2
27
8-14-79
NTIS
23 cop 6

MASTER

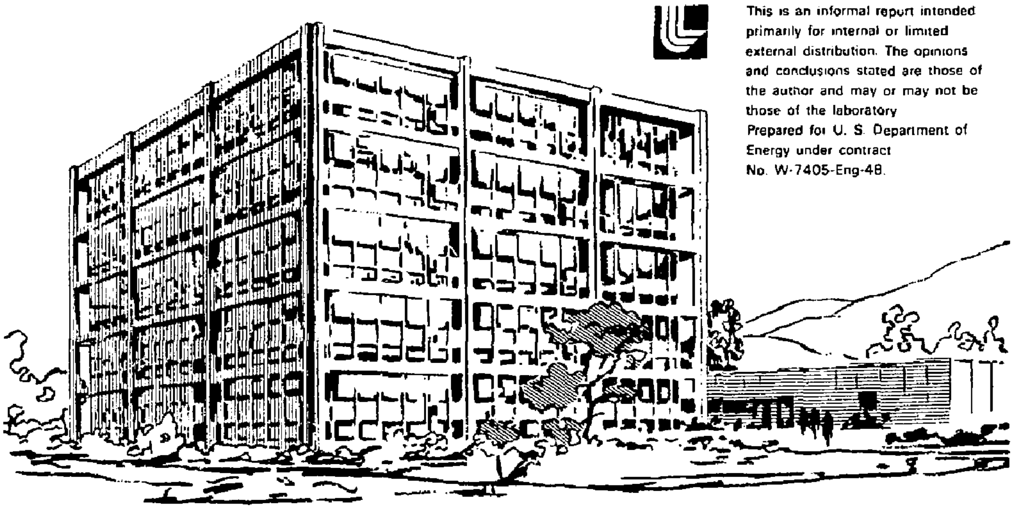
UCID- 18185

Lawrence Livermore Laboratory

LCPT: A PROGRAM FOR FINDING LINEAR CANONICAL TRANSFORMATIONS

Bruce W. Char, University of California, Berkeley, and
Brendan McNamara, LLL

May 21, 1979



Lcpt: a program for finding linear canonical transformations

Bruce W. Char + ~

Computer Science Division
EECS Department
University of California, Berkeley
Berkeley, California 94720

Brendan McNamara

Lawrence Livermore Laboratory
Livermore, California 94550

ABSTRACT

This article describes a MACSYMA* program to compute symbolically a canonical linear transformation between coordinate systems. The difficulties in implementation of this "canonical" small physics problem are also discussed, along with the implications that may be drawn from such difficulties about widespread MACSYMA usage by the community of computational/theoretical physicists.

1. Introduction

Hamiltonian mechanics problems can be solved, partially or completely, by finding canonical transformations to new coordinates in which the Hamiltonian is simpler. A recent paper [McNamara] describes three different Lie transforms which use the method of averaging for oscillatory systems, examine resonant islands in phase space, and provide super convergent algorithms for getting high order terms in perturbation series more easily. The paper also uses more ordinary canonical transformations to action-angle variables or bilinear point transformations, to prepare the Hamiltonians for application of the Lie perturbation

+Work performed while a summer visitor at Lawrence Livermore Laboratory. Work performed under the auspices of the U.S. Dept. of Energy by the Lawrence Livermore Laboratory under contract number W-7405-ENG-48.

~Preparation of this paper supported in part by the National Science Foundation under grant MCS # 78-07291, and the U S. Dept. of Energy under contract DE-AT03-76SF00034 and PA DE-AS03-79ER10358.

*Developed by the Mathlab group at the Massachusetts Institute of Technology's Laboratory for Computer Science, currently supported, in part, by the U.S. Department of Energy under Contract Number E(11-1)-3070 and by the National Aeronautics and Space Administration under Grant NSG 1323.

[illegible]

methods. Another paper [Char] describes how the basic algebra of the Lie transform methods was implemented on MACSYMA. The most difficult mathematical operations to implement involved Poisson brackets, and integrating and averaging operators. This work is intended to be part of a complete MACSYMA system for doing Hamiltonian mechanics— a collection of tools, rather than a monolithic code to do everything. However, even the more trivial operators needed for doing Hamiltonian mechanics have turned out to be complicated to implement.

Here we describe the program for finding linear canonical transformations. The mathematical problem is unusual, in that one only needs to specify the desired properties of a part of the transformation. The rest is left to be chosen by the program to be as close as possible to the identity. This fact, that the problem is underdetermined, contributed to some of the programming complexity.

In Part I of this paper, a procedure for finding a Linear Canonical Point Transformation (LCPT), given a list of desired linear relations between the domain and range coordinate systems, is scribed. The procedure's implementation in MACSYMA so as to facilitate interactive usage is also discussed. In part II we use *lcpt* as an example of a typical small-scale application problem in physics to illustrate the various implementation problems currently encountered with such computations in MACSYMA.

PART I--The problem, its solution, and implementation in MACSYMA

2. Basic definitions

The *Hamiltonian function* $h(p_1,p_2,\cdots p_{dim},q_1,q_2,\cdots q_{dim})$ --often written as $h(p,q)$--is an expression of the total energy of a mechanical system, relating momentum p and position q, vectors in *dim* components, by

$$\frac{\partial h}{\partial p_i}(p,q) = \dot{q}_i \qquad \frac{\partial h}{\partial q_i}(p,q) = -\dot{p}_i \qquad (i=1,2,\ldots dim)$$

where $\dot{q}$ and $\dot{p}$ represent the time derivatives of q and p, respectively*. A transformation of coordinates $T:(p,q)\rightarrow(P,Q)$ also transforms the Hamiltonian h(p,q) into the function H(P,Q). T is called a *canonical* transformation if the Hamiltonian relations hold for H as well:

$$\frac{\partial H}{\partial P_i}(P,Q) = -\dot{Q}_i \qquad \frac{\partial H}{\partial Q_i}(P,Q) = \dot{P}_i \qquad (i=1,\ldots dim)$$

*While time is generally a parameter of the Hamiltonian $h(p,q,t)$, we limit our discussion to time-independent systems. The computational method discussed here can be easily extended to handle the time-dependent case (see [Goldstein]).

Examples

If $h(p,q) := \alpha p_1 + \beta p_2$, let

*T*1:

$$P_1 = \alpha p_1 + \beta p_2 \qquad Q_1 = \frac{p_1}{2\alpha} - \frac{q_2}{2\beta}$$

$$P_2 = \alpha p_1 - \beta p_2 \qquad Q_2 = \frac{q_1}{2\alpha} - \frac{q_2}{2\beta}$$

Then $H(P,Q) := P_1$. T1 is a canonical transformation, since

$$\frac{\partial H}{\partial P_1}(P,Q) = 1 = \dot{Q}_1 = \frac{\dot{q}_1}{2\alpha} + \frac{\dot{q}_2}{2\beta} = \frac{\frac{\partial h}{\partial p_1}}{2\alpha} + \frac{\frac{\partial h}{\partial p_2}}{2\beta}$$

$$\frac{\partial H}{\partial P_2}(P,Q) = 0 = \dot{Q}_2$$

$$\frac{\partial H}{\partial Q_1}(P,Q) = 0 = \dot{P}_1 \quad \frac{\partial H}{\partial Q_2}(P,Q) = 0 = \dot{P}_2$$

However, if we let

*T*2:

$$P_1 = p_1^{\frac{1}{3}} \qquad Q_1 = q_1$$

$$P_2 = p_2^{\frac{1}{3}} \qquad Q_2 = q_2$$

then $H(P,Q) := \alpha P_1^3 + \beta P_2^3$ is not a canonical transformation, since

$\dot{Q}_1 = \dot{q}_1 = \alpha$, but $\frac{\partial H}{\partial P_1}(P,Q) = 3\alpha P_1^2 \neq \alpha$.

3. Finding canonical transformations via generating functions

Finding invariant properties of a Hamiltonian system (e.g. transforms to action-angle variables [Goldstein], Lie transforms to find adiabatic invariants [McNamara], etc.) often motivates the computation of canonical transformations. In many cases, the problem then is to find a canonical transformation that is consistent with desired goal-relations between the original and transformed coordinate variables. The well-known method of generating functions can be used to attack such problems. We present in the remainder of this section a brief review of generating functions, referring the interested reader to [Goldstein] for a more complete presentation of the method.

Generating functions have 2 *dim* independent variables as arguments, in one of four possible arrangements:

(3.1a) $f_1(q,Q)$, where

$$\frac{\partial f_1}{\partial q_i}(q,Q) = p_i \qquad \frac{\partial f_1}{\partial Q_i}(q,Q) = -P_i$$

(3.1b) $f_2(q,P)$, where

$$\frac{\partial f_2}{\partial q_i}(q,P) = p_i \qquad \frac{\partial f_2}{\partial P_i}(q,P) = Q_i$$

(3.1c) $f_3(p,Q)$, where

$$\frac{\partial f_3}{\partial p_i}(p,Q) = -q_i \qquad \frac{\partial f_3}{\partial Q_i}(p,Q) = -P_i$$

(3.1d) $f_4(p,P)$, where

$$\frac{\partial f_4}{\partial p_i}(p,P) = -q_i \qquad \frac{\partial f_4}{\partial P_i}(p,P) = Q_i$$

While the basis for these relations can be derived from "Hamilton's principle" [Goldstein, p. 225], for the purpose of computation they can be assumed as given. In general, the process of finding a canonical transformation consistent with desired coordinate relations and one of (3.1a-d) involves solving an arbitrary system of partial differential equations. However, restricting the form of the old-new coordinate relations to only linear ones, makes the solution considerably easier.

4. Solution in the linear case

If the desired relationships between old and new coordinates are linear* (e.g. $P_1 = \alpha p_1 + \beta p_2$ in the first example above), then the form of the generating function, if it exists, is also linear*:

$$f(v1,v2) := v1 \cdot a + v1 \cdot M \cdot v2^T + v2 \cdot c$$

where $v1$ is either p or q, $v2$ is either P or Q, and

$$a = \begin{bmatrix} a_1 \\ a_2 \\ \cdot \\ \cdot \\ \cdot \\ a_{dim} \end{bmatrix} \quad c = \begin{bmatrix} c_1 \\ c_2 \\ \cdot \\ \cdot \\ \cdot \\ c_{dim} \end{bmatrix} \quad M = \begin{bmatrix} m_{11} & \cdots & m_{1dim} \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ m_{dim1} & \cdots & m_{dimdim} \end{bmatrix}$$

*Strictly speaking, the desired relationships must be such that $P=g(p)$ and $Q=h(q)$ or $P=g(q)$ and $Q=h(p)$ for some linear functions g and h, and that g and h do not violate the restriction that all the Q_i and P_i be mutually independent. The "desired" linear relations generated by actual physical problems are usually in this form.

are matrices of constants.

The generating function relations, in the case $v1 = p$, $v2 = Q$, are then

$$\begin{bmatrix} -q_1 \\ \cdot \\ \cdot \\ \cdot \\ -q_{dim} \end{bmatrix} = \frac{\partial f}{\partial p}(p,Q) = a + M \cdot Q^T \qquad \begin{bmatrix} -P_1 \\ \cdot \\ \cdot \\ \cdot \\ -P_{dim} \end{bmatrix} = \frac{\partial f}{\partial Q}(p,Q) = c + p \cdot M$$

which represent $2\,dim$ linear equations in $2\,dim$ independent variables (the p_i and Q_i), with $2\,dim + dim^2$ unknowns (a,c, and M).

Thus, the problem in the linear case can be summarized as: Given a list of desired relations among origin (p,q) and target (P,Q) coordinate systems,

a) select one of the four functional forms (3.1a-d) consistent with the independence condition for the generating function variables, and

b) solve for a, c, and M. If all the components of a, c, and M have values forced upon them by the defining relations, the undetermined coefficients can be chosen to follow the guidelines of some arbitrary style, e.g. let elements of a and c be zero, M the identity.

To continue the example above, if we desire

$$P_1 = \alpha p_1 + \beta p_2$$

then

$$-\alpha p_1 - \beta p_2 = c_1 + m_{11} p_1 + m_{12} p_2$$

implying

$$-\alpha = m_{11} \qquad c_1 = 0 \qquad -\beta = m_{21}$$

with m_{21}, m_{22}, c_2, and all of a undefined.

5. The MACSYMA implementation

We now present an overview of the MACSYMA implementation of the procedure suggested by the discussion in the preceding section.

5.1. Design decisions

We decided to let the user give values to the undetermined coefficients in M, a, and c rather than use a fixed set of rules for determining them. This was to allow interactive "design" of the target coordinate system's Hamiltonian through trial-and-retrial of candidate values for the undetermined coefficients, after part of a transformation had been generated by constraints input to *lcpt*.

We attempted to make the program text reflect the steps of the procedure outlined in section 3 cleanly and clearly, with comprehensibility chosen over efficiency considerations within the basic framework. We discuss the performance of the MACSYMA implementation in section 6. The success of the attempt at "readable" programming is discussed starting in section 7.

5.2. A user's guide to *lcpt*

Detailed information on *lcpt* can be found in Appendix C, as well as the program listing in Appendix A. Appendices B1 and B2 contain examples of interactive lcpt use. Basically, one loads the *lcpt* routines in by giving the MACSYMA command

```
(C1)  batch(htlcpt,>,dsk,char);
```

When the command has finished (i.e., when the BATCH DONE message is printed), the user then enters the definition of the original Hamiltonian, e.g.,

```
(C16)    H(P,Q):=P[1]^2/2+P[2];
```

The list of desired relations between the original (LP,LQ) and target (BP,BQ) coordinate systems can then be entered:

```
(C17)    LCDF:[BP[2]=-N^2+N*LP[1],LP[2]=2*BP[1]-BP[2]];
```

as well as any desired relations between the time derivatives of the origin and target coordinate variables:

```
(C18)    LCDOTDF:[];
```

The user may also desire to label the lists of origin and target variable names:

```
(C19)    ORIGIN:[LP,LQ];

(C20)    TARGET:[BP,BQ];
```

as well as the dimensionality of the coordinate systems:

```
(C21)    DIM:2;
```

Lcpt may then be invoked by giving it all the pertinent information:

```
(C22)    LCPT(LCDF,LCDOTDF,ORIGIN,TARGET,DIM,H);
```

The results printed include: origin coordinates in terms of target coordinates and vice versa, the generating function selected, and the transformed Hamiltonian. Various error or failure messages will be printed if no generating function can be found. Both the generating function and the transformed Hamiltonian are given unique names (within one MACSYMA session) and may be used subsequently like any other MACSYMA function.

6. Limitations, performance

The present implementation's execution is limited mainly by the space and time costs of the algorithm used by SOLVE for linear systems, as the dimension *dim* of the underlying coordinate system grows. Practically speaking, this means that systems with *dim* > 6 cannot be feasibly solved by *lcpt*. There are ways to avoid exponential intermediate expression growth (e.g. by re-arranging the lcdf equations so as to allow substitution of other[v1] and other[v2] from the generating function equations (2a) and (2b), instead of solving them for bp and bq); this could easily be installed in future versions if the need arises. The point we wish to emphasize is that program was designed for execution for small *dim*; its inability to handle large systems of high dimension is not an important defect. See Appendices B1 and B2 for execution times for typical *dim* = 2 systems.

Part II Towards reducing the complexity of the programming task

7. Programming Simple Procedures

The *lcpt* procedure can be outlined relatively simply. Furthermore, for the kinds of transformations the MACSYMA implementation was meant to find (e.g. see Appendix B1 and B2), the entire calculation can be done by hand quite feasibly. The real advantages to programming it are of convenience: getting the computer to do essentially tedious algebra, and the (supposed) greater speed and reliability of computer versus human calculation. The convenience is compounded because of the interactive nature of the "target system Hamiltonian design" problem.

Unlike many of the programs in the published literature, *lcpt* is not a long, complex calculation where all of the expected payoff is in a few informative runs. In general, programs of this sort are worthwhile only if they can be used widely (say, as part of a library), or if they were cheap to construct (and therefore to dispose of). A realistic hope would be that *lcpt* is the sort of calculation that a physicist could work out without much consultation with (human) system experts.

Despite the relative simplicity of the *lcpt* procedure, the actual MACSYMA code to implement it runs several hundred lines, and took several weeks to produce. This, as Fitch has noted [Fitch], is symptomatic of applications programming on symbolic systems. In the sequel, we suggest reasons for the complexity of programming *lcpt*, and how the difficulty of it and similar programming tasks might be reduced.

8. Gaps in procedural knowledge

There were many gaps between the information presented in the source text [Goldstein], and the explicit procedure contained in the *lcpt* code. For example, the text actually describes the inverse of the procedure implemented (namely, how to figure out what the coordinate transformation is, given a generating function), did not explicitly state how to decide which form of the generating function (3.1a-d) to use, etc. The implementation problems here are compounded: much of the actual procedure was left implicit in the book and thus must be developed by the programmer; the major part of the effort and program text of *lcpt* occurred for the same tacit details.

We can expect this situation to change with time as the influence of symbolic systems becomes more widespread, just as many current calculus and numerical analysis texts are influenced by the computer-oriented approach (e.g. [McNeary], [Gear]). However, it is clearly unreasonable to expect a textbook's readership to need or want to go through an exposition of general physical principles in program-explicit form. Rather, attention naturally turns towards easing the coding process after the algorithm has been developed from the source text and explicitly stated.

9. Why should anyone care about the readability of MACSYMA programs?

A language for mathematical symbol manipulation should serve as a vehicle for clarification. The ideal from the reader's point of view is to make the mathematics appear conspicuously, and as similar as possible to the notation and conventions of the mathematical source. Presumably, a language which allows this sort of abstract representation will develop a literature of well-written programs that are more accessible to source-readers, and "easier" (i.e. cheaper) for source-writers to produce. *Lcpt* is neither particularly easy to read nor was easy to write. We believe that the MACSYMA language can be improved to support communication of mathematical concepts.

Sandewall notes that typical Lisp code is filled with "difficulties of many kinds, all trivial and uninteresting" [Sandewall, p.62]; that the comprehensibility of Lisp code comes from breaking up the idioms of the task into functions. Since the MACSYMA language is derived from Lisp, programs written in it also tend to be collections of many small functions or array definitions. (see [Char], [Golden] for examples). Thus, current MACSYMA programs are not comprehensible in the same fashion as, say, well-written Pascal programs--variables will not be type declared, subsidiary functions do not have to appear within the lexical scope of their calling procedures, etc. While writing *lcpt*, we found ourselves constantly faced with "three temptations":

The temptation *to do an entire computation in one expression*, as opposed to line-at-a-time chunks ("APLification"). Since MACSYMA functions cannot be called as concisely as APL operators, this leads to collections of highly nested function calls. Because there are few MACSYMA "idioms" of function call sequences (see [Perlis]), this degrades comprehensibility.

The temptation *to make every small subcomputation a function call* ("Balkanization"). At some point the proliferation of function definitions means that the defining text is so far removed from the invocation and there are so many function names that the reader loses track of what is going on. Too many functions also tends to make the invoking lines of computation look like an APLism.

The temptation *to select lists when other data structures or non-lists would be more appropriate ("list- think").* For example, in *lcpt*, one could always refer to the p and q variables as origin[1] and origin[2], but that would obscure the physical reason why they were given different names in the first place, e.g.

diff(apply(h,origin),origin[1])=diff(origin[2],t),

but

diff(apply(h,origin),origin[2])=-diff(origin[1],t)?

While the Lisp-basis of MACSYMA can be used for good, the three temptations can turn it into a tool for devilish obscurity. Constant vigilance is required to guard against lapses of style.

10. Notation for Hamiltonian Physics

Some of the verbosity of *lcpt* comes from developing and programming convenient representations for the operations implicit in the standard notation of Hamiltonian physics. We consider the following to be examples of the sorts of things that should be included in a good library of symbolic physics notation for a "physics environment" of a symbol manipulation system:

1. A dot operator, the derivative with respect to the time variable (by default, say, t), e.g. "f dot" means diff(f,t), "f dot dot" means diff(f,t,2).
2. a) Physics texts traditionally establish a notation context (e.g. stating that p and q are vectors of a certain length, part of a particular coordinate system). Once this has been established, the vectors and their components are used rather loosely, e.g.

$$\frac{\partial h}{\partial p} = \dot{q}$$

$$\frac{\partial h}{\partial p_i} = \dot{q}_i$$

and

$$<\frac{\partial h}{\partial p_1}, \frac{\partial h}{\partial p_2}, \cdots \frac{\partial h}{\partial p_{dim}}> = <\dot{q}_1, \dot{q}_2, \ldots \dot{q}_{dim}>$$

or even

$$<\frac{\partial h}{\partial p_1} = \dot{q}_1, \frac{\partial h}{\partial p_2} = \dot{q}_2, \frac{\partial h}{\partial p_{dim}} = \dot{q}_{dim}>$$

all are used to refer to the same equality. Thus, given the user's declaration that

$$\frac{\partial h}{\partial p} = \dot{q},$$

the system should be able to produce the componentized lists of equalities upon demand. For example, if the dimensionality of the vectors *p* and *q* has been declared, then it should be no special effort to derive from

diff(h(p,q),q) = -(p dot)

the componentized vector/list described by

diff($h(p,q),q_i$) = -(p_i dot), i=1,...*dim*

b) Note also that $\frac{\partial h}{\partial q}$ is given no arguments; once the p-q relation is set up, and the Hamiltonian h defined for the p-q coordinate system, by default all references to h, $\frac{\partial h}{\partial q}$, etc. imply the use of p and q unless explicit exception is made.

c) Sometimes the same symbol is used for both variable and function name, as in

$$Q_i = Q_i(q,p,t)$$

with context providing the disambiguation in further references to the symbol. A useful physics manipulation system should be able to record such relations declared in function declarations, coordinate system declarations, etc. and then be prepared to handle the notational conventions without further instruction by the programmer.

11. Conventional programming notation

As well as physics-based notational extensions, there are also system augmentations we favor so that data and subprocedures can be more conveniently defined and manipulated. For example, in the lcpt problem, currently the only connection the reader has relating the origin system variables referred to as origin[1] and origin[2], the componentized representations

```
        lsmp:[origin[1][1],origin[1][2]...],
        lsmq:[origin[2][1],origin[2][2]...],
```

the list of all components of the origin system

```
        oldvar:append(lsmp,lsmq),
```

and the list of all coordinate variables

```
        allvar:append(origin,target)
```

are the names. Not only does this make keeping track of the hierarchy of data a difficult task, it makes assignments to the p/q variables unnecessarily tedious: neither

```
        lsmp:makelist(origin[1][i],i,1,dim),
        lsmq:makelist(origin[2][i],i,1,dim)
```

nor

```
        map(":",[lsmp,lsmq],
            map(lambda([x],makelist(x[i],i,1,dim)),origin))
```

seem to adequately reflect the structure of the assignments being made.

The introduction of new functional forms would also be highly convenient in many situations in *tcpt*. MACSYMA already allows the introduction of syntax extensions for new or existing functions. However, introducing entirely new forms[+] is often difficult without substantial knowledge of the system underpinnings, usually difficult to view by users. Given the limited resources of most system support groups to cater to all the users' wishes, more facilities should be provided for user-supplied extensions. Having a system with modes and modular structure

[+]Everyone has a favorite set of improvements to MACSYMA syntax. Two from ours would be: a CASE statement to clarify multi-branch IF THEN ELSES (and to keep the program formatter GRIND from displaying such nestings entirely within in the last ten columns of output!), and a projection operator Π to apply succinctly multi-argument functions to composite data structures, e.g. abbreviating

```
        map(lambda([listel],ev(listel,diff)),vectorlist)
```

to

```
        map((Π ev(*,diff)),vectorlist)
```

definitions (see [Griss], [Barton]) would allow users to build upon existing notation without rewriting or having to know the internal workings of the basic system-supplied routine. Having a language that supports user-defined data structures would allow the programmer to make the connections in a hierarchy of symbols more apparent to the reader.

While there is a real danger that given the rope, users will hang themselves by developing mutually incompatible applications environments and utility routines, we feel that the community of symbolic manipulation system users is sophisticated enough to use such tools wisely. Furthermore, some large-scale experimental development is necessary before conflicting or embryonic viewpoints can be brought into consensus.

12. The behavior of system-supplied routines: the case of SOLVE

It is often difficult to massage a problem into a form SOLVE can accept as input, or to manipulate the output of SOLVE into a form amenable for further symbol crunching. We do not intend to discuss here any computational errors of SOLVE (we didn't find any while testing the program), but wish to enumerate the sticking points we had with it in writing *lcpt*:

1. In many applications, one can assume (by an accompanying proof, usually) that there's at most one solution, or one may desire to choose arbitrarily one solution from a set of solutions. Yet the default result of SOLVE is (a list of) one or more lists of labels, which means both FIRST and EV have to be invoked in order to get the actual solution (so that, for example, it can be used as input to SUBST, as it is in *lcpt*).* Thus, in *lcpt*, some of the SOLVE computation has to be undone via the *lcpt* routine PICKONES.
2. The code for manufacturing the list of generating function coefficients for SOLVE to solve for, once EQUATECOEFS had produced a system of equations, was quite involved: the direct approach

```
lvars:append
      (append(apply(makelist,
            [makelist(m[i,j],j,1,dim),i,1,dim])),
       makelist(a[1,j],j,1,dim),
       makelist(c[j,1],j,1,dim)),
solve(eqns,lvars)
```

   is baroque, while

*As of April 1979, a PROGRAMMODE flag was installed in order to allow SOLVE to return a list of lists of equations instead of labels. Thus, use of this flag would make the EV unnecessary, although the programmer still would have to pay attention to the particular forms that SOLVE's answer may take.

```
mata:genmatrix(a,dim,1),
matc:genmatrix(c,1,dim),
matm:genmatrix(m,dim,dim),
lvars:listofvars([mata,matc,matm]),
solve(eqns,lvars),
```

is still verbose. Given that the system could easily be instructed as to the dimensions of a, c, and m (since that information had to be supplied anyway to define the generating function)

```
solve(eqns,[a,c,m])
```

seems a desirable and feasible alternative. The situation is a case of SOLVE not being versatile enough.

3. SOLVE's input and output restrictions in one case did too little, in one case, too much:

   a) If there are more non-trivial equations than variables, then the error message "equations inconsistent" is given, even if the "extra" equations contain none of the variables to be solved for. (In that case, inconsistency is not indicated, merely irrelevance.) Often in the automatic production of equations (as in the EQUATECOEFS process in lcpt), the desired equations are produced in a fashion that generates many more than the ones useful to the solution. This is an example of SOLVE wanting input in a form different from that naturally generated by the problem; the user has to work to supply it.

   b) Conversely, if SOLVE is given more variables than (linear) problem-equations, it returns a list (of lists of labels) of solution-equations that makes no mention of those variables missing from the problem-equations. The *lcpt* procedure INSERTFREE figures out the missing variables and equates them to "arbitrary values". While admittedly this is not always the desired outcome when missing variables occur, the action of INSERTFREE could be provided as an option. This is an example of SOLVE's output being in the wrong form; the user has to work to change things.

There is no easy way out of these difficulties. One alternative is to "unbundle" the actions of SOLVE. For example, in the linear case, there could be one routine with one equation and one unknown as input, returning one solution-equation instead of a list or a label, another routine to handle matrix equations, another one for systems of linear equations given as a list, etc. This would satisfy individuals in particular cases, but would lead to a proliferation of routines. It could conceivably take an expert to decide which one to use in a given situation, and the current "general problem solving" ability of SOLVE, limited as it may be, would be lost. Perhaps a hierarchy of routines should be available--a general solver that tries its best to figure out which problem its input presents it with, and many testing and solving primitives that are available to the expert or specialist. Besides the formal parameters of the problem, a general solver should also be built with enough sophistication to allow command or programmatic direction of its selection of algorithm, its "understanding" of the problem, storage allocation schemes, etc. Cost functions, relational equalities and inequalities, and keywords or phrases could all influence such a general solver, as an alternative to the currently implemented flag sch·me. The problems with SOLVE indicate stumbling blocks any polyalgorithm designer faces in building a problem-solver of such scope.

13. Conclusion, Summary

In summary, we find that the significant effort in programming *lcpt* is expended on

a) developing a procedural statement of the knowledge taken for granted by physicists

b) overcoming strictly technical programming difficulties to allow a succinct, comprehensible description of the *lcpt* procedure.

While the first task will always confront someone who wants to automate physics computations, the second seems largely soluble by symbolic systems designers and user groups, and is itself a considerable barrier towards widespread, casual use of symbolic systems. Since there is no consensus as to what constitutes a good set of operators and structures in the various applications areas of symbolic manipulation, the move towards achieving such a set must for the present be in the form of allowing users freedom to develop the language and notation themselves, and to encourage the development and sharing of intermediate level operations for commonly performed mathematical manipulat ons. Without allowing for easy implementation of human engineering impı ›vements, symbolic system usage will be worthwhile for use only in massive, specialized efforts.

Acknowledgments

We wish to acknowledge useful discussions with Richard Fateman and Barry Trager, who shared their experiences and MACSYMA anecdotes with us.

REFERENCES

[Barton]Barton, David. *Mode package for SCA.* MIT LCS Internal Memorandum, 1978.

[Char]Char, B. and McNamara, B. "LIEPROC: A MACSYMA Program for Finding Adiabatic Invariants of Simple Hamiltonian Systems via the Lie Transform," Lawrence Livermore Laboratory Report UCRL-81974, November, 1978.

[Fitch]Fitch, John. "Mechanizing the Solution of Perturbation Problems," *Fourth International Colloquium on Advanced Computing Methods in Theoretical Physics.* Sponsored by Universite d'Aix-Marseille II, Universite de Provence, Centre National de la Recherche Scientifique, Direction des Recherches et Moyens d'Essais, 1977. pp. 93-98.

[Gear]Gear, C. William. *Numerical Initial Value Problems in Ordinary Differential Equations.* Prentice-Hall: Englewood Cliffs, NJ, 1971.

[Golden]Golden, Jeffrey P. "MACSYMA's Symbolic Ordinary Differential Equation Solver," in *Proceedings of the 1977 MACSYMA User's Conference,* NASA: Washington, D.C., 1977. pp. 1-10.

[Goldstein]Goldstein, Herbert. *Classical Mechanics.* Addison-Wesley: Reading, Massachusetts, 1950.

[Griss]Griss, M. The Definition and Use of Data Structures in REDUCE, in *Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation,* R.D. Jenks, ed. ACM: New York, 1976. pp. 53-59.

[McNamara]McNamara, B. "Super Convergent Adiabatic Invariants with Resonant Denominators by Lie Transforms," *Journal of Mathematical Physics,* vol. 19, no. 10, 1978, pp. 2154-2164.

[McNeary]McNeary, Samuel. *Introduction to Computational Methods for Students of Calculus.* Prentice-Hall: Englewood Cliffs, NJ, 1973.

[Perlis]Perlis, Alan J., and Rugaber, Spencer. *The APL Idiom List,* Yale Research Report #87, April 1977.

[Sandewall]Sandewall, Erik. *Computing Surveys,* vol. 10, no. 1, March 1978, pp. 35-71.

Appendix A
Program listing for LCPT

```
     /*closure computes the transitive closure of MAT, a DIM x DIM
1-0 matrix.  See algorithm in Gries, Compiler Construction
for Digital Computers*/


closure(mat,dim):=block([a],a:copymatrix(mat),
     for I thru dim do
        (for j thru dim do
           (if a[j,I] = 1
              then for k thru dim do a[j,k]:max(a[j,k],a[i,k]))),
     return(a))$

     /*Define "dot" as d by dt */

postfix("dot")$

"dot"(x):=diff(x,t)$

     /*apl-like iota.  iota[k] is the list of
     numbers 1 thru k, presumably the null list if k is zero or negative*/

iota[k]:=makelist(jk,jk,1,k)$

     /*Returns the position (one-indexed) of item in list, 0 if not found */

indexin(list,item):=block([len,ji],
     len:length(list),
     return(catch(
          for ji:1 thru len do if list[ji]=item then throw(ji),
               0)))$

     /*If lvar is not in lgt (see lcpt and lcpt0 for what lgt
     means), then insertfree will insert a lvar=%randomsym equation into
     lgt and return the altered list as a result*/

insertfree(lgt,lvar):=block([i],
     for i in lvar do if freeof(i,lgt) then lgt:cons(i=makesym(%r),lgt),
     return(lgt))$

     /*makesym returns a symbol name beginning with the value of atsym
     (which must be an atomic symbol), ending with an internally generated
     suffix which is different each time makesym is called within a macsyma
     session*/

makesym(atsym):=concat(atsym,?gensym())$

     /*discardfree returns the list lgt expunged of all equations
     that do not contain any of the variables in lvar*/

discardfree(lgt,lvar):=block([],
     kill(testf),
     testf(x):=if freeof(lvar,x) then [] else [x],
     apply(append,map(testf,lgt)))$
```

```
        /*Pickones returns the first set of solutions of lsys in
        lvar, where lsys and lvar are both lists as they would be as inputs
        to SOLVE.  There is an error if there is no solution.*/

pickones(lsys,lvars):=ev(first(solve(lsys,lvars)),eval)$



        /*Given an equation eq and a list of independent variables
        in the equation, listofindvars, equatecoefs returns a list of equations
        that result when the coefficients of an independent variable on both
        sides of the equals sign are equated.  Since not all
        variables will appear on both sides of eq, there may be some 0=coef
        equations, as well as some 0=0 equations.  In addition, the
        constant terms on both sides of the = sign are equated (those terms
        free of all independent variables*/

equatecoefs(eq,listofindvars):=block([i,len,l,a],
        l:[],
          for fir in listofindvars do
             (eq:expand(eq),
             while (hip:hipow(eq,fir))>0 do
                  (a:coeff(eq,fir,hip),
                  l:cons(a,l),
                  eq:expand(subst(fir^hip=0,eq))
                  )
             ),
             /*Throw in constants, too*/
     if eq#'(0=0) then l:cons(eq,l),
     return(l))$


                       /*lcpt  finds a  linear
canonical point transformation satisfying the constraints
given by the user through the variables lcdf, a list of
equations stating the desired relations that should hold
between the variables of the origin coordinate system, and
the target system, and lcdotdf, a list of equations stating
the desired relations that should hold between the
dot-derivatives of those variables.  origin and target are
lists giving the names of the p/q variables of the two
coordinate systems respectively (e.g. [lp,lq] and
[bigp,bigq])--the momentum variable name is the first
element of the list, the position variable name the second.
dim is an integer-valued variable stating the dimension of
the system, h the name of the hamiltonian function of the
origin system (defined as h(p,q):=....) */

lcpt(lcdf,lcdotdf,origin,target,dim,h):=block([oldispfl,lbigq,lsmq,lbigp,
     lsmp,lnewvar,loldvar,a,i,j,k,dep,allvar,otlen,dp,close,genvar1,genvar2,
     genpos,doap,endflag,vx1,vx2,depv,vix,genok],
          /*initialize arrays*/
     kill(vmake,discnotsub,sm,m,a,c,other,def,dep,dp,signpg),
          /*vmake returns a list of components of the vector vname*/
```

```
vmake(vname):=makelist(vname[i],i,1,dim),
      /*list of origin and target variables,subscripted*/
lnewvar:apply(append,map(":",[lbigp,lbigq],map(vmake,target))),
loldvar:apply(append,map(":",[lsmp,lsmq],map(vmake,origin))),
allvar:append(origin,target),
      /*list of possible generating function variable pairs*/
genpos:[[origin[1],target[1]],[origin[1],target[2]],[origin[2],
      target[1]],[origin[2],target[2]]],
      /*Generate symbolic coefficients of matrices a,c, and m*/
mata:genmatrix(a,dim,1),
matc:genmatrix(c,1,dim),
matm: genmatrix(m,dim,dim),
      /*other returns the other vector name of the pair
      of canonical vector names in either origin or target*/
other[x]:=if member(x,origin) then origin[3-indexin(origin,x)]
      else if member(x,target) then target[3-indexin(target,x)]
               else error(print("no other:",x)),
      /*Sign for relation between partial derivatives of
      generating function and variables*/
signpg[var]:=if var=origin[1] then -1
      else if var = origin[2] then 1
          else if var=target[1] then 1
               else if var=target[2] then -1,
      /*Find suitable pair of variables for generating function*/
discnotsub(x):=block([var],
      return(if length(x)#1
          then []
          else if member(var:inpart(x,0),allvar)=true
               then [var] else [])),
dep[i,j]:=0,
for eq in lcdf do
(     /*get list of subscripted vars in eq (without
      subscripts). Keep only origin, target variables*/
      lsubvar:apply(append,map(discnotsub,listofvars(eq))),
      for listel in lsubvar do
          for listel2 in lsubvar do
               dep[listel,listel2]:1
),
      /*otlen is the length of origin and target combined, i.e.
      four with typical p,q lists*/
otlen:length(target)+length(origin),
dp[i,j]:=dep[allvar[i],allvar[j]],
      /*close is the transitive closure of dependency matrix*/
close:closure(genmatrix(dp,otlen,otlen),otlen),
genvar1:[],
      /*Complem(l) returns a list of those integers 1-otlen which are
      NOT in the list l*/
complem(l):=apply(append,
      makelist(if member(i,l) then [] else [i], i , 1,otlen)),
      /*compiota[k] produces a list-vector that
      is of length otlen, all ones except for the k-th position,
      which is zero.*/
compiota[k]:=makelist(if k=i then 0 else 1,i,1,otlen),
endflag:false,
      for i in genpos unless endflag = true do
      (catch(
            vx1:apply(indexin,[allvar,i[1]]),
            vx2:apply(indexin,[allvar,i[2]]),
            /*Must be independent */
            if close[vx1,vx2]=1 then throw([]),
            for listel in complem([vx1,vx2]) do
                  /*All others must be dependent on vx1 or vx2,
                  if dependent on anything other than itself.*/
                  if compiota[listel].col(close,listel)#0 and
                        close [listel,vx1]=0 and close[listel,vx2]=0
                                    then throw([]),
            /*Success! Pick these as the generating function variables*/
            genvar1:i[1],genvar2:i[2],endflag:true
            )
      ),
      if genvar1=[]
        then error("Couldn't find valid generating function variables"),
      return(lcpt0(lcdf,lcdotdf,origin,target,genvar1,genvar2,dim,h)))$


lcpt0(lcdf,lcdotdf,origin,target,genvar1,genvar2,dim,h):=block([dispflag,lv1,lv2,
      geneq,gt1,gt2,i,l1,l2,dgt1,dgt2,lhamrel,lm,lmsoleq,lensol,lout,ik,
      appl12,gt,gtinnew,gtinold],
            /*list of generating function variables, subscripted */
      map(":",[lv1,lv2],map(vmake,[genvar1,genvar2])),
            /*Compute generating function f(genvar1,genvar2)*/
      geneq:(lv1).mata+(lv1).matm.transpose(lv2)+ matc.transpose(lv2),
            /*Also define generating equations in terms of new coords*/
            /*Turn off display of intermediate equations */
      oldispfl:dispflag,
      dispflag:false,
            /*genvarequations(var,genfunceq) is a template for production
            of the equations (3.1a-d) of section 3 of the accompanying
            paper*/
      genvarequations(var,genfunceq):=makelist(other[var][i]=
            signpg[var]*diff(genfunceq,var[i]),i,1,dim),
      gt1:pickones(genvarequations(genvar2,geneq),lnewvar),
      gt2:pickones(genvarequations(genvar1,geneq),lnewvar),
            /*Find relations among m[i,j] implied by lcdf
            and generating equations*/
      lcdf:subst(append(gt1,gt2),lcdf),
      l1: if lcdf # [] then apply(append,
                  map(lambda([x],equatecoefs(x,loldvar)),lcdf))
            else [],
            /*List of new and old target-dot variables (derivatives)
            with respect to time */
      depends(target[1],t),depends(target[2],t),
      depends(origin[1],t),depends(origin[2],t),
      lnd:(lnewvar dot),
      lod:(loldvar dot),
            /*Express given dotequations in terms
            of target dot variables*/
      lcdotdf:append(lcdotdf,(lcdf dot)),
if length(lcdotdf)>0 then
  (
        /*Generate time derivs. of gen. eqs.*/
  dgt1:(gt1 dot),
  dgt2:(gt2 dot),
        /*find values of qdot and pdot from hamiltonian */
  lhamrel:append(makelist((lsmp[i] dot)=-diff(apply(h,origin),
              lsmq[i]),i,1,dim),
        makelist((lsmq[i] dot)=diff(apply(h,origin),lsmp[i])
              ,i,1,dim)
              ),
        /*Substitute q/p dot values for q/p dot symbols*/
  lcdotdf:subst(append(dgt1,dgt2),lcdotdf),
  lcdotdf:subst(lhamrel,lcdotdf),
  l2:apply(append,map(lambda([x],equatecoefs(x,loldvar)),lcdotdf))
  )
else l2:[],
        /*Generate list of m-variables, and solve for them */
  lm:listofvars([mata,matm,matc]),
  freefmac(x):=if freeof(m,x) and freeof(a,x) and freeof(c,x)
        then [] else [x],
  appl12:apply(append,map(freefmac,append(l1,l2))),
  lmsoleq:ev(solve(appl12,lm),eval),
  print(if (lensol: length(lmsoleq))=0 then "No Solution"
        else if lensol=1 then "solution"
              else "multiple solution"),
  lout:[],
  for ik:1 thru lensol do
  (
        geneq:subst(lmsoleq[ik],[mata,matm,matc]),
        substarbval(x):=if length(x)>1
              then (varname:inpart(x,0),
                    if varname='a or varname ='c
                          then geneq:subst([x=0],geneq)
                          else if varname ='m
                                then geneq:subst([x=
                                          if inpart(x,1)
                                          =inpart(x,2)
                                          then 1 else 0],
                                      geneq)
                    ),
        map(substarbval,listofvars(geneq)),
        genfuncname:makesym('f),
        apply(define,[funmake(genfuncname,[genvar1,genvar2]),
              lv1.geneq[1]+lv1.geneq[2].transpose(lv2)
                    +geneq[3].transpose(lv2)]),
        geneq:apply(genfuncname,[lv1,lv2]),
              /*Restore display flag*/
        dispflag:oldispfl,
        /*Display generating function, variable transforms*/
        gt:append(apply(genvarequations,[genvar2,geneq]),
              apply(genvarequations,[genvar1,geneq])),
        gtinnew:solve(insertfree(discardfree(gt,lnewvar),lnewvar),lnewv ar),
        gtinold:solve(insertfree(discardfree(gt,loldvar),loldvar),loldv ar),
          thname:makesym('targeth),
          apply(define,[funmake(thname,target),
                    subst(ev(gtinold,eval),apply(h,origin))]),
          if dispflag#false then
          (     print(" "),print("Generating function is"),
                ldisp(apply(dispfun,[genfuncname])),
                print(" "),print("Target Hamiltonian is"),
                ldisp(apply(dispfun,[thname]))
          ),
          lout:cons([first(gtinnew),first(gtinold),genfuncname,
                thname],lout)
     ),
     return(lout))$
```

Appendix B1

One example of LCPT usage

```
(D20)                                  [DSK, CHAR]

(C21) BATCH(HTDEM,>)$

(C22) ORIGIN:[LP,LQ]$

(C23) TARGET:[P,Q]$

(C24) H(P,Q):=P[1]^2/2+P[2]$

(C25) LCDF:[P[2]=-N^2+N*LP[1],LP[2]=2*P[1]-P[2]]$

(C26) LCDOTDF:[]$

(C27) LCPT(LCDF,LCDOTDF,ORIGIN,TARGET,2,H);

MAKEL FASL DSK MAXOUT being loaded
Loading done

GENMAT FASL DSK MAXOUT being loaded
Loading done
A
 1, 1
warning - unbound element - GENMATRIX
C
 1, 1
warning - unbound element - GENMATRIX
M
 1, 1
warning - unbound element - GENMATRIX

MDOT FASL DSK MACSYM being load^d
Loading done

SOLVE FASL DSK MACSYM being loaded
Loading done
SOLUTION

CONCAT FASL DSK MAXOUT being loaded
Loading done
SOLUTION
```

(E37) $$Q_1 = 2\ LQ_2$$

(E38) $$P_2 = LP_1\ N - N^2$$

(E39) $$Q_2 = - \frac{LQ_2\ N - LQ_1}{N}$$

(E40) $$P_1 = - \frac{N^2 - LP_1\ N - LP_2}{2}$$

SOLUTION

(E41) $$LQ_2 = \frac{Q_1}{2}$$

(E42) $$LP_1 = \frac{N^2 + P_2}{N}$$

(E43) $$LQ_1 = \frac{(2\ Q_2 + Q_1)\ N}{2}$$

(E44) $$LP_2 = 2\ P_1 - P_2$$

GENERATING FUNCTION IS

(E45) $$FG0060(LP,\ Q) := Q_2\ N^2 + \frac{Q_1\ N^2}{2} + LP_1\ (-\ Q_2\ N - \frac{Q_1\ N}{2}) - \frac{Q_1\ LP_2}{2}$$

TARGET HAMILTONIAN IS

(E46) $$TARGETHG0079(P,\ Q) := \frac{(N^2 + P_2)^2}{2\ N^2} - P_2 + 2\ P_1$$

```
(D46) [[[E37, E38, E39, E40], [E41, E42, E43, E44], FG0060, TARGETHG0079]]

(C47) TIME(%);
TIME or [TOTALTIME, GCTIME] in msecs.:
(D47)                               [[7155, 3009]]

(C49) CLOSEFILE(HTOUT,>)$
```

Appendix B2

Another example of LCPT usage

```
(D18)                                    [DSK, CHAR]

(C19) BATCH(TESTLC,>)$

(C20) DEPENDS(BQ,T)$

(C21) DEPENDS(BP,T)$

(C22) DEPENDS(SMP,T)$

(C23) DEPENDS(SMQ,T)$

(C24) LCDF:[BP[1] = SMP[2]*BETA+SMP[1]*ALPHA]$

(C25) LCDOTDF:[DIFF(BQ[1],T) = 1,DIFF(BQ[2],T) = 0]$

(C26) TARGET:[BP,BQ]$

(C27) ORIGIN:[SMP,SMQ]$

(C28) H(P,Q):=ALPHA*P[1]+BETA*P[2]$

(C29) LCPT(LCDF,LCDOTDF,ORIGIN,TARGET,2,H);

MAKEL FASL DSK MAXOUT being loaded
Loading done

GENMAT FASL DSK MAXOUT being loaded
Loading done
A
 1, 1
warning - unbound element - GENMATRIX
C
 1, 1
warning - unbound element - GENMATRIX
M
 1, 1
warning - unbound element - GENMATRIX

MDOT FASL DSK MACSYM being loaded
Loading done

SOLVE FASL DSK MACSYM being loaded
Loading done

ALGSYS FASL DSK MACSYM being loaded
Loading done
SOLUTION

CONCAT FASL DSK MAXOUT being loaded
Loading done
SOLUTION

(E33)                         BP   = SMP  BETA + SMP  ALPHA
                                1       2           1

                             (%R1 + SMQ ) %R4 + (- %R2 - SMQ ) %R3
                                       1                    2
(E34)               BQ  = - --------------------------------------
                      1                %R3 BETA - %R4 ALPHA
```

```
(E35)                 BP   = - %R5 - SMP   %R4 - SMP   %R3
                        2                 2           1

                       (%R1 + SMQ  ) BETA + (- %R2 - SMQ  ) ALPHA
                                 1                      2
(E36)           BQ   = - ----------------------------------------
                  2                %R3 BETA - %R4 ALPHA
SOLUTION

                              (%R5 + BP  ) BETA + BP   %R4
                                       2            1
(E37)                SMP   = - -------------------------
                        1         %R3 BETA - %R4 ALPHA

(E38)                  SMQ   = BQ   ALPHA - BQ   %R3 - %R1
                          1      1            2

(E39)                  SMQ   = BQ   BETA - BQ   %R4 - %R2
                          2      1           2

                             (%R5 + BP  ) ALPHA + BP   %R3
                                      2             1
(E40)                   SMP   = ---------------------------
                           2       %R3 BETA - %R4 ALPHA

GENERATING FUNCTION IS
(E41) FG0188(SMP, BQ) := SMP   (BQ   %R4 - BQ   BETA)
                            2     2          1

                   + SMP   (BQ   %R3 - BQ   ALPHA) + BQ   %R5 + SMP   %R2 + SMP   %R1
                        1     2          1               2           2           1


TARGET HAMILTONIAN IS
                              ((%R5 + BP  ) ALPHA + BP   %R3) BETA
                                        2             1
(E42) TARGETHG0219(BP, BQ) := ----------------------------------
                                   %R3 BETA - %R4 ALPHA

                                             ALPHA ((%R5 + BP  ) BETA + BP   %R4)
                                                             2            1
                                           - --------------------------------
                                                   %R3 BETA - %R4 ALPHA

(D42) [[[E33, E34, E35, E36], [E37, E38, E39, E40], FG0188, TARGETHG0219]]


(C44) TIME(%);
TIME or [TOTALTIME, GCTIME] in msecs.:
(D44)                          [[12450, 4867]]

(C45) CLOSEFILE(HTOUT2,>)$
```

APPENDIX C
Detailed description of *lcpt*

1. Input to the procedure

Procedure *lcpt* (origin,target,lcdf,lcdotdf,dim,h) has as input parameters:

·**origin:** a list of the vector names of the original coordinate system's variables, e.g. [*lp* ,*lq*]. Momentum is first, position second in the list.

target: a similar list of names for the desired coordinate system, e.g. [*bp* ,*bq*].

lcdf: a list of defining equations, e.g.

$$[bp[1] = \alpha * lp[1] + \beta * lp[2]].$$

lcdotdf: a list of defining relations among the time derivatives of the coordinates (e.g. [*diff* (*bq*[1],*t*) = 1, *diff* (*bq*[0],*t*) = 0])+

dim: the dimension of the coordinate systems (origin and target have the same dimension).

h: the name of the origin system's Hamiltonian function, defined within the MACSYMA prior to the invocation of *lcpt*.

2. Output of procedure

The output of *lcpt* is a list whose components consist of:

a) A list of labels for the equations expressing the target coordinates (bp,bq) in terms of origin coordinates (*lp* ,*lq*).

b) A list of labels for the equations expressing (*lp* ,*lq*) in terms of (*bp* ,*bq*).

c) The generating function *f* (*genvar1* ,*genvar2*), where *genvar1* and *genvar2* are the vector names of the generating function variables, selected from [*lp* ,*lq* ,*bp* ,*bq*].

d) The Hamiltonian of the target coordinate system, *H*(*bp* ,*bq*).

If the global variable DISPFLAG is true, then the equations produced in a) and b) above will be printed out at the terminal during the computation, in addition to the list of pointers returned as the result. Demonstration input and output for *lcpt* can be found in Appendices B1 and B2.

+Note: MACSYMA must be informed beforehand that the coordinate symbols in fact do depend on t via the *depends* command, e.g. depends(bq,t).

3. Outline of MACSYMA version of lcpt

The details of the *lcpt* implementation are outlined below. The numbering corresponds to that in the program listing, for the MACSYMA function lcpt of Appendix B.

1. Select generating function variables *genvar1* and *genvar2*, given the interdependencies implicit in equations lcdf. Since *genvar1* and *genvar2* must be independent, this involves constructing a dependency matrix, taking its transitive closure, and choosing (arbitrarily) one of the four possibilities

$$[(lp,bp),(lp,bq),(lq,bp),(lq,bq)]$$

which is consistent with the independence restriction.

2. Compute generating equations

$$signpg[genvar1]\cdot other[genvar1] = \frac{\partial f}{\partial genvar1}(genvar1,genvar2) \qquad (2a)$$
$$= a + genvar2\cdot M$$
$$signpg[genvar2]\cdot other[genvar1] = \frac{\partial f}{\partial genvar2}(genvar1,genvar2) \qquad (2b)$$
$$= genvar1\cdot M + c$$

where

$$other[x] := \begin{cases} lp & \text{if } x = lq \\ lq & \text{if } x = lp \\ bp & \text{if } x = bq \\ bq & \text{if } x = bp \end{cases}$$

(i.e. the other variable of the pair of Hamiltonian coordinates), and

$$signpg[x] := \begin{cases} -1 & \text{if } x = lp \\ 1 & \text{if } x = lq \\ 1 & \text{if } x = bp \\ -1 & \text{if } x = bq \end{cases}$$

Signpg reflects the necessary sign changes for the four possible generating functions.

3. Rearrange (2a,b) to express (*other*[*genvar1*],*other*[*genvar2*]) in terms of (*genvar2*,*genvar1*):

$$other[genvar1] = signpg[genvar1]\,(a + genvar2\cdot M) \qquad (3a)$$
$$other[genvar2] = signpg[genvar2](a + genvar1\cdot M) \qquad (3b)$$

4. Put (3a,b) in another form:

$$bp_i = \text{a linear function of}(other[genvar1], genvar2) \tag{4a}$$

$$bq_i = \text{a linear function of}(other[genvar2], genvar1). \tag{4b}$$

(This can always be done, since it is a system of 2 * *dim* linear equations with the 4 * *dim* variables.)

5. Substitute the right hand sides of (4a,b) for bp_i and bq_i in lcdf. This produces equations with only *lp* and *lq* as independent variables. Equate the right and left hand side coefficients of the (*lp*,*lq*) variables. This generates up to 2**dim***len*(*lcdf*) linear equations involving *a*, *M*, and *c*, completely free of any coordinate variables.

6. Differentiate (3a) and (3b) with respect to time, generating the equations:

$$other[genvar1] = \dot{genvar2} \cdot M \tag{6a}$$

$$other[genvar2] = \dot{genvar1} \cdot M \tag{6b}$$

Solve these equations for $(\dot{bp}, \dot{bq})$.

7. Equate (6a,b) with (3a,b). For $\dot{lq}$, and $\dot{lp}$, substitute the expressions for $\frac{\partial h}{\partial lp}$ and $-\frac{\partial h}{\partial lq}$ respectively. This results in a system of equations whose independent variables are *only* those found in the expressions for $\frac{\partial h}{\partial lp}$ and $\frac{\partial h}{\partial lq}$. Equate the coefficients of each independent variable taken from the right and left hand side of each equation, as well as the constant terms of each equation.

8. Solve the system formed from combining the results of (5) and (7) for the coefficients *M*, *a*, and *c*. Assign "arbitrary symbols" *%Ri* (i varying) to those coefficients not determined by the given equations.

9. If DISPFLAG is true, output equations:

$$(bp, bq) \text{ in terms of } (lp, lq), \tag{9a}$$

$$(lp, lq) \text{ in terms of } (bp, bq), \tag{9b}$$

$$\text{the generating function } f(genvar1, genvar2) \tag{9c}$$

by substituting the result of (8) into the basic generating function expressions given by (2a,b) and solving for the appropriate variables. Also print the

$$\text{the transformed Hamiltonian } H(bp, bq). \tag{9d}$$

The value returned by *lcpt* is a list consisting of: a list of labels for (9a), a list of labels for (9b), the name of the generating function (9c), and the name of the Hamiltonian function (9e).