The Journal of the Astronautical Sciences, Vol. 46, No. 1, January-March 1998, pp. 65–81

# Automatic Rendering of Astrodynamics Expressions for Efficient Evaluation

**Liam M. Healy[1] and Jeffrey J. Travisano[2]**

**Abstract**

In this paper, we describe the automatic rendering of expressions computed using symbolic manipulation. Computations from astrodynamics frequently can be put in a fixed hierarchy of polynomials and Fourier series. Once in this form, FORTRAN subprograms can be generated automatically in a form that lends itself to numerical evaluation. The goal of the current work is to present an approach for using symbolic manipulation techniques to produce a Fortran representation of the normalized Hamiltonian and other supporting equations representing as many of the actual physical effects on satellites as possible.

## Introduction

One of the motivations for the development of von Zeipel-Poincaré normalization in celestial mechanics is the desire for efficient numerical prediction of satellite orbits; more efficient, at least, than numerical integration. A key element then is the rendering of the analytical expressions to source code for a numerical program. If the symbolic computation is done with a computer algebra package, the potential for direct computerized generation of efficient numerical code is great. The correct rendering of the expressions as compilable subprograms has several aspects. We describe here a program, LANGTRAN, that performs these tasks and has been used for the generation of a sophisticated orbit propagator.

In order to modernize the propagator used for satellite catalog maintenance, theories that represent an improvement on Brouwer's theory [1] are needed. Coffey et al. [2] describe the program AOPP with an improved geopotential model for propagation. LANGTRAN was used to generate the FORTRAN source code representing the elimination of the parallax transformation [3], the short-period transformation through the geopotential $J_9$, and the long-period transformation (elimination of perigee) through geopotential $J_{16}$, as well as the final secular

[1]Research Physicist, Naval Research Laboratory, Code 8233, Washington, DC 20375-5355.
[2]Senior Software Engineer, The Boeing Company, 7483 Candlewood Road, Suite 100L, Hanover, MD 21076; work completed while employed by AlliedSignal Corporation at the Naval Research Laboratory.

equations of motion. Additionally, the tesseral harmonics and lunar perturbation were treated through degree four, and solar perturbation through degree two. In total, this encompasses tens of thousands of terms (see Table 1). By contrast, Brouwer's development involved, for the short-period transformation, 75 terms; for the long-period transformation, 155 terms; and for the secular (normalized) Hamiltonian, 43 terms.

This paper describes LANGTRAN, a program written in Common LISP [4] for automatically generating (from expressions developed in symbolic algebra) working numerical subroutines for execution in a larger program. The primary motivation for its development was the AOPP program; we shall make reference to the AOPP code generated but because of its size, it is not included here. Instead, part of a propagator for a simple $J_2$ perturbation is used for illustrative purposes in this paper.

The difficulty of producing source code for numerical evaluation of astrodynamics expressions has received attention in the past. Eckert et al. [5] dealt with some of the problems in the 1960s such as substitution of numerical values and common factor removal. Coffey and Deprit [6] and later Miller [7] addressed the issue of numerical evaluation and Fourier series. Here, we consider expressions that are dominated by polynomial evaluation, though Fourier series may be present.

In the context of general expression generation, Gates and van Hulzen [8, 9] developed the program GENTRAN to work with symbolic algebra systems such as MACSYMA and REDUCE to render arbitrary expressions as FORTRAN source code. This is an immense task; the nature of astrodynamics expressions are such that the full range of mathematics need not be addressed. On the other hand, since expressions we do have can stretch in length to tens or even hundreds of

**TABLE 1. Size of Series in AOPP Theory[1]**

| Transformation | Number of Terms |
|---|---|
| Parallax transformation | |
| $J_2$ First order | 40 |
| $J_9$ Second order | 27671 |
| Short-period transformation | |
| $J_2$ First order | 84 |
| $J_9$ Second order | 3495 |
| Long-period transformation | |
| $J_9$ First order | 2611 |
| $J_{16}$ First order | 51327 |
| Normalized Hamiltonian | |
| $J_9$ Second order | 134 |
| $J_{16}$ Second order | 708 |
| $J_9$ Third order | 9793 |
| Tesseral (3 iterations) transformations | |
| Degree 2 | 994 |
| Degree 3 | 6074 |
| Degree 4 | 13262 |
| Moon, $P_2$, $P_3$, $P_3$ | 10448 |
| Sun, $P_2$ | 1665 |

[1]Not all these series will ultimately be used.

thousands of terms (see Table 1), many of the techniques Gates and van Hulzen use (such as common subexpression elimination) are used here.

The computer algebra system used here is MAO, a system developed for LISP machines expressly for the purpose of manipulating large algebraic expressions of a regular form [10]. MAO is a structured algebraic manipulator; a hierarchy of algebras (polynomial, Fourier, or lazy series) is specified, each with coefficients in another defined algebra. Such a fixed structure makes it relatively easy to generate source code for numerical evaluation.

These techniques are meant to automate the process of converting analytically derived expressions into numerical source code that can be executed efficiently, often many thousands of times a day for years or decades on end. Obviously, important considerations are error-free conversion of analytical results into source code, arrangement of expressions for optimal evaluation, particularly techniques of a nature that standard compiler optimization features cannot mimic, and expression in a form that all compilers can handle. A manual coding strategy is acceptable for very short expressions, but is unworkable for the lengths of expressions we consider here. A semi-automated strategy has been used in the past, but is still subject to error and would not combine all the techniques described here.

The essential steps LANGTRAN uses to convert expressions from MAO are summarized below. There are two alternate forms in which it can put expressions, the *common subexpression elimination* and the *array polynomial.* For either form, there are in addition several steps in common.

1. For the common subexpression elimination

   (a) Convert expression from MAO to LISP.
   (b) Apply any special simplifications specified.
   (c) Scan for common subexpressions, and remove by defining a variable equal to that expression at the beginning.

   For array polynomial,

   (a) Convert expression from MAO to a list of terms, each term expressed as a list of the coefficient and exponents for each of the polynomial variables (LMX form).
   (b) Rearrange monomials so that each can be calculated from previous ones and variables.
   (c) Generate array of coefficients, successive monomial calculations, and loops to sum.

2. Insert explicitly specified intermediate quantities.
3. From a database of celestial mechanics relations, construct all intermediate quantities not explicitly given.
4. Apply some basic simplifications.
5. Convert to FORTRAN (or whatever desired output language).

Generalization to problems outside of celestial mechanics is straightforward, provided that the expressions consist of multivariate polynomials and trigonometric functions of multiple arguments. The only part of this program that is domain-specific is the database of relations used by the function constructor; creation of a new database would allow use in a new application.

## Orbit Theory

The AOPP program is designed to advance the theory used for orbit propagation beyond that described by Brouwer [1]. In particular, there are three main areas of development: higher zonal harmonics, tesseral harmonics, and third body forces. Provided here is a brief summary of the mathematics of AOPP as it pertains to LANGTRAN; it is adapted from Coffey et al. [2].

The goal of this computation is to produce a normalized Hamiltonian representing as many of the actual physical effects on satellites as possible. A normalized, or secular, Hamiltonian, by virtue of its being independent of the coordinates, is easy to propagate; no numerical integration is needed. The normalization is effected with the Lie transformation technique described by Deprit [11]. In order to facilitate the elimination of coordinates, it is sometimes advisable not to do the elimination directly but to go through intermediate stages first. Hence Deprit [3] came up with the technique of the elimination of the parallax, in which the explicit $1/r^n$ terms for $n > 2$ are eliminated but the $1/r^2$ terms remain. Then, in a separate transformation, one would eliminate the $1/r^2$ terms. This is the "short-period" transformation. Finally, one would eliminate the dependence on the argument of perigee. This is the "long-period" transformation, so-called because the argument of perigee changes quite slowly.

The Hamiltonian can be divided into several pieces, $\mathcal{H} = \mathcal{H}_{00} + \mathcal{V} + \mathcal{B}$, where $\mathcal{H}_{00}$ is the two-body Hamiltonian, $\mathcal{V}$ is the geopotential, and $\mathcal{B}$ is the third-body Hamiltonian. The principal term is the Keplerian

$$\mathcal{H}_{00} = \frac{1}{2}\|\mathbf{X}\|^2 - \frac{\mu}{r} \tag{1}$$

where $\mathbf{x}$ and $\mathbf{X}$ stand respectively for the position and the velocity of the satellite, $r$ denotes the radial distance $\|\mathbf{x}\|$, and $\mu$ is the Keplerian constant of the Earth.

$$\mathcal{V} = -\frac{\mu}{r}\sum_{n\geq 2}\left(\frac{\alpha}{r}\right)^n \sum_{0\leq m\leq n}(C_{n,m}\cos m\lambda + S_{n,m}\sin m\lambda)P_{n,m}(\sin\beta) \tag{2}$$

the angles $\lambda$ and $\beta$ are the longitude and latitude of the satellite in an Earth-fixed frame, while $\alpha$ is the radius of the Earth. The function $P_{n,m}(z)$ is the Legendre polynomial of degree $n$ and order $m$. The zonal harmonics are those with $m = 0$; the tesserals are all others. The dimensionless quantities $(\alpha/r)^n$ times $C_{n,m}$ ($C_{n,0}$ is usually designated $-J_n$) or $S_{n,m}$, give the perturbation parameters in which the Hamiltonian is expanded. For the Earth, $J_2$ is taken to be first order, and all the others are taken as second order.

The third-body perturbations (principally the sun and the moon) are represented by a similar expansion $\mathcal{B}$,

$$\mathcal{B} = -\frac{\mu_3}{r_3}\sum_{n\geq 2}\left(\frac{r}{r_3}\right)^n P_n(\cos S_3) \tag{3}$$

where $\mu_3$ is the Keplerian constant of the perturbing body and $S_3$ is the synodic angle between the perturbing body and the satellite, $P_n$ is the Legendre polynomial with $m = 0$, and $r_3$ is the distance to the third body. The perturbation parameter is the ratio of the semi-major axes $a/a_3$.

Nonsingular variables are defined because the traditional variables such as the Whittaker (polar) variables exhibit singularities in the transformations at important orbits like circular ($e = 0$) and equatorial ($I = 0$). The transformations of these variables still exhibit singularities, but they are fewer in number and the location can be picked for each satellite to be far away from its orbit (e.g., at $I = \pi$ for posigrade satellites). Nonsingular variables have the unfortunate consequence of greatly complicating and lengthening the algebra.

The Lie transformation technique computes the generator of the transformations by a succession of integration and Poisson brackets on the Hamiltonian expanded by order. Because of complexity of the expressions, particularly with nonsingular variables, the algebra quickly becomes overwhelming, and computer algebra is a necessity. Table 1, adapted from Coffey et al. [2], shows the number of terms involved in each part of the computation. All these expressions were computed using MAO and rendered in FORTRAN using LANGTRAN.

## LISP to FORTRAN Translation

LANGTRAN provides the following in processing a MAO expression(s) for output as a FORTRAN subprogram for numerical evaluation:

- Generation of complete, proper, and human-readable Fortran subprograms (subroutines, functions, programs) for compilation.
- Minor simplification of mathematics, effected with a pattern matcher and replacement system, essentially that described by Abelson et al. [12].
- Substitution of numerical values, if they never are to be changed in the course of running the numerical program.
- Conversion of numerical types of constants, e.g. from rational to double precision.
- Declaration of all variables including type, dimension (if array), file or (for Fortran 90) module in which included.
- Automatic parallelization of a calculation across different initial values, when producing output in a parallel language.
- Construction of intermediate expressions between what is input and what is needed by the expression(s), based only on a relations table of general known relationships.
- Rearrangement of expressions for compilability and efficiency.

The core system for generation of syntactically correct FORTRAN 77 from LISP is `L4`, written by Bruce Miller of the National Institute of Standards and Technology, as part of a larger system to generate Fourier series for numerical evaluation [7]. This system provides the ability to generate a complete FORTRAN subprogram by invoking a form that specifies the type of subprogram, the inputs, outputs, local variables and comments, and finally the LISP form to be translated. This form may consist of any mathematical operation and function, a few control functions (such as `if`), and limited input and output. This is adequate for the purposes of generating a numerical subroutine.

LANGTRAN is able to generate LISP [4], FORTRAN 77 [13], Fortran 90 [14], Connection Machine FORTRAN (CMF) [15], or Fortran 90 with High Performance Fortran (HPF) [16], the last two for parallelized programs. These programming languages permit parallelism using a fine-grained, or SPMD (Single Program, Multiple Data), model. Most scalars become vectors, and arrays acquire

an extra dimension, so that the same calculation may be repeated on different data. There is assumed to be no communication. For multiple satellite propagation, the calculation for a particular satellite is all done on one processor, though different satellites may be done on different processors. Because there is no interchange of data for separate satellites, there is no need to communicate between the processors.

## Expression Generation

There are a variety of techniques described in the literature for evaluation of polynomials; see, for example, Knuth [17]. Unfortunately, the most popular, such as Horner's rule, become much less useful in complicated sparse multivariate polynomials with Fourier terms, such as ones encountered here. We have thus settled on two alternate forms. The first involves direct generation of the expression, with common subexpression elimination. The second treats the evaluation of a polynomial as the inner product of two arrays.

Common subexpression elimination is fast but tends to generate large numbers of continuation lines, which might cause a problem with the FORTRAN compiler. Array polynomial generation, while usually not generating too many continuation lines, takes significantly more time to generate, especially for large expressions.

## Common Subexpression Elimination

The direct insertion of an expression is the most obvious way to render it in a numerical program. In this case, it is merely necessary to make an accurate translation from the symbolic representation to the appropriate target language. Improvements in efficiency and readability can be achieved, however, with common subexpression elimination. This is a technique, well known in compiler technology [18], for removal and precomputation of expressions that occur more than once in a computation. It is quite easy to do in LISP; one descends the expression tree, saving all expressions in a table, and counting duplicates. Upon generating the output code, if a given expression occurs more than once, a variable is invented for it and it is defined at the beginning of the target code. That variable is then used in place of the expression everywhere it occurs. If an expression is used only once, it is just used directly in the computation.

Simple exponentiation of a variable is a frequently removed subexpression. Since MAO represents polynomials fully expanded, each polynomial variable typically would be exponentiated many times in the course of a computation. By pre-evaluating these exponents, many unnecessary exponentiations may be eliminated. One may consider various optimizations for enhancing the computation of the exponential itself (see Knuth [17]), but as this operation takes up but a small fraction of the overall time, and we typically have consecutive exponents, this optimization was deemed to be not worth the effort of implementation. Furthermore, potential numerical instability due to differencing of close large numbers was not addressed, mainly because the variables of the polynomials can take any value.

For exponent or trigonometric function removal, LANGTRAN will try to generate a sensible variable name, e.g. `SIN2TH` for $\sin 2\theta$ which enhances the readability of the numeric code produced. For more complex expressions, it will generate a sequential name, e.g. `A00001`, `A00002`, etc. The example

shows some intermediate values computed. In this example there were 15 subexpressions eliminated (not all are shown), including `A0001` through `A0006`, `S2`, `S4`, `NNN1`, `PN2`, `PN4`, `J22`, `C2G`, `E2`, and `ETA2`. Experience shows that for larger expressions there is approximately one term eliminated for every four lines (120 characters) of source code. For example, one subroutine developed as part of the AOPP effort had 1333 terms eliminated out of 5631 lines of code.

*Array Polynomial*

The generation of expressions directly with subexpression elimination can lead to practical difficulties for very large expressions because for FORTRAN 77 standard specifies a limitation of nineteen continuation lines. Although most compilers are more generous than this, there is usually still some limitation on the number of continuation lines which can easily be exceeded for even modest calculations.

One solution to this problem is to simply artificially break the expressions when they reach a certain length and accumulate in a sum, as is done by the expression segmentation facility of GENTRAN [8,9]. But we can do better than this, by making use of the character of the expressions we are evaluating. Since we are dealing predominantly in polynomials in a limited number of variables, it is sensible to break up the expression as a sum of terms, each term being a product of a coefficient and a monomial. For example, consider the polynomial

$$\frac{3}{4}x^2 + \frac{7}{8}x^2y + z^4 \tag{4}$$

We may compute this expression in two stages: first, with explicit values of the polynomial variables $x$, $y$, and $z$, we evaluate the monomials, computing in a specific order to take advantage of previously computed monomials. Then, we take the dot product of two arrays, the coefficient array $[\frac{3}{4}, \frac{7}{8}, \ldots]$ and the monomial array, $[x^2, x^2y, \ldots]$. The motivation for doing this is twofold. First, by making the final computation a dot product, we may take advantage of vector processing enhancements prevalent in hardware architectures. Second, with the monomial computation separated, they can be computed more optimally in terms of other monomials, usually as a single product, rather than as products of powers of the variables.

Array polynomial generation works in two stages. The first takes the MAO algebraic object(s), i.e., the expression(s), and regenerates it in the "list MAO expression" or LMX. In this form, each expression is represented as a list of lists, all of the same length. The first is a list of symbols for all the variables. The subsequent lists represent the terms of a fully expanded expression, as a coefficient followed by exponents. Thus equation (4) defined in an algebra with variables $t, x, y, z$ would be `((t x y z) (3/4 0 2 0 0) (7/8 0 2 1 0) (1 0 0 0 4))` in LMX form. All code generation is based on this form.

From the LMX, the path to generating a FORTRAN-like pseudo-LISP which can be subsequently translated into FORTRAN has several steps. First, all common factors are removed: for every monomial variable, the minimum exponent over the polynomial is extracted to serve as a base multiplier; with this information, the exponents relative to the base, and the Fourier angle multipliers for each term are collected.

To facilitate the later numerical calculations, the terms are sorted according to a particular ordering of exponents. This is a partial ordering that puts one monomial after another if the later one can be constructed from the earlier. "Constructed" here means that the later monomial can be computed by multiplying the earlier by nonnegative powers of the variables. For example, $x^2y$ would come before $x^2y^2$ because the latter is obtained from the former by multiplying by $y$. Thus, if a monomial can be computed by multiplying two other monomials together, those two will come prior to it. The listing in the Appendix shows the example as an array polynomial; lines 19 onward are an illustration of ordered monomials.

The terms must be organized so that those that correspond to a single expression are contiguous (when the computation involves more than one expression), and so that polynomial terms multiplying a given trigonometric term are grouped together. The last part of the FORTRAN output module consists of do-loops over each contiguous block of the trigonometric grouping that perform the dot product of coefficient and monomial, with the trigonometric term multiplied at the end, and a final term that includes the common factor. The final result is the sum times the common factor.

The final computational module is constructed from the following information. A DATA statement defines the coefficient values (lines 5–7 in the example). The monomial definitions are generated in the sorted order, and are defined in terms of a pair previous monomials if possible (e.g. line 30). This pair is determined by searching the ordered list of monomials. If such a pair does not exist, the possibility of defining it in terms of a previously-defined monomial and variable(s) is explored (e.g. line 36). Finally, if this is not possible, it is defined entirely in terms of the variables (e.g. line 28). In practice, the vast majority of monomials are defined as the product of a pair of previously-defined monomials, or as a single monomial.

A study of the FORTRAN generated for AOPP in both the common subexpression elimination and array polynomial forms performed by the Silicon Graphics analysis tool CaseVision shows that, as one might suspect, the array polynomial code has fewer arithmetic operations but more loads and stores. With optimization turned off, this creates a penalty for the array polynomial form because each monomial and coefficient must be saved and fetched from memory, a relatively time-consuming process.[1] With optimization in effect, however, the penalty of multiple loads and stores can be mitigated, and the array polynomial form runs faster. Table 2 shows the results of repeated calling of a subroutine for evaluation of a single Poisson bracket in the elimination of perigee, in both array polynomial and common subexpression elimination forms where these results can be clearly seen. However, with larger expressions that require slow memory load/stores, much of the advantage is lost. The source text advantage (few if any continuation lines) remains, of course, and so we have decided to use the array polynomial form for actual code production, regardless of size.

## Function Construction

One of the difficult tasks in doing astrodynamics calculations is conversion between different variables. For example, one may have elements specified in Whittaker variables (radius, angle, right ascension of the ascending node, and their

[1]On some modern processors, 200 instructions can be executed in the time it takes to fetch an out-of-cache word from memory.

**TABLE 2. Comparison of Array Polynomial (AP) and Common Subexpression Elimination (CSE) Methods[1]**

| Quantity | CSE | AP |
|---|---|---|
| Total execution time, no optimization | 4.47 s | 7.20 s |
| Total execution time, with optimization "-O2" | 3.72 s | 3.22 s |
| Floating point operations | 51.8 M | 36.4 M |
| Loads | 70.1 M | 141.0 M |
| Stores | 23.8 M | 76.5 M |
| Instructions | 161.4 M | 395.4 M |

[1]This calculation is part of the computation of a particular Poisson bracket in the elimination perigee ($\{f_{01}, W_1\}$ for $yp_3$); it has 1058 terms and was called 10,000 times to generate these statistics. Counts are all for non-optimized form.

conjugate momenta) but wish to find the eccentricity. A human doing this would try to remember or look up the relevant formulas, perhaps using intermediate quantities before getting the final answer.

In order to smooth the automatic generation of executable numeric programs, we have incorporated a "function constructor" into LANGTRAN. This consists of two pieces: procedures that can build a desired overall relationship (desired quantities as a function of known quantities) from elemental relationships, and a relations table of those elemental relationships applicable to a specific discipline; in this case, celestial mechanics and astrodynamics. Given a list of *known* quantities, it will generate the necessary mathematical expressions, including intermediate quantities, to compute the *needed* quantities.

Consider, for example, the generation of the Delaunay equations of motion given in the example. In calling the LANGTRAN functions, we have supplied only our expressions and the names of quantities that will be input: the array of six Delaunay variables $(l, g, h, L, \Theta, N)$, $J_2$, and the Keplerian constant $\mu$. The expression we give it, however, requires not only $J_2$ and $\mu$ but also the mean motion $n$, the semi-latus rectum $p$, the eccentricity $e$, the sine of the inclination $\sin I$, $\eta = \sqrt{1 - e^2}$, and $\beta = 1/(1 + \eta)$. The function constructor must build all the intermediate relationships from what is supplied to what is needed, using the relations table.

The chain of definitions it creates from the relations table to get the needed quantities is as follows:

$$\eta(\Theta, L) = \Theta/L \qquad \beta(\eta) = 1/(1 + \eta)$$

$$\cos I(N, \Theta) = N/\Theta \qquad \sin I(\cos I) = \sqrt{1 - \cos^2 I}$$

$$e(\eta) = \sqrt{1 - \eta^2} \qquad p(\Theta, \mu) = \Theta^2/\mu$$

$$a(L, \mu) = L^2/\mu \qquad n(a, L) = L/a^2$$

Each equation above corresponds to an entry in the relations table; they will be used in an order such that all needed quantities are produced and each may be calculated from quantities specified previously. There are many other relations in the relations table which the constructor may use and discard in the process of

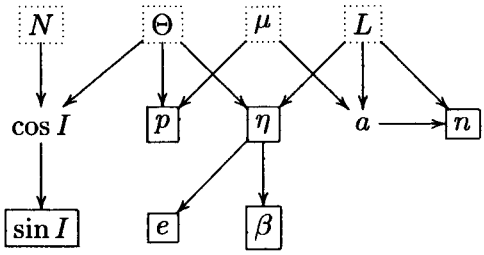


**FIG. 1.** Construction of Quantities Desired (Solid Box) from those Known (Dotted Box).

finding a path to the needed definitions; this is illustrated below. We may look at the quantities as a tree (Fig. 1), starting with the known quantities at the top, and computing successive quantities until we have everything we need. In Fig. 1, the unboxed quantities are only needed as intermediates; they may not appear explicitly in the output because, if they are unneeded elsewhere, they will be substituted by their definition in terms of the boxed quantities.

The resultant FORTRAN is shown in the array polynomial example, lines 11–16 (with S2 representing the square of the sine of the inclination). Quantities that appear only once, such as $a$ and $\beta$ in the example, are substituted directly into the expression, rather than being defined as separate variables. This happens recursively so that the mean motion $n$ ends up defined directly in terms of the momentum $L$ and the Keplerian constant $\mu$.

The function constructor searches the relations table by using a breadth-first search with node queue (see Winston and Horn [19]). We make a list out of all needed quantities; this will be our initial "pathlist." If this pathlist is not a subset of what we have, we get from the relations table all definitions of the first quantity of the first path in the pathlist. For each of the definitions, the quantities needed, together with the definition and all the remaining quantities, becomes a new node. This node is placed at the end of the queue. It is important to place it at the *end* of the queue, making this a breadth-first search, because many, if not most, paths explored are "dead ends" or circular definitions, so we would like to expand successive wanted quantities before we recursively expand definitions pulled from the relations table.

As each node is taken in succession, it is checked to see if the definition chain is complete; that is, it produces all the needed quantities in terms of the known ones. As a shortcut, the entire set of nodes is checked at each recursion to avoid the necessity of checking and expanding each node between the head of the queue and a successful solution. It is possible to set a special (global) variable to indicate that the user should be queried before accepting an expansion. This is to allow control over how the chain is built; there is frequently more than one definition and this allows the selection of a preferred one.

To illustrate, suppose we need the eccentricity $e$ and have an array of the Delaunay variables $\mathbf{D} = (l, g, h, L, \Theta, N)$. Using a restricted relations table that contains only the definitions $L(\mathbf{D})$, $L(a, \mu)$, $p(\Theta, \mu)$, $\eta(p, a)$, $\eta(\Theta, L)$, $a(L, \mu)$, $a(n, \mu)$, $\Theta(\mathbf{D})$, $e(\eta)$ among relevant quantities, the queue looks like the following at each step:

1. Need $e$.
2. Need $\eta$ and use $e(\eta)$.

3. Need $p, a$ and use $\eta(p, a)$, $e(\eta)$.
   Need $\Theta, L$ and use $\eta(\Theta, L)$, $e(\eta)$.
4. Need $\Theta, L$ and use $\eta(\Theta, L)$, $e(\eta)$.
   Need $a, \Theta, \mu$ and use $p(\Theta, \mu)$, $\eta(p, a)$, $e(\eta)$.
5. Need $a, \Theta, \mu$ and use $p(\Theta, \mu)$, $\eta(p, a)$, $e(\eta)$.
   Need $L, \mathbf{D}$, and use $L(\mathbf{D})$, $\eta(\Theta, L)$, $e(\eta)$.
6. Need $L, \mathbf{D}$, and use $L(\mathbf{D})$, $\eta(\Theta, L)$, $e(\eta)$.
   Need $\Theta$, $n$, $\mu$, and use $a(n, \mu)$, $p(\Theta, \mu)$, $\eta(p, a)$, $e(\eta)$.
   Need $\Theta$, $L$, $\mu$, and use $a(L, \mu)$, $p(\Theta, \mu)$, $\eta(p, a)$, $e(\eta)$.
7. Need $\Theta$, $n$, $\mu$, and use $a(n, \mu)$, $p(\Theta, \mu)$, $\eta(p, a)$, $e(\eta)$.
   Need $\Theta$, $L$, $\mu$, and use $a(L, \mu)$, $p(\Theta, \mu)$, $\eta(p, a)$, $e(\eta)$.
   Need $\mathbf{D}, a, \mu$, and use $L(a, \mu)$, $\Theta(\mathbf{D})$, $\eta(\Theta, L)$, $e(\eta)$.
   Need $\mathbf{D}$, and use $L(\mathbf{D})$, $\Theta(\mathbf{D})$, $\eta(\Theta, L)$, $e(\eta)$.

The queue now contains in its fourth entry a solution: a computational path from the Delaunay array $\mathbf{D}$ to the calculation of the eccentricity $e$ using definitions in the relations table. Since the queue is scanned for a solution before expansion, this is the end of the search.

In general, the path obtained for any quantity is dependent on the set of definitions. Sometimes, enlarging the set of definitions means that a different, though still correct, definition will be obtained. In this example, the particular path shown is dependent on the restricted relations table used; when using the full relations table at the end of the paper, the definition will be different.

If no good paths are found and there are no expansions left to be performed, an error is signaled indicating that a variable can not be solved. This usually is due to oversight on the part of the user in giving known variables; sometimes it is necessary to expand the relations table to include new definitions.

It is possible to define macros for use in the particular relations table which makes specification of the functional relationships a little easier. For example, the *Pythagorean complement,* $\sqrt{1 - x^2}$, occurs often enough to define as a macro for the celestial mechanics relations table.

## Pattern Matching and Substitution

The Pattern Matching and Substitution (PMS) subsystem is built around a translation of the program by Abelson et al. [12]. Their program, designed to illustrate capabilities of LISP, is a general purpose pattern matcher with the ability to do substitutions. We have created some simplification rules for mathematics. The intent is not to provide a replacement for a general purpose symbolic manipulation program, but to make more efficient and readable the expressions generated by MAO and the function constructor.

Although we have defined many math simplification rules, we have determined that there are but a few that are important to this application. These rules are

- Product with division: $a \cdot \frac{1}{b} \rightarrow \frac{a}{b}$.
- Lonely products: $a \cdot 1 \rightarrow a$.
- Trigonometry: $\sin \arccos x = \cos \arcsin x \rightarrow \sqrt{1 - x^2}$.
- Inverses: $\cos \arccos x = \sin \arcsin x \rightarrow x$, $\sqrt{x^2} = (\sqrt{x})^2 \rightarrow x$, $(\sqrt{x})^{2n} \rightarrow x^n$, $1/\frac{1}{x} \rightarrow x$, $-(-x) \rightarrow x$.

One must be careful in applying these relations because of the multiple-valued nature of some functions such as the square root. The application of these rules to the problems described does not affect the mathematical validity of the result.

In addition to the rules above applied automatically, the opportunity is given to the user to define a special rule or rules to apply in the conversion from MAO to LISP for specific problems. A special rule was defined and used for trigonometric expressions in the series. Since the six variables in the expression use the same values for the angles in trigonometric expressions, substitution was used to reduce execution time. The trigonometric terms were evaluated in a separate function prior to computation of the transformations, with the results stored in an array. The transformation functions referenced this array instead of performing the trigonometric functions themselves.

## Conclusions

The computations of astrodynamics are sufficiently complex that automatic transfer from computer algebra to numerical source code is a necessity to prevent transcription error. The program LANGTRAN provides the translation from MAO expression to serial or parallel numerical programming languages such as FORTRAN 77, Fortran 90 or Fortran 90 + HPF. For multivariate polynomials, we tried two renderings of the expressions: direct with common subexpression elimination, and as arrays of coefficients and monomials. The latter is slightly faster for modest size problems, and does not have too many continuation lines. The former is faster to generate but often has more continuation lines than FORTRAN allows.

Automatic generation of the expressions is facilitated by a "function constructor" which automatically finds needed variables from those supplied, based on a relations table which contains many relations of celestial mechanics. A pattern match and substitution program provides a means of altering the final program for efficiency. Thus a complete executable subroutine is obtained without need for hand alteration.

### Acknowledgments

We thank Bruce Miller of the National Institute for Standards and Technology for his program L4 which does the actual FORTRAN code generation in LANGTRAN. We also thank Bill Elliott of Duke University for his assistance in analyzing the performance of LANGTRAN output during the summer of 1994. The Naval Space Command and the Naval Research Laboratory supported this work.

## Appendix A: Example of FORTRAN Source Code Output by LANGTRAN

This section shows examples of source code generated by LANGTRAN. The examples given here are the equations of motion in Delaunay variables for the Delaunay normalized Hamiltonian in the satellite main problem ($\mathcal{V}$ consists of $J_2$ term only, and $\mathcal{B} = 0$),

$$\dot{l} = n + \delta n \frac{\alpha^2}{p^2} J_2 \eta \left( -\frac{9}{4} s^2 + \frac{3}{2} \right) + \delta^2 n \frac{\alpha^4}{p^4} J_2^2 \Big[ \eta^3 \left( \frac{75}{128} s^4 + \frac{15}{16} s^2 - \frac{15}{16} \right)$$

$$+ \eta^2 \left( \frac{27}{8} s^4 - \frac{9}{2} s^2 + \frac{3}{2} \right) + \eta \left( \frac{315}{128} s^4 - \frac{45}{8} s^2 + \frac{45}{16} \right) + \left( e^2 \eta \left( \frac{225}{64} s^4 \right. \right.$$

$$- \frac{105}{32} s^2\Big) + e^2\beta\Big(\frac{45}{32} s^4 - \frac{21}{16} s^2\Big) - \frac{45}{32} s^4 + \frac{21}{16} s^2\Big) \cos 2g\Big] + \mathcal{O}(\delta^3)$$

$$\dot{g} = \delta n \frac{\alpha^2}{p^2} J_2\Big(-\frac{15}{4} s^2 + 3\Big) + \delta^2 n \frac{\alpha^4}{p^4} J_2^2\Big(\eta^2\Big(\frac{135}{128} s^4 + \frac{27}{32} s^2 - \frac{21}{16}\Big) + \eta\Big(\frac{135}{16} s^4 - \frac{99}{8} s^2 + \frac{9}{2}\Big) + \frac{1155}{128} s^4 - \frac{645}{32} s^2 + \frac{165}{16} + \Big[e^2\Big(\frac{405}{64} s^4 - \frac{237}{32} s^2 + \frac{21}{16}\Big) + \frac{45}{32} s^4 - \frac{21}{16} s^2\Big) \cos 2g\Big] + \mathcal{O}(\delta^3)$$

$$\dot{h} = \delta \frac{\Theta n}{N} \frac{\alpha^2}{p^2} J_2\Big(\frac{3}{2} s^2 - \frac{3}{2}\Big) + \delta^2 \frac{\Theta n}{N} \frac{\alpha^4}{p^4} J_2^2\Big[\eta^2\Big(-\frac{15}{32} s^4 + \frac{3}{32} s^2 + \frac{3}{8}\Big) + \eta\Big(-\frac{27}{8} s^4 + \frac{45}{8} s^2 - \frac{9}{4}\Big) - \frac{105}{32} s^4 + \frac{225}{32} s^2 - \frac{15}{4} + e^2\Big(-\frac{45}{16} s^4 + \frac{33}{8} s^2 - \frac{21}{16}\Big) \cos 2g\Big] + \mathcal{O}(\delta^3)$$

$$\dot{G} = \delta^2 \Theta n \frac{\alpha^4}{p^4} J_2^2 e^2\Big(-\frac{45}{32} s^4 + \frac{21}{16} s^2\Big) \sin 2g + \mathcal{O}(\delta^3)$$

In the FORTRAN source code output by LANGTRAN shown, some lines have been elided to save space.

*Common Subexpression Elimination FORTRAN Subroutine*

```
1           SUBROUTINE INCREM(DEL,OUT)
2     C     Time derivatives of Delaunay variables for Hamiltonian with
3     C           all degrees of freedom secular; first order J2 only.
4     ...
5           A00006 = DEL(5)/DEL(4)
6           ETA = A00006
7     ...
8           OUT(1) = N+(1.5D0-2.25DO*S2)*ETA*J2*PN2*N+((-0.9375D0+0.5859375D0*
9           . S4+0.9375D0*S2)*ETA**3+(1.5D0+3.375D0*S4-4.5D0*S2)*ETA2+(2.8125D0
10          . +2.4609375Do*S4-5.625D0*S2)*ETA+((3.515625D0*S4-3.28125D0*S2)*E2*
11          . ETA+(A00005+A00004)*E2/(1.0D0+ETA)+A00003+A00002)*C2G)*J22*PN4*N
12    ...
13    C
14    C     ****************************************************************
15    C     End of Subroutine INCREM
16          END
17
```

*Array Polynomial FORTRAN Subroutine*

```
1                SUBROUTINE INCREM(DEL,OUT)
2         C      Time derivatives of Delaunay variables for Hamiltonian with
3         C             all degrees of freedom secular; first order J2 only.
4         ....
5                DATA (COEFS(ITERM),ITERM=0,51)/1.0D0,1.5D0,-2.25D0,2.8125D0,
6                . -5.625D0,2.4609375D0,1.5D0,-4.5D0,3.375D0,-0.9375D0,0.9375D0,
7         ...
8         C      ************************************************************
9         C      Subroutine Body
10        C
11               ETA = DEL(4)/DEL(3)
12               N = DEL(3)/(DEL(3)**2/MU)**2
13               P = DEL(4)**2/MU
14               S2 = 1.0D0-(DEL(5)/DEL(4))**2
15               PN2 = P**(-2)
16               E2 = 1.0D0-ETA**2
17        C
18        C      Computation of monomials, mostly from other monomials
19               MONOM(49) = 1.0D0                      !
20               MONOM(51) = MONOM(49)                  !
21               MONOM(48) = MONOM(49)                  !
22               MONOM(34) = MONOM(49)                  !
23               MONOM(18) = MONOM(49)                  !
24               MONOM(0) = MONOM(49)                   !
25               MONOM(50) = S2                         ! S^2
26               MONOM(35) = MONOM(50)                  ! S^2
27               MONOM(19) = MONOM(50)                  ! S^2
28               MONOM(20) = PN2*J2                     ! P^-2 J2
29               MONOM(36) = MONOM(20)                  ! P^-2 J2
30               MONOM(21) = MONOM(50)*MONOM(20)        ! P^-2 J2 S^2
31               MONOM(29) = MONOM(21)                  ! P^-2 J2 S^2
32               MONOM(37) = MONOM(21)                  ! P^-2 J2 S^2
33               MONOM(22) = MONOM(50)*MONOM(21)        ! P^-2 J2 S^4
34               MONOM(30) = MONOM(22)                  ! P^-2 J2 S^4
35               MONOM(38) = MONOM(22)                  ! P^-2 J2 S^4
36               MONOM(1) = MONOM(20)*ETA               ! P^-2 J2 ETA
37               MONOM(23) = MONOM(1)                   ! P^-2 J2 ETA
38               MONOM(39) = MONOM(1)                   ! P^-2 J2 ETA
39        ...
40        C
41        C      Sum coefficient*monomial for OUT(0)
42               CUMPOL = 0.0D0
43               DO 1 ITERM = 0,11
44                 CUMPOL = CUMPOL+COEFS(ITERM)*MONOM(ITERM)
45          1    CONTINUE
46               CUMHAR = CUMPOL
47               CUMPOL = 0.0D0
48               DO 2 ITERM = 12,17
49                 CUMPOL = CUMPOL+COEFS(ITERM)*MONOM(ITERM)
50          2    CONTINUE
51               CUMHAR = CUMHAR+CUMPOL*COS(2.0D0*DEL(1))
52               OUT(0) = CUMHAR*N
53        ...
54               END
```

## Appendix B: Celestial Mechanics Relations Table

The following is the contents (at present) of the celestial mechanics relations table used in function construction. For brevity, the construction and extraction of quantities from arrays (such as the array of Delaunay variables) has been left off. We use the definitions $\mathscr{C} = e\cos f$, $\mathscr{S} = e\sin f$, $C = e\cos g$, $S = e\sin g$, $c = \cos I$, $s = \sin I$. $\mathcal{H}$ is the two-body Hamiltonian. The functions $\mathcal{R}_x$, $\mathcal{R}_y$, $\mathcal{R}_z$ are the three components of the Euler angle rotation. The first two arguments are the first two angles of rotation, and the last two arguments are the radius times the sine and cosine of the last rotation, respectively. Specifying the arguments this way makes rotation of momenta feasible.

$\theta(f, g) = f + g$ $\qquad$ $r(p, \mathscr{C}) = p/\mathscr{C}$

$R(p, \mathscr{S}, \Theta) = \Theta\mathscr{S}/p$

$l(E, e) = E - e\sin E$ $\qquad$ $E(l, e)$ see below

$g(C, S) = \arctan(C, S)^{\dagger}$ $\qquad$ $f(\mathscr{C}, \mathscr{S}) = \arctan(\mathscr{C}, \mathscr{S})$

$\mathcal{H}(\mu, L) = -\mu^2/L^2$ $\qquad$ $g(\theta, f) = \theta - f$

$$\mathcal{H}(r, R, \Theta, \mu) = \sqrt{R^2 + \left(\frac{\Theta}{r}\right)^2} - \frac{\mu}{r}$$

$\phi(f, l) = f - 1$ $\qquad$ $p(\Theta, \mu) = \Theta^2/\mu$

$\mathscr{C}(p, r) = p/r - 1$ $\qquad$ $\mathscr{C}(e, f) = e\cos f$

$\mathscr{S}(p, R, \Theta) = pR/\Theta$ $\qquad$ $\mathscr{S}(e, f) = e\sin f$

$$f(E, e) = 2\arctan\left(\sqrt{\frac{1+e}{1-e}}\tan\frac{E}{2}\right)$$

$$E(e, f) = 2\arctan\left(\sqrt{\frac{1-e}{1+e}}\tan\frac{f}{2}\right)$$

$L(\mathcal{H}, \mu) = \mu/\sqrt{-2\mathcal{H}}$

$e(\eta) = \sqrt{1-\eta^2}$ $\qquad$ $\eta(e) = \sqrt{1-e^2}$

$\beta(\eta) = 1/(1+\eta)$ $\qquad$ $\eta(\beta) = 1/\beta - 1$

$e(\mathscr{S}, \mathscr{C}) = \sqrt{\mathscr{S}^2 + \mathscr{C}^2}$

$e(S, C) = \sqrt{S^2 + C^2}$

$\eta(\Theta, L) = \Theta/L$ $\qquad$ $\eta(p, a) = \sqrt{p/a}$

$s(c) = \sqrt{1-c^2}$ $\qquad$ $c(s, N) = \mathrm{sgn}(N)\sqrt{1-s^2}$ [‡]

$c(N, \Theta) = N/\Theta$ $\qquad$ $I(c) = \arccos(c)$

[†]Arctangents of two arguments respect the quadrant, in the manner of the function `atan2` in FORTRAN.

[‡]$N$ is necessary to handle retrograde orbits correctly.

$$C(\mathscr{C}, \mathscr{S}, \theta) = \mathscr{C}\cos\theta + \mathscr{S}\sin\theta$$

$$S(\mathscr{C}, \mathscr{S}, \theta) = \mathscr{C}\sin\theta - \mathscr{S}\cos\theta$$

$$C(e, g) = e\cos g \qquad S(e, g) = e\sin g$$

$$a(\eta, p) = p/\eta^2 \qquad p(a, \eta) = a\eta^2$$

$$a(n, \mu) = (\mu/n^2)^{1/3} \qquad a(L, \mu) = L^2/\mu$$

$$n(\mu, L) = \mu^2/L^3 \qquad n(a, L) = L/a^2$$

$$T(n, 2\pi) = 2\pi/n \qquad n(T, 2\pi) = 2\pi/T$$

$$L(a, \mu) = \sqrt{a\mu}$$

$$\hat{G}(\Theta, L) = \Theta/L \qquad \hat{H}(N, L) = N/L$$

$$c(\hat{G}, \hat{H}) = \hat{H}/\hat{G} \qquad \eta(\hat{G}) = \hat{G}$$

$$x(\nu, I, r, \theta) = \mathcal{R}_x(\nu, I, r\cos\theta, r\sin\theta)$$

$$y(\nu, I, r, \theta) = \mathcal{R}_y(\nu, I, r\cos\theta, r\sin\theta)$$

$$z(\nu, I, r, \theta) = \mathcal{R}_z(\nu, I, r\cos\theta, r\sin\theta)$$

$$X(\nu, I, r, \theta, R, \Theta) = \mathcal{R}_x\left(\nu, I, r\cos\theta - \frac{\Theta}{r}\sin\theta, r\sin\theta + \frac{\Theta}{r}\cos\theta\right)$$

$$Y(\nu, I, r, \theta, R, \Theta) = \mathcal{R}_y\left(\nu, I, r\cos\theta - \frac{\Theta}{r}\sin\theta, r\sin\theta + \frac{\Theta}{r}\cos\theta\right)$$

$$Z(\nu, I, r, \theta, R, \Theta) = \mathcal{R}_z\left(\nu, I, r\cos\theta - \frac{\Theta}{r}\sin\theta, r\sin\theta + \frac{\Theta}{r}\cos\theta\right)$$

$$xw_1(\theta, \nu) = \theta \pm \nu \qquad yw_1(\Theta) = \Theta$$

$$xw_2(r) = r \qquad yw_2(R) = R$$

$$xw_3(\Theta, N, \nu) = \sqrt{2(\Theta \mp N)}\cos\nu$$

$$yw_3(\Theta, N, \nu) = \sqrt{2(\Theta \mp N)}\sin\nu$$

$$xp_1(l, g, \nu) = l + g \pm \nu \qquad yp_1(L) = L$$

$$xp_2(g, L, \Theta, \nu) = \sqrt{2(L - \Theta)}\cos(g \pm \nu)$$

$$yp_2(g, L, \Theta, \nu) = \sqrt{2(L - \Theta)}\sin(g \pm \nu)$$

$$xp_3(\Theta, N, \nu) = \sqrt{2(\Theta \mp N)}\cos\nu$$

$$yp_3(\Theta, N, \nu) = \sqrt{2(\Theta \mp N)}\sin\nu$$

The solution of Kepler's equation, the eccentric anomaly in terms of mean anomaly and eccentricity $E(l, e)$, has vexed celestial mechanics for centuries; we have no magical solution to present here! Instead, the necessity for such a solution in function construction causes a call to a subprogram that solves the

problem iteratively. The Whittaker and Poincaré nonsingular variables ($xw_i$, $yw_i$, $xp_i$, $yp_i$) are each actually two sets of variables. The sign specified by $\pm$ or $\mp$ is chosen upper if the orbit is posigrade (singularity is at $I = \pi$) and chosen lower if the orbit is retrograde (singularity is at $I = 0$).

## References


[1] BROUWER, D. "Solution to the Problem of Artificial Satellite Theory without Drag," *Astronomical Journal,* Vol. 64, 1959, pp. 378–397.

[2] COFFEY, S.L., NEAL, H.L., SEGERMAN, A.M., and TRAVISANO, J.J. "An Analytic Orbit Propagation Program for Satellite Catalog Maintenance," AAS Paper 95-426, AAS/AIAA Astrodynamics Specialist Conference, Halifax, Nova Scotia, August 1995.

[3] DEPRIT, A. "The Elimination of the Parallax in Satellite Theory," *Celestial Mechanics,* Vol. 24, No. 2, 1981, pp. 111–153.

[4] STEELE, JR., G.L. *Common Lisp, the Language,* Digital Press, 1990.

[5] ECKERT, W.J., WALKER, M.J., and ECKERT, D. "Transformations of the Lunar Coordinates and Orbital Parameters," *Astronomical Journal,* Vol. 71, No. 5, 1966, pp. 314–332.

[6] COFFEY, S.L., and DEPRIT, A. "Fast Evaluation of Fourier Series," *Astronomy and Astrophysics,* Vol. 81, No. 3, 1980, pp. 310–315.

[7] MILLER, B.R. "A Program Generator for Efficient Evaluation of Fourier Series," *Proceedings of ACM-SIGSAM, International Symposium on Symbolic and Algebraic Computation,* Portland, Oregon, July 1989, pp. 199–206.

[8] GATES, B.L. "GENTRAN: An Automatic Code Generation Facility for REDUCE," *SIGSAM Bulletin,* Vol. 19, No. 3, 1985, pp. 24–42.

[9] GATES, B.L., and VAN HULZEN, J.A. "Automatic Generation of Optimized Programs," *European Conference on Computer Algebra,* Linz, Austria, April 1985, pp. 583–584.

[10] COFFEY, S.L., DEPRIT, A., DEPRIT, E., HEALY, L.M., and MILLER, B.R. "A Toolbox for Nonlinear Dynamics," *Computer Aided Proofs in Analysis,* Springer-Verlag, New York, 1991, pp. 97–115.

[11] DEPRIT, A. "Canonical Transformations Depending on a Small Parameter," *Celestial Mechanics,* Vol. 1, No. 1, 1969, pp. 12–30.

[12] ABELSON, H., HALIFANT, M., KATZENELSON, J., and SUSSMAN, G.J. "The LISP Experience," *Annual Review of Computer Science,* Annual Reviews, Inc., Palo Alto, California, 1988, pp. 167–195.

[13] American National Standard Programming Language FORTRAN, American National Standards Institute, ANSI X3.9-1978, April 1978.

[14] ADAMS, J.C., BRAINERD, W.S., MARTIN, J.T., SMITH, B.T., and WAGENER, J.L. *Fortran 90 Handbook: Complete ANSI/ISO Reference,* Intertext Publications, McGraw-Hill, New York, 1992.

[15] Connection Machine Fortran Programming Guide and CM Fortran Reference Manual, Thinking Machines Corporation, Cambridge, Massachusetts, 1991.

[16] KOELBEL, C.H., LOVEMAN, D.B., SCHREIBER, R.S., STEELE JR., G.L., and ZOSEL, M.E. *The High Performance Fortran Handbook,* MIT Press, Cambridge, Massachusetts, 1994.

[17] KNUTH, D.E. *Seminumerical Algorithms,* Addison-Wesley, Reading, Massachusetts, 1981.

[18] AHO, A.V., SETHI, R., and ULLMAN, J.D. *Compilers Principles, Techniques, Tools,* Addison-Wesley, Reading, Massachusetts, 1986.

[19] WINSTON, P.H., and HORN, B.K.P. *LISP,* Addison-Wesley, Reading, Massachusetts, 1984.