Design of the Stiff Integrators in the NAG Library

M. Berzins
Department of Computer Studies
University of Leeds
Leeds LS2 9JT

R.W. Brankin
Numerical Algorithms Group Ltd.
Mayfield House
256 Banbury Road
Oxford OX2 7DE

I. Gladwell
Department of Mathematics
University of Manchester
Manchester M13 9PL

July 17, 1987

Abstract

This paper describes the design philosophy behind the recent replacement of the NAG Ordinary Differential Equation (ODE) stiff integrators. This replacement was is intended to update the ODE chapter algorithmically but, more importantly in the context of this paper, it provides a more flexible interface than has been available in the past. This interface is designed to permit a wide variety of problem and method definitions, to provide flexibility when introducing new methods or problem definitions in the future and to allow straightforward use of the software as the integrator in packages implementing, for example, the method of lines for parabolic partial differential equations.

1 Introduction

We present an overview of the design considerations which determined the structure and details of the new generation of NAG stiff solvers. These solvers replace an existing set of routines which are functionally inferior and which have a much less user-friendly interface. It is our aim to emphasize the points of continuity and the points of change in the new design.

The aims of the redesign can be described broadly as follows:

(i) to provide continuity and stability for the users of the current NAG stiff solvers;

(ii) to provide improved easy-to-use interfaces and a general simplification of interfaces;

(iii) to extend the functionality of the solvers to implicit and differential/algebraic systems;

(iv) to provide flexibly structured open ended software (based on implicit methods using modified Newton or functional iteration) so that adding new integration methods is a simple matter;

(v) to provide a design and an interface which makes using the solvers from other packages a relatively simple matter; and

(vi) to provide an interface which permits the solver to be used at the "assembly language" level in packages written in languages more "robust" than Fortran 77 (for example in ADA) when a mixed language environment is available.

In Section 2, we give a brief historical introduction to the NAG stiff solvers and, insofar as it is pertinent, to other NAG ODE solvers. In Section 3, the main design considerations for the new stiff solvers are outlined. In Section 4, we describe the actual design in more detail and some novel features of a new easy-to-use routine. Finally, in Section 5, three practical examples are used to show how the improved design permits a relatively simple use of the codes in a package environment.

2 Historical Development

In the mid-1970's one of the authors (Gladwell [1]) redesigned the NAG Library ODE (D02) chapter to provide what were at that time state-of-the-art codes which used a common interface across all the initial value methods employed. The methods were based on Runge-Kutta-Merson, Adams and Backward Differentiation Formula (BDF) algorithms. For each algorithm the design is similar to that illustrated in Figure 1 for the BDF integrators. There are four levels of software. At the uppermost level are a set of easy-to-use interval oriented integrators designed to solve many of the simple problems arising frequently in practice and to provide an easy introduction to ODE solving for naive users, who are a significant proportion of the market for Library software. The four routines listed below each solve the stiff system of order N

$$y' = f(x, y), y(a) = A, \ x > a \tag{2.1}$$

and have some additional features:

(i) D02EAF: Integrates across a range using a numerically determined Jacobian and a scalar relative error test with an absolute error threshold related to machine roundoff.

(ii) D02EBF: Integrates across a range permitting intermediate output at user defined points (through a user supplied subroutine). A choice of analytical or numerical Jacobian is permitted and user specified scalar relative, absolute or mixed error control is allowed.

(iii) D02EGF: Specified as D02EAF but integrates to the first point after a where a specified component of y takes a given value.

(iv) D02EHF: Integrates as D02EBF but without intermediate output; instead integrates to the first point after a where a specified function $h(x, y)$ changes sign.

At the next lower level of Figure 1, a single general purpose integrator, D02QBF, is supplied for the user with a problem which does not fit the specifications of the D02ExF routines. D02QBF is interval oriented but

with a comprehensive set of interrupts which permit also step or output oriented mode of operation and provides more flexible error control than the easy-to-use routines.

The design of D02QBF was similar in many respects to many other ODE codes of its era. DVERK [3], DE/STEP [4], RKF45 [5], the DEPAC codes of the SLATEC Library [6], and even codes of more recent design such as the LSODE collection [7] and the ODEPACK standard interface [8] share a number of features in common with D02QBF. Amongst these codes it is a matter of taste which particular interface is to be preferred. Suffice it to say that D02QBF was designed specifically for use in a Library where the user is presented only with the compiled (object) code, is permitted neither to communicate with the Library routines through COMMON nor to make internal modifications to the code.

The other codes at the second level in Figure 1 are the interpolation routines, D02XHF and D02XGF. They provide a continuous approximation to the solution by the same interpolation procedure used internally by the code but differ in approximating either a single specified component of the solution or all the components simultaneously. To compute the solution these codes need access to scaled derivative information produced by the integrator. Prudence, and to a lesser extent library design, dictated that this information be passed by workspace. This mode of communication is desirable for the following reasons. When two pieces of code, such as an integrator and an associated interpolant, may be used independently it is necessary to specify precisely their method of intercommunication. In a library, the use of COMMON for this purpose implies revealing more of the internal organisation of the codes to the user than is strictly necessary. Also, in an overlaying or in a multiprocessing environment, the COMMON blocks must be saved separately and so their use should be avoided.

Note, that the D02ExF routines make direct calls to the routines at the second level and also, in some cases, to routines at the lowest level; these latter are themselves NAG Library routines callable directly from a user's program. In contrast D02QBF calls a number of auxiliary routines directly which are not "visible" to the user, and are not documented in NAG Library manuals and were taken, with minor modifications, from the Gear (Revision 3) package [9].

The other ODE solvers in the NAG Library have a slightly simpler design, described in [1]. The aim of the design was to permit users to switch between codes implementing different algorithms with minimum modification to the calls in their programs. There is every reason to believe that this has proved a popular innovation.

Despite the success of the design outlined above there are a number of deficiencies and restrictions from the point of view of use in a Library environment. In the next section we show how the design may be improved to provide a more flexible environment for the solution of stiff systems. The improved design is described in more detail in Section 4 and a schematic representation of this improved design is given in Figures 2a–c.

3 Design Considerations

In this section the reasons behind the choice of interface for the new NAG stiff solvers are discussed. In many respects the design is borrowed from that of the stiff integrators in the partial differential equation (PDE) package SPRINT [10] but this design has been modified to provide a cleaner and simpler ODE solving interface. In Section 5 we describe how this redesigned interface is to be reused in a set of PDE solvers for the NAG Library.

Some of our observations apply equally to ODE solvers in general whilst others are specific to codes in a Library environment. Amongst these latter observations many are relevant to other areas besides ODE solving.

3.1 Continuity and its Impact on Design

In the introductory section we described the interfaces used in the earlier NAG Library stiff solvers and remarked that this design was not specific to the stiff solvers. When making a major modification of a chapter of Library software it must be borne in mind that this software may have many adherents already. It is important to these users that their programs continue to run, preferably with no modification at all but at worst with a few minor modifications. This could be achieved by leaving the old software unchanged but ceasing to recommend it and just adding the new software to the Library. However this approach would deny existing users the benefits

to be gained by using the more modern software as normally they would see no reason to change working programs. Also, in the long term, this catholic approach could have serious consequences for library management.

Accordingly, the D02ExF routines are replaced by others with almost identical interfaces (there are some changes in workspace definition and size). Also, the general purpose routine D02QBF has been replaced by another, D02QDF, with an almost identical interface. This allows the user to switch from any of the above mentioned routines to the corresponding, as yet unchanged, Runge-Kutta-Merson and Adams codes in the NAG Library with minimal programming effort.

The new software has two major additions. First, in response to user prompting, an easy-to-use code D02EJF has been supplied which combines intermediate output and root finding in a simple driver routine. At the same time, for reasons of continuity and uniformity the temptation to withdraw D02EGF has been resisted. The availability of two root finders, D02EGF and D02EHF, with similar capability may confuse users who do not appreciate the gains apparent to the software designer of solving only the problem of finding a sign change in a component of the solution rather than the general root finding problem. With the advent of D02EJF the multiplicity of codes for similar problems is likely to prove even more confusing and so the documentation designed to assist the user has been changed so that in almost all circumstances the new routine is recommended, the other D02ExF routines being retained for the above stated reasons of continuity and of uniformity with other codes available in the ODE chapter. In the long term it is intended to provide routines implementing Adams and Runge-Kutta methods with a similar interface to that of D02EJF. This novel interface is considered in some detail in subsection 4.2.

The second novel feature of the new software is its greater functionality as it provides a variety of methods and a range of problem structures within a single framework. This will be described in subsection 4.1, after we have explained the design requirements for the software.

3.2 Functionality or Simplicity?

We have provided an extensible framework which permits the software designer to add new integration algorithms within the structure We have

provided a set of integrators:

(i) BDF (an adaptation of LSODE [7] with error estimates based on the work of Petzold [11]),

(ii) blended formulae (based on the code of Skeel and Kong [12]), and

(iii) a type insensitive θ-method with an error estimate [13] (for use in the method of lines only in the first instance).

We also provide a choice of linear algebra for varying structures of Jacobian $(\partial f_i/\partial y_j)$. In the first release full, banded and sparse matrix structures are provided. The linear algebra codes used are already available in the NAG Library; in particular the code for sparse problems is based on the Harwell Library code MA28 [14].

It is our intention to provide an interface capable of handling both explicit problems of type (2.1) and implicit (including differential/algebraic) problems which may be written in the quite general linearly implicit form

$$M(t,y)\dot{y} = g(t,y) \tag{3.1}$$

where M can, and often will, be singular. The software does not even require that the user specifies M and g directly. Instead, given any y and $\dot{y}$, the residual

$$M(t,y)\dot{y} - g(t,y) \tag{3.2}$$

and the product

$$M(t,y)\dot{y} \tag{3.3}$$

must be specified separately. Though this might seem an obtuse approach it does in fact permit great algorithmic flexibility.

All these options are provided through a single interface without employing subroutine arguments which are sometimes redundant; for example arguments defining bandwidths are redundant in the case of a full matrix and are inadequate to describe the sparse case. Such redundant parameters are common in interfaces of modern stiff solvers and are a potential source of confusion and error for the inexperienced user. This redundancy can be a particularly serious matter for Library software where errors such as the mistaken omission of redundant arguments from a parameter list because

they are "not needed" can be almost untraceable on many systems due to their "knock on" effect. This is compounded by the lack of information available to the user about the internal workings of the routine.

The option of handling both explicit and implicit ODE's through the same interface is clearly attainable simply by employing the most general interface needed to include both the problems. Then users with a problem which can be written simply in the form (2.1) must adopt instead the relatively complicated interface required for the system (3.1). Also, if this "solution" were adopted, for the D02ExF routines to link to the more general interface without change from their earlier specifications which were designed to solve (2.1) would require the user to supply fixed name routines defining the differential system. These considerations and the significant internal simplifications which can be made when solving the explicit problem (2.1) directly led us to supply two interfaces, one each for (2.1) and (3.1). Note that the resulting "explicit" code is simply a stripped down version of the "implicit" code and that one should obtain identical results to within the effects of roundoff by solving (2.1) using either of the two codes.

This design should allow developments in the future to permit, for example, the straightforward addition of other integrators, in particular a type insensitive code [15] and the differential/algebraic solver DASSL [16], and of other Jacobian structures, in particular unsymmetric almost block diagonal systems [18] and symmetric positive definite banded or profile systems. In this, the design has been successful. The code for some of these additions has proved to be simple to incorporate. However, our overall design cannot be pushed too far without further modification. For example, if second derivative methods [17] were used, we would require the derivative of f in (2.1) (that is the second derivative y'') and this is not requested in the current interface though it would be a fairly straightforward matter to include it. A more serious problem would arise with the inclusion of iterative or "matrix free" linear algebra. In the underlying design we have assumed that there will be a factorisation phase and a solution phase as is invariably the case with direct methods. However an iterative scheme which simulated these phases would be easy to incorporate.

3.3 Use in Packages

An increasingly common use of stiff ODE solvers is in the method of lines for solving PDE's (in particular for parabolic/elliptic equations such as arise in reaction kinetics and fluid dynamics problems). In general, a semi-discretization of such equations using finite differences or finite elements leads to a system of equations of the form (3.1) which will represent a differential/algebraic system when derived from a mixed system such as a parabolic/elliptic system. For use in a package for such problems, ODE software should have a number of properties:

(i) It should be able to operate in a one-step mode so that the user and the code can inspect and possibly adjust the solution at each time step, for example by remeshing in the space dimensions.

(ii) It should provide a template into which it is a simple matter to map the parameters of the package.

(iii) It should be *robust* in the following regards:

  (a) the possibility of internal breakdowns in the integrator should be minimised;

  (b) all exits and error exits should be through the interface, and there should be no side effects; and

  (c) as far as possible all input should be checkable and should be checked.

  The aim here is to ensure that all errors reported by the package are reported from package level; the user should be unaware of the properties of the underlying ODE software.

(iv) The ODE software should be portable so that the only hindrance to portability is the package itself. In this respect the fact of our software meeting NAG Library standards is a major advantage. The codes make full use of NAG's machine constants [19] and BLAS [20] (where appropriate). Machine specific coding is restricted to modules which are the responsibility of specific implementations of the

NAG Library. Considerable attempts have also been made to ensure portability across machine architectures. Hence, for example, there has been emphasis on using structures and portable directives to assist vectorisation.

(v) Perhaps the most important feature of ODE software for use in a package environment is that the problem definition (2.1) or (3.1) be supplied by reverse communication. In fact we provide two pieces of reverse communication software; one requires f in (2.1) and the other requires the expressions (3.2) and (3.3). Though individually each of the three applications described in Section 5 could be achieved using a conventional forward communication interface, the simple implementation of all three is a direct result of the provision of the reverse communication facility.

(vi) The reverse communication feature and the design decision to pass information only in one-dimensional array parameters imply that these routines could be used in mixed language packages at the "assembly code" level. That is, other language versions of the forward communication interfaces or of external packages could be written calling the Fortran reverse communication routines. For example, in Figure 4 below the whole code would be written in a language other than Fortran but would call the Fortran reverse communication routine D02NxF. Of course the availability of a suitable framework for mixed language programming is machine and operating system dependent.

4 Structure of the NAG Stiff ODE Solvers

As has been indicated in the previous two sections, the aim of our software is to provide new facilities combined with greater flexibility than in the past. The structure of the software is outlined in Figures 2a–c, which should be compared with Figure 1. The basic structure is the same but that there are two significant differences. First, the existence of the reverse communication codes has been exploited so that all the forward communication codes call them directly. This includes the easy-to-use routines D02ExF which, if the practice of Figure 1 were to be followed, would call

D02QDF, the replacement for D02QBF. Second, there is a proliferation of forward communication comprehensive routines D02NxF, essentially one for each linear algebra structure available for both (2.1) and (3.1). This design is forced on us by the restrictions of Fortran 77 since we must pass the Jacobian for the current structure of (2.1) or (3.1) and since we have determined to use no redundant parameters, visible or invisible to the user, except in those routines which are direct successors to those in Figure 1. It would have been possible to reduce to one the number of routines associated with each of (2.1) and (3.1) only if we had been willing to rely on linkers not checking the number of parameters and their types carefully. However, our emphasis on robustness precludes us from indulging in trickery of this type. In fact this profusion of routines is not an obstacle for the user as we shall see in subsection 4.1. Then, in subsection 4.2 we consider in detail the specification of subroutine D02EJF. Finally, in subsection 4.3 we discuss briefly the successors to the earlier NAG stiff solvers.

4.1 The D02N Subroutines

Each of the D02N forward communication routines is designed to be called in essentially the way represented by the pseudocode in Figure 3.

```
c
c     declarations
c
      EXTERNAL EQN, JAC, MONITR
      call linear algebra setup routine (sets RWORK)
      call integrator setup routine (sets RWORK)
c
c     set T,TOUT,Y,(YDOT for implicit problems,) RTOL,ATOL,
c     ITOL (and ITRACE to monitor course of integration if
c     required), IFAIL.
c
      CALL D02NxF(NEQ,NEQMAX,T,TOUT,Y,YDOT,RWORK,RTOL,ATOL,
     *     ITOL,INFORM,EQN,JAC,MONITR,ITRACE,IFAIL, and
     *     linear algebra parameters)
c
```

```
c     check IFAIL(the error indicator)
c     post processing:
c       optional linear algebra diagnostic call, sparse case
c       only (reads from INFORM)
c       optional integrator diagnostic call (reads from INFORM)
c     optionally reset TOUT and go back to the call of DO2NxF
c
      STOP
      END
      SUBROUTINE EQN(T,Y,YDOT,NEQ)
c
c     user supplied routine to define the system of
c     differential equations or to calculate a residual
c     in the case of implicit systems
c
      RETURN
      END
c
c     optional user supplied Jacobian forming routine (JAC)
c     optional user supplied  integration monitor  (MONITR)
c
```

Figure 3 A typical call to a forward communcation DO2N routine.

The first call to any integrator must be preceded by a call to *two* setup routines. One of these defines the integrator to be used and all the options appropriate to that choice of integrator; each option will take a default value if set accordingly. The other setup routine defines the structure of the Jacobian and options in the associated software. This option passing scheme is of particular importance in the case of sparse equation software where various combinations are permitted of numerical and analytical determination of the non-zero sparsity structure and of the corresponding numerical values of the non-zero elements. The integrator called must correspond to the linear algebra routine for its particular structure. An error is flagged if the two calls do not match. Also a check is included to test that both an integrator and a linear algebra setup routine have been called

prior to the call of the integrator itself.

The call of the integrator has a particularly simple form due to the extraction of all the options associated with the *choice* of integrator and the *choice* of linear algebra structure. The remaining arguments are concerned with the definition of (2.1) or (3.1), the error tolerances, a variable defining the task of the integrator (for example, one-step or interval mode of integration), variables to define error exits and the printing of trace information, a monitor routine described in detail in [22] and workspace used in part to pass information from the setup routines to the integrator. The user is not expected to set or access this workspace at any stage.

Essentially the calls to all the forward communication integrators are the same. There are however variations in the subroutines defining the ODE's (or residuals) and the Jacobian. It is hoped that this commonality of user interface will permit users to switch between explicit and implicit ODE definition and between linear algebra structures fairly painlessly. That is, the user can gain experience on simple structures and with simple definitions and then change to formats more appropriate to his problem simply by changing setup calls and redefining the subroutines for the differential equation and Jacobian. This feature will become more important as other structures such as almost block diagonal solvers are included; in this case the simpler alternative of using a banded solver might be tried first.

In addition to the forward communication routines we provide just *two* reverse communication routines, one for explicit and one for implicit ODE's. There is no proliferation of routines for the different types of linear algebra as the Jacobian evaluation is part of the reverse communication scheme; this design will be sufficiently general for future additions to the linear algebra facilities. The other major parts of the reverse communication scheme are the evaluation of the ODE (or residual) and the call of the monitor. Hence these routines have no subroutine or function arguments.

Precisely the same setup calls are required before a call to a reverse communication routine as to a forward communication routine, and the same continuation and diagnostic equivalents are available. The only difference is that any linear algebra setup call is permitted with each reverse communication routine. That is, the position with linear algebra structure choice is the same as that with integrator method choice in this case.

Inevitably a call to a reverse communication routine is rather complicated. To assist the user, an example is given in the NAG documentation which is intended as a template. A pseudo code version is reproduced in Figure 4. Unlike for the forward communication routines, we have not attempted to provide a clean interface. We have concentrated on efficiency and so in particular we do not copy arrays on each entry; for this reason we require the user to plant information in the correct area of workspace. We do not feel this is a major disadvantage as these reverse communication routines are intended mainly for package use. In fact, their success can be judged from their use as the building blocks for the forward communication routines, see Figure 2b, and by the experience reported in the next section.

```
c
c      declarations
c
       call linear algebra setup routine
       call integrator setup routine
       IREVCM=0
       CALL D02NxF(NEQ,NEQMAX,T,TOUT,Y,YDOT,RWORK,RTOL,ATOL,
      *  ITOL,INFORM,IREVCM,ITRACE,IFAIL)
       check IFAIL (the error indicator)
       IF (IREVCM > 0) THEN
          IF(1 < IREVCM < 5) THEN
             IF(IREVCM = 2) THEN
                supply the Jacobian
             ELSE IF (IREVCM = 3) THEN
                perform monitoring tasks
             ELSE IF (IREVCM = 4) THEN
                indicates an unsuccessful step (normally call
                   D02NxF again directly)
             END IF
          ELSE
             supply definition of the system of differential
             equations or of the residual
          ENDIF
          goto call of D02NxF
```

```
      ENDIF
c
c     post processing:
c         optional linear algebra diagnostic call (sparse case)
c         optional integrator diagnostic call
c     if not finished reset TOUT and goto call of D02NxF
c
      STOP
      END
```

Figure 4 Pseudocode for a call to a reverse communication integrator.

The integrators have several novel features and modifications from their predecessors such as LSODE including:

(i) On a call to the implicit ODE solvers, the initial values of the derivatives of the dependent variables may be unknown. On request, the integrator will attempt to calculate these values before starting the integration proper, see [22] for details.

(ii) In all the integrators a special starting step size algorithm similar to that described in [2] is used. This is designed to permit the integrator to get "on scale" immediately. The earlier NAG stiff solvers used a primitive version of this algorithm [1], but many other stiff solvers do not insist on starting "on scale".

(iii) When calling any of the integrators associated with the sparse setup routine the corresponding internal routine adopts a special strategy if the sparsity structure is to be determined numerically. It is often the case that at the initial point the solution has zero components. This causes the initial Jacobian sparsity structure to differ from that occurring later in the integration. Because of this and also because numerical determination of structure is less reliable close to zeros of components, we choose to mistrust the sparsity structure determined numerically at the initial point and on the *first* occasion we need to re-evaluate the Jacobian, we also re-evaluate its structure. This has proved a significant improvement when solving several problems over the usual strategy of evaluating the structure only at the initial point.

A utility to allow the user to force recomputation of the sparsity pattern at any time in the integration has also been provided.

(iv) The reverse communication routines can return just before a step is taken; this and a step size override facility are included to permit efficient treatment of problems with discontinuities. The pseudocode in [23] shows how this might be exploited.

(v) If, during the course of the integration, the user wishes to change the number of differential equations he may be able to avoid a restart and use at least some information already generated. Though a new factorisation of the Jacobian may be needed, the user may be able to start the new problem with the current order and stepsize. The facility to change the number of equations, and especially to reduce the number, has been available for some time elsewhere [10,22]. We are providing these features but in a novel way. There are two routines, one for subtracting a specified component and one for adding a component into a specified position in the list. All the internal reorganisation is handled invisibly and the user is asked to supply extra information, when adding a component, in a form which is immediately comprehensible. It is hoped that this facility will prove particularly useful when, for example, remeshing in solving parabolic partial differential equations by the method of lines.

(vi) Two interpolation routines are supplied. One, D02XJF, corresponding to D02XGF and D02XHF in the earlier design (see Figure 1), is essentially the code used internally to the integrator for prediction and error estimation. The other, D02XKF, is a C^1 interpolant which is recommended for general use in conjunction with the BDF method and, indeed, is used in internal calls from the D02ExF routines for root finding and intermediate output. Both interpolation routines are designed to return just the first $M(\leq N)$ components. Hence one can achieve significant efficiencies for large systems when only a few components are needed by the simple device of numbering the components of interest as the first.

(vii) An "error" was made in many earlier BDF codes, whereby before returning after a successful step, the order of the method was changed to that needed for the next step. This is corrected so that the interpolants now use the correct order of polynomial when evaluated on a step just taken.

4.2 Subroutine D02EJF

As mentioned in Section 3.1 an easy-to-use routine D02EJF has been supplied which combines all the facilities of the earlier D02ExF routines. It has been our aim in providing this new routine to simplify the calling sequence as far as possible. The resulting arguments are:

T REAL—initial point of integration (and final point or root on output);

TCRIT REAL—final point of integration;

N INTEGER—number of ODE's;

Y REAL array of length N—solution at initial point (and solution at TCRIT or at root on output);

FCN external SUBROUTINE—defines ODE's;

JAC external SUBROUTINE—defines Jacobian or is a NAG dummy;

TOL REAL—local error tolerance;

RELABS CHARACTER*1—determines whether error control with TOL is mixed, absolute or relative (with a threshold related to machine accuracy) which is also the default;

OUT external SUBROUTINE—used for intermediate output or is a NAG dummy (on each call it defines the next output point);

G external REAL FUNCTION—defines function whose first root is required or is a NAG dummy;

LW INTEGER—length of workspace;

W REAL array of length LW—workspace—not accessed directly by the user; and

IFAIL INTEGER—error indicator and controller of error output.

It is difficult to see how the number of arguments could be reduced further without reducing the facilities provided except, for example, by requiring the user to communicate through workspace. Whether or not the user supplies a Jacobian, whether intermediate output is required and whether it is a root finding problem are all specified without using indicators. Instead, the user is required to use a *named* NAG dummy in each case where he does *not* wish to supply the corresponding external subroutine. D02EJF determines for itself what is the situation by calling each of these routines at the initial point, as it must in any case if all the options are being used; on this call the NAG dummies set internal markers not specified for the user. The only danger with this approach is that the user may specify incorrectly the name of a NAG dummy. In this case he will either get an unsatisfied external on linkage or, in the unlikely event of specifying the wrong but an existing external routine, unpredictable results. Both problems could arise in the normal course of events so should not concern us overmuch; in any case a strict linker will usually trap the latter case.

4.3 The Successors of the Earlier NAG Stiff Solvers

We have to preserved the interface of the earlier NAG stiff solvers to provide a degree of continuity for users with production programs. Hence we have built the D02ExF routines on the reverse communication explicit solver D02NMF making appropriate setup calls internally (as in Figure 2c). As far as the user is concerned the interfaces and specifications are unchanged except that the same arguments as used previously are used to define an improved local error control. Also the workspace has been redefined and slightly extended — hence users must be warned.

Routine D02QDF has an almost identical interface to its predecessor D02QBF although some of the interrupts and their method of access are changed a little as is the workspace definition. One additional feature of D02QDF is to permit banded as well as full matrix linear algebra. This extension brings this code into line with LSODE [7] in almost all respects;

however even with this extension the NAG Library manual contains a strong recommendation that the D02NxF routines be used instead.

5 Package Use of the NAG Stiff Solvers

We report our experiences with the NAG stiff solvers when used:

(i) to solve time dependent partial differential equations in two space dimensions using a finite element space discretization;

(ii) to develop NAG Library software for the solution of systems of one space dimensional parabolic/elliptic PDE's; and

(iii) to solve the systems of differential/algebraic equations arising in the dynamic simulation of chemical engineering process plant.

First, we describe the use of the reverse communication routine for solving equations of type (3.1) arising from use of the NAG Finite Element Library (FEL) [24]. At present there are no time integration routines in the FEL but methods of solution of time-dependent problems are described via example programs. One principal method is direct integration, after a finite element mesh is used to discretize the solution in the spatial coordinates. For linear problems, depending on the type of PDE, this may yield an implicit system of second order ODE's

$$B\ddot{a} + C\dot{a} + Da + f = 0 \tag{5.1}$$

or an implicit system of first order ODE's

$$C\dot{a} + Da + f = 0, \tag{5.2}$$

where a is the vector of displacements of the nodes on the finite element mesh, f is some forcing term, and B, C and D are matrices assembled independently of the time variable. The derivatives with respect to time are then replaced by finite differences and thus integration is performed by some low order constant stepsize scheme, typically the Crank Nicolson method for first order systems or Newmark's method for second order systems. Often, a more efficient and a more accurate technique is to use a higher

order ODE integration method for solving implicit systems of type (5.2), such as the BDF method implemented in the NAG stiff solvers. Note that second order systems of type (5.1) can be reposed as first order systems of the following form

$$Q\dot{y} + Py + g = 0, \tag{5.3}$$

where

$$P = \begin{bmatrix} D & O \\ O & -I \end{bmatrix}, Q = \begin{bmatrix} C & B \\ I & O \end{bmatrix}, g = \begin{bmatrix} f \\ 0 \end{bmatrix}, y = \begin{bmatrix} a \\ \dot{a} \end{bmatrix}.$$

A forward communication routine could be used to solve the resulting first order systems, except for the fact there is no convenient way for the user to supply the matrices B, C and D to the residual forming routine. For the general case, the use of COMMON is not permissible as the size of B, C and D depends on the parameters of the problem. An alternative would have been to design the residual forming routine with a user specified workspace parameter, thus requiring the user to pack this information therein. We view this technique as cumbersome and error prone. The most convenient solution is to call the reverse communication implicit routine and indeed this approach has been adopted successfully by McCauley and Smith [25]. Since B, C and D are constant throughout the integration, the construction of problem specific driver routines for the reverse communication routine for commonly occurring PDE's is a relatively simple task.

The above example shows how the reverse communication interface can be used to provide finite element solutions in conjunction with FEL. We now consider how the reverse communication interface can be used to ease the construction of method of lines library software for general systems of PDE's in one space dimension. The general class of problems may be written as

$$C(x,t,u)u_t = R(x,t,u,u_x)_x + F(x,t,u,u_x), (x,t) \in (0,1) \times (0,\infty] \tag{5.4}$$

$$\beta_0(t)R(0,t,u,u_x) = g_0(t,u(0,t))$$
$$\beta_1(t)R(1,t,u,u_x) = g_1(t,u(1,t))$$

together with an appropriate initial condition. In calling software to solve such a problem the user defines the PDE by means of two variable name

subroutines, one describing the PDE functions C, R and F and the other the boundary conditions functions β_0, β_1, g_0 and g_1. In a Library context these should not be fixed name subroutines and so the names must be passed to the routine that performs the spatial discretization (that is defines the initial value problem in time). It is only possible to do this in Fortran 77 without changing the body of the ODE integrator if the integrator is written in the reverse communication style. The calling sequence to the PDE software will, in general, be more complicated than that of the ODE software since the user requires access to the full range of integrators and linear algebra routines. The PDE driver need not be concerned with performing the time integration; it is enough to interface to the reverse communication interface. Using this approach the first author has constructed new routines for solving parabolic PDE's without the code duplication that previously existed in the parabolic PDE (D03P) and the ODE (D02) chapters of the NAG Library when each used their own BDF codes. The open-ended nature of the ODE software interface has made it very easy to supply a specialised type insensitive θ-method integrator that is tuned for the solution of method of lines PDE problems.

As a final application, consider the construction of a package for the dynamic simulation of a large chemical engineering process plant. This example provides an illustration of another situation where reverse communication allows the problem to be formulated in a natural manner and allows a clean interface to be constructed between the application driver and the time integrator. Kuru and Westerberg [26] take as the general model of a chemical engineering process the differential/algebraic system

$$\frac{dx}{dt} = f(x, t, u, z) \tag{5.5}$$

$$0 = g(x, t, u, z) \tag{5.6}$$

with an appropriate initial condition. Here u is a given vector of r control variables, x is a vector of n state variables and z is a vector of m algebraic variables. In realistic simulations the total number of variables may be of the order of thousands but $n \leq m$ and the differential/algebraic system has a sparsity pattern that depends on the physical connections in the process plant. The functions f and g will also depend on a wide range of physical constants and individual models of subprocesses. One approach to

solving the system of differential algebraic equations is to use the algebraic equations to eliminate the algebraic variables, that is to solve (5.6) for

$$z = \hat{g}(x, u, t), \tag{5.7}$$

for instance by using specialised techniques based on the direction flows of liquids and gases. The problem can then be formulated as an ODE system in normal form

$$\frac{dx}{dt} = f(x, t, u, \hat{g}(x, u, t)) \tag{5.8}$$

Every evaluation of the function f thus requires the solution of the system of equations (5.6). The complex nature and the size of this system of equations together with the complexity of the network and its control structure make it unrealistic to define the function f by a subroutine. One of the early tests of the reverse communication interface involved joint work with chemical engineers of Leeds University on constructing experimental software for this type of dynamic simulation problem.

6 Conclusion

We have reported on major developments in the stiff system solvers in the NAG Library. We have provided a set of Fortran 77 routines designed to be both simple to use and flexible enough for the package writer to build upon. Our successful experiments in package building have validated the design used. Considerable effort has been expended in providing state-of-the-art integrators with an interface which is easy to use and difficult to fool — a very important point in designing Library software. As far as the continuity of the NAG Library is concerned, we have ensured that the earlier driver routines are preserved with very minor changes, we have provided a new all-embracing driver routine, and we have replaced the previous comprehensive routine by one with more facilities.

Acknowledgements

We wish to thank W. Seward for her careful criticisms of an earlier draft of this paper.

References

[1] Gladwell, I. (1979) Initial Value Routines in the NAG Library, *ACM Trans. Math. Soft.*, **5**, 386–400.

[2] Gladwell, I., Shampine, L.F. and Brankin, R.W. (1987) Automatic Selection of the Initial Stepsize for an ODE solver, *J. Comput. Appl. Math.*, **18**, 175–192.

[3] Hull, T.E., Enright, W.H. and Jackson, K.R. (1976) User's Guide for DVERK — a subroutine for Non-Stiff ODE's, Rept. 100, Dept. Comp. Sci., University of Toronto.

[4] Shampine, L.F. and Gordon, M.K. (1975) *Computer Solution of Ordinary Differential Equations: The Initial Value Problem*, W.H. Freeman, San Francisco.

[5] Shampine, L.F. and Watts, H.A. (1976) Practical Solution of Ordinary Differential Equations by Runge-Kutta Methods, Rept. SAND76–0585, Sandia National Laboratories, Albuquerque, NM.

[6] Shampine, L.F. and Watts, H.A. (1980) DEPAC — Design of a User Oriented Package of ODE Solvers, Rept. SAND79–2374, Sandia National Laboratories, Albuquerque, NM.

[7] Hindmarsh, A.C. (1980) LSODE and LSODI, Two New Initial Value Ordinary Differential Equation Solvers, *SIGNUM*, **15**, 10–11.

[8] Hindmarsh, A.C. (1982) ODEPACK — A Systematised Collection of ODE Solvers, in "*Advances in Computer Methods IV*", eds. R. Vichnevetzky and R.S. Stepleman, IMACS, New Brunswick, NJ.

[9] Hindmarsh, A.C. (1978) GEAR: Ordinary Differential Equation Solver, Rept. UCID–30001, Rev.3, Lawrence Livermore Laboratory, Livermore, CA.

[10] Berzins, M. and Furzeland, R.M. (1985) A User's Manual for Sprint — A Versatile Software Package for Solving Systems of Algebraic, Ordinary and Partial Differential Equations, Rept. TNER 85.058, Shell Research Ltd., Thornton.

[11] Petzold, L. (1982) Differential/Algebraic Equations are not ODE's, *SIAM J. Sci. and Stat. Comp.*, **3**, 367–384.

[12] Skeel, R.D. and Kong, A.D. (1977) Blended Linear Multistep Methods, *ACM Trans. Math. Soft.*, **3**, 326—345.

[13] Berzins, M. and Furzeland, R.M. (1986) A type insensitive method for the solution of stiff and non-stiff differential equations, Rept. 204, Dept. Comp. Stud., University of Leeds.

[14] Duff, I. (1977) MA28 — a set of Fortan subroutines for sparse unsymmetric linear equations, AERE Rept. R8730, HMSO, London.

[15] Petzold, L.R. (1983) Automatic Selection of Methods for Solving Stiff and Non-Stiff Systems of ODE's, *SIAM J. Sci. and Stat. Comp.*, **4**, 136–148.

[16] Petzold, L.R. (1982) A Description of DASSL: A Differential/Algebraic System Solver, in "*Transactions on Scientific Computation I*" ed. R.S. Stepleman, IMACS, New Brunswick, NJ.

[17] Addison, C.A. (1979) Implementing a Stiff Method Based on Second Derivative Formulas, Rept. 130, Dept. Comp. Sci. , University of Toronto.

[18] Brankin, R.W., Gladwell, I. and Hay, R.I. (1987) Codes for Almost Block Diagonal Systems and their Extensions, in preparation.

[19] X02 Chapter,(1987) *Mark 12 NAG Fortran Library Manual*, NAG Ltd., Oxford.

[20] F06 Chapter, (1987)*Mark 12 NAG Fortran Library Manual*, NAG Ltd., Oxford.

[21] Gladwell, I. (1985) NAG Library ODE Chapter and Short Term Plans for its Extension, *SIGNUM*, **20**, 35–40.

[22] Berzins, M. Dew, P.M. and Furzeland, R.M. (1986) Developing PDE Software Using the Method of Lines and Differential-Algebraic Integrators.(Highlighted talk presented at 1986 ODE conference, Albuquerque) Rept. 220, Dept. Comp. Stud., University of Leeds.

[23] Enright, W.H., Jackson, K.R., Norsett, S.P. and Thomsen, P.G. (1986) Effective Solution of Discontinuous IVP's using a Runge-Kutta Formula Pair with Interpolants, Num. Anal. Rept. 113, University of Manchester.

[24] *Mark 2 NAG Finite Element Library Manual*, (1985) NAG Ltd., Oxford.

[25] McCauley, A. and Smith, I.M. (1986). Private communication.

[26] Kuru, S. and Westerberg, A.W. (1985) A Newton Raphson Based Strategy for Exploiting Latency in Dynamic Simulation, *Computers and Chem. Eng.*, **9**, 175–182.

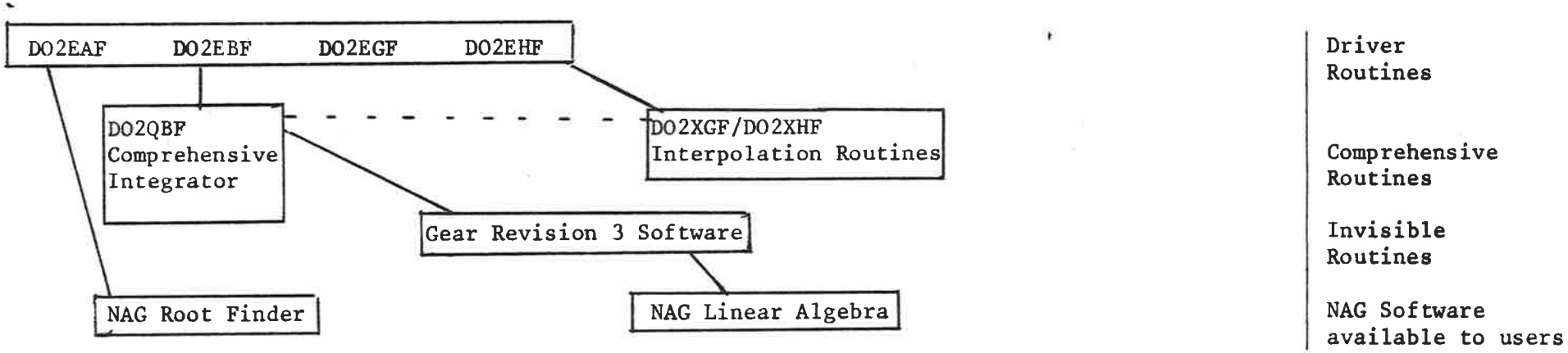

Figure 1 NAG Stiff Solvers before Mark 12 of the NAG Library

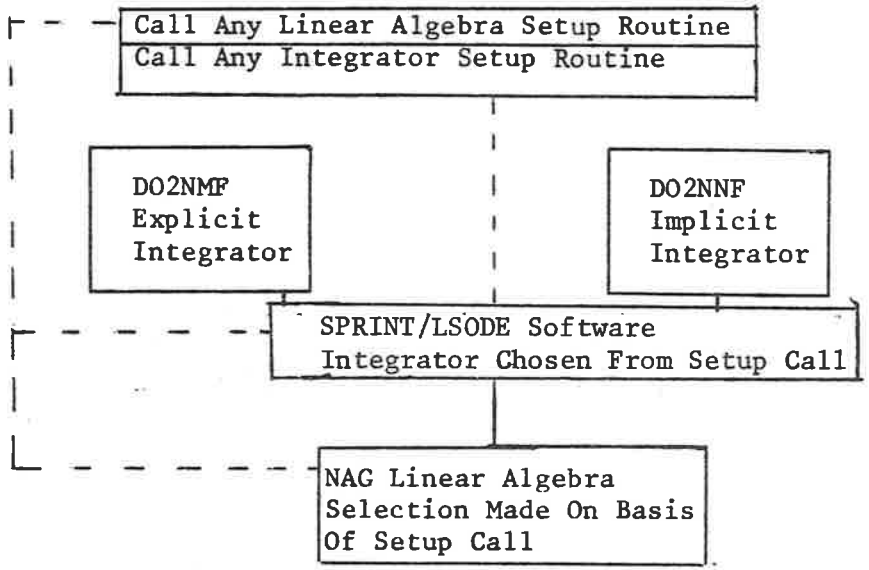

Figure 2a Reverse Communication DO2N Calls

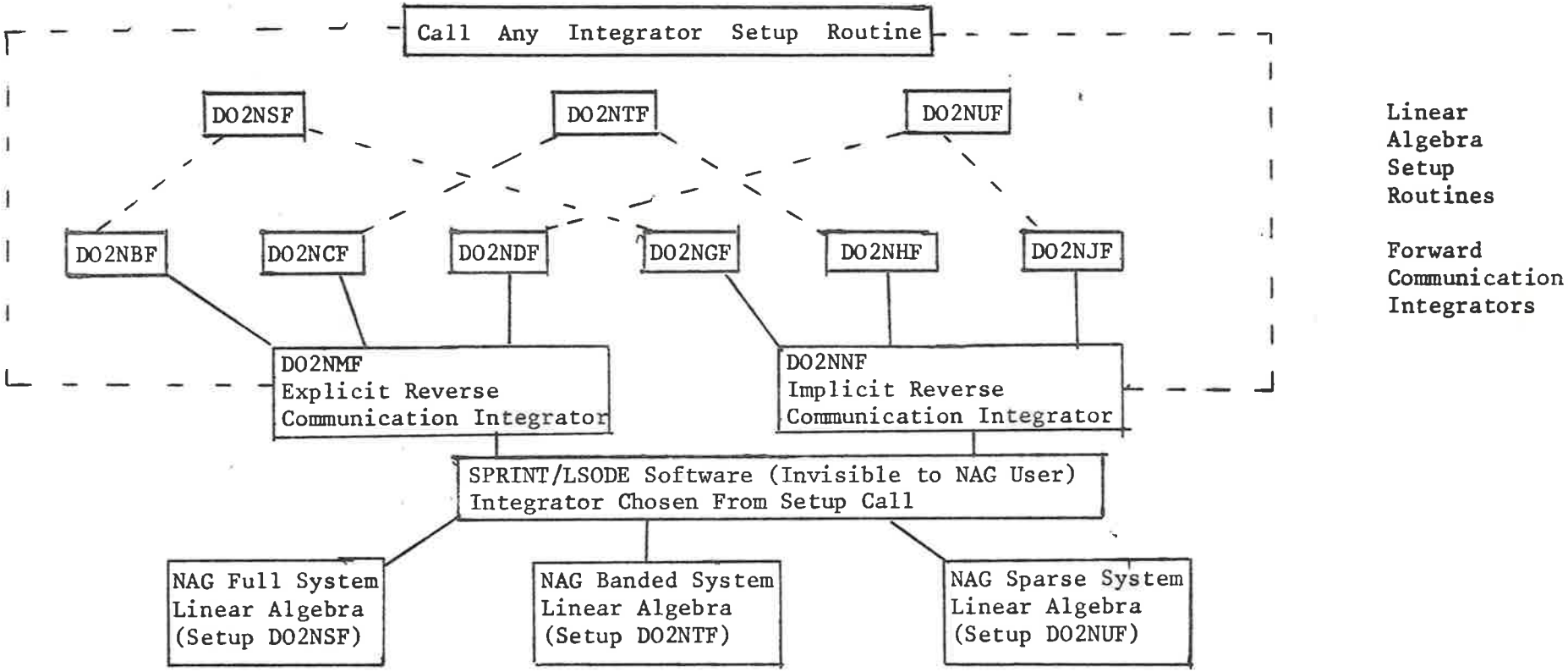

Figure 2b Forward Communication DO2N Routines

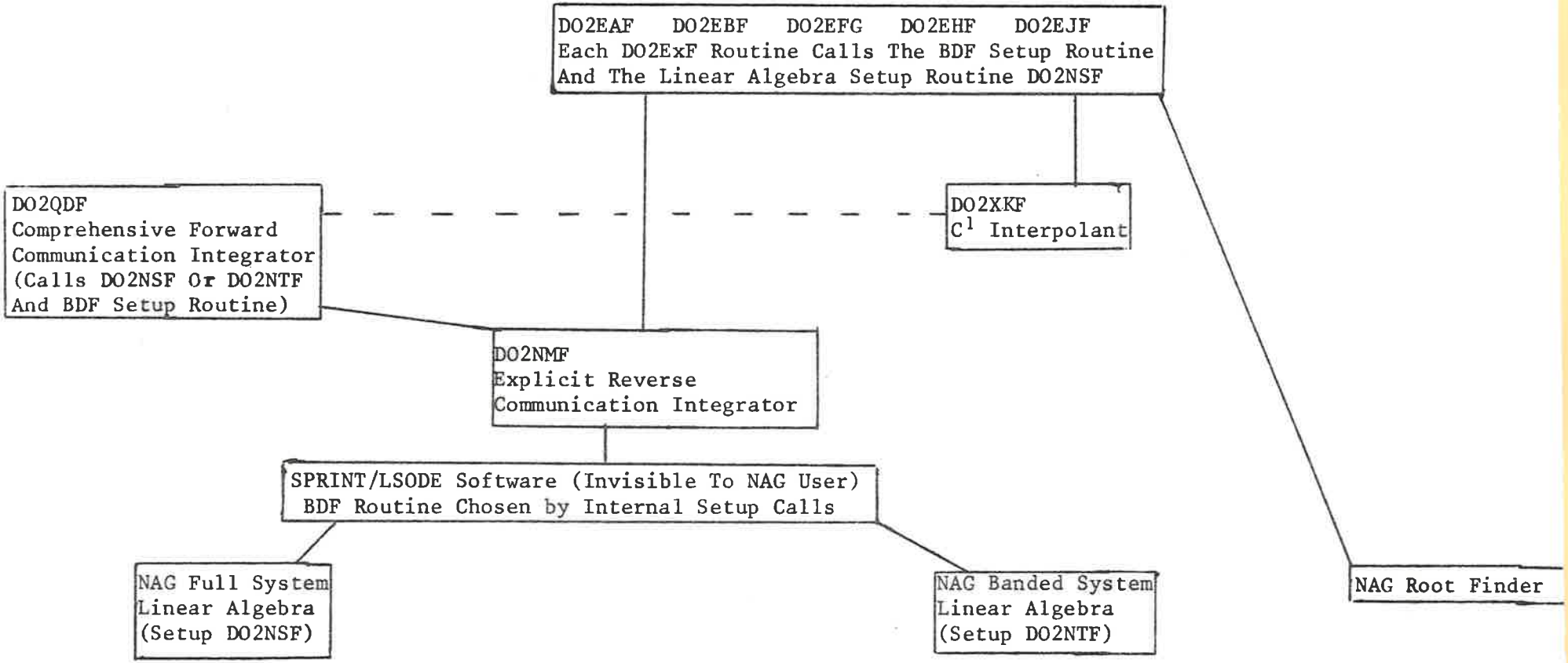

Figure 2c Driver and Comprehensive Stiff Integrators Calling DO2N