REDUCE Meets CAMAL

J. P. Fitch
School of Mathematical Sciences
University of Bath
BATH, BA2 7AY, United Kingdom

Abstract

It is generally accepted that special purpose algebraic systems are more efficient than general purpose ones, but as machines get faster this does not matter. An experiment has been performed to see if using the ideas of the special purpose algebra system CAMAL(F) it is possible to make the general purpose system REDUCE perform calculations in celestial mechanics as efficiently as CAMAL did twenty years ago. To this end a prototype Fourier module is created for REDUCE, and it is tested on some small and medium-sized problems taken from the CAMAL test suite. The largest calculation is the determination of the Lunar Disturbing Function to the sixth order. An assessment is made as to the progress, or lack of it, which computer algebra has made, and how efficiently we are using modern hardware.

1 Introduction

A number of years ago there emerged the divide between general-purpose algebra systems and special purpose one. Here we investigate how far the improvements in software and more predominantly hardware have enabled the general systems to perform as well as the earlier special ones. It is similar in some respects to the Possion program for MACSYMA [8] which was written in response to a similar challenge.

The particular subject for investigation is the Fourier series manipulator which had its origins in the Cambridge University Institute for Theoretical Astronomy, and later became the F subsystem of CAMAL [3, 10]. In the late 1960s this system was used for both the Delaunay Lunar Theory [7, 2] and the Hill Lunar Theory [5], as well as other related calculations. Its particular area of application had a number of peculiar operations on which the general

speed depended. These are outlined below in the section describing how CAMAL worked. There have been a number of subsequent special systems for celestial mechanics, but these tend to be restricted to the group of the originator.

The main body of the paper describes an experiment to create within the REDUCE system a sub-system for the efficient manipulation of Fourier series. This prototype program is then assessed against both the normal (general) REDUCE and the extant CAMAL results. The tests are run on a number of small problems typical of those for which CAMAL was used, and one medium-sized problem, the calculation of the Lunar Disturbing Function. The mathematical background to this problem is also presented for completeness. It is important as a problem as it is the first stage in the development of a Delaunay Lunar Theory.

The paper ends with an assessment of how close the performance of a modern REDUCE on modern equipment is to the (almost) defunct CAMAL of eighteen years ago.

2 How CAMAL Worked

The Cambridge Algebra System was initially written in assembler for the Titan computer, but later was rewritten a number of times, and matured in BCPL, a version which was ported to IBM mainframes and a number of microcomputers. In this section a brief review of the main data structures and special algorithms is presented.

2.1 CAMAL Data Structures

CAMAL is a hierarchical system, with the representation of polynomials being completely independent of the representations of the angular parts.

The angular part had to represent a polynomial coefficient, either a sine or cosine function and a linear sum of angles. In the problems for which CAMAL was designed there are 6 angles only, and so the design restricted the number, initially to six on the 24 bit-halfword TITAN, and later to eight angles on the 32-bit IBM 370, each with fixed names (usually u through z). All that is needed is to remember the coefficients of the linear sum. As typical problems are perturbations, it was reasonable to restrict the coefficients to small integers, as could be represented in a byte with a guard bit. This allowed the representation to pack everything into four words.

```
[ NextTerm, Coefficient, Angles0-3, Angles4-7 ]
```

The function was coded by a single bit in the `Coefficient` field. This gives a particularly compact representation. For example the Fourier term $\sin(u - 2v + w - 3x)$ would be represented as

```
[ NULL, "1"|0x1, 0x017e017d, 0x00000000 ]
```
or
```
[ NULL, "1"|0x1, 1:-2:1:-3, 0:0:0:0 ]
```

where `"1"` is a pointer to the representation of the polynomial 1. In all this representation of the term took 48 bytes. As the complexity of a term increased the store requirements to no grow much; the expression $(7/4)ae^3f^5\cos(u - 2v + 3w - 4x + 5y + 6z)$ also takes 48 bytes. There is a canonicalisation operation to ensure that the leading angle is positive, and $\sin(0)$ gets removed. It should be noted that $\cos(0)$ is a valid and necessary representation.

The polynomial part was similarly represented, as a chain of terms with packed exponents for a fixed number of variables. There is no particular significance in this except that the terms were held in *increasing* total order, rather than the decreasing order which is normal in general purpose systems. This had a number of important effects on the efficiency of polynomial multiplication in the presence of a truncation to a certain order. We will return to this point later. Full details of the representation can be found in [9].

The space administration system was based on explicit return rather than garbage collection. This meant that the system was sometimes harder to write, but it did mean that much attention was focussed on efficient reuse of space. It was possible for the user to assist in this by marking when an expression was needed no longer, and the compiler then arranged to recycle the space as part of the actual operation. This degree of control was another assistance in running of large problems on relatively small machines.

2.2 Automatic Linearisation

In order to maintain Fourier series in a canonical form it is necessary to apply the transformations for linearising products of sine and cosines. These will be familiar to readers of the REDUCE test program as

$$\cos\theta\cos\phi \quad \Rightarrow \quad (\cos(\theta+\phi) + \cos(\theta-\phi))/2, \tag{1}$$

$$\cos\theta\sin\phi \quad \Rightarrow \quad (\sin(\theta+\phi) - \sin(\theta-\phi))/2, \tag{2}$$

$$\sin\theta\sin\phi \Rightarrow (\cos(\theta-\phi)-\cos(\theta+\phi))/2, \quad (3)$$
$$\cos^2\theta \Rightarrow (1+\cos(2\theta))/2, \quad (4)$$
$$\sin^2\theta \Rightarrow (1-\cos(2\theta))/2. \quad (5)$$

In CAMAL these transformations are coded directly into the multiplication routines, and no action is necessary on the part of the user to invoke them. Of course they cannot be turned off either.

2.3 Differentiation and Integration

The differentiation of a Fourier series with respect to an angle is particularly simple. The integration of a Fourier series is a little more interesting. The terms like $\cos(nu+\ldots)$ are easily integrated with respect to u, but the treatment of terms independent of the angle would normally introduce a secular term. By convention in Fourier series these secular terms are ignored, and the constant of integration is taken as just the terms independent of the angle in the integrand. This is equivalent to the substitution rules

$$\begin{aligned}\sin(n\theta) &\Rightarrow -(1/n)\cos(n\theta)\\ \cos(n\theta) &\Rightarrow (1/n)\sin(n\theta)\end{aligned}$$

In CAMAL these operations were coded directly, and independently of the differentiation and integration of the polynomial coefficients.

2.4 Harmonic Substitution

An operation which is of great importance in Fourier operations is the *harmonic substitution*. This is the substitution of the sum of some angles and a general expression for an angle. In order to preserve the format, the mechanism uses the translations

$$\begin{aligned}\sin(\theta+A) &\Rightarrow \sin(\theta)\cos(A)+\cos(\theta)\sin(A)\\ \cos(\theta+A) &\Rightarrow \cos(\theta)\cos(A)-\sin(\theta)\sin(A)\end{aligned}$$

and then assuming that the value A is small it can be replaced by its expansion:

$$\begin{aligned}\sin(\theta+A) \Rightarrow\ & \sin(\theta)\{1-A^2/2!+A^4/4!\ldots\}+\\ & \cos(\theta)\{A-A^3/3!+A^5/5!\ldots\}\end{aligned}$$

$$\cos(\theta + A) \Rightarrow \cos(\theta)\{1 - A^2/2! + A^4/4! \ldots\} - \sin(\theta)\{A - A^3/3! + A^5/5! \ldots\}$$

If a truncation is set for large powers of the polynomial variables then the series will terminate. In CAMAL the `HSUB` operation took five arguments; the original expression, the angle for which there is a substitution, the new angular part, the expression part (A in the above), and the number of terms required.

The actual coding of the operation was not as expressed above, but by the use of Taylor's theorem. As has been noted above the differentiation of a harmonic series is particularly easy.

2.5 Truncation of Series

The main use of Fourier series systems is in generating perturbation expansions, and this implies that the calculations are performed to some degree of the small quantities. In the original CAMAL all variables were assumed to be equally small (a restriction removed in later versions). By maintaining polynomials in increasing maximum order it is possible to truncate the multiplication of two polynomials. Assume that we are multiplying the two polynomials

$$A = a_0 + a_1 + a_2 + \ldots$$
$$B = b_0 + b_1 + b_2 + \ldots$$

If we are generating the partial answer

$$a_i(b_0 + b_1 + b_2 + \ldots)$$

then if for some j the product $a_i b_j$ vanishes, then so will all products $a_i b_k$ for $k > j$. This means that the later terms need not be generated. In the product of $1 + x + x^2 + x^3 + \ldots + x^{10}$ and $1 + y + y^2 + y^3 + \ldots + y^10$ to a total order of 10 instead of generating 100 term products only 55 are needed. The ordering can also make the merging of the new terms into the answer easier.

3 Towards a CAMAL Module

For the purposes of this work it was necessary to reproduce as many of the ideas of CAMAL as feasible within the REDUCE framework and philosophy.

It was not intended at this stage to produce a complete product, and so for simplicity a number of compromises were made with the "no restrictions" principle in REDUCE and the space and time efficiency of CAMAL. This section describes the basic design decisions.

3.1 Data Structures

In a fashion similar to CAMAL a two level data representation is used. The coefficients are the standard quotients of REDUCE, and their representation need not concern us further. The angular part is similar to that of CAMAL, but the ability to pack angle multipliers and use a single bit for the function are not readily available in Standard LISP, so instead a longer vector is used. Two versions were written. One used a balanced tree rather than a linear list for the Fourier terms, this being a feature of CAMAL which was considered but never coded. The other uses a simple linear representation for sums. The angle multipliers are held in a separate vector in order to allow for future flexibility. This leads to a representation as a vector of length 6 or 4;

```
Version1:  [ BalanceBits, Coeff, Function, Angles, LeftTree, RightTree ]
Version2:  [ Coeff, Function, Angles, Next ]
```

where the `Angles` field is a vector of length 8, for the multipliers. It was decided to forego packing as for portability we do not know how many to pack into a small integer. The tree system used is AVL, which needs 2 bits to maintain balance information, but these are coded as a complete integer field in the vector. We can expect the improvements implicit in a binary tree to be advantageous for large expressions, but the additional overhead may reduce its utility for smaller expressions.

A separate vector is kept relating the position of an angle to its print name, and on the property list of each angle the allocation of its position is kept. So long as the user declares which variables are to be treated as angles this mechanism gives flexibility which was lacking in CAMAL.

3.2 Linearisation

As in the CAMAL system the linearisation of products of sines and cosines is done not by pattern matching but by direct calculation at the heart of the product function, where the transformations (1) through (3) are made in the product of terms function. A side effect of this is that there are no

simple relations which can be used from within the Fourier multiplication, and so a full addition of partial products is required. There is no need to apply linearisations elsewhere as a special case. Addition, differentiation and integration cannot generate such products, and where they can occur in substitution the natural algorithm uses the internal multiplication function anyway.

3.3 Substitution

Substitution is the main operation of Fourier series. It is useful to consider three different cases of substitutions.

1. Angle Expression for Angle:
2. Angle Expression + Fourier Expression for Angle:
3. Fourier Expression for Polynomial Variable.

The first of these is straightforward, and does not require any further comment. The second substitution requires a little more care, but is not significantly difficult to implement. The method follows the algorithm used in CAMAL, using TAYLOR series. Indeed this is the main special case for substitution.

The problem is the last case. Typically many variables used in a Fourier series program have had a WEIGHT assigned to them. This means that substitution must take account of any possible WEIGHTs for variables. The standard code in REDUCE does this in effect by translating the expression to prefix form, and recalculating the value. A Fourier series has a large number of coefficients, and so this operations are repeated rather too often. At present this is the largest problem area with the internal code, as will be seen in the discussion of the Disturbing Function calculation.

4 Integration with REDUCE

The Fourier module needs to be seen as part of REDUCE rather than as a separate language. This can be seen as having internal and external parts.

4.1 Internal Interface

The Fourier expressions need to co-exist with the normal REDUCE syntax and semantics. The prototype version does this by (ab)using the module

method, based in part on the TPS code [1]. Of course Fourier series are not constant, and so are not really domain elements. However by asserting that Fourier series form a ring of constants REDUCE can arrange to direct basic operations to the Fourier code for addition, subtraction, multiplication and the like.

The main interface which needs to be provided is a simplification function for Fourier expressions. This needs to provide compilation for linear sums of angles, as well as constructing sine and cosine functions, and creating canonical forms.

4.2 User Interface

The creation of `HDIFF` and `HINT` functions for differentiation disguises this. An unsatisfactory aspect of the interface is that the tokens `SIN` and `COS` are already in use. The prototype uses the operator form

```
fourier sin(u)
```

to introduce harmonically represented sine functions. An alternative of using the tokens `F_SIN` and `F_COS` is also available.

It is necessary to declare the names of the angles, which is achieved with the declaration

```
harmonic theta, phi;
```

At present there is no protection against using a variable as both an angle and a polynomial varaible. This will nooed to be done in a user-oriented version.

5 The Simple Experiments

The REDUCE test file contains a simple example of a Fourier calculation, determining the value of $(a_1\cos(wt)+a_3\cos(3wt)+b_1\sin(wt)+b_3\sin(3wt))^3$. For the purposes of this system this is too trivial to do more than confirm the correct answers.

The simplest non-trivial calculation for a Fourier series manipulator is to solve Kepler's equation for the eccentric anomoly E in terms of the mean anomoly u, and the eccentricity of an orbit e, considered as a small quantity

$$E = u + e\sin E$$

The solution procedes by repeated approximation. Clearly the initial approximation is $E_0 = u$. The n^{th} approximation can be written as $u + A_n$, and so A_n can be calculated by

$$A_k = e \sin(u + A_{k-1})$$

This is of course precisely the case for which the HSUB operation is designed, and so in order to calculate $E_n - u$ all one requires is the code

```
bige := fourier 0;
for k:=1:n do <<
  wtlevel k;
  bige:=fourier e * hsub(fourier(sin u), u, u, bige, k);
>>;
write "Kepler Eqn solution:", bige$
```

It is possible to create a regular REDUCE program to simulate this (as is done for example in Barton and Fitch[4], page 254). Comparing these two programs indicates substantial advantages to the Fourier module, as could be expected.

Solving Kepler's Equation

Order	REDUCE	Fourier Module
5	9.16	2.48
6	17.40	4.56
7	33.48	8.06
8	62.76	13.54
9	116.06	21.84
10	212.12	34.54
11	381.78	53.94
12	692.56	82.96
13	1247.54	125.86
14	2298.08	187.20
15	4176.04	275.60
16	7504.80	398.62
17	13459.80	569.26
18	***	800.00
19	***	1116.92
20	***	1536.40

These results were with the linear representation of Fourier series. The tree representation was slightly slower. The ten-fold speed-up for the 13th order is most satisfactory.

6 A Medium-Sized Problem

Fourier series manipulators are primarily designed for large-scale calculations, but for the demonstration purposes of this project a medium problem is considered. The first stage in calculating the orbit of the Moon using the Delaunay theory (of perturbed elliptic motion for the restricted 3-body problem) is to calculate the energy of the Moon's motion about the Earth — the Hamiltonian of the system. This is the calculation we use for comparisons.

6.1 Mathematical Background

The full calculation is described in detail in [6], but a brief description is given here for completeness, and to grasp the extent of the calculation.

Referring to the figure 1 which gives the cordinate system, the basic equations are

$$S = (1-\gamma^2)\cos(f+g+h-f'-g'-h') + \gamma^2 cos(f+g-h+f'+g'+h') \quad (6)$$

$$r = a(1-e\cos E) \quad (7)$$

$$l = E - e\sin E \quad (8)$$

$$a = \frac{r\mathbf{d}E}{\mathbf{d}l} \quad (9)$$

$$\frac{r^2\mathbf{d}f}{\mathbf{d}l} = a^2(1-e^2)^{\frac{1}{2}} \quad (10)$$

$$R = m'\frac{a^2}{a'^3}\frac{a'}{r'}\left\{\left(\frac{r}{a}\right)^2\left(\frac{a'}{r'}\right)^2 P_2(S) + \left(\frac{a}{a'}\right)\left(\frac{r}{a}\right)^3\left(\frac{a'}{r'}\right)^3 P_3(S) + \ldots\right\} \quad (11)$$

There are similar equations to (7) to (10) for the quantities r', a', e', l', E' and f' which refer to the position of the Sun rather than the Moon. The problem is to calculate the expression R as an expansion in terms of the quantities e, e', γ, a/a', l, g, h, l', g' and h'. The first three quantities are small quantities of the first order, and a/a' is of second order.

The steps required are

1. Solve the Kepler equation (8)

2. Substiture into (7) to give r/a in terms of e and l.

3. Calculate a/r from (9) and f from (10)

4. Substitute for f and f' into S using (6)

5. Calculate R from S, a'/r' and r/a

The program is given in the Appendix.

6.2 Results

The Lunar Disturbing function was calculated by a direct coding of the previous sections' mathematics. The program was taken from Barton and Fitch [4] with just small changes to generalise it for any order, and to make it acceptable for Reduce3.4. The Fourier program followed the same pattern, but obviously used the `HSUB` operation as appropriate and the harmonic integration. It is very similar to the CAMAL program in [4].

The disturbing function was calculated to orders 2, 4 and 6 using Cambridge LISP on an HLH Orion 1/05 (Intergraph Clipper), with the three programs α) Reduce3.4, β) Reduce3.4 + Camal Linear Module and γ) Reduce3.4 + Camal AVL Module. The timings for CPU seconds (excluding garbage collection time) are summarised the following table:

Order of DDF	Reduce	Camal Linear	Camal Tree
2	23.68	11.22	12.9
4	429.44	213.56	260.64
6	>7500	3084.62	3445.54

If these numbers are normalised so REDUCE calculating the DDF is 100 units for each order the table becomes

Order of DDF	Reduce	Camal Linear	Camal Tree
2	100	47.38	54.48
4	100	49.73	60.69
6	100	<41.13	<45.94

From this we conclude that a doubling of speed is about correct, and although the balanced tree system is slower as the problem size increases the gap between it and the simpler linear system is narrowing.

It is disappointing that the ratio is not better, nor the absolute time less. It is worth noting in this context that Jefferys claimed that the sixth order DDF took 30s on a CDC6600 with TRIGMAN in 1970 [11], and Barton and Fitch took about 1s for the second order DDF on TITAN with CAMAL [4]. A closer look at the relative times for individual sections of the program shows that the substitution case of replacing a polynomial variable by a Fourier series is only marginally faster than the simple REDUCE program. In the DDF program this operation is only used once in a major form, substituting into the Legendre polynomials, which have been previously calculated by Rodrigues formula. This suggests that we replace this with the recurrence relationship.

Making this change actually slows down the normal REDUCE by a small amount but makes a significant change to the Fourier module; it reduces the run time for the 6th order DDF from 3084.62s to 2002.02s. This gives some indication of the problems with benchmarks. What is clear is that the current implementation of substitution of a Fourier series for a polynomial variable is inadequate.

7 Conclusion

The Fourier module is far from complete. The operations necessary for the solution of Duffing's and Hill's equations are not yet written, although they should not cause much problem. The main defficiency is the treatment of series truncation; at present it relies on the REDUCE WTLEVEL mechanism, and this seems too coarse for efficient truncation. It would be possible to re-write the polynomial manipulator as well, while retaining the REDUCE syntax, but that seems rather more than one would hope.

The real failure so far is the large time lag between the REDUCE-based system on a modern workstation against a mainframe of 25 years ago running a special system. The CAMAL Disturbing function program could calculate the tenth order with a maximum of 32K words (about 192Kbytes) whereas this system failed to calculate the eigth order in 4Mbytes (taking 2000s before failing). I have in my archives the output from the standard CAMAL test suite, which includes a sixth order DDF on an IBM 370/165 run on 2 June 1978, taking 22.50s and using a maximum of 15459 words of memory for heap — or about 62Kbytes. A rough estimate is that the Orion 1/05 is comparable in speed to the 360/165, but with more real memory and virtual memory.

However, a simple Fourier manipulator has been created for REDUCE which performs between twice and three times the speed of REDUCE using pattern matching. It has been shown that this system is capable of performing the calculations of celestial mechanics, but it still seriously lags behind the efficiency of the specialist systems of twenty years before. It is perhaps fortunate that it was not been possible to compare it with a modern specialist system.

There is still work to do to provide a convenient user interface, but it is intended to develop the system in this direction. It would be pleasant to have again a system of the efficiency of CAMAL(F).

I would like to thank Codemist Ltd for the provision of computing resources for this project, and David Barton who taught be so much about Fourier series and celstial mechanics. Thank are also due to the National Health Service, without whom this work and paper could not have been produced.

Appendix: The DDF Function

```
array p(n/2+2);
harmonic u,v,w,x,y,z;
weight e=1, b=1, d=1, a=1;

%% Generate Legendre Polynomials to sufficient order
for i:=2:n/2+2 do <<
  p(i):=(h*h-1)^i;
  for j:=1:i do p(i):=df(p(i),h)/(2j)
>>;

%%%%%%%%%%%%%% Step1: Solve Kepler equation
bige := fourier 0;
for k:=1:n do <<
  wtlevel k;
  bige:=fourier e * hsub(fourier(sin u), u, u, bige, k);
>>;

%% Ensure we do not calculate things of too high an order
wtlevel n;
```

```
%%%%%%%%%%%%%%% Step 2: Calculate r/a in terms of e and l
dd:=-e*e; hh:=3/2; j:=1; cc := 1;
for i:=1:n/2 do <<
  j:=i*j; hh:=hh-1; cc:=cc+hh*(dd^i)/j
>>;
bb:=hsub(fourier(1-e*cos u), u, u, bige, n);
aa:=fourier 1+hdiff(bige,u); ff:=hint(aa*aa*fourier cc,u);

%%%%%%%%%%%%%%% Step 3: a/r and f
uu := hsub(bb,u,v); uu:=hsub(uu,e,b);
vv := hsub(aa,u,v); vv:=hsub(vv,e,b);
ww := hsub(ff,u,v); ww:=hsub(ww,e,b);

%%%%%%%%%%%%%%% Step 4: Substitute f and f' into S
yy:=ff-ww; zz:=ff+ww;
xx:=hsub(fourier((1-d*d)*cos(u)),u,u-v+w-x-y+z,yy,n)+
    hsub(fourier(d*d*cos(v)),v,u+v+w+x+y-z,zz,n);

%%%%%%%%%%%%%%% Step 5: Calculate R
zz:=bb*vv; yy:=zz*zz*vv;

on fourier;
for i := 2:n/2+2 do <<
  wtlevel n+4-2i; p(i) := hsub(p(i), h, xx) >>;

wtlevel n;
for i:=n/2+2 step -1 until 3 do
    p(n/2+2):=fourier(a*a)*zz*p(n/2+2)+p(i-1);
yy*p(n/2+2);
```

References

[1] A. Barnes and J. A. Padget. Univariate power series expansions in Reduce. In S. Watanabe and M. Nagata, editors, *Proceedings of ISSAC'90*, pages 82–7. ACM, Addison-Wesley, 1990.

[2] D. Barton. *Astronomical Journal*, 72:1281–7, 1967.

[3] D. Barton. A scheme for manipulative algebra on a computer. *Computer Journal*, 9:340–4, 1967.

[4] D. Barton and J. P. Fitch. The application of symbolic algebra system to physics. *Reports on Progress in Physics*, 35:235–314, 1972.

[5] Stephen R. Bourne. Literal expressions for the co-ordinates of the moon. I. the first degree terms. *Celestial Mechanics*, 6:167–186, 1972.

[6] E. W. Brown. *An Introductory Treatise on the Lunar Theory.* Cambridge University Press, 1896.

[7] C. Delaunay. *Théorie du Mouvement de la Lune.* (Extraits des Mém. Acad. Sci.). Mallet-Bachelier, Paris, 1860.

[8] Richard J. Fateman. On the multiplication of poisson series. *Celestial Mechanics*, 10(2):243–249, October 1974.

[9] J. P. Fitch. Syllabus for algebraic manipulation lectures in cambridge. *SIGSAM Bulletin*, 32:15, 1975.

[10] J. P. Fitch. *CAMAL User's Manual.* University of Cambridge Computer Laboratory, 2nd edition, 1983.

[11] W. H. Jeffereys. *Celestial Mechanics*, 2:474–80, 1970.