INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION

INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
34100 TRIESTE (ITALY) - P.O.B. 586 - MIRAMARE - STRADA COSTIERA 11 - TELEPHONE: 2240-1
CABLE: CENTRATOM - TELEX 460392 - I

SECOND SCHOOL ON ADVANCED TECHNIQUES
IN COMPUTATIONAL PHYSICS
(18 January - 12 February 1988)

SMR.282/ 19

Solving Algebraic Problems with REDUCE

Reference Material

JOHN FITCH

School of Mathematics, University of Bath, England

J. Symbolic Computation (1985) **1**, 211–227

Solving Algebraic Problems with REDUCE

JOHN FITCH

School of Mathematics, University of Bath, Claverton Down, Bath, England

(*Received* 26 *February* 1985)

The algebra system REDUCE is introduced by the exposition of a number of sample problems. While these problems will inevitably be small in scale, and biased towards the interests of the author, the aim is to show how many of the algebraic problems that are faced by scientists and engineers can be readily solved with the powerful tool of computer algebra. No attempt is made to explain the inner workings and design of REDUCE, although there is some reference to the international REDUCE user and implementor community.

Introduction

In the first issue of the *Journal of Symbolic Computation* the major algebra system, MACSYMA, was presented by Pavelle & Wang (1985). In that paper the emphasis was largely on the general capabilities of computer algebra. The present paper, as well as introducing the capabilities of the REDUCE system, will emphasise the problem solving potential of algebra systems. The sample problems that are used are drawn from a range of disciplines, but it is impossible to ensure that the particular interests of all readers are mentioned. I have my own application interests and this bias will be obvious. Even if I ignore your application I trust that one or more of the examples will act as a pointer to how computer algebra can be used in your service. A simpler step-by-step tutorial on how to use REDUCE may be found in Stoutemyer (1978). There are a number of more specialist papers that have been written over the years presenting applications of computer algebra (see for example Barton & Fitch, 1972a, b; Cohen *et al.*, 1976; Fitch, 1979 and Brown & Hearn, 1979). Here the approach is simpler, with more details of the programs.

When giving small examples there is always the suspicion that the times taken are very long. To demonstrate that this is not the case the execution time will sometimes be given. All output in the paper was generated on a High Level Hardware Orion Supermicrocomputer running Cambridge LISP and REDUCE 3·1 in 1·5 Mbytes. The Orion is comparable to a VAX 11/750 in speed. In a later section the range of processors, both larger and smaller, for which REDUCE has been implemented will be given.

A novice seeing computer algebra for the first time is often very impressed by its ability to handle large integers. A session like that given in Fig. 1 can be surprising, but the technology is really very simple. For a demonstration of this one only has to read parts of Knuth (1981). However, this short sample serves to indicate the general style of REDUCE. After loading the program and the banner REDUCE prompts with a one followed by a colon. We will see later how these prompts can be used, but for the present one can view them as labels on the statements. The user interactively types expressions or statements to

```
REDUCE 3.1 (Apr-15-84 (31  Jan 1985)) : 7  Feb 1985
on HLH Orion/Cambridge LISP at the University of Bath under OTS

1: ON TIME;

TIME: 140 MS

2: 2**10;

1024

TIME: 60 MS

3: 2**100;

1267650600228229401496703205376

TIME: 100 MS

4: 2**1000;

1071508607186267320948425049060001810561404811705
5336074437503883703510511249361224931983788156958
5812759467291755314682518714528569231404359845775
7469857480393456777482423098542107460506237114187
7954182153046474983581941267398767559165543946077
0629145711964776865421676604298316526243868372056
68069376

TIME: 620 MS

5: BYE;
```

Fig. 1. Big integers in REDUCE.

terminated by a semicolon, and REDUCE replies with the answer. The first statement arranges that the time taken for each statement is printed. In this implementation of REDUCE the input is always mapped to lower-case, so to aid reading the scripts the user input will be always in upper-case. The statement BYE causes an exit from REDUCE.

This example also draws attention to one of the main reasons for using algebra systems, that the results are exact. The thousandth power of two is not approximated immediately by a floating point number. This attention to exact answers is at one and the same time a power of computer algebra and a drawback. Although the times for calculating these powers of two are fairly fast, they are slower than floating point calculation. Indeed, in general algebraic calculations are apparently slower than pure numerical ones. Of course, there are problems which cannot be tackled in pure numerical terms. The decision whether to use algebraic or numerical means depends critically on the problem. There are many problems which are best solved by a mixture of algebraic and numeric methods. In the examples that follow the emphasis will be on the algebraic aspects.

An Algebraic Calculator

REDUCE can be used in much the same way that a pocket calculator can be used for numerical calculation, except that as yet it is not available on a computer that can be carried regularly in a pocket. A typical example of this kind of use is supplied by the problems faced in high school mathematics. At an early stage in mathematical instruction one is asked to verify that two expressions are equal; perhaps that $(x+y)^2$ is the same as $x^2+2xy+y^2$. In REDUCE this calculation is very simple

```
4: (X+Y)**2;

 2           2
X  + 2*X*Y + Y

TIME: 100 MS

5: (X+Y)**2 - X**2 - 2*X*Y - Y**2;

0

TIME: 200 MS
```

The first calculation shows how the brackets in the input expression are multiplied out and the result displayed in a simple two dimensional form. From the answer to statement 4 we can visually check the result. A more complete approach is in statement 5 where the zero is made explicit. This is one respect in which REDUCE differs from MACSYMA. The normal action of REDUCE is to multiply out all brackets, and combine any similar terms, while MACSYMA does this in response to a command. It is possible to make REDUCE produce its output in a bracketed form at a time cost.

```
6: ON FACTOR;

TIME: 80 MS

7: X**2-2*X*Y+Y**2;

       2
(X - Y)

TIME: 260 MS
```

These examples also show how REDUCE treats names as indeterminates in the absence of other information. It is not always convenient or possible to express a problem simply as a series of expressions. To help in these cases REDUCE provides two simple mechanisms. The first is that the result of any calculation can be referenced in subsequent input by using WS followed by the number of the relevant prompt.

```
8: WS(4) - WS(7);

4*X*Y

TIME: 100 MS
```

The other mechanism is to use assignment. In this next section remember that the ACTOR switch is still on so the input expression is factored where possible

```
9: A:=X**3-1;

         2
A := (X  + X + 1)*(X - 1)

TIME: 260 MS

10: A-3;

 3
X  - 4

TIME: 300 MS
```

REDUCE has a number of flags like FACTOR that control the style of printing of algebraic pressions, such as the order of variables, partial fraction representation and the like, but is not necessary to use or understand them to use REDUCE for real problems; their main e is to improve the style of output to match conventions, as in the quantum theory ample below. With this short introduction to the syntax and style of REDUCE it is ssible to start to solve problems. Further syntax and facilities will be introduced from ne to time.

The *f* and *g* Series

Physical situations are frequently described in terms of expressions which are defined by recurrence relations. A particular example of this situation has been often used in demonstrations of algebra systems, the *f* and *g* series. Lagrange (1869) showed that the co-ordinates of a body in Keplerian motion about an attracting mass can be written as the sum of two terms, a function *f* times the initial position and a function *g* times the initial value of the velocity. These functions can be expanded as Taylor series, and the coefficients of the Taylor series satisfy a recurrence relation. It was shown by Cipolletti (1872) that the coefficients of the expansions can be expressed as polynomials in three variables, conventionally called μ, σ and ε. The recurrence relations for *f* and *g* are

$$f\langle n\rangle = \mathrm{d}f\langle n-1\rangle/\mathrm{d}t - \mu g\langle n-1\rangle$$
$$g\langle n\rangle = \mathrm{d}g\langle n-1\rangle/\mathrm{d}t + f\langle n-1\rangle$$

where

$$\mathrm{d}\mu/\mathrm{d}t = -3\mu\sigma$$
$$\mathrm{d}\sigma/\mathrm{d}t = \varepsilon - 2\sigma^2$$
$$\mathrm{d}\varepsilon/\mathrm{d}t = -\sigma(\mu + 2\varepsilon)$$

and

$$f\langle 0\rangle = 1$$
$$g\langle 0\rangle = 0.$$

Using these relations it is easy to generate the first few terms of the Taylor expansion by hand, but the probability of error increases as the expressions get longer. With an algebra system this problem poses no difficulty. The input and part of the output from REDUCE calculating the first 12 terms in the series is given in Fig. 2. A few notes will elucidate this figure. The differentiation operation d*y*/d*x* is denoted in REDUCE by DF(Y, X), and this

```
4: ARRAY FF(12), GG(12);

TIME: 80 MS

5: DEPS:= -SIG*(MU+2*EPS);

deps := sig*( - 2*eps - mu)

TIME: 320 MS

6: DMU:= -3*MU*SIG;

dmu :=  - 3*mu*sig

TIME: 180 MS

7: DSIG:= EPS-2*SIG**2;

                   2
dsig := eps - 2*sig

TIME: 200 MS

8: FF(0):= 1$

TIME: 60 MS
```

Fig. 2. Calculating the *f* and *g* series.

```
9: GG(0):= 0$

TIME: 60 MS


10:
FOR I:= 1:12 DO
 BEGIN
   FF(I):= -MU*GG(I-1) + DEPS*DF(FF(I-1),EPS) +
                         DMU*DF(FF(I-1),MU) + DSIG*DF(FF(I-1),SIG);
   WRITE "F(",I,") := ", FF(I);
   GG(I):=  FF(I-1)     + DEPS*DF(GG(I-1),EPS) +
                         DMU*DF(GG(I-1),MU) + DSIG*DF(GG(I-1),SIG);
   WRITE "G(",I,") := ", GG(I)
  END;

F(1) := 0

G(1) := 1

F(2) :=  - mu

G(2) := 0

F(3) := 3*mu*sig

G(3) :=  - mu

                                2
F(4) := mu*(3*eps + mu - 15*sig )

G(4) := 6*mu*sig

..........output edited out.................

                                       4
G(12) := 66*mu*sig*(1488375*eps  + 914400*

               3                 3   2
            eps *mu - 25798500*eps *sig  +

                        2   2                 2
            139806*eps *mu  - 15233400*eps *

                2                 2    4
            mu*sig  + 116093250*eps *sig  +

                        3                 2
            5388*eps*mu  - 2211300*eps*mu *

              2                        4
            sig  + 54348840*eps*mu*sig  -

                                6         4
            187960500*eps*sig  + 31*mu  -

                    3   2                2    4
            72800*mu *sig  + 4987710*mu *sig

                               6
             - 50122800*mu*sig  + 99201375*

              8
            sig )

TIME: 20260 MS
```

Fig. 2. (cont.)

fragment uses arrays, which have to be declared. The use of a $ instead of a semicolon at the end of a statement suppresses printing of the result. The main new feature is the introduction of small programs as distinct from single expressions, here represented by a FOR loop, whose meaning is similar to that in languages of the ALGOL family.

From the output it is possible to realise the necessity for algebraic assistance in this calculation. As the order increases the polynomials become much more complicated and rapidly become too difficult for hand algebra. The numerical coefficients also grow to large integers.

Three Dimensional Modelling

All the examples so far have been polynomial. It is obviously necessary to handle more general expressions, and in particular expressions involving the elementary functions of sine, cosine, exponential and logarithm functions. REDUCE knows about these functions, how they differentiate and some of how they combine. As a simple case study we will consider the equations for combining rotations about the x and y axes in three dimensions. In addition to demonstrating transcendental functions this example naturally brings in the matrix data type. The example comes from an undergraduate course on computer graphics, and has been considerably simplified for presentation here.

The position of a point w expressed as a vector after a rotation of θ about the x axis is given by **RX** w, where **RX** is the matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}$$

and similarly the rotation about the y axis of an angle ϕ is expressed in the matrix **RY**

$$\begin{bmatrix} \cos\phi & 0 & -\sin\phi \\ 0 & 1 & 0 \\ \sin\phi & 0 & \cos\phi \end{bmatrix}$$

The problem is to determine the co-ordinates of the point w after first the x axis rotation by θ and then the y axis rotation by ϕ. It would be possible to express this problem using two-dimensional arrays, but it is more natural to use REDUCE's matrix facility.

In Fig. 3 the two matrices **RX** and **RY** are declared and initialised. As an alternative it is possible to set individual elements of the array, but the MAT function is usually more convenient for setting all the values of an array. In statement 7 the two matrices are multiplied using a natural notation, and the value of the product printed. While this is formally correct it may be more useful to change the representation of products of sine functions by linearising them. There is no built-in facility for this in REDUCE as there are in some specialised Fourier series systems (e.g. CAMAL (Fitch, 1983)), but three simple patterns can be declared which implement this simplification. Patterns are introduced with the LET statement. In this case we want the products to be simplified for any arguments of the trigonometric functions, so the pattern is preceded by the FOR ALL construction. Without this the patterns would only work for the particular functions SIN(A) COS(A) SIN(B) and COS(B).

The expectation for the destination of such calculations is a FORTRAN program for transformation of pictures. REDUCE in common with most algebra systems provides a

```
3: ON TIMES

TIME: 220 MS

4: MATRIX RX(3,3), RY(3,3);

TIME: 120 MS

5: RX:= MAT(    (1,0,0),
                (0,COS(TH),SIN(TH)),
                (0,-SIN(TH),COS(TH)) );

rx(1,1) := 1

rx(1,2) := 0

rx(1,3) := 0

rx(2,1) := 0

rx(2,2) := cos(th)

rx(2,3) := sin(th)

rx(3,1) := 0

rx(3,2) :=  - sin(th)

rx(3,3) := cos(th)

TIME: 940 MS

6: RY:= MAT(    (COS(PHI),0,-SIN(PHI)),
                (0,1,0),
                (SIN(PHI),0,COS(PHI)) );

ry(1,1) := cos(phi)

ry(1,2) := 0

ry(1,3) :=  - sin(phi)

ry(2,1) := 0

ry(2,2) := 1

ry(2,3) := 0

ry(3,1) := sin(phi)

ry(3,2) := 0

ry(3,3) := cos(phi)

TIME: 980 MS

7: RX*RY;

mat(1,1) := cos(phi)

mat(1,2) := 0

mat(1,3) :=  - sin(phi)

mat(2,1) := sin(phi)*sin(th)

mat(2,2) := cos(th)
```

Fig. 3. Simple rotation matrix calculations.

```
mat(2,3) := cos(phi)*sin(th)

mat(3,1) := cos(th)*sin(phi)

mat(3,2) :=  - sin(th)

mat(3,3) := cos(phi)*cos(th)

TIME: 920 MS

8: FOR ALL A,B LET SIN(A)*COS(B) = (SIN(A+B)+SIN(A-B))/2;

TIME: 320 MS

9: FOR ALL A,B LET SIN(A)*SIN(B) = (COS(A-B)-COS(A+B))/2;

TIME: 300 MS

10: FORALL A,B LET COS(A)*COS(B) = (COS(A+B)+COS(A-B))/2;

TIME: 320 MS

11: WS(7);

mat(1,1) := cos(phi)

mat(1,2) := 0

mat(1,3) :=  - sin(phi)

mat(2,1) := (cos(phi - th) - cos(phi + th))/2

mat(2,2) := cos(th)

mat(2,3) := ( - sin(phi - th) + sin(phi + th)
             )/2

mat(3,1) := (sin(phi - th) + sin(phi + th))/2

mat(3,2) :=  - sin(th)

mat(3,3) := (cos(phi - th) + cos(phi + th))/2

TIME: 2020 MS
```

Fig. 3. (cont.)

facility for generating a FORTRAN program. After the statement ON FORT printing is performed in a FORTRAN style. An example of this for the f and g series example can be found in Fig. 4. We will see later that if the needed language is PASCAL (say) it is not too difficult to do that instead.

In REDUCE it is possible to do other operations on matrices simply and efficiently, such as add them, invert them, calculate the trace, and transpose them.

Factorisation

It has already been seen in the introductory part of this paper that REDUCE can factorise polynomial expressions. The factoriser code is a very large part of REDUCE in source lines and incorporates the state of the art algorithms (Moore & Norman, 1981). In practice it is rarely necessary to perform factorisation except in certain specialised fields, but here a

```
  ANS=66.*MU*SIG*(1488375.*EPS**4+914400.*EPS**3*MU-
. 25798500.*EPS**3*SIG**2+139806.*EPS**2*MU**2-15233400.*EPS
. **2*MU*SIG**2+116093250.*EPS**2*SIG**4+5388.*EPS*MU**3-
. 2211300.*EPS*MU**2*SIG**2+54348840.*EPS*MU*SIG**4-
. 187960500.*EPS*SIG**6+31.*MU**4-72800.*MU**3*SIG**2+
. 4987710.*MU**2*SIG**4-50122800.*MU*SIG**6+99201375.*SIG**8
. )
```

Fig. 4. FORTRAN output from REDUCE.

simple example can be used to illustrate not only factorisation but also the use of procedures in REDUCE.

As well as using the switch FACTOR to control output style factorisation can also be invoked directly, and the factors assigned to elements of an array. Our example uses this form to investigate the size of coefficients in the factorisation of $x^n - 1$ for different values of n. For low values of n all the non-zero coefficients are plus or minus one, but this is not true for large values of n. The program in Fig. 5 defines a procedure which from a value for n (called POW in the program) determines the largest absolute value of the coefficients of the factorisation of $x^n - 1$. It calls FACTORIZE to place the factors into an array and the function COEFF to put coefficients into another array. The procedure was compiled by the standard LISP compiler, but in this example the majority of the time is spent on the factorisation. The sample values of 105 and 1155 for the power are not chosen at random, but are the smallest powers for which the coefficient is 2 and 3 respectively. Indeed, it has been shown (Vaughan, 1974) that the maximum coefficient grows without bound, but very slowly.

Integration Example

One of the other large packages in REDUCE is the integration function. It is rare that simple integration is used in real problems but it does happen. The integration method offered by REDUCE is algorithm based, and follows the revised integration method of Risch & Norman (Fitch, 1981). While this is not a complete decision procedure in practice it solves most integrations with which it is presented. Like the factorisation package the integrator is also capable of explaining the steps of the calculation.

Rather than take a real example for integration the script of Fig. 6 shows a few simple integrals so the scope of the integrator can be seen. In broad terms it can handle integrals involving rational functions, exponential and logarithmic functions, and the trigonometric functions. There is a limited capacity for algebraic forms and there is an extensibility feature.

Quantum Mechanics Example

The earliest REDUCE system was built to solve problems in quantum electro-dynamics. It would therefore seem inappropriate to ignore this area in any introductory tutorial to REDUCE. However, quantum theory is in no way a speciality of the present author, who therefore begs the indulgence of experts for this simplified explanation.

Our sample problem involves the interaction of an electron and a photon, in which an electron and a photon emerge. The problem can be posed in terms of Lorentz four vectors and the gamma matrices. The variables which represent the four dimensional vectors need to be declared as such, which is achieved with the statement

VECTOR EI;

```
1: ON TIME;

TIME: 120 MS

2: ARRAY FF(100), CC(100);

TIME: 160 MS

3:
ALGEBRAIC PROCEDURE LARGECOEFF POW;
BEGIN SCALAR N, M, MAX, Y;
    Y:=X**POW-1;
    MAX:=0;
    N:=FACTORIZE(Y,FF);
    FOR I:=1:N DO
    BEGIN
        M:=COEFF(FF(I),X,CC);
        FOR J:=0:M DO
        BEGIN
            IF CC(J)<0 THEN CC(J):=-CC(J);
            IF CC(J)>MAX THEN MAX:=CC(J)
        END;
    END;
    RETURN MAX
END;

** 852 bytes   700 msecs    compiling largecoeff

largecoeff

TIME: 1500 MS

4: LARGECOEFF(3);

1

TIME: 300 MS

5: LARGECOEFF(105);

2

TIME: 4360 MS

6: LARGECOEFF(1155);

3

TIME: 161060 MS

7: END;
```

Fig. 5. Factorisation and algebraic procedures.

In the case of particles whose mass is known the simple declaration

MASS PI = M;

would associate the mass M with **PI** and declare **PI** as a vector. The normal product operator is insufficient for products involving vectors, so REDUCE uses the dot as a product between Lorentz vectors. As our example is concerned with real particles we know how they react with each other, that is to say what the dot products are between them. In particular we know that the dot product with itself is the square of the mass times a unit 4×4 matrix. In practice it turns out to be unnecessary to mention the unit matrix, so we would like the simplification

$$\mathbf{PI} . \mathbf{PI} \Rightarrow M^{**}2.$$

The pattern matching facility already introduced can be used to implement these simplifications, but as rules of this kind are so common REDUCE has a statement

MSHELL PI;

which declares this relationship, provided that a previous MASS statement has indicated the value. The other product simplifications can be introduced with LET.

The script in Fig. 7 gives a REDUCE program for calculating the Compton scattering cross-section barring a simple factor. Full details of the formulation can be found in Bjorken & Drell (1964, ch. 7). The notation is that **PI** and **KI** are the incident electron and photon respectively, and **PF** and **KF** are the final electron and photon. **E** and **EP** (= **E**′) are two polarisation vectors. The heart of the calculation is to calculate

$$(1/4)\ \mathrm{trace}\left((\gamma.\mathbf{Pf}+M)\left(\frac{\gamma.\mathbf{E}\,\gamma.\mathbf{E}'\,\gamma.\mathbf{Ki}}{2\,\mathbf{Ki}.\mathbf{Pi}}+\frac{\gamma.\mathbf{E}\,\gamma.\mathbf{E}'\,\gamma.\mathbf{Kf}}{2\,\mathbf{Kf}.\mathbf{Pi}}\right)(\gamma.\mathbf{Pi}+M)\left(\frac{\gamma.\mathbf{Ki}\,\gamma.\mathbf{E}\,\gamma.\mathbf{E}'}{2\,\mathbf{Ki}.\mathbf{Pi}}+\frac{\gamma.\mathbf{Kf}\,\gamma.\mathbf{E}'\,\gamma.\mathbf{E}}{2\,\mathbf{Kf}.\mathbf{Pi}}\right)\right).$$

```
1: INT(X,X);

 2
x /2

2: INT(X**5*LOG(X)**3,X);

  6              3              2
(x *(36*log(x)  - 18*log(x)  + 6*log(x) - 1))/216

3: INT(E**(-X**2),X);

         2
       (x )
int(1/e     ,x)

4: INT(1/(X**8-1),X);

( - 2*atan(( - sqrt(2) + 2*x)/sqrt(2))*sqrt(2) - 2*atan((sqrt(2) + 2*x)/

                                                                2
 sqrt(2))*sqrt(2) - 4*atan(x) + log( - sqrt(2)*x + x  + 1)*sqrt(2) - log(

                 2
 sqrt(2)*x + x  + 1)*sqrt(2) + 2*log(x - 1) - 2*log(x + 1))/16

5: INT(1/SIN(X),X);

log(tan(x/2))

6: INT(X*COS(X),X);

cos(x) + sin(x)*x

7: FOR ALL A LET DF(DILOG(A),A)=LOG(A)/(A+1);

8: INT(X*DILOG(X),X);

                2                         2                       2
(4*dilog(x)*x  - 4*dilog(x) - 2*log(x)*x  + 4*log(x)*x + x  - 4*x)/8

9: END;
```

Fig. 6. Some integrals.

```
REDUCE 3.1 (Apr-15-84 (31  Jan 1985)) : 12  Feb 1985
on HLH Orion/Cambridge LISP at the University of Bath under OTS

1: ON TIME,DIV;

TIME: 100 MS

2: MASS KI= 0, KF= 0, PI= M, PF= M;

TIME: 160 MS

3: VECTOR E,EP;

TIME: 60 MS

4: MSHELL KI,KF,PI,PF;

TIME: 220 MS

5: LET PI.E= 0,  PI.EP= 0, PI.PF= M**2+KI.KF, PI.KI= M*K,
    PI.KF= M*KP, PF.E= -KF.E, PF.EP= KI.EP,    PF.KI= M*KP,
    PF.KF= M*K,  KI.E= 0, KI.KF= M*(K-KP),   KF.EP= 0,
    E.E= -1,  EP.FP= -1;

TIME: 1260 MS

6: (G(L,PF)+M)*(G(L,EP)*G(L,E)*G(L,KI)/(2*KI.PI) +
            G(L,E)*G(L,EP)*G(L,KF)/(2*KF.PI)) *
   (G(L,PI)+M)*(G(L,KI)*G(L,E)*G(L,EP)/(2*KI.PI) +
            G(L,KF)*G(L,EP)*G(L,E)/(2*KF.PI)) $

TIME: 3220 MS

7: WRITE "THE COMPTON CROSS-SECTION IS ",WS;

                                       2            (-1)          (-1)
THE COMPTON CROSS-SECTION IS 2*e . ep  + 1/2*k*kp      + 1/2*k     *kp - 1

TIME: 420 MS

8: BYE;

End of lisp run after 5.56+22.18 SECS - 31.4% store used
```

Fig. 7. Compton cross-section.

The contractions of the momenta with the gamma matrices is indicated in REDUCE with the function *G*, whose first argument is a label on the fermion line in a Feynman diagram. If more than one line is present, then in effect there is a separate set of gamma matrices needed. In the present problem there is only one line which the program labels as *L*. As a slight gloss on an earlier example this special use of *G* is the reason why the arrays to hold the *f* and *g* series above were called *FF* and *GG*. The program of Fig. 7 has been adapted from the program distributed with REDUCE as a test. It has been modifed to avoid some shorthand notations, details of which can be found in the REDUCE manual (Hearn, 1984). The first line of the program contains a change to the printing style by switching the flag DIV on. This means that denominators are represented as negative powers, the style of expression used in Bjorken & Drell (1964).

Duffing's Equation

All the examples presented in this tutorial so far have been very simple and short. In case the reader may be getting the impression that REDUCE and computer algebra are only useful for small problems, in this section will be presented a realistic problem, that has been used in computer algebra for some years (see, for example, Martin, 1967; Barton & Fitch, 1972*a* and Fitch *et al.*, 1981). The problem is also a special case of a number of problems that arise in many branches of physics. The problem is to produce an approximate solution to the ordinary differential equation

$$\frac{d^2x}{dt^2}+x = ex^3$$

where $x(0)=a$ and $dx/dt=0$ and e is a small quantity. The solution required is that which does not have any secular terms, that is, terms that grow with time. A method for solving this problem is the technique of Linstedt (1882) and Poincaré (1893), where we introduce a stretched time ct, with the constant c constructed so the solution is periodic. Changing the independent variable to $v=ct$ we obtain

$$c^2x''+x=ex^3$$

where the prime denotes differentiation with respect to v. The value of x will be approximated in powers of the small quantity e. The zeroth order approximation is determined by ignoring e

$$x_0 = a\cos(ct) = a\cos(v)$$
$$c_0 = 1.$$

We next make the approximation step, by assuming that the true solution is of the form the nth approximation plus a small correction. We will attempt to produce an equation for the $(n+1)$th order correction

$$x = x_n+\varepsilon+O(n+2)$$
$$c = c_n+\eta+O(n+2).$$

Substituting this into the transformed equation, and ignoring all terms of obviously higher order than $n+1$, one obtains

$$c_n\varepsilon''+\varepsilon+2c_n\eta x_n'' = ex_n^3-c_n^2x_n''-x_n+O(n+2).$$

If we now apply our knowledge that ε is small, the $c_n\varepsilon''$ term can be replaced by $c_0\varepsilon''$, and a similar argument reduces the term in η, so we get

$$c_0\varepsilon''+\varepsilon-2\eta a\cos(v) = ex^3-c_n^2x''-x+O(n+2).$$

A closer look at the right-hand side of this equation will show that as x is a cosine series (by consideration of the boundary conditions), so is the RHS. Let us write this as

$$\sum_{n=1}^{\inf} B\cos(nv).$$

As long as n is not 1 it is easy to spot that

$$\sum_{n\neq 1}\frac{B_n\cos(nv)}{(1-n^2)}$$

is an integral. By choosing η to be $-B_1/(2a)$ we have a complete integration, and so we can produce the approximation to x to any finite order. To do this in REDUCE it is necessary to introduce patterns to work in Fourier series, similar to the patterns used in

the rotation matrix example. We also have to construct the particular integral. The only subtlety here is to arrange that the pattern does not get re-used over and over again. The complete REDUCE program is given in Fig. 8 and the output to fourth order in Fig. 9. This problem is not ideally suited to REDUCE, being the subject of many specialised algebra systems, but it can handle it with judicious use of patterns and substitution.

This example is just one of a whole class of problems of expansion of functions in series. Other examples can be found in Barton & Fitch (1972*a*) and Fitch *et al.* (1981). The general approach outlined in this section can be applied to a very large number of these cases. The author has used it widely in astronomical work, and as an aid to various numerical processes.

Survey of Other Applications

This introductory review has indicated that REDUCE may be useful in the areas of simple calculation, quantum electro-dynamics and approximation methods. With the distribution of REDUCE comes a small test file that includes a number of examples, including some used in this tutorial, but also including a *pro forma* general relativity program. There have been applications of REDUCE in a large number of fields (Fitch, 1979). These include: quantum mechanics, general relativity, optics, structural mechanics, ordinary differential equation theory, electronics, geometry, fluid mechanics, centrifuge design, windmill design, helicopter rotor design, astrophysics, celestial mechanics, economics, stress analysis, plasma physics, finite element analysis, control theory, image processing, symbolic execution, and antenna design. It is beyond the scope of the current work to consider these, but a register of papers that cite the algebraic work of REDUCE in their field has been maintained for some years and the range is considerable (Hearn, 1985). There are currently over 200 papers in that list.

There can be no doubt that REDUCE is a tool that is needed in many fields of science and engineering. Recently, H. I. Cohen and myself undertook a programme for the education of industrial and technical companies in both Sweden and Britain in computer algebra,

```
ORDER A;
FOR ALL A, B LET COS(A)*COS(B) = (COS(A+B)+COS(A-B))/2;
FOR ALL A LET COS(A)**2 = (1+COS(2*A))/2;
X:=A*COS(V);
C:=1;
OPERATOR CS;
FOR N=1:4 DO BEGIN
    LET E**(N+1)=0;
    R := E * X**3 - C**2 * DF(X,V,2) - X;
COMMENT GET THE COS(V) TERM;
    COSTERM:=R-SUB(COS(V)=0,R);
    R:=SUB(COS(V)=0,R);
    C:= C - COSTERM/(2*A);
COMMENT PERFORM THE INTEGRATION;
    FOR ALL J LET COS(J*V) = CS(J*V)/(1-J**2);
    TEMP := R;
    FOR ALL J CLEAR COS(J*V);
    FOR ALL A LET CS(A)=COS(A);
    X := X + TEMP;
    FOR ALL A CLEAR CS(A);
END;
X;
C;
END;
```

Fig. 8. Program for approximate solution of Duffing's equation.

```
                8               4                8               4                 8
 (a*(147000*a *cos(9*v)*e  - 9333280*a *cos(8*v)*e  + 125357820*a *cos(7*v

      4              8             4                  8            4
   )*e  + 94445760*a *cos(6*v)*e  - 1434307140*a *cos(5*v)*e  +

                 8            4                  8            4
   1363384848*a *cos(4*v)*e  + 3721989285*a *cos(3*v)*e  - 28523633040*

   8             4               8  4              6            3
   a *cos(2*v)*e  - 52012598400*a *e  - 4704000*a *cos(7*v)*e  +

              6            3                 6            3             6
   239662080*a *cos(6*v)*e  - 2261212800*a *cos(5*v)*e  - 279605760*a *

              3                  6            3                  6            3
   cos(4*v)*e  + 11660745600*a *cos(3*v)*e  + 27049881600*a *cos(2*v)*e

                6  3                4            2                  4
    - 93731904000*a *e  + 150528000*a *cos(5*v)*e  - 5178163200*a *cos(4

        2                 4            2                  4            2
   *v)*e  + 16482816000*a *cos(3*v)*e  + 24987648000*a *cos(2*v)*e  -

                4  2                2                         2
   152635392000*a *e  - 4816896000*a *cos(3*v)*e + 57802752000*a *cos(2*

                      2
   v)*e - 173408256000*a *e + 154140672000*cos(v)))/154140672000


              8             4           6             3          4            2
 ( - 2310027*a *cos(v)*e  - 1347456*a *cos(v)*e  + 98304*a *cos(v)*e  -

           2
 3145728*a *cos(v)*e + 2097152)/2097152
```

Fig. 9. Solution to Duffing's equation to fourth order.

and we discovered that once these techniques had been shown to them they immediately recognised that they would be able to ask the questions that they had been suppressing as incapable of solution. Many of these companies are now acquiring algebra systems. At present it is unfortunately still the case that much algebra is being done by hand.

REDUCE: Availability and Scope

The REDUCE system was originally the work of Hearn who wanted a tool to solve problems in quantum electro-dynamics. From this beginning it has grown into a general purpose algebra system with a wide distribution, and an international community of algebra researchers refining and extending it. REDUCE is available on a large number of different processors and operating systems. Table 1 summarises the availability. As new implementations are fairly frequent this table is likely to be incomplete in some respects. However, it shows the range.

REDUCE is written in a dialect of LISP that has shown itself to be portable (Marti *et al.*, 1979). At present there are two main families of implementation, one based on PSL (Griss *et al.*, 1982) and the other on Cambridge LISP (Fitch & Norman, 1977) and there are other implementations based on various standard LISP systems and adaptations of other LISPS.

Table 1. Systems supporting REDUCE ((*) = old version)

IBM 360 Series and similar	MVT, MVS, CMS, MTS, TSO
DEC 10 and 20	Tops-10, Tops-20
DEC VAX	VMS, UNIX
68 000, various manufacturers	
SUN	UNIX
HP9000 series 200	PASCAL, HPUS
Sage	Tripos
Pinnacle	Tripos, CP/M68K
HLH Orion	UNIX
Acorn 32016	PANOS
GEC System 63	UNIX
Apollo Domain	Aegis
CRAY-1	CTSS
Symbolics 3600	
Xerox Dolphin, Dandelion	
DG Eclipse MV	AOS/VS
Honeywell 68/DPS	Multics (*)
CDC Cyber series	NOS, NOS/BE, SCOPE (*)
UNIVAC 1100	(*)
Burroughs B6000, B7000	(*)

In this tutorial it is inevitable that only some of the capabilities of REDUCE have been described. It has been inappropriate to discuss the SOLVE package that can solve certain classes of equations, or the arbitrary precision floating point number package that can be used very effectively to determine the source of some numerical errors. Only the very simplest use of the high energy physics package has been mentioned.

One of the important aspects of REDUCE is that the source is distributed with the system. Thus it is possible for interested scientists to look into extending REDUCE with new capabilities. Indeed, the factorisation, integration, solving and big float packages began in this way. REDUCE is a system that is still evolving. As new algorithms are discovered the system is revised and modified. At present the author knows of new packages in an advanced state of development or testing for integer factorisation, heuristic GCD (Davenport & Padget, 1985), exterior calculus, common subexpression detection (Wang *et al.*, 1984), integration of algebraic functions, algebraic numbers, factorisation over algebraic fields and Taylor series. REDUCE is a living system that is always being reviewed to provide the best possible service to the scientist or engineer who needs algebraic computation. There are many people over the world who can maintain REDUCE. It is for this reason that it was suggested earlier that writing a PASCAL output package would be comparatively easy.

The author is indebted to the members of the Computing Group at the University of Bath for advice and understanding while this paper was written, and to Tony Hearn for providing some additional information about REDUCE. The opinions about REDUCE expressed in this paper are, however, entirely mine.

References

Barton, D., Fitch, J. P. (1972*a*). Applications of algebraic manipulation programs in physics. *Rep. Prog. Phys.* **35**, 235–314.
Barton, D., Fitch, J. P. (1972*b*). A review of algebraic manipulative programs and their application. *Comput. J.* **15**, 362–381.
Bjorken, J. D., Drell, S. D. (1964). *Relativistic Quantum Mechanics*. New York: McGraw-Hill.

Brown, W. S., Hearn, A. C. (1979). Applications of symbolic algebraic computation. *Comput. Phys. Commun.* **17**, 207–215.

Cipolletti, D. (1872). Espressioni generali dello sviluppo in serie delle coordinate di un corpo celeste. *Publ. Osservatorio di Firenze.*

Cohen, H. I., Leringe, O., Sundblad, Y. (1976). The use of algebraic computing in general relativity. *Gen. Relativ. Gravit.* **7**, 269–286.

Davenport, J. H., Padget, J. A. (1985). HEUGCD: How elementary upper-bounds give cheaper data. *Proc. EUROCAL 85.* (to appear).

Fitch, J. P., Norman, A. C. (1977). Implementing LISP in a high level language. *Software—Prac. Experien.* **7**, 713–725.

Fitch, J. P. (1979). A survey of symbolic computation in physics. Symbolic and Algebraic Computation, *Proc. EUROSAM 79. Lecture Notes in Computer Science, 72.* New York: Springer-Verlag.

Fitch, J. P. (1981). User-based integration software. *Proc. SYMSAC 81*, pp. 245–248. Snowbird, Utah.

Fitch, J. P., Norman, A. C., Moore, P. M. A. (1981). The automatic derivation of periodic solutions to a class of weakly nonlinear differential equations. *Proc. SYMSAC 81*, pp. 239–244. Snowbird, Utah.

Fitch, J. P. (1983). *CAMAL User's Manual*, 2nd edn. University of Cambridge Computer Laboratory.

Griss, M. L., Benson, E., Maguire, G. Q., Jr (1982). PSL: A portable LISP system. *Proc. ACM Symposium on LISP and Functional Programming.*

Hearn, A. C. (1984). *REDUCE User's Manual: Version 3.1.* Santa Monica: The Rand Corporation.

Hearn, A. C. (1985). *REDUCE Publications List.* (Available from the author.)

Knuth, D. E. (1981). *The Art of Computer Programming, Vol. 2.* Reading, Mass: Addison-Wesley.

Lagrange, J. L. (1869). *Ouvres*, Vol. **IV**, 500.

Lindstedt, A. (1882). Ueber die Integration einer fuer die Stoerungstheorie wichtigen Differentialgleichung. *Astron Nach* **104**, Col 211–20.

Martin, W. A. (1967). *Symbolic Mathematical Laboratory.* PhD Thesis, MIT, MAC-TR-36.

Marti, J. B., Hearn, A. C., Griss, M. L., Griss, C. (1979). Standard LISP report. *SIGPLAN Notices* **14**, **10**, 48–68.

Moore, P. M. A., Norman, A. C. (1981). Implementing a polynomial factorisation and GCD package. *Proc. SYMSAC 81*, pp. 109–116. Snowbird, Utah.

Pavelle, R., Wang, P. S. (1985). MACSYMA from F to G. *J. Symbolic Computation* **1**, 69–100.

Poincaré, H. (1893). *Les Méthodes Nouvelles de la Méchanique Céleste, Vol. 2.* Paris: Gauthier-Villars. *(Available in English as NASA TTF-450, 1967.)*

Stoutemyer, D. R. (1978). REDUCE interactive lessons 1–5. *REDUCE Newsletter* **2**, **5**, **6** and **7**, 1978–79.

Vaughan, R. C. (1974). Bounds for the coefficients of cyclotomic polynomials. *Mich. Math. J.* **21**, 289–295.

Wang, P. S., Chang, T. Y. P., van Hulzen, J. A. (1984). Code generation and optimization for finite element analysis. *Proc. EUROSAM 84, Lecture Notes in Computer Science 174*, pp. 237–247. New York: Springer-Verlag.