep-a13*ep=0,-a23*ep-a28+10*a17*ep+6*a16*ep=0,15*a29*ep+6*a32*ep=0,
-1/2*a27-a24*ep+5*a19*ep+9*a11*ep=0,9*a31*ep+10*a28*ep=0,5*a27*ep+
12*a30*ep=0,-1/2*a14-3/2*a10*ep-12*a3*ep+3*a7*ep=0,3*a19*ep-24*a6*ep-3/
2*a22*ep-a25=0,3*a28*ep-36*a12*ep=0,-48*a34*ep=0,-12*a9*ep+6*a15*ep+
5*a14*ep-1/2*a23-3/2*a21*ep=0,6*a27*ep+10*a25*ep-24*a18*ep=0,-36*a33*ep=
0,9*a26*ep+5*a23*ep-12*a16*ep=0,-24*a32*ep=0,-12*a31*ep=0,-12*a8*ep-
42*ep+3*a13*ep-1/2*a22-2*a20*ep=0,3*a23*ep-24*a17*ep=0,-36*a29*ep=0,
5*a22*ep-12*a19*ep+6*a24*ep=0,-24*a28*ep=0,-12*a27*ep=0,3*a21*ep-
12*a14*ep=0,-24*a25*ep=0,-12*a23*ep=0,-12*a22*ep=0};
```

10. References

All85. William Allaway, Private Communication 1985.

Bar68. E.H. Bareiss, Sylvester's Identity and Multistep Integer Preserving Gaussian Elimination, *Mathematics of Computation* **22**(1968).

Buc84. Bruno Buchberger, Groebner Basis: An Algorithmic Method in Polynomial Ideal Theory, in *Recent Trends in Multidimensional System Theory*, ed. N. K. Bose, D. Reidel (1984). to appear

Cha83. B.W. Char, K.O. Geddes, W.M. Gentleman, and G.H. Gonnet, The design of Maple: A compact, portable, and powerful computer algebra system, *Proceedings of Eurocal '83*, pp. 101-115 (April, 1983). Springer-Verlag Lecture Notes in Computer Science no. 162.

Cha84. Bruce W. Char, Gregory J. Fee, Keith O. Geddes, Gaston H. Gonnet, Michael B. Monagan, and Stephen M. Watt, On the Design and Performance of the Maple System, pp. 199-219 in *Proceedings of the 1984 Macsyma User's Conference*, (July, 1984). University of Waterloo Computer Science Department Research Report CS-84-13.

Cza86. Steve R. Czapor and Keith O. Geddes, *On Implementing Buchberger's Algorithm for Grobner Bases*, To appear in SYMSAC July 1986.

Dev85a. Michel P. Devine, Simulating Beta Splines with Multiple-Knot B-Splines., *CS788 Course Project*, Department of Computer Science, University of Waterloo., (August 1985).

Dev85. J. Stan Devitt, Private communication 1985.

Fla86. Philippe Flajolet, Solution to Steady State Doubly-Ended Pushdown Stack, *Proceedings of Math. Foundations of Computer Science*, (1986). Bratislava,

Gen76. W.M. Gentleman and S.C. Johnson, Analysis of Algorithms, A Case Study: Determinants of Matrices with Polynomial Entries, *ACM Transactions on Mathematical Software* **2** pp. 232-241 (September 1976).

Gon83. G.H. Gonnet, B.W. Char, and K.O. Geddes, Solution of a general system of equations, *ACM SIGSAM Bulletin*, (August, 1983).

Gon86. Gaston H. Gonnet, *New Results for Random Determination of Equivalence of Expressions*, To appear in SYMSAC July 1986.

Hea71. Anthony C. Hearn, Reduce 2: A System and Language for Algebraic Manipulation, *Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation*, pp. 128-133 Association for Computing Machinery, (1971).

Hor75. E. Horowitz and S. Sahni, On Computing the Exact Determinant of Matrices with Polynomial Entires, *Journal of the Association for Computing Machinery* **22**(1) pp. 38-50 (January 1975).

Kal85. Eric Kaltofen., Computing with Polynomials Given by Straight-Line Programs: I Greatest Common Divisors, *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, (May 1985).

Knu81. Donald E. Knuth, *Seminumerical Algorithms, 2nd edition*, Addison-Wesley, Menlo Park, California (1981). The Art of Computer Programming, vol. 2.

Lip68. John Lipson, Symbolic Methods for the Computer Solution of Linear Equations With Applications to Flow Graphs., *Proc. Summer Institute on Symbolic Mathematical Computation.*, (1968).

McC73. Michael T. McClellan, The Exact Solution of Systems of Linear Equations, *JACM* **20**(4)(1973).

Mos74. Joel Moses, Macsyma - The Fifth Year, *Proceedings of the Eurosam 74 Conference*, (August 1974).

Pav85. Richard Pavelle, *Macsyma Newsletter* **II**(4) pp. 5-6 (Oct 1985).

Rim84. Ken Rimey, Problem Section (Rotating Fluids): A System of Polynomial Equations and a Solution by an Unusual Method, *Sigsam bulletin* **18**(1) pp. 30-32 ACM, (February 1984).

Sha86. W. Shadwick, Private Communication 1986.

Ste84. Stanly Steinberg, *Very Large Linear system of Equations*, Private communication 1984.

Ziv82. Nivio Ziviani, The Fringe Analysis of Search Trees, *PhD Thesis*, Department of Computer Science, University of Waterloo., (1982).