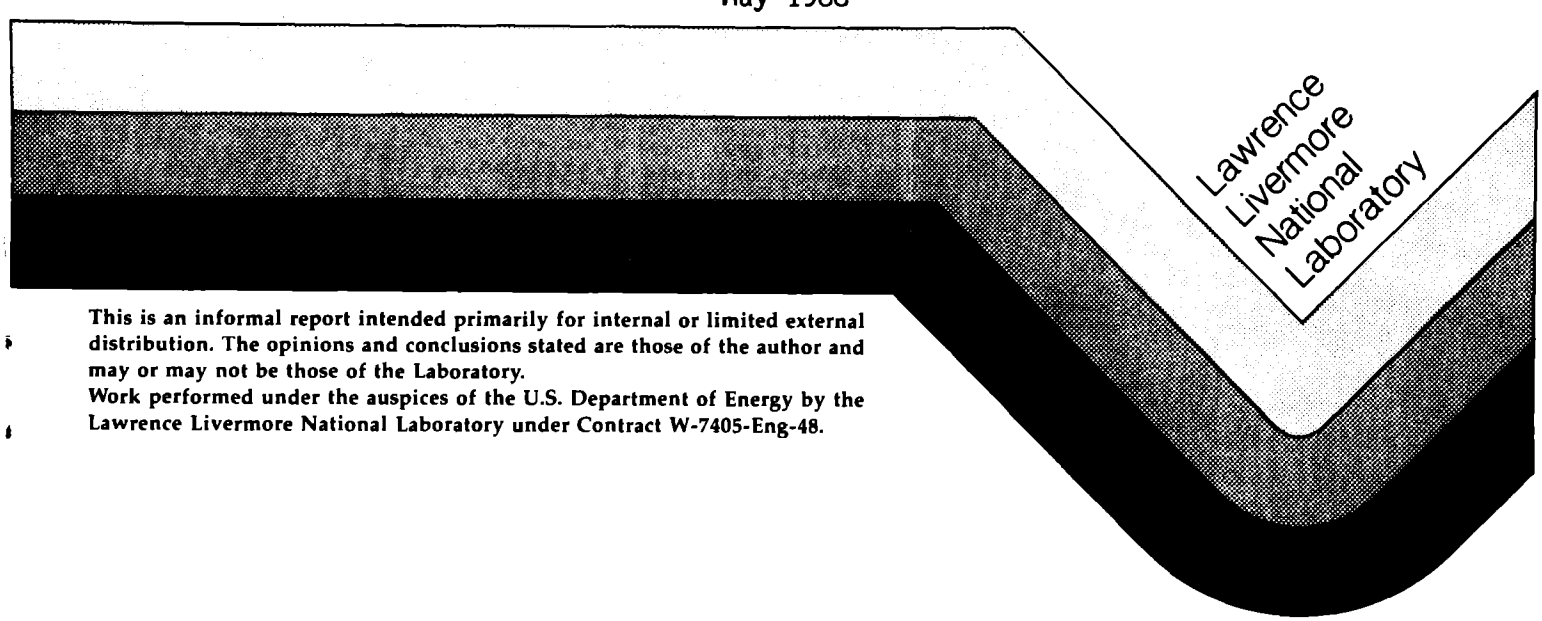
UCID- 21422

CIRCULATION COPY
SUBJECT TO RECALL
IN TWO WEEKS

KRYSI, An ODE Solver Combining a Semi-Implicit Runge-Kutta Method and a Preconditioned Krylov Method

Alan C. Hindmarsh
Syvert P. Nørsett

May 1988



This is an informal report intended primarily for internal or limited external distribution. The opinions and conclusions stated are those of the author and may or may not be those of the Laboratory.
Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

Printed in the United States of America
Available from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road
Springfield, VA 22161

Price Code	Page Range
A01	Microfiche
Papercopy Prices	
A02	001 - 050
A03	051 - 100
A04	101 - 200
A05	201 - 300
A06	301 - 400
A07	401 - 500
A08	501 - 600
A09	601

KRYSI, An ODE Solver Combining a Semi-Implicit Runge-Kutta Method and a Preconditioned Krylov Method*

Alan C. Hindmarsh
Computing & Mathematics Research Div.
Lawrence Livermore National Laboratory

Syvert Paul Nørsett
Div. of Mathematical Sciences
University of Trondheim
7034 Trondheim-NTH, Norway

ABSTRACT

This report describes KRYSI, a solver for ODE initial value problems that uses a combination of two powerful techniques in the case of stiff systems. KRYSI is based on the SIMPLE solver, and (like SIMPLE) uses a 3-stage 3rd order SDIRK method. When solving the implicit stage equations in the stiff case, both use some form of Newton iteration. But there, KRYSI (unlike SIMPLE) uses a Krylov subspace iteration method, called SPIGMR : Scaled Preconditioned Incomplete Generalized Minimum Residual method. The SPIGMR algorithm is outlined in the SDIRK context. No explicit Jacobian storage is required, except where used in preconditioning. The KRYSI package and its usage are described briefly. One test (demonstration) problem is given, along with a description of two preconditioners that are natural for its solution by KRYSI. The KRYSI solution results are given, and compared with those of LSODPK, a solver that combines a BDF method with the same SPIGMR algorithm.

1. INTRODUCTION.

The Fortran package SIMPLE [9] of Nørsett and Thomsen is a solver for stiff and nonstiff systems of ordinary differential equations (ODEs), written

$$dy/dt = f(t,y) \quad , \quad y \in R^N \quad , \tag{1}$$

with a given initial value vector $y(t_0) = y_0$. The method used is a 3-stage 3rd order SDIRK (singly diagonally implicit Runge-Kutta) method, documented in [8,9] and references listed there. At each stage of each time step, SIMPLE must solve a system of nonlinear algeberaic equations of the form

$$F(Y_i) \equiv Y_i - h\gamma f(t_n + c_i h, Y_i) - b_i = 0 \quad . \tag{2}$$

Here h is the current time step size, $h = t_{n+1} - t_n$, i is the stage index, γ is the diagonal RK coefficient a_{ii}, the c_i are the RK abscissae coefficients, and b_i is a linear combination of known values of f from previous stages.

The solution of the systems (2) is of central importance in achieving an accurate and efficient numerical solution of stiff systems (1). In the case of large N, this part of the total algorithm can easily dominate all the others in both computational cost and memory requirements. It is here that the solver KRYSI differs radically from

SIMPLE. The major difference is that, where SIMPLE uses a direct linear system solver within a modified Newton iteration, KRYSI uses a preconditioned Krylov subspace iteration within a (so-called) inexact Newton iteration. Various Krylov methods, which have recently had much attention in their own right as linear system algorithms, have the potential for greatly reducing the storage and computational costs in solving stiff systems. Earlier efforts in this area were carried out by Gear and Saad [7], Chan and Jackson [5], and Brown and Hindmarsh [3]. More recently, very encouraging results were obtained by Brown and Hindmarsh [4] with a solver called LSODPK (based on the general solver LSODE) that uses a BDF method in the stiff case.

2. METHOD DESCRIPTION.

The SIMPLE solver treats the system (2) with a combination of fixed-point and modified Newton iteration. Iterates Y^s approximating Y_i are generated from a prediction Y^0 that uses an appropriate extrapolation formula (also 3rd order). Fixed-point iterates are defined by

$$Y^{s+1} = F(Y^s) \quad . \tag{3}$$

while modified Newton iterates are defined by

$$(I-h\gamma J)(Y^{s+1}-Y^s) = -F(Y^s) \quad . \tag{4}$$

In (4), $J = \partial f/\partial y$ is the Jacobian, evaluated at some nearby point (t,y), and kept fixed over subsequent stages, and also over some subsequent time steps. The decision algorithm for switching between (3) and (4), and for updating J in (4), is summarized in [9]. Iteration is stopped with a convergence test. Of course, when the problem is truly stiff, this algorithm chooses the Newton method almost exclusively. In the Newton case, the linear systems (4) are solved directly, with a dense matrix LU factorization of $-J+(1/h\gamma)I$ followed by back-substitutions as needed.

The KRYSI solver retains, as far as possible, the strategy in SIMPLE for choosing between fixed-point (nonstiff) and Newton-like (stiff) iterations. But at each point in the algorithm where an algebraic system (2) must be solved and the system (1) is considered stiff, the approach taken is a combination of a Inexact Newton Method [6] and a preconditioned Krylov (linear) iteration [4,10]. An Inexact Newton

Method is one in which the Newton correction $\Delta Y^s = Y^{s+1} - Y^s$ does not solve (4) exactly, but instead satisfies

$$F'(Y^s)\,\Delta Y^s = -F(Y^s) + r^s \qquad (5)$$

with the residual vector r^s subject to some tolerance control. Note that the matrix in (5) is the exact Jacobian of the funtion F in (2), so that, aside from the residual r^s, (5) represents a true Newton iteration, rather than a modified Newton iteration.

The linear system to be solved (approximately) can be restated as

$$Ax = b \qquad (6)$$

with

$$\left.\begin{aligned} A &= I - h\gamma J(t_n + c_i h, Y^s) \quad , \\ b &= -F(Y^s) \quad , \quad \text{and} \\ x &= \Delta Y^s = Y^{s+1} - Y^s \quad . \end{aligned}\right\} \qquad (7)$$

An iterative method applied to (6) produces iterates x_ℓ from some initial guess x_0, and includes a convergence test to stop the iteration.

The particular iterative method used in KRYSI is called SPIGMR : Scaled Preconditioned Incomplete Generalized Minimum Residual [4]. It has evolved from a basic Generalized Minimum Residual (GMRES) method due to Saad and Schultz [11]. It is one of several so-called Krylov subspace iteration methods, which are characterized by the feature that each iteration only requires A in operator form, i.e. it only requires values of the matrix-vector product Av. A Krylov method

finds successive approximations x_ℓ by adding to x_0 a correction vector chosen from an ℓ-dimensional subspace (the Krylov subspace). GMRES makes a particular choice that minimizes the residual in norm. The "incomplete" (or truncated) version is a generalization in which each new basis vector for the Krylov supspace is made orthogonal to only a specified number of previous basis vectors, rather than all of them, and allows for reduced expense when A is nearly symmetric. Scaling was added to account for the weights (closely related to error tolerances) that are involved in all norms of error-like vectors throughout the ODE solver. Finally, provision was made for user-supplied preconditioning by matrices on both the left and right of the matrix A, to enhance the robustness of the method. More specifically, a diagonal scaling matrix D is defined such that the weighted root-mean-square norm

$$\|v\|_{WRMS} = \|D^{-1}v\|_2 \tag{8}$$

is the approriate choice consistent with the ODE error control, and preconditioners P_1 and P_2 are to be supplied such that each P_j is relatively easy to invert, while P_1P_2 approximates A in some sense. Then the SPIGMR algorithm amounts to the application of the incomplete GMRES algorithm to the equivalent system

$$\tilde{A}\,\tilde{x} = \tilde{b} \qquad \text{with}$$

$$\tilde{A} = D^{-1}P_1^{-1}AP_2^{-1}D, \quad \tilde{x} = D^{-1}P_2x, \quad \tilde{b} = D^{-1}P_1^{-1}b \tag{9}$$

instead of to (6).

Perhaps the most important aspect of the entire algorithm is the manner in which the product Av is obtained within each iteration. For any given vector v, we have by (7)

$$Av = v - h\gamma\, J(t,Y^s)v \quad . \tag{10}$$

A major goal of the Krylov method is to avoid having to construct J explicitly for this (or any) purpose. We do this by replacing the product Jv in (10) by a difference quotient approximation

$$J(t,Y)v \approx [f(t,Y+\sigma v)-f(t,Y)]/\sigma \tag{11}$$

with σ a suitably small parameter. In fact, since the diagonal elements of D in (8) are proportional to tolerances, and thus a vector whose WRMS norm is 1 is considered a "small" vector, we use $\sigma = 1/\|v\|_{WRMS}$ in (11).

The SPIGMR algorithm is given in more specific terms below. The maximum number of iterations is ℓ_{max}. An incompleteness parameter p is given, with the complete (non-truncated) case given by $p = \ell_{max}$. The stopping criterion is $\rho_\ell = \|b-Ax_\ell\|_{WRMS} < \delta$, or $\|r^s\|_{WRMS} < \delta$ in (5). The test constant δ is $\delta_1\, \epsilon_1$, where $\epsilon_1 = .5$ is the stopping test constant used in the nonlinear iterations (correction less than ϵ_1 in weighted norm), and $\delta_1 < 1$ is a heuristic constant. The norm ρ_ℓ is computed indirectly from other data generated in the algorithm; the approximate solution vector x_ℓ itself is not computed until the last iteration (on convergence). The initial guess is taken to be $x_0 = 0$, although the algorithm is given below for general x_0. The implemented algorithm actually allows the case $\ell_{max} = 0$, and in that case it returns either $x_0 = 0$, if $\|b\|_{WRMS} < \delta$, or $x = P_2^{-1}P_1^{-1}b$ otherwise.

Other algorithmic details are given in [4], and a convergence theory for it in the inexact Newton setting is given in [2].

SPIGMR Algorithm:

1. (a) $r_0 = b - Ax_0$: stop if $\|r_0\|_{WRMS} < \delta$.

   (b) $\tilde{r}_0 = D^{-1}P_1^{-1}r_0$, compute $\|\tilde{r}_0\|_2 = \|P_1^{-1}r_0\|_{WRMS}$,

   $\delta' = \delta\|P_1^{-1}r_0\|_{WRMS}/\|r_0\|_{WRMS}$, $\tilde{v}_1 = \tilde{r}_0/\|\tilde{r}_0\|_2$.

2. For $\ell = 1,2,\ldots,\ell_{max}$, do:

   (a) Compute $\tilde{A}\,\tilde{v}_\ell = D^{-1}P_1^{-1}A\,P_2^{-1}D\tilde{v}_\ell$.

   (b) $\tilde{w}_{\ell+1} = \tilde{A}\,\tilde{v}_\ell - \sum_{i=i_0}^{\ell} \tilde{h}_{i\ell}\tilde{v}_i$, where $i_0 = \max(1, \ell-p+1)$,
   $\tilde{h}_{i\ell} = (\tilde{A}\,\tilde{v}_\ell, \tilde{v}_i)$.

   (c) $\tilde{h}_{\ell+1,\ell} = \|\tilde{w}_{\ell+1}\|_2$, $\tilde{v}_{\ell+1} = \tilde{w}_{\ell+1}/\tilde{h}_{\ell+1,\ell}$.

   (d) Update QR factorization of $\tilde{H}_\ell = (\tilde{h}_{ij})$ (an $(\ell+1) \times \ell$ matrix).

   (e) Compute residual ρ_ℓ indirectly.

   (f) If $\rho_\ell < \delta'$, go to Step 3; otherwise go to (a).

3. Compute $\|\tilde{r}_0\|_2 Q_\ell^T e_1 = (\bar{g}_\ell, g)^T$. $\tilde{z} = \tilde{V}_\ell \bar{R}_\ell^{-1} \bar{g}_\ell$. $x_\ell = x_0 + P_2^{-1} D\tilde{z}$.

The inclusion of the SPIGMR algorithm into the SDIRK algorithm implemented in SIMPLE is rather straightforward. The modified Newton algorithm in SIMPLE becomes an inexact Newton algorithm. The logic in SIMPLE associated with re-evaluating J is used in KRYSI to decide whether to call on the user to re-evaluate parts of J (if any) that are involved in the preconditioner matrices P_1 and P_2. In the event that SPIGMR fails to converge (which has no analog in SIMPLE), the algorithm in KRYSI either re-evaluates the P_j and tries the time step again (if the P_j involve Jacobian data and are out of date), or else reduces h to h/4 and repeats the time step. The factor 1/4 is purely heuristic.

There are two minor algorithmic differences between the KRYSI and LSODPK versions of SPIGMR. One is that on the first Newton iteration in each RK stage, at least one linear iteration is performed (an immediate return in Step 1(a) is disallowed), in order to avoid a failure of the stiffness estimation procedure in KRYSI (which uses values of a norm of $x = \Delta Y^S$). The other is that in the LSODPK version of SPIGMR, if $\ell = \ell_{max}$ iterations have been done and the stopping test $\rho_\ell < \delta'$ has still not passed, SPIGMR is nevertheless considered to have converged if either (a) $\rho_\ell \leq 1$, or (b) this is the first Newton iteration and $\rho_\ell \leq \|r_0\|_{WRMS}$. In the KRYSI version, only condition (a) is allowed.

3. CODE DESCRIPTION.

Since the SIMPLE code is fully described elsewhere [9], what follows is a description of the differences between the KRYSI solver and SIMPLE.

The transformation of SIMPLE to KRYSI was greatly facilitated by a) the modular structure of SIMPLE, and b) the existence of the solver LSODPK, which implements various preconditioned Krylov methods in combination with the BDF method for ODE systems. Of the Krylov methods in LSODPK, the one most highly recommended in [4] for general situations is SPIGMR. Furthermore, the modular structure of LSODPK made it possible to move the SPIGMR algorithm into the KRYSI code with very little effort.

The changes made to SIMPLE were largely confined to three areas, which are summarized below.

(1) The name of the driver subroutine SIMPLE was changed to KRYSI, with the call sequence augmented by several items related to the SPIGMR algorithm, and corresponding minor changes were made to the subroutine body. One change was first made in SIMPLE that was not related to the algebraic system method, but rather to the tolerances: The absolute tolerance parameter AEPS was changed from a scalar to an array, since this is a commonly needed feature. At the same time the relative tolerance REPS is limited to value $\geq$ 100*(unit roundoff), with the machine unit roundoff supplied by a separate routine (R1MACH in single precision, D1MACH in double precision). Call sequence items added to KRYSI include names of routines to preset and solve the preconditioner matrices, work spaces (real and integer) associated

with the SPIGMR algorithm and (separately) with preconditioning, a flag to indicate the presence/absence of optional inputs, and a flag to indicate the type of preconditioning (none, left or right only, or both sides). The SIMPLE arguments DF, IDF, AJAC, ALU, and PWORK were removed. The Common block /STATS/ containing run statistics was altered to include quantities related to SPIGMR. The optional inputs are the constants ℓ_{max} (= MAXL), p (= KMP), and δ_1 (= DELT) discussed in the previous section. Subroutine KRYSI includes added coding to compute and save the error weights REPS|Y(i)|+AEPS(i), and to handle an unrecoverable error in either the evaluation of a preconditioner or in the SPIGMR iteration.

(2) In place of the routines DFDEC, DF, DFN, DECOMP and SGEFA/DGEFA in the SIMPLE sover, for the calculation of the Jacobian J and factorization of $-J+(1/h_7)I$, KRYSI has a subroutine PKSET devoted to the preprocessing associated with the preconditioners, if any. PKSET calls the user-supplied routine JAC to compute any part of the Jacobian that either preconditioner might use, say J_{part}, and also to do a factorization of the corresponding matrices $I-h_7J_{part}$ if a direct method is to be used on these preconditioners. A sophisticated user may even save the matrix J_{part} in separate storage, and if circumstances warrant it, redo only the factorization when h_7 has changed since the last call to JAC. One differnce between the JAC routine in KRYSI and that in LSODPK is that the former must compute the current f(t,y) if needed (e.g. for difference quotients), while this vector is available for use by JAC in LSODPK.

(3) In place of the routines SOLVE and SGESL/DGESL in the SIMPLE solver, KRYSI has subroutines SOLPK, SPIGMR, ATV, ORTHOG, SHEQR/DHEQR, SHELS/DHELS (the first letter of the name depending on the precision,

in the last two), and some of the BLAS routines, which, as a group, carry out the SPIGMR algorithm. These routines were taken from the LSODPK solver, and used with only relatively minor changes. Considerable shortening of the interface routine SOLPK was done (it calls one of four iterative linear solvers in LSODPK, but only SPIGMR in KRYSI). Some changes were made in the call sequences of SOLPK, SPIGMR, and ATV, mainly just changes in variable names. The programming of all norm calculations was changed, because KRYSI keeps the error weights themselves, while LSODPK keeps their reciprocals (trading divide operations for multiply operations). The SPIGMR routine also calls the user-supplied routine PSOL to compute the solutions u of preconditioner systems $P_j u = v$ for given vectors v. Also, ATV calls the user's right-hand side routine F in accordance with (11).

A complete block diagram of the KRYSI package, excluding BLAS routines, is shown in fig. 1. The user's calling program and user-supplied subroutines all appear on the left (with F repeated for simplicity).

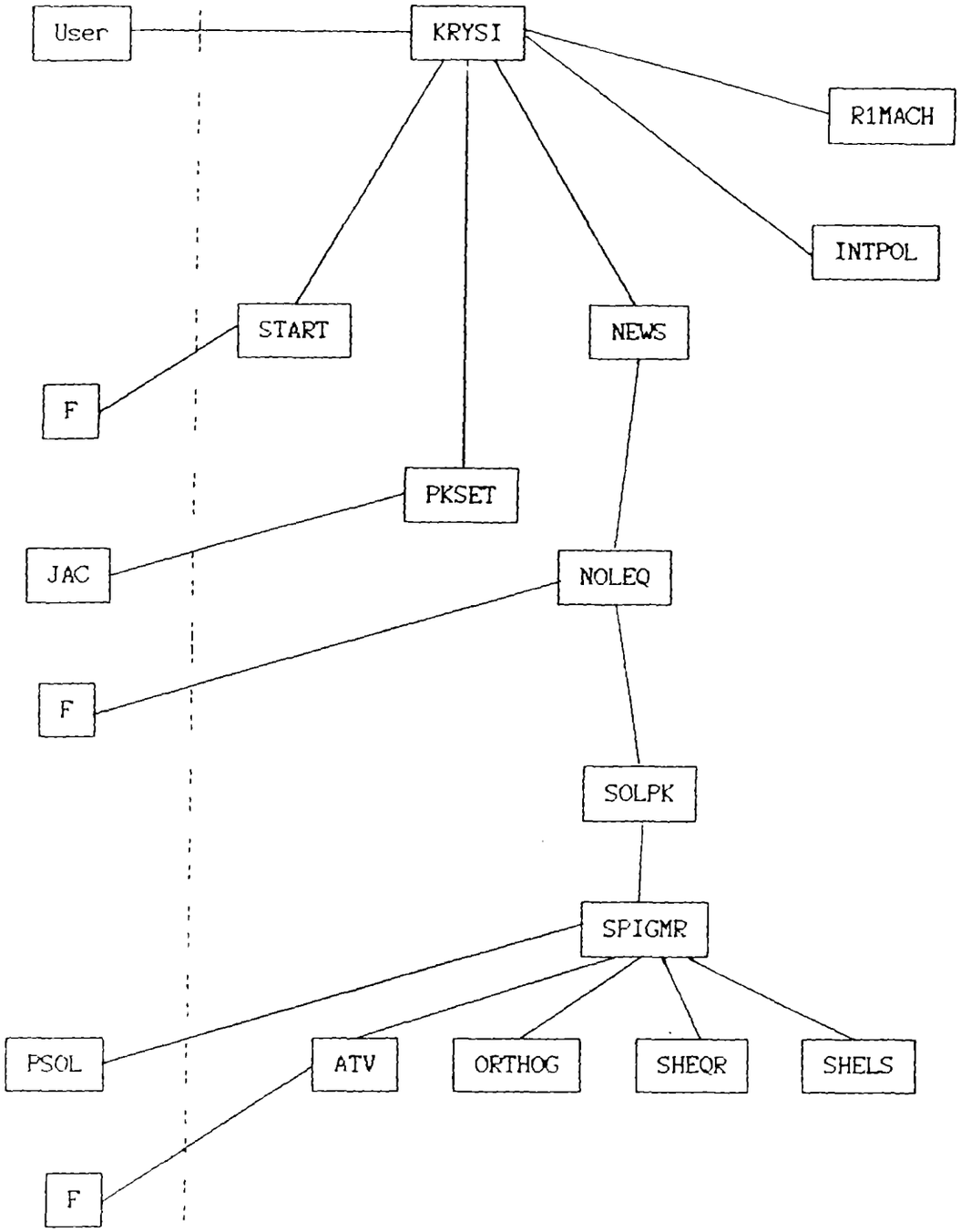


Figure 1. Block diagram of KRYSI.

The primary goal of the replacement of a traditional method by a Krylov method is the reduction in storage requirements for large systems. So consider the storage needs of the SIMPLE and KRYSI solvers. Both use a work array of size 14 N, plus the (length N) Y and AEPS arrays, for ODE algorithm matters that are not directly related to the algebraic system problem. Beyond that, SIMPLE requires $2N^2+N$ words of storage for the Jacobian, the factored Newton matrix, and pivot information. In place of this, KRYSI requires $(\ell_{max}+2)N+\ell_{max}^2+4\ell_{max}+1$ words (all real) for the GMRES algorithm, plus real and integer arrays APRE and IPRE for use in preconditioning, with contents and lengths that are completely up to the user. Thus a direct comparison is difficult, but we can consider some typical values from tests described in [4]. The default value of ℓ_{max} is 5, and this is usually sufficient when preconditioning is used. Thus the typical GMRES work space is 7N+46 words. Test problems from various reaction-transport systems in 1, 2, and 3 space dimensions have been solved with BDF/Krylov method combinations [3,4], and the sum of the lengths of the real and integer preconditioner work arrays is typically some reasonable multiple of N, say $k_{pre}N$. Values of k_{pre} in the tests range from 0 to 21. Thus the typical ratio of total storage requirements for SIMPLE to KRYSI is

$$(2N^2+17N)/(16N+7N+46+k_{pre}N) \approx (2N+17)/(23+k_{pre})$$

Thus for (say) $k_{pre} \sim 20$ and N large, a reduction in storage by a factor of about N/20 is achieved. For any $N \geq 100$, this represents a rather dramatic reduction.

4. USAGE OF KRYSI.

The user instructions for the KRYSI solver are given in the initial blocks of comment lines, which are reproduced below. These instructions are the same as for SIMPLE except for arguments related to the preconditioned Krylov iteration method. Since the latter is invoked only in the stiff case, a user with a nonstiff problem will see no advantage to KRYSI over SIMPLE.

```
      SUBROUTINE KRYSI(N,F,JAC,PSOL,Y,T,TEND,AEPS,REPS,APRE,IPRE,
     &                 WORK,WK,IWK,IOPT,JPRE,IFLAG,OUTPUT,IPT)
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C     KRYSI SOLVER FOR INTEGRATING Y'=F(T,Y) , Y(T)=GIVEN AS Y
C     OVER THE INTERVAL (T,TEND) USING
C     A SEMI-IMPLICIT RK-METHOD OF NORSETT AND THOMSEN,
C     IN COMBINATION WITH A PRECONDITIONED KRYLOV SUBSPACE ITERATION
C     METHOD (SCALED INCOMPLETE GENERALIZED MINIMUM RESIDUAL METHOD).
C     SEE COMMENTS IN SUBROUTINES NEWS AND SPIGMR.
C
C     VERSION OF 18 SEPTEMBER 1987.
C
C     THIS VERSION IS IN SINGLE PRECISION.
C
C     INPUT PARAMETERS ARE:
C               N    : INTEGER GIVING THE NUMBER OF EQUATIONS
C
C               F    : SUBROUTINE FOR RIGHTHAND SIDE.
C                      SUBROUTINE F(T,Y,R)
C                      REAL T,Y(N),R(N)
C                      R=F(T,Y)
C
C               JAC  : SUBROUTINE FOR THE COMPUTING JACOBIAN ELEMENTS
C                      NEEDED FOR PRECONDITIONING.
C                      SUBROUTINE JAC(F,N,T,Y,EWT,FTY,FTEM,GAH,
C                                     APRE,IPRE,JFLAG)
C                      REAL Y(N),EWT(N),FTY(N),FTEM(N),
C                                      APRE(1),IPRE(1)
C                      THIS ROUTINE MUST EVALUATE AND PREPROCESS ANY
C                      PARTS OF THE JACOBIAN MATRIX DF/DY USED IN THE
C                      LEFT AND RIGHT PRECONDITIONER MATRICES, P1 AND P2.
C                      FTY IS SPACE FOR THE CURRENT VALUE OF F(T,Y),
C                      AND SHOULD BE LOADED BY A CALL TO F IF NEEDED.
C                      FTEM IS WORK SPACE, E.G. FOR VALUES OF F(T,Y+E)
C                      FOR USE IN DIFFERENCE QUOTIENTS APPROXIMATIONS.
C                      ON COMPUTING JACOBIAN ELEMENTS, JAC MUST MULTIPLY
C                      ALL COMPUTED ELEMENTS BY -GAH AND ADD THE
C                      IDENTITY MATRIX, THEN DO ANY FACTORING
C                      OPERATIONS NEEDED FOR LATER SOLUTION OF LINEAR
C                      SYSTEMS.
```

```
C                  JAC MAY SAVE JACOBIAN ELEMENTS FOR REUSE, WITH
C                  ONLY A CORRECTION OF THE GAH FACTOR, AND A
C                  REFACTORING OF THE MATRIX.  IF THIS STRATEGY IS
C                  USED, JAC SHOULD REEVALUATE JACOBIAN ELEMENTS
C                  (AND ASSEMBLE AND FACTOR THE MATRIX) WHEN
C                  JFLAG = -1 ON INPUT, AND OTHERWISE ONLY REASSEMBLE
C                  USING THE SAVED ELEMENTS AND REFACTOR.
C                  THE MATRIX P1*P2 SHOULD APPROXIMATE
C                    IDENTITY - GAH*(DF/DY).
C                  ON RETURN JAC SHOULD SET JFLAG AS FOLLOWS..
C                  JFLAG = 1  IF SUCCESSFUL, WITH NO EVALUATION OF
C                             RELEVANT JACOBIAN DATA
C                  JFLAG = 2  IF SUCCESSFUL, WITH EVALUATION OF
C                             RELEVANT JACOBIAN DATA
C                  JFLAG = -3 IF NOT SUCCESSFUL (THE STEP WILL
C                             REDUCED AND RETRIED).
C
C            PSOL : SUBROUTINE FOR SOLVING LINEAR SYSTEMS WITH A
C                   PRECONDITIONER (P1 OR P2) AS COEFFICIENT MATRIX.
C                   SUBROUTINE PSOL(N,T,Y,SAVF,TEMP,GAH,APRE,IPRE,
C                                   B,LR,IER)
C                   REAL Y(N),SAVF(N),TEMP(N),B(N),
C                                   APRE(1),IPRE(1)
C                   THIS ROUTINE MUST SOLVE A LINEAR SYSTEM WITH B
C                   AS RIGHT-HAND SIDE AND ONE OF THE PRECONDITIONERS
C                   P1 OR P2 AS COEFFICIENT MATRIX, AND RETURN THE
C                   SOLUTION IN B.  LR IS A LEFT-RIGHT FLAG (INPUT).
C                   PSOL IS TO USE P1 IF LR = 1 AND P2 IF LR = 2.
C                   PSOL CAN USE DATA GENERATED IN THE JAC ROUTINE
C                   AND SAVED IN APRE AND IPRE.  THE ARGUMENT GAH IS
C                   THE CURRENT VALUE OF THE SCALAR APPEARING IN
C                   THE LINEAR SYSTEM.  IF THE OLD VALUE, AT THE
C                   TIME OF THE LAST JAC CALL, IS NEEDED, IT MUST
C                   HAVE BEEN SAVED BY JAC IN APRE.
C                   TEMP IS A WORK ARRAY OF LENGTH N.
C                   ON RETURN, PSOL SHOULD SET IER AS FOLLOWS:
C                      IER = 0 IF PSOL WAS SUCCESSFUL,
C                      IER .GT. 0 IF A RECOVERABLE ERROR OCCURRED,
C                              MEANING THAT THE STEP WILL BE RETRIED
C                              WITH THE SAME STEP SIZE BUT WITH A
C                              CALL TO JAC FIRST TO UPDATE P1 AND P2,
C                              OR THEIR FACTORIZATIONS,
C                      IER .LT. 0 IF AN UNRECOVERABLE ERROR OCCURRED
C                              (TIME STEP SIZE WILL BE REDUCED).
C
C            IOPT : FLAG FOR PRESENCE OF OPTIONAL INPUTS.
C                   IOPT = 0 MEANS DEFAULT VALUES ARE USED.
C                   IOPT = 1 MEANS INPUTS ARE READ FROM...
C                   WK(1)  = DELT = KRYLOV CONVERGENCE TEST CONSTANT
C                            DEFAULT IS DELT = .05.
C                   IWK(1) = MAXL = MAXIMUM NUMBER OF VECTORS SAVED IN
C                            KRYLOV ITERATION.  DEFAULT IS MAXL = 5.
C                   IWK(2) = KMP = NUMBER OF VECTORS ON WHICH
C                            ORTHOGONALIZATION IS DONE IN KRYLOV
C                            ITERATION.  DEFAULT IS KMP = MAXL.
C                            IF JACOBIAN IS SYMMETRIC, USE KMP=2.
C
C            JPRE : PRECONDITIONER TYPE FLAG.
C                   JPRE = 0 FOR NO PRECONDITIONING.
C                   JPRE = 1 FOR LEFT-ONLY PRECONDITIONING.
C                   JPRE = 2 FOR RIGHT-ONLY PRECONDITIONING.
C                   JPRE = 3 FOR PRECONDITIONING ON BOTH SIDES.
C
C            Y(N) : ARRAY CONTAINING THE STARTING VECTOR
C                   IN RETURNING Y=Y(TEND)
C
C            T    : STARTING POINT OF INTEGRATION
C
C            TEND : ENDPOINT OF INTEGRATION
```

```
C
C          AEPS,REPS  : ERROR TOLERANCE CONTROL PARAMETERS.
C                       THE ERROR IS CONTROLLED IN THE FOLLOWING WAY:
C                       ABS(LOCAL ERROR(I)) < AEPS(I) + REPS * ABS(Y(I))
C
C      THE WORKING AREA IS GIVEN AS :
C               WORK  : REAL ARRAY OF LENGTH 14*N
C          APRE,IPRE  : REAL AND INTEGER WORK SPACES FOR PRECONDITIONING.
C                       THE LENGTHS AND STRUCTURE OF APRE AND IPRE ARE
C                       UNDER USER CONTROL IN THE JAC AND PSOL ROUTINES.
C             WK,IWK  : REAL AND INTEGER WORK ARRAYS FOR KRYLOV
C                       ITERATION METHOD AND OPTIONAL INPUTS.
C                       WK LENGTH = 7*N + 46 FOR DEFAULT INPUTS.
C                       FOR GENERAL VALUES OF THE OPTIONAL INPUTS
C                       MAXL AND KMP, THE LENGTH OF WK IS
C                            (MAXL+2)*N + MAXL*(MAXL+4) + 1
C                       IWK LENGTH = 2.
C
C      CONTROL VARIABLE:
C                IFLAG: INTEGER FOR OUTPUT INFORMATION.
C                       =1 INDICATES SUCCESSFULL COMPLETION OF STEP
C                          WITH  OLD JACBIAN
C                       =2 AS FOR =1 BUT WITH NEW JACOBIAN IN LAST STEP
C
C      OUTPUT           : IPT = 1 IF OUTPUT ROUTINE IS SUPPLIED ELSE 0
C                       SUBROUTINE OUTPUT IS A SUBROUTINE GIVEN BY USER
C                       CALL OUTPUT(X,Y) WHERE :
C                            X : NEXT OUTPUT POINT . T < X < TEND WITH
C                            SOLUTIONS AT T AND TEND ALWAYS GIVEN.
C                            Y(X) IS HANDED OVER TO USER WHO IS ASKED
C                            TO GIVE THE NEXT OUTPUTPOINT
C
C IN ADDITION TO THE CALL SEQUENCE, INFORMATION IS ALSO SUPPLIED
C IN THE COMMON BLOCK
C     COMMON /STATS/ NS,NFX,NF,NNI,NNS,NLI,NPE,NPF,NPS,NCFL
C THE STATISTICS IN /STATS/ HAVE THE FOLLOWING MEANING..
C   NS    = NUMBER OF TIME STEPS
C   NFX   = NUMBER OF STEPS WITH FIXPOINT ITERATION
C   NF    = NUMBER OF F EVALUATIONS
C   NNI   = NUMBER OF NONLINEAR ITERATIONS (FIXPOINT OR NEWTON)
C   NNS   = NUMBER OF NEWTON ITERATIONS
C   NLI   = NUMBER OF LINEAR ITERATIONS (BY KRYLOV METHOD)
C   NPE   = NUMBER OF PRECONDITIONER EVALUATIONS
C   NPF   = NUMBER OF PRECONDITIONER FACTORIZATIONS
C   NPS   = NUMBER OF PRECONDITIONER SOLVES, I.E. PSOL CALLS
C   NCFL  = NUMBER OF CONVERGENCE FAILURES OF THE LINEAR ITERATIVE SOLVER
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

5. EXAMPLE PROBLEM.

The KRYSI solver has been subjected to very little testing so far, especially by comparison with LSODPK. For the earliest development and debugging phases, a small problem (3-species Robertson kinetics problem) was used, and is successfully solved by KRYSI. But its small size makes it meaningless as a test of the capabilities of KRYSI. A larger problem, described below, has been incorporated into a demonstration program for KRYSI. It is also used as the demonstration problem for LSODPK, and is one of the test problems described in [4] (namely Test problem 2) with slight differences in parameter values. Various reduced versions of this problem were also used for debugging, but are of no interest otherwise.

The demonstration problem is a semi-discrete form of a system of reaction-diffusion PDEs arising from multi-species food web models. It is a predator-prey model in which the various animal species experience interaction and simple diffusion. The general form of these models, for s species in two dimensions, is

$$\partial c^i/\partial t = f_i(x,y,t,c) + d_i(c^i_{xx} + c^i_{yy}) \quad (i = 1,2,\ldots,s),$$

with

$$f_i(x,y,t,c) = c^i\left(b_i + \sum_{j=1}^{s} a_{ij} c^j\right) \quad .$$

The interaction and diffusion coefficients (a_{ij}, b_i, d_i) could be functions of (x,y,t) in general. The choices made for this test problem are for a simple model of p prey and p predator species $(s = 2p)$, arranged in that order in the concentration vector c. We

take the various coefficients to be as follows :

$$a_{ii} = -1 \quad (\text{all } i)$$

$$a_{ij} = -\frac{1}{2}\cdot 10^{-6} \quad (i \le p, \quad j > p)$$

$$a_{ij} = 10^{3} \quad (i > p, \quad j \le p)$$

(all other $a_{ij} = 0$),

$$b_i = (1+\alpha xy) \quad (i \le p)$$

$$b_i = -(1+\alpha xy) \quad (i > p)$$

$$d_i = 1 \quad (i \le p)$$

$$d_i = .05 \quad (i > p)$$

The domain is the unit square $0 \le x,y \le 1$, and $0 \le t \le 10$. The boundary conditions are of Neumann type (zero normal derivatives) everywhere. The coefficients are such that a unique stable equilibrium is guaranteed to exist in the constant coefficient case $\alpha = 0$ [1], namely $c = -A^{-1}b$, and empirically the same appears to be true for $\alpha > 0$. In this problem we take $\alpha = 1$.

The initial conditions used for this problem are taken to be simple peaked functions that satisfy the boundary conditions, given by the polynomial function

$$c^i(x,y) = 10+i[16x(1-x)y(1-y)]^2 \quad (1 \le i \le s),$$

which varies between 10 and 10+i.

This PDE system (plus boundary conditions) is discretized in space by way of central differences on a uniform M×M mesh, resulting in an ODE system of size $N = 2p\,M^2$. The ODE for species i at the point (x_j, y_k) is

$$\dot{c}^i_{jk} = f_i(x_j, y_k, t, c_{jk}) + d_i\left(\frac{c^i_{j+1,k} - 2c^i_{jk} + c^i_{j-1,k}}{\Delta x^2} + \frac{c^i_{j,k+1} - 2c^i_{jk} + c^i_{j,k-1}}{\Delta y^2}\right)$$

$$(i = 1,..,s; \quad j \text{ and } k = 1,\ldots,M)$$

where $\Delta x = \Delta y = 1/(M-1)$, and $c_{jk} = \{c^i_{jk} : i = 1,\ldots,s\}$.

The boundary conditions are simulated by taking

$$c^i_{0k} = c^i_{2k} \quad , \quad c^i_{M+1,k} = c^i_{M-1,k} \qquad (\text{all } k)$$

$$c^i_{j,0} = c^i_{j,2} \quad , \quad c^i_{j,M+1} = c^i_{j,M-1} \qquad (\text{all } j)$$

for all i. The ODE system $\dot{y} = f$ is organized with a vector y ordered by i, then j, then k: $y = (c_{11}, c_{21}, \ldots, c_{M,1}, \ldots, c_{MM})$.

The ODE system is stiff, and an estimate of the spectrum is easily obtained from the interaction terms f_i, for which the dominant eigenvalues are about $-10^4 p(1+\alpha xy)$ for the components at a mesh point (x,y). However, the diffusion terms cause the profiles to flatten out at steady state, so that the equilibrium values of any species c^i are

spread by a factor of only about 1.08 rather than $1+\alpha = 2$ (though the spread factor exceeds 2 during the transient). The discrete diffusion terms contribute significantly to the stiffness also.

The particular sizes chosen here are $p = 4$ ($s = 8$) and $M = 6$ ($N = 288$). However, the program is written for general values of these parameters, and requires minimal changes if different values are substituted.

Two preconditioners are used with this problem. One is based on the diffusion terms only and is applied as a left preconditioner P_1. The other is based on the interaction terms f_i only, and gives a right preconditioner P_2. Both are based on rather general ideas that apply to reaction-transport problems of various kinds, and are described fully in [4]. But they will be summarized here for the sake of completeness. The combining of two preconditioners in this way resembles an operator-splitting approach, but is more robust than traditional operator splitting because of the various convergence tests and error controls.

The diffusion-based preconditioner arises by ignoring completely the interaction contributions to the Jacobian J in the system $(I-h\gamma J)u = v$. The resulting approximate system consists of s decoupled $M \times M$ systems, each of which corresponds to the semi-discrete form of a simple diffusion equation

$$\partial c/\partial t = d(c_{xx}+c_{yy}) \quad .$$

These linear systems are amenable to various classical iterative methods. We have chosen to apply five Gauss-Seidel iterations to the

M×M system. If the coefficient matrix is written as D-L-U (D diagonal, L lower triangular, U upper triangular), then the initial guess u^0 has the form $(D-L)^{-1}v$, and the iteration is given by

$$u^{m+1} = G\,u^m + (D-L)^{-1}v \qquad (m = 0,1,\ldots)$$
$$G = (D-L)^{-1}U$$

The interaction-based preconditioner completely ignores the diffusion terms, and so gives a block-diagonal matrix

$$P_2 = I - h\gamma\ \mathrm{diag}(R_{11}, R_{21}, \ldots, R_{MM})$$

in which R_{jk} approximates the s×s interaction Jacobian $\partial f_l/\partial c$ at the spatial point (x_j, y_k). Considerable further economy (in storage and computation) is gained if, instead of forming and using all M^2 of these s×s matrices, we partition the grid into groups of mesh points and take only one point in each group at which to evaluate R_{jk}, using this matrix in P_2 for all of the points in the group. We have chosen a simple 2×2 Cartesian partition of the 6×6 mesh, so that R_{jk} is evaluated at each of only 4 points, each of those representing a group of 9 points. Each R_{jk} evaluated is approximated by difference quotients, and the matrices $I - h\gamma R_{jk}$ are LU-factored for later solution of the linear systems $P_2 u = v$.

The demonstration program integrates the above ODE system with KRYSI using tolerances REPS = AEPS(i) = 10^{-4}, and prints output at $t = 10^{-8}, 10^{-7}, \ldots, 10^{-1}, 1, 2, \ldots, 10$. Default values were used for the SPIGMR parameters. The results reported below are from runs on a CRAY-1 at LLNL. The accuracy of all solution values was quite acceptable and consistent with the tolerances used. The various

counters of interest are :

NS	= 223	=	no. time steps
NFX	= 145	=	no. fixed-point (nonstiff) steps
NF	= 2237	=	no. f evaluations (excluding calls in START)
NNI	= 1677	=	no. nonlinear iterations (fixed-point or Newton)
NNS	= 457	=	no. Newton iterations (no. linear systems solved)
NLI	= 560	=	no. linear (Krylov) iterations
NPE	= 20	=	no. preconditioner evaluations
NPS	= 1658	=	no. preconditioner solves
RT	= 8.48	=	runtime in CPU sec. (excluding output)

It should be noted that the number of factorizations of the block-diagonal preconditioner matrices was also NPE (no saving and re-use of the R_{jk} was done). There were no convergence failures of the SPIGMR algorithm. The fixed point steps were all taken on an initial interval, roughly $0 \leq t \leq 10^{-3}$, and 1205 f evaluations were done on these steps.

Some observations are easy to make. The nonstiff part of the problem (145 steps) used an average of 8.3 f evaluations per step, or 2.8 per RK-stage. The stiff part (78 steps) used an average of 5.9 Newton iterations per step, or 1.95 per stage, and 1.23 linear iterations per Newton iteration. The f counts from Newton iterations (one each) and linear iterations (one each) add up to 1017, which differs from the observed value 2237 - 1205 = 1032 for the stiff interval. The discrepancy (learned from an extra diagnostic run) is due to five

steps in which fixed point iteration was attempted again, but rejected immediately, using an additional 15 f values in the process. The SPIGMR algorithm is having no difficulty with the linear systems, although the average iteration count (= average Krylov subspace dimension) of 1.23 is somewhat deceptive. The average over the last 24 steps, covering $1 \leq t \leq 10$, is 1.90.

The storage requirement for KRYSI on this problem was 6960 words (real and integer), or about 24.2 N. This can be compared with the storage required for SIMPLE to solve the same problem, namely (2N+17)N = 593N = 170,784, higher by a factor of about 24.5.

For comparison, runs of the LSODPK demonstration program were made, solving the same problem with the same SPIGMR input parameters and on the same machine. When run with the same tolerances (all 10^{-4}), the statistics were as follows:

```
NS   =   163   (all steps considered stiff)
NF   =   433
NNI  =   192   (all Newton iterations)
NLI  =   240
NPE  =    29
NPS  =   806
RT   =  2.06
```

However, a direct comparison is not fair if the accuracy achieved is different for the two solvers. Thus a high accuracy solution (with 10^{-7} tolerances) was generated, and a calculation of the maximum relative error was added to both demonstration programs. (Relative error is the appropriate choice, since all components stay well above

the absolute tolerance.) As a result, the KRYSI run with 10^{-4} tolerances was found to have a maximum relative error of $1.3 \cdot 10^{-4}$, while that of the corresponding LSODPK run was $15 \cdot 10^{-4}$, over 11 times larger.

To perform a fair comparison, further runs were made with LSODPK with tighter tolerances until the maximum error reached was (roughly) the same value as for KRYSI. This happened at a tolerance of $5 \cdot 10^{-6}$, giving a maximum relative error of $1.2 \cdot 10^{-4}$. The statistics for this run were as follows :

NS	=	257
NF	=	685
NNI	=	303
NLI	=	381
NPE	=	34
NPS	=	1282
RT	=	3.23

If we regard the time interval as divided into stiff and nonstiff parts at $t = 10^{-3}$, then LSODPK spent 124 steps and 294 f evaluations (2.4 per step) in the nonstiff interval, and 133 steps and 391 f evaluations (2.9 per step) in the stiff interval. The overall average number of linear iterations per Newton iteration was 1.26, and in $1 \leq t \leq 10$ (45 steps) it was 2.11.

The comparison of the last run with the KRYSI run (both achieving nearly the same accuracy) can be summarized as follows : LSODPK takes considerably more steps than KRYSI in the stiff interval (133 vs 78), and slightly fewer steps in the nonstiff interval, but a much smaller number of nonlinear iterations per step in both intervals. The run time ratio, 2.63, roughly reflects those higher costs per step in KRYSI. Since the SPIGMR algorithm by itself is performing very similarly in the two runs, the cost difference can be attributed entirely to the relative performance of the SDIRK and BDF methods on this problem. On a problem in which BDF would have difficulty for reasons of stability, the relationship between the two solvers might be just the reverse. The total storage requirement of LSODPK on this problem was about 20.3 N, slightly less than that of KRYSI.

6. CONCLUSIONS.

The KRYSI package has been developed as an experimental solver for stiff systems. It combines an SDIRK algorithm as the ODE integration method with SPIGMR (Scaled Preconditioned Incomplete Generalized Minimum Residual method) as the iterative solver for the linear systems that arise within the Newton method solution of each RK stage. The combinatioon appears to work, although so far it has been tested on only one problem of realistic proportions.

The usage of KRYSI requires preconditioner matrices from the user, and to this extent it relies on the ability of the user to identify the dominant contributions to the stiffness of the problem, and construct efficient preconditioners accordingly.

On the other hand, KRYSI includes many features of "black box" solvers, including local error control in the integration steps, and convergence control in the nonlinear and linear iterations. Thus for a large and difficult stiff system, KRYSI combines the best properties of two contrasting approaches - a powerful easy-to-use general (but inefficient) solver, and a completely problem-specific and efficient (but hard to set up) ad hoc solution algorithm.

KRYSI resembles the LSODPK solver, which (in the stiff case) combines a BDF integration algorithm with SPIGMR and other KRYLOV iterative linear system algorithms. In the one comparison test done with both solvers, LSODPK was more efficient, but for reasons that relate to the merits of SDIRK vs BDF on the particular problem, and not to the use of an iterative linear system solver.

Clearly, much more testing is needed, and suitable tuning within KRYSI is likely to enhance its performance. The SDIRK algorithm in KRYSI could benefit from some enhancements that are used in BDF solvers (e.g. anticipated convergence of Newton iterations). The SPIGMR algorithm in KRYSI could probably benefit from tuning ideas related to the context of a multi-stage SDIRK method (e.g. reuse in later stages of Krylov subspace basis vectors generated in the first stage).

7. ACKOWLEDGEMENTS.

The bulk of the work described here was done during a visit by the first author to NTH, Trondheim, Norway, in September 1986. The support of the Department of Mathematical Sciences at NTH and of STATOIL (VISTA Program) is gratefully acknowledged.

The support of Lawrence Livermore National Laboratory, during and after that visit is also gratefully ackowledged.

REFERENCES.

[1] P.N. Brown, Decay to Uniform States in Food Webs, SIAM J. Appl. Math., 46 (1986), pp. 376-392.

[2] P.N. Brown, A Local Convergence Theory for Combined Inexact-Newton/Finite-Difference Projection Methods, SIAM J. Num. Anal., 24 (1987), pp. 407-434.

[3] P.N. Brown and A.C. Hindmarsh, Matrix-Free Methods for Stiff Systems of ODEs, SIAM J. Num. Anal., 23 (1986), pp. 610-638.

[4] P.N. Brown and A.C. Hindmarsh, Reduced Storage Matrix Methods in Stiff ODE Systems, LLNL Report UCRL-95088, Rev. 1, June 1987 ; to appear in J. Appl. Math. & Comp.

[5] T.F. Chan and K.R. Jackson, The Use of Iterative Linear Equation Solvers in Codes for Large Systems of Stiff IVPs for ODEs, SIAM J. Sci. Stat. Comp., 7 (1986), pp. 378-417.

[6] R.S. Dembo, S.C. Eisenstat, and T. Steihaug, Inexact Newton Methods, SIAM J. Num. Anal., 19 (1982), pp. 400-408.

[7] C.W. Gear and Y. Saad, Iterative Solution of Linear Equations in ODE Codes, SIAM J. Sci. Stat. Comp., 4 (1983), pp. 583-601.

[8] S.P. Nørsett and P.G. Thomsen, Imbedded SDIRK-Methods of Basic Order Three, BIT 24 (1984), pp. 634-646.

[9] S.P. Nørsett and P.G. Thomsen, SIMPLE, a Stiff System Solver. To appear in NTH Technical Report Series - Mathematics of Computation.

[10] Y. Saad, Krylov Subspace Methods for Solving Large Unsymmetric Linear Systems, Math. Comp., 37 (1981), pp. 1105-1126.

[11] Y. Saad and M.H. Schultz, GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems, SIAM J. Sci. Stat. Comp., 7 (1986), pp. 856-869.