SLICOT Working Note 2002-4

Definition and Implementation of a SLICOT Interface and a MATLAB Gateway for the Solution of Nonlinear Equations Systems [1]

Fernando Alvarruiz[2], Vicente Hernández[2]

March 2002

[1]This document presents research results of the European Community BRITE-EURAM III Thematic Networks Programme NICONET (contract number BRRT-CT97-5040) and is distributed by the Working Group on Software WGS. *WGS secretariat*: Mrs. Ida Tassens, ESAT - Katholieke Universiteit Leuven, Kasteelpark Arenberg 10, 3001-Leuven-Heverlee, BELGIUM. This report is available by anonymous ftp from `wgs.esat.kuleuven.ac.be/pub/WGS/REPORTS/SLWN2002-4.ps.Z`
[2]Departamento de Sistemas Informáticos y Computación (DSIC). Universidad Politécnica de Valencia (UPV). Valencia. Spain. *fbermejo@dsic.upv.es*, *vhernand@dsic.upv.es*.

Contents

1 Introduction

The solution of nonlinear equations systems is necessary in order to cope with nonlinear control problems (Task V in NICONET). Following an approach similar to the work done in NICONET for other problems in task V, such as the solution of ordinary differential equations (ODE) and differential algebraic equations (DAE) systems, SLICOT and MATLAB interfaces have been implemented on top of existing software packages for the solution of nonlinear equations systems. For the moment, only one such library, known as KINSOL, has been considered.

This paper presents SLICOT and MATLAB interfaces for the KINSOL package. The SLICOT interface enables the user to call the KINSOL package by means of a subroutine with a SLICOT-compliant calling sequence ([3]). By means of the MATLAB interface the user can call the package from MATLAB, defining the problem by means of MATLAB functions.

The interfaces could be extended in the future in order to consider other nonlinear equations systems solvers, although some restructuring of the interfaces would be necessary.

2 The KINSOL Package

The KINSOL package ([1]) has been developed at the Center for Applied Scientific Computing, Lawrence Livermore National Laboratory and can be freely downloaded from the web (http://www.llnl.gov/CASC/PVODE). Strongly connected with KINSOL are the packages PVODE and IDA, developed by the same Center. The three packages are complementary, forming a software tool for the solution of ODE and DAE systems.

KINSOL (Krylov Inexact Newton SOLver) is a general purpose solver for nonlinear systems of equations. Its most notable feature is that it uses Krylov Inexact Newton techniques in the system's approximate solution. KINSOL is written in ANSI C, although it contains interface routines so that it can be called also from Fortran programs.

KINSOL is available both in serial and parallel forms. In fact, the package is arranged so that selecting one of two forms of a single module in the compilation process allows the entire package to be created in either serial or parallel form. This document is only concerned with the serial form of KINSOL.

The nonlinear system of equations

$$F(u) = 0,$$

where $F(u)$ is a nonlinear function from R^n to R^n, is solved by this package. The Newton method used results in the solution of linear systems of the form

$$J(u)x = b,$$

where $J(u)$ is the Jacobian of F at u. The solution of these systems by a Krylov method requires products of the form $J(u)v$, which are approximated by a difference quotient of the form

$$J(u)v \approx \frac{F(u + \sigma v) - F(u)}{\sigma}.$$

Thus, the Jacobian need not be formed explicitly.

A Fortran program making use of the KINSOL package must follow the following sequence of steps:

1. Allocate space for KINSOL, by calling the function `FSKINMALLOC`.

2. Set up the linear solver. This is done by calling a routine with the name `FKINSPGMR<A><B>`, where `<A>` is a digit that can be 0, 1 or 2, and `<B>` is a digit with value 0 or 1. Thus, there are 6 different routines. Which one to choose depends on the user-defined routines that are provided, as we will see later.

3. Solve the nonlinear system $F(u) = 0$, by calling the function `FKINSOL`.

4. Free the memory, by calling the function `FKINFREE`.

In addition to this, there are some user-defined routines that have to be taken into account. One of them must be necessarily defined by the user, while the others are optional. These routines must have the following names and calling sequences:

- SUBROUTINE KFUN (NEQ, UU, FVAL)
  (Required). This function evaluates the nonlinear function F.
- SUBROUTINE KPSOL (NEQ, UU, USCALE, FVAL, FSCALE, VTEM, FTEM, UROUND, NFE, IER)
  (Optional). This is used to solve the linear system $Px = b$, where P is a preconditioner matrix. If this function is not provided, then no preconditioning is done.
- SUBROUTINE KPRECO (NEQ, UU, USCALE, FVAL, FSCALE, VTEMP1, VTEMP2, UROUND, NFE, IER)
  (Optional). This is used to evaluate and preprocess any Jacobian-related data needed by KPSOL.
- SUBROUTINE FATIMES(V, Z, NEWU, UU, IER)
  (Optional). This funcion performs the matrix-vector multiply $J(u)v$, where $J(u)$ is an approximate Jacobian matrix for that iteration.

The subroutine FKINSPGRM<A><B> to be used in step 2 above depends on which optional user-defined routines are provided. <A>$= 0$ means that neither KPRECO nor KPSOL routines are provided; <A>$= 1$ means that KPSOL is provided, but KPRECO is not; <A>$= 2$ indicates that both KPRECO and KPSOL are provided. <B>$= 0$ indicates that the routine FATIMES is not provided, while <B>$= 1$ indicates that FATIMES is provided.

3 SLICOT Interface

The SLICOT interface contains a single user-callable routine that performs the sequence of 4 steps described above. The calling sequence of the routine presents the following form:

```
SUBROUTINE KINSOL( GSTRAT, LINSU, NEQ, OPTIN, MAXL, MAXLRST,
                   MSBPRE, UU, USCALE, FSCALE, CONSTR,
                   IOPT, ROPT, TOL1, TOL2, INFO)
```

NEQ corresponds to the number of equations (and unknowns) in the sistem. MAXL, MAXLRST and MSBPRE are arguments related to the Krylov method for the linear system. UU contains on input the initial point, and on exit the solution of the system. IOPT and ROPT are integer and real arrays, respectively, used to provide additional input/output options. The argument LINSU corresponds to the name of the FKINSPGMR<A><B> routine to be used to set up the linear solver (step 2). Arguments indicating tolerances (TOL1 and TOL2) and the error indicator INFO, are named according to the SLICOT standard ([3]). Note that no work arrays are used, since KINSOL allocates and deallocates its own memory space. For a more detailed description of the arguments, see Appendix A of this Working Note.

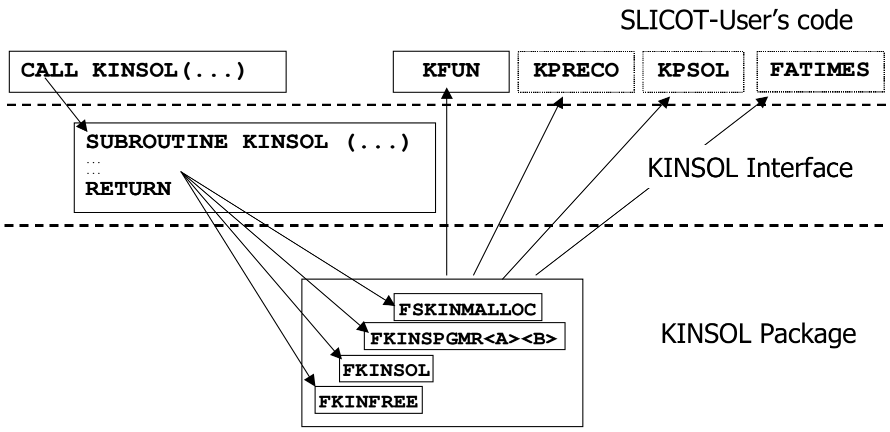

Figure 1: Application using the SLICOT interface.

The user-defined subroutines (KFUN, KPRECO, KPSOL and FATIMES) are not altered by the SLICOT interface. Figure 1 shows a diagram where we can see the different layers of an application that makes use of the SLICOT interface described here.

4 MATLAB Interface

The main user-callable routine in the MATLAB interface for KINSOL presents the following form

```
function [uu, kinout] = kinsol(kfun, uu0, opts)
```

where `kinout` and `opts` are optional arguments. As we can see, the number of arguments reduces considerably with respect to the SLICOT interface. This has been done by grouping the input arguments in the structure `opts`, and the output arguments in the structure `kinout`. This keeps the interface easy to use while at the same time powerful.

The argument `kfun` is the function that evaluates F. `uu0` is the initial point. `uu` is the solution of the nonlinear system, if no error ocurred.

The argument `opts` is a structure containing input options. Each option corresponds either to an argument of the user-callable routine of the SLICOT interface, or to a parameter in the arrays `IOPT` or `ROPT`, which are arguments of the same routine. To create this structure, the user can call the function

```
function kopt = kinsoptions()
```

which creates a structure of input options with empty fields. The user can then assign a value to specific fields as needed, before passing the structure to the `kinsol` function.

The output argument kinout is a structrure containing additional information about the solution of the nonlinear system (e.g. number of nonlinear iterations or number of function evaluations). Each element of the structure corresponds to an output parameter in the arrays IOPT or ROPT.

The MATLAB user-defined functions present the following forms (see Appendix B for a more detailed description):

```
function fval = kfun(uu)

function [nfe, ierr] = kpreco(uu, uscale, fval, fscale, kfun, uround, nfe)

function [x, nfe, ierr] = kpsol(uu, uscale, fval, fscale, r, kfun, uround, nfe)

function [z, ierr] = fatimes(v, new_uu, uu)
```

The MATLAB interface is implemented by means of a gateway written in Fortran, plus two m-files. The gateway has an entry point from MATLAB, which calls the KINSOL package, and one routine for each of the user-provided functions (kfun, kpreco, kpsol and fatimes). These gateway routines only make the call to the corresponding MATLAB user function, returning the results to the KINSOL package.

The two m-files of the interface correspond to the two user-callable functions of this MATLAB interface. These functions deal with the structures opts and kinout, constructing them, forming the calling sequence of the KINSOL gateway from the information contained in opts, and filling kinout with the information returned by the gateway. These functions can be found in Appendix B of this Working Note.

Figure 2 shows a diagram where we can see the different layers involved when the MATLAB interface for KINSOL is used.

5 Examples of use

5.1 A trivial diagonal example

We consider a trivial diagonal example, taken from the KINSOL package, only for illustrating the use of the interfaces described here. In particular, the nonlinear function F is

$$F(u)_i = x_i^2 - i^2, \quad i = 1, \ldots, 128,$$

and we choose as starting point $x^\star$, with $x_i^\star = 2i$.

5.1.1 SLICOT interface

A Fortran program using the SLICOT interface for solving the problem could be as follows.

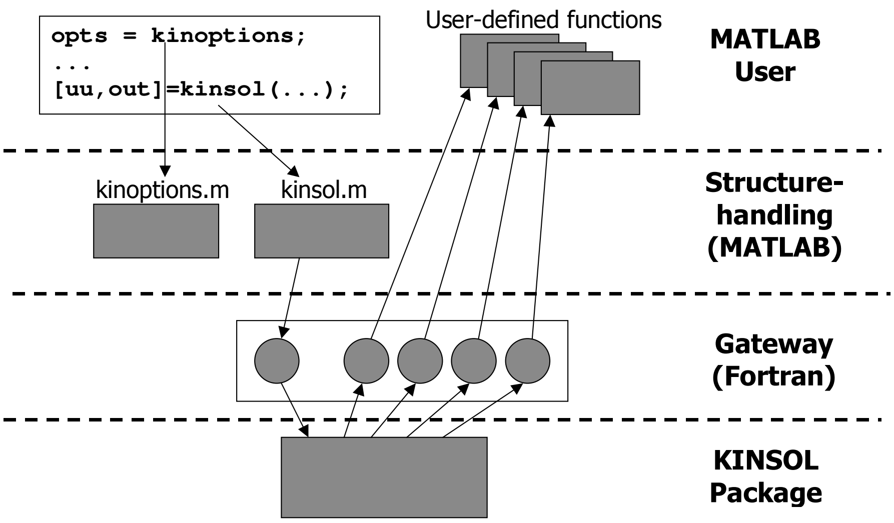

Figure 2: Diagram of the MATLAB interface.

```
      PROGRAM DIAGSF
C    ***************************************************************************
C
C     file: diagsf.f
C       Simple diagonal test with SLICOT interface, using user-supplied
C       preconditioner setup and solve routines (supplied in fortran, below).
C       This example does a basic test of the code suite by solving the system
C                             f(u) = 0.  for
C                             f(u) = u(i)^2 - i^2 .
C       no scaling is done.
C       execute line: diagsf
C
C     CONTRIBUTOR
C
C     Fernando Alvarruiz, Vicente Hernandez
C     Universidad Politecnica de Valencia, Spain
C
C    *--------------------------------------------------------------------------

      PARAMETER(PROBSIZE=128)

      INTEGER NEQ, IER, GSTRAT
      INTEGER IOPT(40)
      LOGICAL INOPT
      DOUBLE PRECISION TOL1, TOL2
```

```
      DOUBLE PRECISION ROPT(40)
      DOUBLE PRECISION UU(PROBSIZE), SCALE(PROBSIZE)
      DOUBLE PRECISION CONSTR(PROBSIZE)
      EXTERNAL FKINSPGMR20

C        Preconditioner data
      DOUBLE PRECISION PP(PROBSIZE)

      COMMON /PCOM/ PP

      NEQ = PROBSIZE
      GSTRAT = 0
      TOL1 = 0.00001
      TOL2 = 0.0001
      INOPT = .FALSE.
      MAXL = 10
      MAXLRST = 2
      MSBPRE  = 5
      MU = 0
      ML = 0

      DO 20 I = 1 , PROBSIZE
         UU(I) = 2.*I
         SCALE(I) = 1.
         CONSTR(I) = 0.
 20   CONTINUE

C * * * * * * * * * * * * * * * * * * * * * *
       WRITE(6,1240)
 1240  FORMAT('diagsf example case. this kinsol example code does a ')
       WRITE(6,1241)
 1241  FORMAT('128 eqn diagonal algebraic system. its purpose is to ')
       WRITE(6,1242)
 1242  FORMAT('demonstrate the use of the SLICOT interface.')
       WRITE(6,1244)
 1244  FORMAT('GSTRAT = INEXACT_NEWTON')

      CALL KINSOL (GSTRAT , FKINSPGMR20, NEQ, INOPT, MAXL, MAXLRST,
     &             MSBPRE, UU, SCALE, SCALE, CONSTR, IOPT, ROPT, TOL1,
     &             TOL2, INFO)

       WRITE(6,1245)INFO
 1245 FORMAT(1X,' Return code is ',I5)
```

```
      WRITE(6,1246)
 1246 FORMAT(1X,'The resultant values of UU are:',///)

      DO 30 I = 1, PROBSIZE, 4
         WRITE(6,1256)I, UU(I), UU(I+1), UU(I+2), UU(I+3)
 1256    FORMAT(I4,4(1X,F10.6))
 30   CONTINUE


       WRITE(6,1267) IOPT(4), IOPT(11), IOPT(5),
     1    IOPT(12), IOPT(13),  IOPT(14)
 1267 FORMAT(1X,'NNI=',I4,' NLI=',I4,' NFE=',I4,' NPE=',I4,
     1       ' NPS=',I4,' NCFL=',I4)

      STOP

      END
```

Following is the definition of the user-defined routines KFUN, KPRECO and KPSOL. It is not necessary for this simple example to use preconditionning. However, it is used in order to illustrate how to do it.

```
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     The function defining the system f(u) = 0. Must be defined by a Fortran
C     function of the following form.

      SUBROUTINE KFUN(NEQ, UU, FVAL)
      INTEGER NEQ
      DOUBLE PRECISION FVAL(*), UU(*)

      DO 10 I = 1 , NEQ
         FVAL(I) = UU(I)*UU(I) - I*I
 10   CONTINUE

      RETURN
      END

C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   The routine KPRECO is the preconditioner setup routine. It is required that
C   this specific name be used in order that the KINSOL code can find and
C   link to it.
C   The argument list must also be as illustrated below:

      SUBROUTINE KPRECO(NEQ, UDATA, USCALE, FDATA, FSCALE,
     1                  VTEMP1, VTEMP2, UROUND, NFE, IER)
```

```
      INTEGER NFE, IER

      DOUBLE PRECISION UDATA(*), USCALE(*), FDATA(*), FSCALE(*)
      DOUBLE PRECISION VTEMP1(*), VTEMP2(*)

      DOUBLE PRECISION UROUND

C     COMMON PCOM * * * * * * * * * * * * * * * * * * * * * * * * * * *
      PARAMETER(PROBSIZE=128)
      DOUBLE PRECISION PP(PROBSIZE)
      COMMON /PCOM/ PP
C     COMMON PCOM * * * * * * * * * * * * * * * * * * * * * * * * * * *

      IER = 0

      DO 10 I = 1 , NEQ
         PP(I) = 0.5/(UDATA(I)+5.)
 10   CONTINUE

      RETURN
      END


C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C   The routine KPSOL is the preconditioner solve routine. It is required that
C   this specific name be used in order that the KINSOL code can find and
C   link to it.
C   The argument list must also be as illustrated below:

      SUBROUTINE KPSOL(NEQ, UDATA, USCALE, FDATA, FSCALE,
     1                 VTEM, FTEM, UROUND, NFE, IER)

      INTEGER NFE, IER

      DOUBLE PRECISION UDATA(*), USCALE(*), FDATA(*), FSCALE(*)
      DOUBLE PRECISION VTEM(*), FTEM(*)

      DOUBLE PRECISION UROUND

C     COMMON PCOM * * * * * * * * * * * * * * * * * * * * * * * * * * *
      PARAMETER(PROBSIZE=128)
      DOUBLE PRECISION PP(PROBSIZE)
      COMMON /PCOM/ PP
C     COMMON PCOM * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
      IER = 0

      DO 10 I = 1 , NEQ
         VTEM(I) = VTEM(I) * PP(I)
 10   CONTINUE

      RETURN
      END
```

When running the program (in a Linux PC), the following output is obtained:

```
diagsf example case. This kinsol example code does a
128 eqn diagonal algebraic system. Its purpose is to
demonstrate the use of the SLICOT interface.
GSTRAT = INEXACT_NEWTON
  return code is     0
 The resultant values of UU are:



   1   1.000000   2.000000   3.000000   4.000000
   5   5.000000   6.000000   7.000000   8.000000
   9   9.000000  10.000000  11.000000  12.000000
  13  13.000000  14.000000  15.000000  16.000000
  17  17.000000  18.000000  19.000000  20.000000
  21  21.000000  22.000000  23.000000  24.000000
  25  25.000000  26.000000  27.000000  28.000000
  29  29.000000  30.000000  31.000000  32.000000
  33  33.000000  34.000000  35.000000  36.000000
  37  37.000000  38.000000  39.000000  40.000000
  41  41.000000  42.000000  43.000000  44.000000
  45  45.000000  46.000000  47.000000  48.000000
  49  49.000000  50.000000  51.000000  52.000000
  53  53.000000  54.000000  55.000000  56.000000
  57  57.000000  58.000000  59.000000  60.000000
  61  61.000000  62.000000  63.000000  64.000000
  65  65.000000  66.000000  67.000000  68.000000
  69  69.000000  70.000000  71.000000  72.000000
  73  73.000000  74.000000  75.000000  76.000000
  77  77.000000  78.000000  79.000000  80.000000
  81  81.000000  82.000000  83.000000  84.000000
  85  85.000000  86.000000  87.000000  88.000000
  89  89.000000  90.000000  91.000000  92.000000
  93  93.000000  94.000000  95.000000  96.000000
```

```
 97  97.000000  98.000000  99.000000 100.000000
101 101.000000 102.000000 103.000000 104.000000
105 105.000000 106.000000 107.000000 108.000000
109 109.000000 110.000000 111.000000 112.000000
113 113.000000 114.000000 115.000000 116.000000
117 117.000000 118.000000 119.000000 120.000000
121 121.000000 122.000000 123.000000 124.000000
125 125.000000 126.000000 127.000000 128.000000
NNI=    7 NLI=   21 NFE=   36 NPE=    2 NPS=   28 NCFL=    0
```

5.1.2 MATLAB interface

To solve the problem from MATLAB, the following simple call is enough

```
[uu, kinout] = kinsol('func', 2*[1:128]');
```

and the function `func` is defined as follows (in this case we do not use preconditionning).

```
function fu = func (uu)
fu = uu.^2 - [1:length(uu)]'.^2;
```

The contents of `uu` after calling the `kinsol` function are as expected, while `kinout` contains the following infomation (results may vary slightly depending on the computer used):

```
kinout =

        NonLinIters: 29
       NumFuncEvals: 315
    NumBetaCondFail: 0
      NumBacktracks: 0
              FNorm: 0
         StepLength: 1.2645e-014
           LinIters: 256
       NumPrecEvals: 0
          NumPSolve: 0
    NumLinConvFails: 24
```

5.2 Predator-prey system

The following example, taken from the KINSOL package, is more illustrative of the power of KINSOL with real problems. The example problem is a Partial Differential Equation model of a multi-species food web [2], in which mutual competition and/or predator-prey relationships in a spatial domain are simulated. For this problem the dependent variable c (concentration of a species in a spatial point) replaces the generic dependent variable u used above. We consider a model with $s = 2p$ species, where both species $1, \ldots, p$ (the prey) and $p+1, \ldots, s$ (the predators) have infinitely fast reaction rates:

$$0 = f_i(x, y, c) + d_i(c^i_{xx} + c^i_{yy}) \quad (i = 1, 2, \ldots, s),$$

where (x, y) are the coordinates of a point in the plane; c^i_{xx} and c^i_{yy} denote the second derivatives with respect to x and y respectively, of the concentration of the i-th species, and

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

The interaction and diffusion coefficients (a_{ij}, b_i, d_i) could be functions of (x, y) in general. The choices made for this test problem are for a simple model of p prey and p predator species, arranged in that order in the vector c. We take the various coefficients to be as follows:

$$\begin{cases} a_{ii} = -1 & (i = 1, 2, \ldots, s) \\ a_{ij} = -0.5 \cdot 10^{-6} & (i \leq p, j > p) \\ a_{ij} = 10^4 & (i > p, j \leq p) \\ a_{ij} = 0 & \text{for all other } i, j \end{cases}$$

$$\begin{cases} b_i = b_i(x, y) = (1 + \alpha x y) & (i \leq p) \\ b_i = b_i(x, y) = -(1 + \alpha x y) & (i > p) \end{cases}$$

and

$$\begin{cases} d_i = 1 & (i \leq p) \\ d_i = 0.5 & (i > p). \end{cases}$$

The domain is the unit square $0 \leq x, y \leq 1$. The boundary conditions are of Neumann type (zero normal derivatives) everywhere. The coefficients are such that a unique stable equilibrium is guaranteed to exist when α is zero [2]. Empirically, a stable equilibrium appears to exist when α is positive, although it may not be unique. In this problem we take $\alpha = 1$. The initial species concentrations used are constant functions by species type, in particular

$$\begin{cases} c_i = 1 & (i \leq p) \\ c_i = 30000 & (i > p). \end{cases}$$

The PDE system (with boundary conditions) was discretized using central differencing on an $MX \times MY$ mesh, with the resulting nonlinear system having size $N = s \cdot MX \cdot MY$.

5.2.1 SLICOT Interface

The following Fortran program implements the solution of this problem using the SLICOT interface to KINSOL.

```
      program predprey
c ********************************************************************
```

```
* File: predprey.f
*
*
*     CONTRIBUTOR
*
*     Fernando Alvarruiz, Vicente Hernandez
*     Universidad Politecnica de Valencia, Spain
*
*     Adapted from:
*        kinxs.c, by Allan G. Taylor and Alan C. Hindmarsh @ LLNL
*        Version of 16 January 2001
*
*-----------------------------------------------------------------------*
*
* Example problem for KINSol, serial machine version.
* This example solves a nonlinear system that arises from a system of
* partial differential equations. The PDE system is a food web
* population model, with predator-prey interaction and diffusion on the
* unit square in two dimensions. The dependent variable vector is
*
*       1   2         ns
* c = (c , c ,  ..., c  )              (denoted by the variable cc)
*
* and the PDE's are as follows:
*
*                    i       i
*         0 = d(i)*(c     + c    )  +  f  (x,y,c)   (i=1,...,ns)
*                    xx      yy         i
*
*   where
*
*                   i              ns         j
*   f  (x,y,c)  =  c  * (b(i)  + sum a(i,j)*c )
*    i                           j=1
*
* The number of species is ns = 2 * np, with the first np being prey and
* the last np being predators. The number np is both the number of prey and
* predator species. The coefficients a(i,j), b(i), d(i) are
*
*   a(i,i) = -AA  (all i)
*   a(i,j) = -GG  (i <= np , j >  np)
*   a(i,j) =  EE  (i >  np,  j <= np)
*   b(i) = BB * (1 + alpha * x * y)  (i <= np)
*   b(i) =-BB * (1 + alpha * x * y)  (i >  np)
*   d(i) = DPREY  (i <= np)
```

```
*   d(i) = DPRED  ( i > np)
*
*  The various scalar parameters are set using parameter statements
*  The boundary conditions are: normal derivative  =  0.
*  The initial guess is constant in x and y, although the final
*  solution is not.
*
*  The PDEs are discretized by central differencing on a MX by MY mesh.
*
*  The nonlinear system is solved by KINSOL using the method specified in
*  local variable gstrat .
*
*  The preconditioner matrix is a block-diagonal matrix based on the
*  partial derivatives of the interaction terms f only.
*
*
* References:
*
* 1. Peter N. Brown and Youcef Saad,
*    Hybrid Krylov Methods for Nonlinear Systems of Equations
*    LLNL report UCRL-97645, November 1987.
*
* 2. Peter N. Brown and Alan C. Hindmarsh,
*    Reduced Storage Matrix Methods in Stiff ODE systems,
*    Lawrence Livermore National Laboratory Report  UCRL-95088, Rev. 1,
*    June 1987, and  Journal of Applied Mathematics and Computation, Vol. 31
*    (May 1989), pp. 40-91. (Presents a description of the time-dependent
*    version of this test problem.)
*****************************************************************************/

      implicit none


c     ********* Common parameters **********************************

      integer NS, NP, MX, MY, NEQ
      double precision AA, EE, GG, BB, DPREY, DPRED, ALPHA, AX, AY,
     &                 DX, DY, FTOL, STOL, PREYIN, PREDIN

c     Number of species
      parameter(NS = 6)

c     Number of prey/predator species
      parameter(NP = NS/2)
```

```
c     Number of x mesh points
      parameter(MX = 8)

c     Number of y mesh points
      parameter(MY = 8)

c     Total range of x variable
      parameter(AX   = 1.0)

c     Total range of y variable
      parameter(AY   = 1.0)

c     Distance between x mesh points
      parameter(DX   = AX/(MX-1))

c     Distance between y mesh points
      parameter(DY   = AY/(MY-1))

c     Number of nonlinear equations
      parameter(NEQ  = NS * MX * MY)

      parameter(AA   = 1.0)
      parameter(EE   = 10000.0)
      parameter(GG   = 0.5e-6)
      parameter(BB   = 1.0)
      parameter(DPREY= 1.0)
      parameter(DPRED= 0.5)
      parameter(ALPHA= 1.0)

c     Tolerances for the solution of the system
      parameter(FTOL = 1.e-7)
      parameter(STOL = 1.e-13)

c     Initial guess for prey concentrations
      parameter(PREYIN = 1.0)

c     Initial guess for predator concentrations
      parameter(PREDIN = 30000.0)

c     ********* End of Common parameters ****************************


c     ********* Common variables ************************************

      double precision Acoef(NS,NS), Bcoef(NS)
```

```
c     Preconditioning data
      double precision P(NS,NS,MX,MY), ipiv(NS,MX,MY)

c     Interaction rates, i.e. result of evaluating f (x,y,c)
c                                                   i
      double precision rates(NS,MX,MY)

      double precision cox(NS), coy(NS)

c     Machine unit roundoff and its square root
      double precision uround, squrnd

      common /pcom/ Acoef, Bcoef, P, ipiv, rates, cox, coy, uround,
     &              squrnd

c     ********* End of common variables ****************************


c     Local variables
      integer i, info, gstrat, maxl, maxlrst, msbpre
      integer iopt(40)
      logical inopt
      double precision ropt(40)
      double precision cc(NS,MX,MY), sc(NS,MX,MY)
      double precision constr(NEQ)
      external fkinspgmr20

      call InitUserData()

      call SetInitialProfiles(cc, sc)

      do i = 1,NEQ
          constr(i) = 0.0
      end do

      gstrat = 0
      inopt = .false.
      maxl = 15
      maxlrst = 2
      msbpre  = 0

      print *, 'Predator-prey test problem --  KINSol (serial version)'
      print *, 'Mesh dimensions =', MX, ' X', MY
      print *, 'Number of species =', NS
```

```
      print *, 'Total system size =', NEQ
      print *, 'Flag globalstrategy =', gstrat,
     & ' (0 = Inex. Newton, 1 = Linesearch)'
      print *, 'Linear solver is SPGMR with maxl =', maxl,
     & ' maxlrst =', maxlrst
      print *, 'Preconditioning uses interaction-only block-diagonal ',
     & 'matrix'
      print *, 'Tolerance parameters:  fnormtol =', FTOL,
     & ' scsteptol =', STOL

      print *, 'Initial profile of concentration'
      print *, 'At all mesh points:', PREYIN,PREYIN,PREYIN,
     &    PREDIN,PREDIN,PREDIN


c     Call KINSol and print output concentration profile

      call kinsol (gstrat , fkinspgmr20, NEQ, inopt, maxl, maxlrst,
     &             msbpre, cc, sc, sc, constr, iopt, ropt, FTOL,
     &             STOL, info)

      if (info .ne. 0) then
         print *, 'KINSOL failed, returning ', info
         stop
      end if

      print *, ''
      print *, ''
      print *, 'Computed equilibrium species concentrations:'
      call PrintOutput(cc)

c     Print final statistics
      print *, ''
      call PrintFinalStats(iopt)

      stop

      end

c * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
c     The function defining the system f(u) = 0 must be defined by a fortran
c     function of the following form.

      subroutine kfun(neqn, cc, fval)
      implicit none
```

```
(... Deleted lines correponding to common parameters and variables ...)

c     Function arguments
      integer neqn
      double precision fval(NS,MX,MY), cc(NS,MX,MY)

c     Local variables
      double precision xx, yy, dcx, dcy
      integer jx, jy, is, up, down, left, right

c     Loop over all mesh points, evaluating rate array at each point
      do jy=1,MY

          yy = (jy-1)*DY
          if (jy .ne. 1) then
              down = jy-1
          else
              down = 2
          end if
          if (jy .ne. MY) then
              up = jy+1
          else
              up = MY-1
          end if

          do jx=1,MX

              xx = (jx-1)*DX
              if (jx .ne. 1) then
                  left = jx-1
              else
                  left = 2
              end if
              if (jx .ne. MX) then
                  right = jx+1
              else
                  right = MX-1
              end if

              call WebRate(xx, yy, cc(1,jx,jy), rates(1,jx,jy))

              do is = 1,NS
                 dcy = coy(is) *
     &                 (cc(is,jx,down)+cc(is,jx,up)-2*cc(is,jx,jy))
```

```
              dcx = cox(is) *
     &              (cc(is,left,jy)+cc(is,right,jy)-2*cc(is,jx,jy))

              fval(is,jx,jy) = dcy + dcx + rates(is,jx,jy)

           end do
        end do
      end do

      return
      end

c * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
c   The routine kpreco is the preconditioner setup routine. It is required that
c   this specific name be used in order that the KINSOL code can find and
c   link to it.
c   The argument list must also be as illustrated below:

      subroutine kpreco(neqn, cc, cscale, fval, fscale,
     1                  vtemp1, vtemp2, urnd, nfe, ier)
      implicit none

(... Deleted lines correponding to common parameters and variables ...)

c     Function arguments
      integer neqn, nfe, ier

      double precision cc(NS,MX,MY), cscale(NS,MX,MY), fval(NS,MX,MY),
     &                 fscale(NS,MX,MY)
      double precision vtemp1(*), vtemp2(*)

      double precision urnd

c     Local variables
      integer jx, jy, j, i, info
      double precision csave, xx, yy, r, r0, fac
      double precision PertRates(NS)
      double precision WNorm

      ier = 0

      fac = WNorm(NEQ, fval, fscale)
      r0 = 1000.0 * uround * fac * NEQ
      if (r0 .eq. 0.0) r0 = 1
```

```
c     Loop over all spatial points; get NS-sized Jacobian block at each
      do jy = 1, MY

          yy = (jy-1)*DY

          do jx = 1, MX

              xx = (jx-1)*DX

c             Compute difference quotients of interaction rate function
              do j = 1,NS

c                 Save the j,jx,jy element of cc
                  csave = cc(j,jx,jy)

                  r = max(squrnd*abs(csave), r0/cscale(j,jx,jy))
c                 Perturb the j,jx,jy element of cc
                  cc(j,jx,jy) = cc(j,jx,jy) + r
                  fac = 1.0/r

                  call WebRate(xx, yy, cc(1,jx,jy), PertRates)

c                 Restore the j,jx,jy element of cc
                  cc(j,jx,jy) = csave

c                 Load the j-th column of difference quotients
                  do i = 1,NS
                      P(i,j,jx,jy) = (PertRates(i)-rates(i,jx,jy))*fac
                  end do
              end do

c             Do LU decomposition of preconditioner block
              call dgetrf(NS, NS, P(1,1,jx,jy), NS, ipiv(1,jx,jy), info)
              if (info .ne. 0) then
                  ier = info
                  return
              end if

          end do
      end do

      return
      end
```

```
c * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
c   The routine kpsol is the preconditioner solve routine. It is required that
c   this specific name be used in order that the KINSOL code can find and
c   link to it.
c   The argument list must also be as illustrated below:

      subroutine kpsol(neqn, cc, cscale, fval, fscale,
     &                 vv, ftem, urnd, nfe, ier)
      implicit none

(... Deleted lines correponding to common parameters and variables ...)

c     Function arguments
      integer neqn, nfe, ier
      double precision cc(NS,MX,MY), cscale(NS,MX,MY), fval(NS,MX,MY),
     &                 fscale(NS,MX,MY)
      double precision vv(NS,MX,MY), ftem(*)
      double precision urnd

c     Local variables
      integer jx, jy, info

      ier = 0

      do jy = 1,MY
          do jx = 1,MX

c             For each (jx,jy), solve a linear system of size NS
              call dgetrs('N', NS, 1, P(1,1,jx,jy), NS, ipiv(1,jx,jy),
     &                    vv(1,jx,jy), NS, info)

          end do
      end do

      return
      end

c ***********************************************************************
      subroutine WebRate (xx, yy, cxy, rxy)
      implicit none

c     This function computes f(x,y,c) at point (xx, yy)

(... Deleted lines correponding to common parameters and variables ...)
```

```
c     Function arguments
      double precision xx, yy
      double precision cxy(NS), rxy(NS)

c     Local variables
      double precision ONE
      parameter (ONE=1.0)
      integer is
      double precision fac

c     ( rxy = Bcoef )
      call dcopy(NS, Bcoef, 1, rxy, 1)

      fac = 1 + ALPHA*xx*yy

c     ( rxy = fac*rxy + Acoef*cxy )
      call dgemv('N', NS, NS, ONE, Acoef, NS, cxy, 1, fac, rxy, 1)

      do is = 1,NS
         rxy(is) = cxy(is) * rxy(is)
      end do

      return
      end

c *********************************************************************
      subroutine InitUserData()
      implicit none

(... Deleted lines correponding to common parameters and variables ...)

c     Local variables
      integer i, j
      double precision dx2, dy2
      double precision dlamch

      uround = dlamch('E')
      squrnd = sqrt(uround)

      dx2 = DX*DX
      dy2 = DY*DY

      do j=1,NP

c        Fill in the portion of Acoef in the four quadrants,
```

```
c         column by column
          do i=1,NP
              Acoef(i,j) = 0.0
              Acoef(i, NP+j) = -GG
              Acoef(NP+i, j) = EE
              Acoef(NP+i, NP+j) = 0.0
          end do

c         and then change the diagonal elements of Acoef to -AA
          Acoef(j,j) = -AA
          Acoef(NP+j, NP+j) = -AA

          Bcoef(j) = BB
          Bcoef(NP+j) = -BB

          cox(j) = DPREY/dx2
          cox(NP+j) = DPRED/dx2

          coy(j) = DPREY/dy2
          coy(NP+j) = DPRED/dy2

      end do

      return
      end

c **********************************************************************
      subroutine SetInitialProfiles(cc, sc)
      implicit none

(... Deleted lines correponding to common parameters and variables ...)

c     Local variables
      double precision cc(NS,MX,MY), sc(NS,MX,MY)
      integer i, j, is

      do j=1,MY
          do i=1,MX
              do is = 1,NP
                  cc(is,i,j) = PREYIN
                  sc(is,i,j) = 1
              end do
              do is = NP+1,NS
                  cc(is,i,j) = PREDIN
                  sc(is,i,j) = 0.00001
```

```
            end do
         end do
      end do

      return
      end


      subroutine PrintOutput(cc)
      implicit none

(... Deleted lines correponding to common parameters ...)

      double precision cc(NS,MX,MY)

c     Local variables
      integer is

      print *, 'At bottom left:'
      do is = 1,NS
         print *, cc(is,1,1)
      end do

      print *, 'At top right:'
      do is = 1,NS
         print *, cc(is,MX,MY)
      end do

      return
      end


      subroutine PrintFinalStats(iopt)
      implicit none

      integer iopt(40)

      print *, 'Final Statistics:'
      print *, ' Non-linear iterations      = ', iopt(4)
      print *, ' Linear iterations          = ', iopt(11)
      print *, ' Num. function evals        = ', iopt(5)
      print *, ' Num. precond evals         = ', iopt(12)
      print *, ' Num. precond solves        = ', iopt(13)
      print *, ' Num. linear conv. failures = ', iopt(14)
```

```
      return
      end


      double precision function WNorm(n, x, w)
c     Weighted 2-norm

      implicit none
      integer n
      double precision x(n), w(n)

      integer i
      double precision s, prodi

      s = 0.0
      do i = 1,n
          prodi = x(i) * w(i);
          s = s + prodi * prodi;
      end do

      WNorm = sqrt(s);
      return
      end
```

The program defines in the first place different constants and variables common to the main program and other subroutines. Other coefficients defining the problem, such as a_{ij} and machine unit roundoff, are initialized in subroutine `InitUserData`. Subroutine `SetInitialProfiles` provides the initial guess for the species concentrations, and also a vector that will be used for scaling both function values and concentrations within KINSOL. Note that the dependent variable c is implemented by means of a 3-dimensional array `cc`, although the KINSOL solver will see it as a 1-dimensional array, ordered in the following way

$$c = [c_1^{x_1 y_1} \dots c_s^{x_1 y_1} \dots c_1^{x_{MX}, y_1} \dots c_s^{x_{MX}, y_1} \; c_1^{x_1 y_2} \dots c_s^{x_1 y_2} \dots c_1^{x_{MX}, y_{MY}} \dots c_s^{x_{MX}, y_{MY}}]^T$$

where $c_i^{x_j y_k}$ denotes the concentration of species i at the point (x_j, y_k).

The implementation of the nonlinear function is contained in subroutine `kfun`, which in turn calls `WebRates` in order to compute the terms $f_i(x, y, c)$.

If we look at the call to the `kinsol` subroutine (the SLICOT interface) in the main program, we can see that the subroutine name `fkinspgmr20` is used as the second argument, wich means that preconditioning will be used, and the user-defined subroutines `kpreco` and `kpsol` must be provided. The preconditioner matrix is a block-diagonal matrix based on the partial derivatives of the interaction terms $f(x, y, c)$ only. This matrix is built and factorized in `kpreco`, using finite differences for computing the partial derivatives. LAPACK is used for LU factorization of the diagonal blocks. Subroutine `kpsol` solves a linear system with the factorized preconditioner as the coefficient matrix. Again LAPACK is used for solving the triangular systems.

The following output is obtained when running the program on a Linux PC (again we must note that results may vary slightly depending on the computer/compiler used).

```
Predator-prey test problem --  KINSol (serial version)
Mesh dimensions = 8 X 8
Number of species = 6
Total system size = 384
Flag globalstrategy = 0 (0 = Inex. Newton, 1 = Linesearch)
Linear solver is SPGMR with maxl = 15 maxlrst = 2
Preconditioning uses interaction-only block-diagonal matrix
Tolerance parameters:  fnormtol =  1.00000001E-07 scsteptol =  9.99999982E-14
Initial profile of concentration
At all mesh points:  1.  1.  1.  30000.  30000.  30000.


Computed equilibrium species concentrations:
At bottom left:
 1.16427931
 1.16427931
 1.16427931
 34927.4876
 34927.4876
 34927.4876
At top right:
 1.25796688
 1.25796688
 1.25796688
 37736.6641
 37736.6641
 37736.6641

Final Statistics:
 Non-linear iterations      =  9
 Linear iterations          =  352
 Num. function evals        =  371
 Num. precond evals         =  1
 Num. precond solves        =  361
 Num. linear conv. failures =  7
```

5.2.2 MATLAB Interface

The file `predprey.m`, which is listed next, is a MATLAB script for the solution of the predator-prey problem using the MATLAB interface to KINSOL.

```
global data
```

```
% Initialization
data = initprpr(8,8,6);
[cc0, sc] = SetInitialProfiles(data);

% Set KINSOL Options
opts = kinsoptions;
opts.Uscale = sc;
opts.Fscale = sc;
opts.FNormTol = 1e-7;
opts.ScStepTol = 1e-13;
opts.MaxLinDim = 15;
opts.MaxLinRestarts = 2;
opts.PrecondSetFunc = 'kpreco';
opts.PrecondSolveFunc = 'kpsol';

% Solve nonlinear system of equations
[cc, kinout] = kinsol('funcprpr', cc0, opts);
```

The script calls auxiliary functions `initprpr` and `SetInitialProfiles`. The former initializes the problem coefficients and stores them in the structure global variable `data`, while the latter gives the inital guess for the concentrations and the scaling vector for KINSOL. These functions are listed next.

```
function data = initprpr(mx, my, ns)

ns2 = ns/2;

data.ax = 1;
data.ay = 1;
data.mx = mx;
data.my = my;
data.dx = data.ax/(data.mx-1);
data.dy = data.ay/(data.my-1);
data.ns = ns;
data.alpha = 1;

BB = 1;
data.bcoef = [ones(1,ns2)*BB -ones(1,ns2)*BB]';

AA = 1;
GG = 0.5e-6;
EE = 10000;
data.acoef = [...
   -diag(ones(ns2,1)*AA)    -ones(ns2)*GG; ...
```

```
      ones(ns2)*EE      -diag(ones(ns2,1)*AA)];

DPREY = 1;
DPRED = 0.5;
a = DPREY/(data.dx*data.dx);
b = DPRED/(data.dx*data.dx);
data.cox = [ones(1,ns2)*a ones(1,ns2)*b]';

a = DPREY/(data.dy*data.dy);
b = DPRED/(data.dy*data.dy);
data.coy = [ones(1,ns2)*a ones(1,ns2)*b]';

% Initialization of Preconditioning data
data.sqruround = sqrt(eps);
data.L = zeros(ns,ns,mx,my);
data.U = zeros(ns,ns,mx,my);


function [cc, sc] = SetInitialProfiles(data)
mx = data.mx;
my = data.my;
ns = data.ns;
np = ns/2;

PREYIN=1;
PREDIN=30000;

cc = ones(ns,mx,my);
sc = ones(ns,mx,my);
cc(1:np,:,:)    = cc(1:np,:,:)    * PREYIN;
cc(np+1:ns,:,:) = cc(np+1:ns,:,:) * PREDIN;
sc(1:np,:,:)    = sc(1:np,:,:)    * 1 ;
sc(np+1:ns,:,:) = sc(np+1:ns,:,:) * 0.00001;

cc = reshape(cc, ns*mx*my, 1);
sc = reshape(sc, ns*mx*my, 1);
```

The evaluation of the function and the preconditioning related functions are presented next:

```
function fval=funcprpr(cc)
% System function for the predator-prey system

global data;

dx = data.dx;
```

```
dy = data.dy;
mx = data.mx;
my = data.my;
ns = data.ns;
cox = data.cox;
coy = data.coy;

rates = zeros(ns, mx, my);
cc = reshape(cc, ns, mx, my);
fval = zeros(ns, mx, my);

down  = [2 [1:my-1]]';
up    = [[2:my] my-1]';
left  = [2 [1:mx-1]]';
right = [[2:mx] mx-1]';

yvec = dy * [0:my-1]';
xvec = dx * [0:mx-1]';

% Loop over all mesh points, evaluating function at each point

for jy = 1:my

    % Get species interaction rate at (:, yy)
    rates(:,:,jy) = WebRateY(xvec, yvec(jy), cc(:,:,jy));

    fval(:,:,jy) = ...
        diag(coy) * (cc(:, :, up(jy)) + cc(:, :, down(jy)) - 2*cc(:, :, jy)) + ...
        diag(cox) * (cc(:, right, jy) + cc(:, left, jy) - 2*cc(:, :, jy)) + ...
        rates(:,:,jy);

end

data.rates = rates;
fval = reshape(fval, ns*mx*my, 1);


function rates = WebRateY(xvec, yy, cy)
% Computes rates for Interaction of species at points
% (xvec(i), yy) for all i

global data;

alpha = data.alpha;
```

```
fac = 1 + alpha * yy * xvec;
rates = data.bcoef * fac' + data.acoef * cy;
rates = cy .* rates;


function [nfe, ierr] = kpreco(cc, cscale, fval, fscale, kfun, uround, nfe)
% Preconditioner setup routine. Generates P

  global data;

  ierr = 0;

  mx = data.mx;
  my = data.my;
  dx = data.dx;
  dy = data.dy;
  ns = data.ns;
  sqruround = data.sqruround;

  cc = reshape(cc, ns, mx, my);
  cscale = reshape(cscale, ns, mx, my);

  r0 = 1000 * eps * norm(fval.*fscale) * length(fval);
  if (r0 == 0) r0=1; end

  yvec = dy * [0:my-1]';
  xvec = dx * [0:mx-1]';

  % Loop over spatial points; get NS-sized Jacobian block at each point
  P = zeros(ns);
  for jy = 1:my

      for jx = 1:mx

          for j = 1:ns
              % Save element of cc
              csave = cc(j,jx,jy);

              % Perturb element of cc
              r = max(sqruround*abs(csave), r0/cscale(j,jx,jy));
              cc(j,jx,jy) = cc(j,jx,jy) + r;

              perturb_rates = WebRate(xvec(jx), yvec(jy), cc(:,jx,jy));

              % Restore element of cc
```

```
            cc(j,jx,jy) = csave;

            % Load j-th column of difference quotients
            P(:,j) = (perturb_rates - data.rates(:,jx,jy)) ./ r;
          end

          % Do LU decomposition of preconditioner block
          [data.L(:,:,jx,jy), data.U(:,:,jx,jy)] = lu(P);
        end
  end


function rates = WebRate(xx, yy, cxy)
% Computes rates for Interaction of species at point (xx, yy)

global data;

alpha = data.alpha;

fac = 1 + alpha*xx*yy;
rates = fac * data.bcoef + data.acoef * cxy;
rates = cxy .* rates;


function [x, nfe, ierr] = kpsol(cc, cscale, fval, fscale, r, funcprpr, uround, nfe)
% Preconditioner setup routine. Generates P
  global data;
  ierr = 0;

  mxmy = data.mx * data.my;
  ns = data.ns;

  r = reshape(r, ns, mxmy);
  x = zeros(ns, mxmy);

  % For each spatial point (jx, jy), solve a NS-sized linear system
  for jxy = 1:mxmy
          x(:,jxy) = data.L(:,:,jxy) \ r(:,jxy);
          x(:,jxy) = data.U(:,:,jxy) \ x(:,jxy);
  end

  x = reshape(x, ns*mxmy, 1);
```

The evaluation of the nonlinear function is done in `funcprpr`, which, similarly to the Fortran implementation, calls the function `WebRateY` in order to evaluate the terms $f(x, y, c)$. However, in this case `funcprpr` presents a single loop, which corresponds to the loop over the y coordinates, i.e. the outermost loop in the corresponding Fortran subroutine `kfun`. The other two loops have been in this case *vectorized* (i.e. they are now implicit, forming part of matrix/vector operations). This vectorization of loops is important in order to obtain acceptable execution times in MATLAB code.

As a consequence of this loop vectorization, `WebRateY` must in this case compute the rates $f(x, y, c)$ not for a single (x, y) point, but for all points with a given y coordinate.

The code for the preconditioning is very similar to the corresponding Fortran code. Note that in order to compute the preconditioner we use function `WebRate` instead of `WebRateY`, because in this case we need to evaluate $f(x, y, c)$ only one (x, y) point at a time.

The final concentrations of the bottom left and top right points, and KINSOL statistical information is presented next (results obtained on a Windows PC with Matlab 6):

```
Final concentrations at bottom left point:

  1.0e+004 *

   0.00011642793077
   0.00011642793077
   0.00011642793077
   3.49274875697278
   3.49274875697278
   3.49274875697278

Final concentrations at top right point:

  1.0e+004 *

   0.00012579668753
   0.00012579668753
   0.00012579668753
   3.77366640744681
   3.77366640744681
   3.77366640744681

Statistical KINSOL information:

kinout =

        NonLinIters: 8
       NumFuncEvals: 325
    NumBetaCondFail: 0
      NumBacktracks: 0
              FNorm: 5.367633801921303e-008
         StepLength: 1.449283603070963e-007
```

```
        LinIters: 308
    NumPrecEvals: 1
       NumPSolve: 316
 NumLinConvFails: 6
```

References

[1] Allan G. Taylor and Alan C. Hindmarsh, *User Documentation for KINSOL, a Nonlinear Solver for Sequential and Parallel Computers*, Center for Applied Scientific Computing, L-561, LLNL, Livermore, CA 94551.

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

[3] Working Group on Software (WGS), *SLICOT Implementation and Documentation Standards*, WGS-Report 96-1, Eindhoven University of Technology, February 1998, ftp://wgs.esat.kuleuven.ac.be/pub/WGS/REPORTS/rep96-1.ps.Z.

A SLICOT interface for KINSOL

We present here the source code of the subroutine KINSOL, which implements the SLICOT interface described in this Working Note.

```
      SUBROUTINE KINSOL (GSTRAT, LINSU, NEQ, OPTIN, MAXL, MAXLRST,
     &                   MSBPRE, UU, USCALE, FSCALE, CONSTR,
     &                   IOPT, ROPT, TOL1, TOL2, INFO)
C
C     WGS COPYRIGHT 2000.
C
C     PURPOSE
C
C     To solve a nonlinear system of equations F(u)=0, where F(u) is
C                                   n    n
C     a nonlinear function from R to R , using Krylov Inexact Newton
C     techniques.
C
C     ARGUMENTS
C
C     GSTRAT  (input) INTEGER
C             Indicates the global strategy to apply the computed
C             increment delta in the solution UU.  Choices are:
C             0 - Inexact Newton.
C             1 - Linesearch.
```

```
C
C     LINSU   SUBROUTINE
C             Linear Solver Set-up Routine. This is the KINSOL routine
C             to be called to set-up the linear solver. The user should
C             specify here one of the 6 different Fortran-callable
C             routines provided by KINSOL for this purpose. The choice
C             to be used depends on which of the optional user-defined
C             routines are provided by the user (see User-defined
C             routines below).
C             LINSU can be one of the following routines: FKINSPGMR00,
C             FKINSPGMR01, FKINSPGMR10, FKINSPGMR11, FKINSPGMR20, and
C             FKINSPGMR21, where the first digit in the name of the
C             function is: 0 if neither KPSOL nor KPRECO routines are
C             provided; 1 if only the preconditioner solve routine
C             (KPSOL) is provided; and 2 if both the preconditioner
C             solve (KPSOL) and setup (KPRECO) routines are provided.
C             The second digit is: 0 if a function FATIMES is not
C             provided; and 1 if a function FATIMES is provided.
C
C     NEQ     (input) INTEGER
C             Number of equations (and unknowns) in the algebraic
C             system.
C
C     OPTIN   (input) LOGICAL
C             Flag indicating whether optional inputs from the user in
C             the arrays IOPT and ROPT are to be used.
C             Pass FALSE to ignore all optional inputs and TRUE to use
C             all optional inputs that are present. Either choice does
C             NOT affect outputs in other positions of IOPT or ROPT.
C
C     MAXL    (input) INTEGER
C             Maximum Krylov dimension for the Linear Solver. Pass 0
C             to use the default value MIN(Neq, 10).
C
C     MAXLRST (input) INTEGER
C             Maximum number of linear solver restarts allowed. Values
C             outside the range 0 to 2*NEQ/MAXL will be restricted to
C             that range. 0, meaning no restarts, is a safe starting
C             value.
C
C     MSBPRE  (input) INTEGER
C             Maximum number of steps calling the solver KPSOL
C             without calling the preconditioner KPRECO. (The default is
C             10).
C
```

```
C     UU      (input/output) DOUBLE PRECISION array, dimension (NEQ)
C             On entry, UU is the initial guess.
C             On exit, if no errors ocurr, UU is the solution of
C             the system KFUN(UU) = 0.
C
C     USCALE  (input) DOUBLE PRECISION array, dimension (NEQ)
C             Array of diagonal elements of the scaling matrix for UU.
C             The elements of USCALE must be positive values. The
C             scaling matrix USCALE should be chosen so that
C             USCALE * UU (as a matrix multiplication) should have all
C             its components with roughly the same magnitude when UU is
C             close to a root of KFUN.
C
C     FSCALE  (input) DOUBLE PRECISION array, dimension (NEQ)
C             Array of diagonal elements of the scaling matrix for
C             KFUN. The elements of FSCALE must be positive values.
C             The scaling matrix FSCALE should be chosen so that
C             FSCALE * KFUN(UU) (as a matrix multiplication) should
C             have all its components with roughly the same magnitude
C             when UU is NOT too near a root of KFUN.
C
C     CONSTR  (input) DOUBLE PRECISION array, dimension (NEQ)
C             Constraints on UU.
C             A positive value in CONSTR(I) implies that the Ith
C             component of UU is to be constrained > 0.
C             A negative value in CONSTR(I) implies that the Ith
C             component of UU is to be constrained < 0.
C             A zero value in CONSTR(I) implies there is no constraint
C             on UU(I).
C
C     IOPT    (input/output) INTEGER array, dimension (40)
C             Array of optional integer inputs and outputs.
C             If OPTIN is TRUE, the user should preset to 0 those
C             locations for which default values are to be used.
C             See Optional Inputs and Outputs, below.
C
C     ROPT    (input/output) DOUBLE PRECISION array, dimension (40)
C             Array of optional double precision inputs and outputs.
C             If OPTIN is TRUE, the user should preset to 0 those
C             locations for which default values are to be used.
C             See Optional Inputs and Outputs, below.
C
C     Tolerances
C
C     TOL1    DOUBLE PRECISION
```

```
C            Stopping tolerance on maxnorm( FSCALE * KFUN(UU) ).
C            If TOL1 is input as 0., then a default value of
C            (uround) to the 1/3 power will be used. uround is the
C            unit roundoff for the machine in use for the calculation.
C
C     TOL2   DOUBLE PRECISION
C            Stopping tolerance on the maximum scaled step
C            UU(K) - UU(K-1).
C            If TOL2 is input as 0., then a default value of (uround)
C            to the 2/3 power will be used. uround is the unit
C            roundoff for the machine in use for the calculation.
C
C     Error Indicator
C
C     INFO   (output) INTEGER
C            See Termination Codes below.
C
C     ----------------------------------------------------------------
C
C     Termination Codes
C
C     (Note: in this documentation we use named constants for
C     certain integer constant values. To see the values of these
C     symbols see Named constants below.)
C
C     The termination values KINS_***** are now given. These are the
C     values of the INFO argument.
C
C     SUCCESS :    means maxnorm(FSCALE*KFUN(UU) <= TOL1, where
C                  maxnorm() is the maximum norm function N_VMaxNorm.
C                  Therefore, UU is probably an approximate root of
C                  KFUN.
C
C     INITIAL_GUESS_OK: means the initial guess UU has been found
C                  to already satisfy the system to the desired
C                  accuracy. No calculation was performed other
C                  than testing UU.
C
C     STEP_LT_STPTOL:  means the scaled distance between the last
C                  two steps is less than TOL2.  UU may be an
C                  approximate root of KFUN, but it is also possible
C                  that the algorithm is making very slow progress
C                  and is not near a root or that TOL2 is too
C                  large
C
```

```
C     LNSRCH_NONCONV: means the LineSearch module failed to reduce
C                  norm(KFUN) sufficiently on the last global step.
C                  Either UU is close to a root of F and no more
C                  accuracy is possible, or the finite-difference
C                  approximation to J*v is inaccurate, or TOL2
C                  is too large. Check the outputs NCFL and NNI: if
C                  NCFL is close to NNI, it may be the case that the
C                  Krylov iteration is converging very slowly. In
C                  this case, the user may want to use precondition-
C                  ing and/or increase the MAXL argument (that is,
C                  increase the max dimension of the Krylov subspace)
C                  by setting MAXL to nonzero (thus not using the
C                  default value of KINSPGMR_MAXL) or if MAXL is being
C                  set, increase its value.
C
C     MAXITER_REACHED: means that the maximum allowable number of
C                  nonlinear iterations has been reached. This is by
C                  default 200, but may be changed through optional
C                  input IOPT(MXITER).
C
C     MXNEWT_5X_EXCEEDED: means 5 consecutive steps of length mxnewt
C                  (maximum Newton stepsize limit) have been taken.
C                  Either norm(F) asymptotes from above to a finite
C                  value in some direction, or mxnewt is too small.
C                  Mxnewt is computed internally (by default) as
C                  mxnewt = 1000*max(norm(USCALE*UU0),1), where
C                  UU0 is the initial guess for UU, and norm() is
C                  the Euclidean norm. Mxnewt can be  set by the
C                  user through optional input ROPT(MXNEWTSTEP).
C
C     LINESEARCH_BCFAIL: means that more than the allowed maximum
C                  number of failures (MXNBCF) occurred when trying
C                  to satisfy the beta condition in the linesearch
C                  algorithm. It is likely that the iteration is
C                  making poor progress.
C
C     KRYLOV_FAILURE: means there was a failure of the Krylov
C                  iteration process to converge.
C
C     PRECONDSET_FAILURE: means there was a nonrecoverable
C                  error in PrecondSet causing the iteration to halt.
C
C     PRECONDSOLVE_FAILURE: means there was a nonrecoverable
C               error in PrecondSolve causing the iteration to halt.
C
```

```
C     NO_MEM:    the KINSol memory pointer received was NULL.
C
C
C     INPUT_ERROR: one or more input parameters or arrays was in
C                  error. See the program output for further info.
C
C     LSOLV_NO_MEM: The linear solver memory pointer (lmem) was
C                received as NULL. The return value from the linear
C                solver needs to be checked and the cause found.
C
C     ---------------------------------------------------------------
C
C     Optional inputs and outputs
C
C     (Note: in this documentation we use named constants for
C     certain integer constant values. To see the values of these
C     symbols see Named constants below.)
C
C     The user should declare two arrays for optional input and
C     output, an IOPT array for optional integer input and output
C     and an ROPT array for optional real input and output. These
C     arrays should both be of size OPT_SIZE.
C     So the user's declaration should look like:
C
C     INTEGER          IOPT(OPT_SIZE)
C     DOUBLE PRECISION ROPT(OPT_SIZE)
C
C     The following definitions are indices into the IOPT and ROPT
C     arrays. A brief description of the contents of these positions
C     follows.
C
C     IOPT(PRINTFL)  (input)  Allows user to select from 4 levels
C                    of output.
C                    =0 no statistics printed   (DEFAULT)
C                    =1 output the nonlinear iteration count, the
C                       scaled norm of KFUN(UU), and number of
C                       KFUN calls.
C                    =2 same as 1 with the addition of global
C                       strategy statistics:
C                       f1 = 0.5*norm(FSCALE*KFUN(UU))**2   and
C                       f1new = 0.5*norm(FSCALE*KFUN(unew))**2 .
C                    =3 same as 2 with the addition of further
C                       Krylov iteration statistics.
C
C     IOPT(MXITER)   (input) Maximum allowable number of nonlinear
```

```
C                    iterations. The default is MXITER_DEFAULT.
C
C     IOPT(PRECOND_NO_INIT) (input) Set to 1 to prevent the initial
C                    call to the routine KPRECO upon a given
C                    call to KINSol. Set to 0 or leave unset to
C                    force the initial call to KPRECO.
C                    Use the choice of 1 only after beginning the
C                    first of a series of calls with a 0 value.
C                    If a value other than 0 or 1 is encountered,
C                    the default, 0, is set in this element of
C                    IOPT and thus the routine KPRECO will
C                    be called upon every call to KINSol, unless
C                    IOPT(PRECOND_NO_INIT) is changed by the user.
C
C     IOPT(ETACHOICE) (input) A flag indicating which of three
C                    methods to use for computing eta, the
C                    coefficient in the linear solver
C                    convergence tolerance eps, given by
C                      eps = (eta+u_round)*norm(KFUN(UU)).
C                    Here, all norms are  the scaled L2 norm.
C                    The linear solver attempts to produce a step
C                    p such that norm(KFUN(UU)+J(UU)*p) <= eps.
C                    Two of the methods for computing eta
C                    calculate a value based on the convergence
C                    process in the routine KINForcingTerm.
C                    The third method does not require
C                    calculation; a constant eta is selected.
C
C                    The default if IOPT(ETACHOICE) is  not
C                    specified is ETACHOICE1, (see below).
C
C                    The allowed values (methods)  are:
C                ETACONSTANT  constant eta, default of 0.1 or user
C                   supplied choice, for which see ROPT(ETACONST),
C
C                ETACHOICE1 (default) which uses choice 1 of
C                   Eisenstat and Walker's paper of SIAM J. Sci.
C                   Comput.,17 (1996), pp 16-32 wherein eta is:
C                           eta(k) =
C     ABS( norm(KFUN(UU(k))) - norm(KFUN(UU(k-1))+J(UU(k-1))*p) )
C                        / norm(KFUN(UU(k-1))),
C
C                ETACHOICE2   which uses choice 2 of
C                   Eisenstat and Walker wherein eta is:
C                   eta(k) = egamma *
```

```
C                ( norm(KFUN(UU(k))) / norm(KFUN(u(k-1))) )^ealpha
C
C                    egamma and ealpha for choice 2, both required,
C                    are from either defaults (egamma = 0.9 ,
C                    ealpha = 2)  or from  user input,
C                    see ROPT(ETAALPHA) and ROPT(ETAGAMMA), below.
C
C                    For eta(k) determined by either Choice 1 or
C                    Choice 2, a value eta_safe is determined, and
C                    the safeguard   eta(k) <- max(eta_safe,eta(k))
C                    is applied to prevent eta(k) from becoming too
C                    small too quickly.
C                     For Choice 1,
C                       eta_safe = eta(k-1)^((1.+sqrt(5.))/2.)
C              and    for Choice 2,
C                       eta_safe = egamma*eta(k-1)^ealpha.
C                    (These safeguards are turned off if they drop
C                    below 0.1 . Also, eta is never allowed to be
C                    less than eta_min = 1.e-4).
C
C     IOPT(NO_MIN_EPS) (input) Set to 1 or greater to remove
C                    protection agains eps becoming too small.
C                    This option is useful for debugging linear
C                    and nonlinear solver interactions. Set to 0
C                    for standard eps minimum value testing.
C
C     IOPT(NNI)      (output) Total number of nonlinear iterations.
C
C     IOPT(NFE)      (output) Total number of calls to the user-
C                     supplied system function KFUN.
C
C     IOPT(NBCF)     (output) Total number of times the beta
C                     condition could not be met in the linesearch
C                     algorithm. The nonlinear iteration is halted
C                     if this value ever exceeds MXNBCF (10).
C
C     IOPT(NBKTRK)   (output) Total number of backtracks in the
C                     linesearch algorithm.
C
C     IOPT(SPGMR_NLI) (output) Number of linear iterations.
C
C     IOPT(SPGMR_NPE) (output) Number of preconditioner evaluations.
C
C     IOPT(SPGMR_NPS) (output) Number of calls made to user's psolve
C                     function.
```

```
C
C     IOPT(SPGMR_NCFL) (output) Number of linear convergence failures.
C
C
C     ROPT(MXNEWTSTEP) (input) Maximum allowable length of a Newton
C                      step. The default value is calculated from
C                      1000*max(norm(USCALE*UU(0),norm(USCALE)).
C
C     ROPT(RELFUNC)  (input) Relative error in computing KFUN(UU)
C                      if known. Default is the machine epsilon.
C
C     ROPT(RELU)     (input) A scalar constraint which restricts
C                      the update of UU to  del(UU)/UU < ROPT(RELU)
C                      The default is no constraint on the relative
C                      step in UU.
C
C     ROPT(ETAGAMMA) (input) The coefficient egamma in the eta
C                      computation. See routine KINForcingTerm
C               (SEE IOPT(ETACHOICE) above for additional info).
C
C     ROPT(ETAALPHA) (input) The coefficient ealpha in the eta
C                      computation. See routine KINForcingTerm
C               (SEE IOPT(ETACHOICE) above for additional info).
C
C     ROPT(ETACONST) (input) A user specified constant value for
C                       eta, used in lieu of that computed by
C                           routine KINForcingTerm
C               (SEE IOPT(ETACHOICE) above for additional info).
C
C     ROPT(FNORM)    (output) The scaled norm at a given iteration:
C                      norm(FSCALE(KFUN(UU)).
C
C     ROPT(STEPL)    (output) Last step length in the global
C                      strategy routine:
C                      KINLineSearch or KINInexactNewton.
C
C     ---------------------------------------------------------------
C
C     User-defined routines
C
C     In order to use this routine, some user-defined routines have to
C     be provided. One of them is required, while the others are
C     optional. These routines are described next.
C
C     KFUN    Required
```

```
C
C          SUBROUTINE KFUN (NEQ, UU, FVAL)
C          INTEGER           NEQ
C          DOUBLE PRECISION  UU(NEQ), FVAL(NEQ)
C
C          PURPOSE
C
C          Evaluates the KFUN function which defines the system
C          to be solved:
C                            KFUN(UU)=0
C
C          ARGUMENTS
C
C          NEQ
C          (input) INTEGER
C          Number of equations (and unknowns) in the algebraic
C          system
C
C          UU
C          (input) DOUBLE PRECISION array, dimension (NEQ)
C          independent variable vector
C
C          FVAL
C          (output) DOUBLE PRECISION array, dimension (NEQ)
C          Result of KFUN(UU)
C
C    KPRECO Optional
C
C          SUBROUTINE KPRECO (NEQ, UU, USCALE, FVAL, FSCALE,
C                             VTEMP1, VTEMP2, UROUND, NFE, IER)
C          INTEGER           NEQ, NFE, IER
C          DOUBLE PRECISION  UROUND
C          DOUBLE PRECISION  UU(NEQ), USCALE(NEQ), FVAL(NEQ),
C                            FSCALE(NEQ), VTEMP1(NEQ), VTEMP2(NEQ)
C
C          PURPOSE
C
C          The user-supplied preconditioner setup function KPRECO and
C          the user-supplied preconditioner solve function KPSOL
C          together must define the right preconditoner matrix P
C          chosen so as to provide an easier system for the Krylov
C          solver to solve. KPRECO is called to provide any matrix
C          data required by the subsequent call(s) to KPSOL. The
C          data is expected to be stored in variables within a
C          COMMON block and the definition of those variables is up
```

```
C             to the user. More specifically, the user-supplied
C             preconditioner setup function KPRECO is to evaluate and
C             preprocess any Jacobian-related data needed by the
C             preconditioner solve function KPSOL. This might include
C             forming a crude approximate Jacobian, and performing an
C             LU factorization on the resulting approximation to J.
C             This function will not be called in advance of every call
C             to KPSOL, but instead will be called only as often as
C             necessary to achieve convergence within the Newton
C             iteration in KINSol.  If the KPSOL function needs no
C             preparation, the KPRECO function need not be provided.
C
C             KPRECO should not modify the contents of the arrays
C             UU or FVAL as those arrays are used elsewhere in the
C             iteration process.
C
C             Each call to the KPRECO function is preceded by a call to
C             the system function KFUN. Thus the KPRECO function can use
C             any auxiliary data that is computed by the KFUN function
C             and saved in a way accessible to KPRECO.
C
C             The two scaling arrays, FSCALE and USCALE, and unit
C             roundoff UROUND are provided to the KPRECO function for
C             possible use in approximating Jacobian data, e.g. by
C             difference quotients. These arrays should also not be
C             altered
C
C             ARGUMENTS
C
C             NEQ
C             (input) INTEGER
C             Number of equations (and unknowns) in the algebraic
C             system.
C
C             UU
C             (input) DOUBLE PRECISION array, dimension (NEQ)
C             Independent variable vector.
C
C             USCALE
C             (input) DOUBLE PRECISION array, dimension (NEQ)
C             See USCALE above.
C
C             FVAL
C             (input) DOUBLE PRECISION array, dimension (NEQ)
C             Current value of KFUN(UU).
```

```
C
C           FSCALE
C           (input) DOUBLE PRECISION array, dimension (NEQ)
C           See FSCALE above.
C
C           VTEMP1
C           DOUBLE PRECISION array, dimension (NEQ)
C           Temporary work array.
C
C           VTEMP2
C           DOUBLE PRECISION array, dimension (NEQ)
C           Temporary work array.
C
C           UROUND
C           (input) DOUBLE PRECISION
C           Machine unit roundoff.
C
C           NFE
C           (input/output) INTEGER
C           Number of calls to KFUN made by the package. The KPRECO
C           routine should update this counter by adding on the
C           number of KFUN calls made in order to approximate the
C           Jacobian, if any.  For example, if the routine calls
C           KFUN a total of W times, then the update is
C           NFE = NFE + W.
C
C           IER
C           (output) INTEGER
C           Error indicator.
C           0 if successful,
C           1 if failure, in which case KINSOL stops.
C
C     KPSOL   Optional
C
C           SUBROUTINE KPSOL (NEQ, UU, USCALE, FVAL, FSCALE, VTEM,
C                             FTEM, UROUND, NFE, IER)
C           INTEGER           NEQ, NFE, IER
C           DOUBLE PRECISION  UU(NEQ), USCALE(NEQ), FVAL(NEQ),
C                             FSCALE(NEQ), VTEM(NEQ), FTEM(NEQ)
C
C           PURPOSE
C
C           The user-supplied preconditioner solve function KPSOL
C           is to solve a linear system P x = r in which the matrix
C           P is the (right) preconditioner matrix P.
```

```
C
C           KPSOL should not modify the contents of the iterate
C           array UU  or the current function value array  FVAL as
C           those are used elsewhere in the iteration process.
C
C           ARGUMENTS
C
C           NEQ
C           (input) INTEGER
C           Number of equations (and unknowns) in the algebraic
C           system.
C
C           UU
C           (input) DOUBLE PRECISION array, dimension (NEQ)
C           Independent variable vector.
C
C           USCALE
C           (input) DOUBLE PRECISION array, dimension (NEQ)
C           See USCALE above.
C
C           FVAL
C           (input) DOUBLE PRECISION array, dimension (NEQ)
C           Current value of KFUN(UU).
C
C           FSCALE
C           (input) DOUBLE PRECISION array, dimension (NEQ)
C           See FSCALE above.
C
C           VTEM
C           (input/output) DOUBLE PRECISION array, dimension (NEQ)
C           On entry, holds the RHS vector r.
C           On exit, holds the result x.
C
C           FTEM
C           DOUBLE PRECISION array, dimension (NEQ)
C           Temporary work array.
C
C           UROUND
C           (input) DOUBLE PRECISION
C           Machine unit roundoff.
C
C           NFE
C           (input/output) INTEGER
C           Number of calls to KFUN made by the package. The KPRECO
C           routine should update this counter by adding on the
```

```
C           number of KFUN calls made in order to carry out the
C           solution, if any.  For example, if the routine calls
C           KFUN a total of W times, then the update is
C           NFE = NFE + W.
C
C           IER
C           (output) INTEGER
C           Error indicator.
C           0 if successful,
C           1 if failure, in which case KINSOL stops.
C
C     FATIMES Optional
C
C           SUBROUTINE FATIMES(V, Z, NEWU, UU, IER)
C           INTEGER           NEWU, IER
C           DOUBLE PRECISION  V(:), Z(:), UU(:)
C
C           PURPOSE
C
C           The user-supplied A times V routine (optional) where
C           A is the Jacobian matrix dF/du, or an approximation to
C           it, and V is a given  vector.  This routine computes the
C           product Z = J V.
C
C           ARGUMENTS
C
C           V
C           (input) DOUBLE PRECISION array, dimension (NEQ)
C           Vector to be multiplied by J
C           (preconditioned and unscaled as received).
C
C           Z
C           (output) DOUBLE PRECISION array, dimension (NEQ)
C           Vector resulting from the application of J to V.
C
C           NEW_UU
C           (input) INTEGER
C           Flag indicating whether or not the UU vector has been
C           changed since the last call to this function (0 means
C           FALSE, 1 TRUE).
C           If this function computes and saves Jacobian data, then
C           this computation can be skipped if NEW_UU = FALSE.
C
C           UU
C           (input) DOUBLE PRECISION array, dimension (NEQ)
```

```
C           Current iterate u.
C
C           IER
C           (output) INTEGER
C           Error indicator.
C           0 if successful,
C           1 if failure, in which case KINSOL stops.
C
C     --------------------------------------------------------------
C
C     Named constants
C
C     Here we specify the value of the named integer constants used
C     in this documentation. We use Fortran code for the specification,
C     so that the user can copy and paste these lines in order to
C     use the named constants in his/her programs.
C
CC     KINSOL return values
CC     Note that the value of these constants differ from those of
CC     the KINSOL package. This is due to the adaptation to the
CC     SLICOT standards.
C      INTEGER KINS_NO_MEM, KINS_INPUT_ERROR, KINS_LSOLV_NO_MEM,
C     &        KINS_SUCCESS, KINS_INITIAL_GUESS_OK,KINS_STEP_LT_STPTOL,
C     &        KINS_LNSRCH_NONCONV, KINS_MAXITER_REACHED,
C     &        KINS_MXNEWT_5X_EXCEEDED, KINS_LINESEARCH_BCFAIL,
C     &        KINS_KRYLOV_FAILURE, KINS_PRECONDSET_FAILURE,
C     &        KINS_PRECONDSOLVE_FAILURE}
C
C      PARAMETER(KINS_NO_MEM=101)
C      PARAMETER(KINS_INPUT_ERROR=102)
C      PARAMETER(KINS_LSOLV_NO_MEM=103)
C      PARAMETER(KINS_SUCCESS=0)
C      PARAMETER(KINS_INITIAL_GUESS_OK=2)
C      PARAMETER(KINS_STEP_LT_STPTOL=3)
C      PARAMETER(KINS_LNSRCH_NONCONV=4)
C      PARAMETER(KINS_MAXITER_REACHED=5)
C      PARAMETER(KINS_MXNEWT_5X_EXCEEDED=6)
C      PARAMETER(KINS_LINESEARCH_BCFAIL=7)
C      PARAMETER(KINS_KRYLOV_FAILURE = 8)
C      PARAMETER(KINS_PRECONDSET_FAILURE=9)
C      PARAMETER(KINS_PRECONDSOLVE_FAILURE=10)
C
C      Size of IOPT, ROPT
C      INTEGER OPT_SIZE
C      PARAMETER(OPT_SIZE=40)
```

```
C
CC     IOPT indices
C      INTEGER PRINTFL, MXITER, PRECOND_NO_INIT, NNI ,NFE ,NBCF, NBKTRK,
C     &        ETACHOICE, NO_MIN_EPS
C      INTEGER SPGMR_NLI, SPGMR_NPE, SPGMR_NPS, SPGMR_NCFL
C
C      PARAMETER(PRINTFL=1)
C      PARAMETER(MXITER=2)
C      PARAMETER(PRECOND_NO_INIT=3)
C      PARAMETER(NNI=4)
C      PARAMETER(NFE=5)
C      PARAMETER(NBCF=6)
C      PARAMETER(NBKTRK=7)
C      PARAMETER(ETACHOICE=8)
C      PARAMETER(NO_MIN_EPS=9)
C      PARAMETER(SPGMR_NLI=11)
C      PARAMETER(SPGMR_NPE=12)
C      PARAMETER(SPGMR_NPS=13)
C      PARAMETER(SPGMR_NCFL=14)
C
CC     ROPT indices
C      INTEGER MXNEWTSTEP , RELFUNC , RELU , FNORM , STEPL,
C     &        ETACONST, ETAGAMMA, ETAALPHA
C
C      PARAMETER(MXNEWTSTEP=1)
C      PARAMETER(RELFUNC=2)
C      PARAMETER(RELU=3)
C      PARAMETER(FNORM=4)
C      PARAMETER(STEPL=5)
C      PARAMETER(ETACONST=6)
C      PARAMETER(ETAGAMMA=7)
C      PARAMETER(ETAALPHA=8)
C
CC     Values for IOPT(ETACHOICE)
C      INTEGER ETACHOICE1, ETACHOICE2, ETACONSTANT
C
C      PARAMETER(ETACHOICE1=0)
C      PARAMETER(ETACHOICE2=1)
C      PARAMETER(ETACONSTANT=2)
C
C     ---------------------------------------------------------------
C
C     METHOD
C
C     KINSOL (Krylov Inexact Newton SOLver) is a general purpose
```

```
C     solver for nonlinear systems of equations. Its most notable
C     feature is that it uses Krylov Inexact Newton techniques in the
C     system's approximate solution.
C     The Newton method used results in the solution of linear systems
C     of the form
C                             J(u)*x = b
C     where J(u) is the Jacobian of F at u. The solution of these
C     systems by a Krylov method requires products of the form J(u)*v,
C     which are approximated by a difference quotient of the form
C                         F(u+sigma*v)-F(u)
C                         -----------------
C                              sigma
C     Thus, the Jacobian need not be formed explicitly.
C
C     REFERENCES
C
C     [1] Allan G. Taylor and Alan C. Hindmarsh, "User Documentation
C         for KINSOL, a Nonlinear Solver for Sequential and Parallel
C         Computers", Center for Applied Scientific Computing, L-561,
C         LLNL, Livermore, CA 94551.
C
C     NUMERICAL ASPECTS
C
C     CONTRIBUTOR
C
C     Fernando Alvarruiz, Vicente Hernandez
C     Universidad Politecnica de Valencia, Spain
C
C     REVISIONS
C
C     KEYWORDS
C
C     Newton Method, Krylov methods
C
      IMPLICIT NONE

      LOGICAL OPTIN
      INTEGER GSTRAT, NEQ, MAXL, MAXLRST, MSBPRE, INFO
      INTEGER IOPT(40)
      DOUBLE PRECISION TOL1, TOL2
      DOUBLE PRECISION UU(NEQ), USCALE(NEQ), FSCALE(NEQ),
     &         CONSTR(NEQ), ROPT(40)
      EXTERNAL LINSU

C     Local variables
```

```
      INTEGER IOPTIN

C     Executable statements
      IF (OPTIN) THEN
         IOPTIN = 1
      ELSE
         IOPTIN = 0
      ENDIF

      CALL FSKINMALLOC (NEQ, INFO)

      IF (INFO .NE. 0) THEN
         INFO = 101
         RETURN
      ENDIF

C     Linear solver setup
      CALL LINSU (MAXL, MAXLRST, MSBPRE)

      CALL FKINSOL (NEQ, UU, GSTRAT, USCALE, FSCALE, TOL1,
     &              TOL2, CONSTR, OPTIN, IOPT, ROPT, INFO)

      CALL FKINFREE

C     Transform the return code to SLICOT standards
      IF (INFO .EQ. 1) THEN
         INFO = 0
      ENDIF
      IF (INFO .LT. 0) THEN
         INFO=100-INFO
      ENDIF

      RETURN
C *** Last line of KINSOL ***
      END
```

B Documentation of the MATLAB interface for KINSOL

We include here the source code of the two m-files for the KINSOL interface (`kinsol.m` and `kinsoptions.m`). This is included because the source code contains detailed comments describing the interface.

B.1 The function kinsol

```
% kinsol - Krylov Inexact Newton Solver
%
% function [uu, kinout] = kinsol (kfun, uu0, opts)
%
%     PURPOSE
%
%     To solve a nonlinear system of equations F(u)=0, where F(u) is
%                                    n    n
%     a nonlinear function from R to R , using Krylov Inexact Newton
%     techniques.
%
%     INPUT ARGUMENTS
%
%     kfun    String
%             Name of a matlab .m file that contains a function with
%             the following form:
%
%             function fval = kfun(uu)
%
%             This function evaluates the function F at uu, i.e. F(uu).
%
%             uu
%             Real vector.
%             Independent variable vector.
%
%             fval
%             Real vector.
%             Result of F(uu).
%
%     uu0     Real vector
%             Initial guess.
%
%     opts    Structure of options (type help kinoptions for a description
%             of the options).
%
%     OUTPUT ARGUMENTS
%
%     kinout  Structure of output information about the solution
%             of the problem. The fields of this structure are
%             presented next.
%
%        NonLinIters  (output) Total number of nonlinear iterations.
%
%        NumFuncEvals (output) Total number of calls to the user-
```

```
%                    supplied system function kfun.
%
%       NumBetaCondFail (output) Total number of times the beta
%                    condition could not be met in the linesearch
%                    algorithm. The nonlinear iteration is halted
%                    if this value ever exceeds MXNBCF (10).
%
%       NumBacktracks (output) Total number of backtracks in the
%                    linesearch algorithm.
%
%       Fnorm        (output) The scaled norm at a given iteration:
%                    norm(fscale(kfun(uu)).
%
%       StepLength  (output) Last step length in the global
%                    strategy routine:
%                    KINLineSearch or KINInexactNewton.
%
%       LinIters    (output) Number of linear iterations.
%
%       NumPrecEvals (output) Number of preconditioner evaluations.
%
%       NumPSolve   (output) Number of calls made to user's
%                    PrecondSolveFunc function.
%
%       NumLinConvFails (output) Number of linear convergence failures.
%
%    METHOD
%
%    KINSOL (Krylov Inexact Newton SOLver) is a general purpose
%    solver for nonlinear systems of equations. Its most notable
%    feature is that it uses Krylov Inexact Newton techniques in the
%    system's approximate solution.
%    The Newton method used results in the solution of linear systems
%    of the form
%                            J(u)*x = b
%    where J(u) is the Jacobian of F at u. The solution of these
%    systems by a Krylov method requires products of the form J(u)*v,
%    which are approximated by a difference quotient of the form
%                         F(u+sigma*v)-F(u)
%                         -----------------
%                              sigma
%    Thus, the Jacobian need not be formed explicitly.
%
%    REFERENCES
%
```

```
%      [1] Allan G. Taylor and Alan C. Hindmarsh, "User Documentation
%          for KINSOL, a Nonlinear Solver for Sequential and Parallel
%          Computers", Center for Applied Scientific Computing, L-561,
%          LLNL, Livermore, CA 94551.
%
%      NUMERICAL ASPECTS
%
%      CONTRIBUTOR
%
%      Fernando Alvarruiz, Vicente Hernandez
%      Universidad Politecnica de Valencia, Spain
%
%      REVISIONS
%
%      KEYWORDS
%
%      Newton Method, Krylov methods
%

function [uu, kinout] = kinsol (kfun, uu0, opts)

n = length(uu0);

iopt = zeros(1,40);
ropt = zeros(1,40);

% Set default values for empty fields of opt
if isempty(opts.GlobalStrategy)
    opts.GlobalStrategy = 'in';
end
if isempty(opts.Constr)
    opts.Constr = zeros(1,n);
end
if isempty(opts.Uscale)
    opts.Uscale = ones(1,n);
end
if isempty(opts.Fscale)
    opts.Fscale = ones(1,n);
end
if isempty(opts.FNormTol)
    opts.FNormTol = 0;
end
if isempty(opts.ScStepTol)
    opts.ScStepTol = 0;
end
```

```
if isempty(opts.MaxNewtStep)
    opts.MaxNewtStep = 0;
end
ropt(1)=opts.MaxNewtStep;
if isempty(opts.Verbosity)
    opts.Verbosity = 0;
end
iopt(1)=opts.Verbosity;
if isempty(opts.MaxIter)
    opts.MaxIter = 0;
end
iopt(2)=opts.MaxIter;
if isempty(opts.RelFunc)
    opts.RelFunc = 0;
end
ropt(2)=opts.RelFunc;
if isempty(opts.RelU)
    opts.RelU = 0;
end
ropt(3)=opts.RelU;
if isempty(opts.PrecondSetFunc)
    opts.PrecondSetFunc = '';
end
if isempty(opts.PreconsSolveFunc)
    opts.PreconsSolveFunc = '';
end
if isempty(opts.UserATimesFunc)
    opts.UserATimesFunc = '';
end
if isempty(opts.MaxLinDim)
    opts.MaxLinDim = 0;
end
if isempty(opts.MaxLinRestarts)
    opts.MaxLinRestarts = 0;
end
if isempty(opts.MaxBeforePrecond)
    opts.MaxBeforePrecond = 0;
end
if isempty(opts.PrecondNoInit)
    opts.PrecondNoInit = 0;
end
iopt(3)=opts.PrecondNoInit;
if isempty(opts.EtaChoice)
    opts.EtaChoice = 0;
end
```

```
iopt(8)=opts.EtaChoice;
if isempty(opts.EtaGamma)
    opts.EtaGamma = 0;
end
ropt(7)=opts.EtaGamma;
if isempty(opts.EtaAlpha)
    opts.EtaAlpha = 0;
end
ropt(8)=opts.EtaAlpha;
if isempty(opts.EtaConst)
    opts.EtaConst = 0;
end
ropt(6)=opts.EtaConst;

% Call the MEX-File
[uu, iopt, ropt, info] = kinsol_gtw (opts.GlobalStrategy, kfun,...
    opts.PrecondSetFunc, opts.PreconsSolveFunc, opts.UserATimesFunc,...
    opts.MaxLinDim, opts.MaxLinRestarts, opts.MaxBeforePrecond,...
    uu0, opts.Uscale, opts.Fscale, opts.Constr, iopt, ropt,...
    opts.FNormTol, opts.ScStepTol);

% Form kinout structure
kinout.NonLinIters = iopt(4);
kinout.NumFuncEvals = iopt(5);
kinout.NumBetaCondFail = iopt(6);
kinout.NumBacktracks = iopt(7);
kinout.FNorm = ropt(4);
kinout.StepLength = ropt(5);
kinout.LinIters = iopt(10);
kinout.NumPrecEvals = iopt(11);
kinout.NumPSolve = iopt(12);
kinout.NumLinConvFails = iopt(13);
```

B.2 The function kinsoptions

```
% KINSOL options
%
% function kopt = kinsoptions()
%
% This function creates a default structure of options to be passed to the
% kinsol function.
%
% In these comments, the following notation is used:
% UU - independent variable for the nonlinear function.
% KFUN - nonlinear function.
```

```
% NEQ - number of equations/unknowns.
%
% The members of the options structure are:
%
% GlobalStrategy: String.
%     Indicates the global strategy to apply the computed
%     increment delta in the solution UU.  Choices are:
%     'in' - Inexact Newton (default).
%     'ls' - Linesearch.
%
% Constr: Vector, dimension (NEQ).
%     Constraints on UU.
%     A positive value in Constr(i) implies that the ith
%     component of UU is to be constrained > 0.
%     A negative value in Constr(i) implies that the ith
%     component of UU is to be constrained < 0.
%     A zero value in Constr(i) implies there is no constraint
%     on UU(i).
%
% Uscale: Vector, dimension (NEQ).
%     Array of diagonal elements of the scaling matrix for UU.
%     The elements of Uscale must be positive values. The
%     scaling matrix Uscale should be chosen so that
%     Uscale * UU (as a matrix multiplication) should have all
%     its components with roughly the same magnitude when UU is
%     close to a root of KFUN.
%
% Fscale: Vector, dimension (NEQ).
%     Array of diagonal elements of the scaling matrix for
%     KFUN. The elements of Fscale must be positive values.
%     The scaling matrix Fscale should be chosen so that
%     Fscale * KFUN(UU) (as a matrix multiplication) should
%     have all its components with roughly the same magnitude
%     when UU is NOT too near a root of KFUN.
%
% FNormTol: Real.
%     Stopping tolerance on maxnorm( Fscale * KFUN(UU) ).
%     If FNormTol is input as 0., then a default value of
%     (uround) to the 1/3 power will be used. uround is the
%     unit roundoff for the machine in use for the calculation.
%
% ScStepTol: Real.
%     Stopping tolerance on the maximum scaled step
%     UU(K) - UU(K-1).
%     If ScStepTol is input as 0., then a default value of (uround)
```

```
%     to the 2/3 power will be used. uround is the unit
%     roundoff for the machine in use for the calculation.
%
% MaxNewtStep: Real.
%     Maximum allowable length of a Newton step. The default value
%     is calculated from 1000*max(norm(uscale*UU(0),norm(uscale)).
%
% Verbosity: Integer.
%     Allows user to select from 4 levels of output.
%     =0 no statistics printed (DEFAULT).
%     =1 output the nonlinear iteration count, the scaled norm of
%        KFUN(UU), and number of KFUN calls.
%     =2 same as 1 with the addition of global strategy statistics:
%        f1 = 0.5*norm(Fscale*KFUN(UU))**2   and
%        f1new = 0.5*norm(Fscale*KFUN(unew))**2 .
%     =3 same as 2 with the addition of further Krylov iteration
%        statistics.
%
% MaxIter: Integer.
%     Maximum allowable number of nonlinear iterations. The default
%     is 200.
%
% RelFunc: Real.
%     Relative error in computing KFUN(UU) if known. Default is the
%     machine epsilon.
%
% RelU: Real.
%     A scalar constraint which restricts
%     the update of UU to  del(UU)/UU < RelU.
%     The default is no constraint on the relative
%     step in UU.
%
% PrecondSetFunc: String.
%     Name of an m-file with the definition of the following function
%
%     [NFE, IERR] = KPRECO(UU, USCALE, FVAL, FSCALE, KFUN, UROUND, NFE)
%
%     where the name of the function, here KPRECO, must be the same as the
%     name of the m-file. This function is described next.
%
%     PURPOSE
%
%     The user-supplied preconditioner setup function KPRECO and
%     the user-supplied preconditioner solve function KPSOL (see below
%     together must define the right preconditoner matrix P
```

```
%     chosen so as to provide an easier system for the Krylov
%     solver to solve. KPRECO is called to provide any matrix
%     data required by the subsequent call(s) to KPSOL. The
%     data is expected to be stored in variables within a
%     COMMON block and the definition of those variables is up
%     to the user. More specifically, the user-supplied
%     preconditioner setup function KPRECO is to evaluate and
%     preprocess any Jacobian-related data needed by the
%     preconditioner solve function KPSOL. This might include
%     forming a crude approximate Jacobian, and performing an
%     LU factorization on the resulting approximation to J.
%     This function will not be called in advance of every call
%     to KPSOL, but instead will be called only as often as
%     necessary to achieve convergence within the Newton
%     iteration in KINSol.  If the KPSOL function needs no
%     preparation, the KPRECO function need not be provided.
%
%     Each call to the KPRECO function is preceded by a call to
%     the system function KFUN. Thus the KPRECO function can use
%     any auxiliary data that is computed by the KFUN function
%     and saved in a way accessible to KPRECO.
%
%     The two scaling arrays, FSCALE and USCALE, and unit
%     roundoff UROUND are provided to the KPRECO function for
%     possible use in approximating Jacobian data, e.g. by
%     difference quotients.
%
%     INPUT ARGUMENTS
%
%     UU: Real array, dimension (NEQ).
%     Independent variable vector.
%
%     USCALE: Real array, dimension(NEQ).
%     See Uscale above.
%
%     FVAL: Real array, dimension (NEQ).
%     Current value of KFUN(UU).
%
%     FSCALE: Real array, dimension(NEQ).
%     See FSCALE above.
%
%     KFUN: String.
%     Name of an m-file defining the KFUN function.
%
%     UROUND: Real.
```

```
%     Machine unit roundoff.
%
%     NFE: Integer.
%     Number of calls to KFUN made by the package so far.
%
%     OUTPUT ARGUMENTS
%
%     NFE: Integer.
%     NFE should be set to the result of updating NFE by adding to it the
%     number of KFUN calls made in order to approximate the
%     Jacobian, if any.  For example, if the routine calls
%     KFUN a total of W times, then the update is
%     NFE = NFE + W.
%
%     IER: Integer.
%     Error indicator.
%     0 if successful,
%     1 if failure, in which case KINSOL stops.
%
% PrecondSolveFunc: String.
%     Name of an m-file with the definition of the following function
%
%     [X, NFE, IERR] = KPSOL(UU, USCALE, FVAL, FSCALE, R, KFUN, UROUND, NFE)
%
%     where the name of the function, here KPSOL, must be the same as the
%     name of the m-file. This function is described next.
%
%     PURPOSE
%
%     The user-supplied preconditioner solve function KPSOL
%     is to solve a linear system P x = r in which the matrix
%     P is the (right) preconditioner matrix P.
%
%     ARGUMENTS
%
%     UU: Real array, dimension (NEQ)
%     Independent variable vector.
%
%     USCALE: Real array, dimension(NEQ).
%     See Uscale above.
%
%     FVAL: Real array, dimension (NEQ).
%     Current value of KFUN(UU).
%
%     FSCALE: Real array, dimension(NEQ).
```

```
%     See Fscale above.
%
%     R: Real array, dimension (NEQ).
%     RHS vector r.
%     On exit, holds the result x.
%
%     KFUN: String.
%     Name of an m-file defining the KFUN function.
%
%     UROUND: Real.
%     Machine unit roundoff.
%
%     NFE: Integer.
%     Number of calls to KFUN made by the package so far.
%
%     OUTPUT ARGUMENTS
%
%     X: Real array, dimension (NEQ).
%     The result x.
%
%     NFE: Integer.
%     NFE should be set to the result of updating NFE by adding to it the
%     number of KFUN calls made in order to carry out the solution,
%     if any. For example, if the routine calls KFUN a total of W times,
%     then the update is
%     NFE = NFE + W.
%
%     IER: Integer.
%     Error indicator.
%     0 if successful,
%     1 if failure, in which case KINSOL stops.
%
% UserATimesFunc: String.
%     Name of an m-file with the definition of the following function
%
%     [Z, IERR] = FATIMES(V, NEW_UU, UU)
%
%     where the name of the function, here FATIMES, must be the same as the
%     name of the m-file. This function is described next.
%
%     PURPOSE
%
%     The user-supplied A times v routine (optional) where
%     A is the Jacobian matrix dF/du, or an approximation to
%     it, and v is a given  vector.  This routine computes the
```

```
%     product z = J*v.
%
%     INPUT ARGUMENTS
%
%     V: Real array, dimension (NEQ).
%     Vector to be multiplied by J
%     (preconditioned and unscaled as received).
%
%     NEW_UU: Integer.
%     Flag indicating whether or not the UU vector has been
%     changed since the last call to this function (0 means
%     FALSE, 1 TRUE).
%     If this function computes and saves Jacobian data, then
%     this computation can be skipped if NEW_UU = FALSE.
%
%     UU: Real array, dimension (NEQ).
%     Current iterate u.
%
%     OUTPUT ARGUMENTS
%
%     Z: Real array, dimension (NEQ).
%     Vector resulting from the application of J to v.
%
%     IER: Integer.
%     Error indicator.
%     0 if successful,
%     1 if failure, in which case KINSOL stops.
%
% MaxLinDim: Integer.
%     Maximum Krylov dimension. This is an
%     optional input to the KINSpgmr solver. Pass 0 to
%     use the default value MIN(NEQ, KINSPGMR_MAXL=10).
%
% MaxLinRestarts: Integer.
%     Is the maximum number of linear solver restarts
%     allowed. Values outside the range 0 to 2*NEQ/MaxLinDim
%     will be restricted to that range. 0, meaning no
%     restarts, is a safe starting value.
%
% MaxBeforePrecond: Integer.
%     Is the maximum number of steps calling the solver
%     PrecondSolveFunc without calling the preconditioner
%     PrecondSetFunc (the default is KINSPGMR_MSBPRE = 10).
%
% PrecondNoInit: Integer.
```

```
%     Set to 1 to prevent the initial
%     call to the routine PrecondSetFunc upon a given
%     call to KINSol. Set to 0 or leave unset to
%     force the initial call to PrecondSetFunc.
%     Use the choice of 1 only after beginning the
%     first of a series of calls with a 0 value.
%     If a value other than 0 or 1 is encountered,
%     the default, 0, is set in this option
%     and thus the routine PrecondSetFunc will
%     be called upon every call to KINSol, unless
%     PrecondNoInit is changed by the user.
%
% EtaChoice: Integer.
%     Flag indicating which of three
%     methods to use for computing eta, the
%     coefficient in the linear solver
%     convergence tolerance eps   given by
%     eps = (eta+u_round)*norm(KFUN(UU)).
%     Here, all norms are  the scaled L2 norm.
%     The linear solver attempts to produce a step
%     p such that norm(KFUN(UU)+J(UU)*p) <= eps.
%     Two of the methods for computing eta
%     calculate a value based on the convergence
%     process in the routine KINForcingTerm.
%     The third method does not require
%     calculation; a constant eta is selected.
%
%     The default if EtaChoice is  not
%     specified is ETACHOICE1 (0), (see below).
%
%     The allowed values (methods)  are:
%     ETACONSTANT (2)  constant eta, default of 0.1 or user
%        supplied choice, for which see EtaConst,
%
%     ETACHOICE1 (0) [default] which uses choice 1 of
%        Eisenstat and Walker's paper of SIAM J. Sci.
%        Comput.,17 (1996), pp 16-32 wherein eta is:
%                          eta(k) =
%   ABS( norm(KFUN(UU(k))) - norm(KFUN(UU(k-1))+J(UU(k-1))*p) )
%                        / norm(KFUN(UU(k-1))),
%
%     ETACHOICE2 (1)  which uses choice 2 of
%        Eisenstat and Walker wherein eta is:
%                     eta(k) = egamma *
%         ( norm(KFUN(UU(k))) / norm(KFUN(u(k-1))) )^ealpha
```

```
%
%        egamma and ealpha for choice 2, both required,
%        are from either defaults (egamma = 0.9 ,
%        ealpha = 2)  or from  user input,
%        see EtaAlpha and EtaGamma, below.
%
%     For eta(k) determined by either Choice 1 or
%     Choice 2, a value eta_safe is determined, and
%     the safeguard   eta(k) <- max(eta_safe,eta(k))
%     is applied to prevent eta(k) from becoming too
%     small to quickly.
%     For Choice 1,
%            eta_safe = eta(k-1)^((1.+sqrt(5.))/2.)
%     and for Choice 2,
%              eta_safe = egamma*eta(k-1)^ealpha.
%     (These safeguards are turned off if they drop
%     below 0.1 . Also, eta is never allowed to be
%     less than eta_min = 1.e-4).
%
% EtaGamma: Real.
%     The coefficient egamma in the eta
%     computation. See routine KINForcingTerm
%     (see EtaChoice above for additional info).
%
% EtaAlpha: Real.
%     The coefficient ealpha in the eta
%     computation. See routine KINForcingTerm
%     (see EtaChoice above for additional info).
%
% EtaConst: Real.
%     A user specified constant value for
%     eta, used in lieu of that computed by
%     routine KINForcingTerm
%     (see EtaChoice above for additional info).
%
function kopt = kinsoptions()

 kopt = struct('GlobalStrategy',[],...
    'Constr',[],...
    'Uscale',[],...
    'Fscale',[],...
    'FNormTol',[],...
    'ScStepTol',[],...
    'MaxNewtStep',[],...
    'Verbosity',[],...
```

```
'MaxIter',[],...
'RelFunc',[],...
'RelU',[],...
'PrecondSetFunc',[],...
'PreconsSolveFunc',[],...
'UserATimesFunc',[],...
'MaxLinDim',[],...
'MaxLinRestarts',[],...
'MaxBeforePrecond',[],...
'PrecondNoInit',[],...
'EtaChoice',[],...
'EtaGamma',[],...
'EtaAlpha',[],...
'EtaConst',[]);
```