ADIFOR Working Note #10:

ADIFOR Case Study: VODE + ADIFOR

by

George Corliss

Abstract. ADIFOR can be used to generate the Jacobians required by VODE in a manner that is easy to use. We provide a template to interface the ADIFOR-generated code with VODE and show how the template is used in a sample system of stiff ordinary differential equations. The ADIFOR-generated code is about 10% faster than the hand-coded Jacobian for this example.

1 VODE

VODE (Variable-coefficient Ordinary Differential Equation solver) [4] is a popular solver for stiff ordinary differential equations. The code is available from `netlib`, is well documented, and is widely used.

We were interested in exploring VODE applications to see how ADIFOR (Automatic DIfferentiation in FORtran) [2] would perform in this context.

The user of VODE must provide a subroutine of the form shown in Listing 1.

```
SUBROUTINE FEX (NEQ, T, Y, YDOT, RPAR, IPAR)
DOUBLE PRECISION T, Y, YDOT, RPAR
DIMENSION Y(NEQ), YDOT(NEQ), RPAR(*), IPAR(*)
```

Listing 1. Template for subroutine FEX

which supplies the vector function f by loading YDOT(i) with $f(i)$. The user of VODE must also provide a subroutine for the Jacobian in the form shown in Listing 2.

```
SUBROUTINE JAC (NEQ, T, Y, ML, MU, PD, NROWPD, RPAR, IPAR)
DOUBLE PRECISION T, Y, PD, RPAR
DIMENSION Y(NEQ), PD(NROWPD,NEQ), RPAR(*), IPAR(*)
```

Listing 2. Template for subroutine JAC

2 Using ADIFOR with VODE

In this section, we go through the steps required to use ADIFOR to generate the Jacobians required by VODE. We assume that the reader is familiar with other reports showing how to use ADIFOR [1,2,3], so we give only the VODE-specific information.

The lesson here is that ADIFOR is a very useful tool which can relieve a user of VODE from the task of hand-coding a routine for computing the Jacobian. The user of VODE + ADIFOR can use the makefile and the template for `VODJAC` given here. The following steps are necessary:

1. Write subroutine `FEX` as always.

2. Make adifor.

3. Edit subroutine `VODJAC` to change `NEQMAX`, if necessary.

4. Make driver program as usual.

ADIFOR can generate a subroutine for computing the Jacobian using a dummy main program shown in Listing 3

```
integer NEQ, IPAR
parameter (NEQ = 3)
DOUBLE PRECISION RPAR, T, Y(NEQ), YDOT(NEQ)
call FEX (NEQ, T, Y, YDOT, RPAR, IPAR)
STOP
END
```

Listing 3. Dummy main program required by ADIFOR

and a file.adf of the form shown in Listing 4.

```
TOP fex
PMAX 3   (whatever is the maximum value of NEQ)
IVARS Y
OVARS YDOT
```

Listing 4. File.adf required by ADIFOR

ADIFOR generates a subroutine

```
 subroutine g$fex$c(g$p$, neq, t, y, g$y, ldg$y, ydot, g$ydot, ldg$
*ydot, rpar, ipar)
```

whose full text appears in Section 5. Subroutine `g$fex$c` is called from the template subroutine `VODJAC`. The subroutine `VODJAC` calls the ADIFOR-generated routine `g$fex$c` and returns the Jacobian in the format expected by VODE.

The user of VODE might have to make two changes in `VODJAC`.

1. Replace the value of `NEQMAX` by the maximum value of `NEQ`.

2. Replace the call `g$fex$c` by one of the user-supplied subroutines defining the right-hand side of the ODE.

The locations where these changes should be made are clearly marked in `VODJAC`, which appears in Section 5.

Then VODE is called, as is shown in Listing 5.

```
      EXTERNAL FEX, VODJAC
        . . .
      CALL VODE(FEX,NEQ,Y,T,TOUT,ITOL,RTOL,ATOL,ITASK,ISTATE,
     1          IOPT,RWORK,LRW,IWORK,LIW,VODJAC,MF,RPAR,IPAR)
                                        ======
```

Listing 5. Calling VODE from the main program

In the next section, we give the results. Subsequent sections describe what we did and include the relevant code.

3 Results

We used the example supplied in the VODE internal documentation. It is a simple chemical kinetics model due to Robinson of a polymerization process in a continuously stirred tank reactor or C-star reactor. It consists of the following three rate equations:

```
dy1/dt = -.04*y1 + 1.e4*y2*y3
dy2/dt = .04*y1 - 1.e4*y2*y3 - 3.e7*y2**2
dy3/dt = 3.e7*y2**2
```

on the interval from $t = 0.0$ to $t = 4 \cdot 10^{10}$, with initial conditions $y_1 = 1.0$, $y_2 = y_3 = 0$. The problem is stiff.

Computing the Jacobian by ADIFOR-generated code gave the same answers as the hand-coded Jacobian routine supplied in the example, and it ran 10% faster.

The following is the output from the original sample program, augmented with timing.

```
At t =  0.4000E+00   y =  0.985164E+00  0.338624E-04  0.148020E-01
At t =  0.4000E+01   y =  0.905510E+00  0.224034E-04  0.944679E-01
At t =  0.4000E+02   y =  0.715801E+00  0.918506E-05  0.284189E+00
At t =  0.4000E+03   y =  0.450542E+00  0.322311E-05  0.549455E+00
At t =  0.4000E+04   y =  0.183206E+00  0.894285E-06  0.816793E+00
At t =  0.4000E+05   y =  0.389815E-01  0.162174E-06  0.961018E+00
At t =  0.4000E+06   y =  0.493610E-02  0.198411E-07  0.995064E+00
At t =  0.4000E+07   y =  0.516592E-03  0.206742E-08  0.999483E+00
At t =  0.4000E+08   y =  0.520168E-04  0.208078E-09  0.999948E+00
At t =  0.4000E+09   y =  0.518902E-05  0.207562E-10  0.999995E+00
At t =  0.4000E+10   y =  0.507970E-06  0.203188E-11  0.999999E+00
At t =  0.4000E+11   y =  0.625102E-07  0.250041E-12  0.100000E+01

No. steps = 562   No. f-s = 809   No. J-s =  11   No. LU-s = 109
 No. nonlinear iterations = 806
 No. nonlinear convergence failures =   0
 No. error test failures =  26

 2.60E-01  Elapsed CPU seconds for solution
```

The following is the output from the same program, except that the Jacobian was computed by using ADIFOR-generated code.

```
At t =  0.4000E+00   y =  0.985164E+00  0.338624E-04  0.148020E-01
At t =  0.4000E+01   y =  0.905510E+00  0.224034E-04  0.944679E-01
At t =  0.4000E+02   y =  0.715801E+00  0.918506E-05  0.284189E+00
At t =  0.4000E+03   y =  0.450542E+00  0.322311E-05  0.549455E+00
At t =  0.4000E+04   y =  0.183206E+00  0.894285E-06  0.816793E+00
At t =  0.4000E+05   y =  0.389815E-01  0.162174E-06  0.961018E+00
At t =  0.4000E+06   y =  0.493610E-02  0.198411E-07  0.995064E+00
At t =  0.4000E+07   y =  0.516592E-03  0.206742E-08  0.999483E+00
At t =  0.4000E+08   y =  0.520168E-04  0.208078E-09  0.999948E+00
At t =  0.4000E+09   y =  0.518902E-05  0.207562E-10  0.999995E+00
At t =  0.4000E+10   y =  0.507970E-06  0.203188E-11  0.999999E+00
At t =  0.4000E+11   y =  0.625102E-07  0.250041E-12  0.100000E+01

No. steps = 562   No. f-s = 809   No. J-s =  11   No. LU-s = 109
 No. nonlinear iterations = 806
 No. nonlinear convergence failures =   0
 No. error test failures =  26

 2.50E-01  Elapsed CPU seconds for solution
```

Times are on a SPARC 2. The timer resolution is 1/60 sec, so these results are a draw. Next, we inserted a loop in the driving program to perform the solution 50 times, deleted writing the solution, and divided the total elapsed times by 50 to obtain an average solution time.

Analytic Jacobian:

```
No. steps = 562   No. f-s = 809   No. J-s =  11   No. LU-s = 109
No. nonlinear iterations = 806
No. nonlinear convergence failures =   0
No. error test failures =  26

2.33E-01  Elapsed CPU seconds for solution
```

ADIFOR-generated Jacobian:

```
No. steps = 562   No. f-s = 809   No. J-s =  11   No. LU-s = 109
No. nonlinear iterations = 806
No. nonlinear convergence failures =   0
No. error test failures =  26

2.01E-01  Elapsed CPU seconds for solution
```

VODE uses an internally generated Jacobian (MF = 22):

```
No. steps = 543   No. f-s = 796   No. J-s =  11   No. LU-s = 107
No. nonlinear iterations = 760
No. nonlinear convergence failures =   0
No. error test failures =  24

1.94E-01  Elapsed CPU seconds for solution
```

In short, the VODE solution using ADIFOR-generated derivatives gave exactly the same results and executed about 10% faster than the solution using the hand-coded analytic derivatives.

Of course, this is only one example, and a simple one at that. No one should be impressed until we can report similar results on *real* problems. Nevertheless, this *does* show that ADIFOR is competitive.

The next section outlines what we did to obtain these results.

4 Example Main Program

The main program run to produce both of these timings was taken from the internal VODE documentation shown in Listing 6. As noted above, we modified the program by

1. inserting timing calls,
2. looping to solve the problem 50 times, and
3. commenting out writing of solution values.

```
      program byrne

c  Purpose: Driver to call VODE with ADIFOR-generated Jacobian.
c  Author:  George Corliss, 13-JUL-1992, after VODE documentation.
c
C EXAMPLE PROBLEM
```

```
C
C The following is a simple example problem, with the coding
C needed for its solution by VODE.  The problem is from chemical
C kinetics, and consists of the following three rate equations..
C     dy1/dt = -.04*y1 + 1.e4*y2*y3
C     dy2/dt = .04*y1 - 1.e4*y2*y3 - 3.e7*y2**2
C     dy3/dt = 3.e7*y2**2
C on the interval from t = 0.0 to t = 4.e10, with initial conditions
C y1 = 1.0, y2 = y3 = 0.  The problem is stiff.
C
C The following coding solves this problem with VODE, using MF = 21
C and printing results at t = .4, 4., ..., 4.e10.  It uses
C ITOL = 2 and ATOL much smaller for y2 than y1 or y3 because
C y2 has much smaller values.
C At the end of the run, statistical quantities of interest are
C printed. (See optional output in the full description below.)
C To generate Fortran source code, replace C in column 1 with a blank
C in the coding below.
C
      EXTERNAL FEX, VODJAC
      DOUBLE PRECISION ATOL, RPAR, RTOL, RWORK, T, TOUT, Y
      DIMENSION Y(3), ATOL(3), RWORK(67), IWORK(33)
c GFC modification for timing:
      real    timer, StartTime, ElapsedTime
      StartTime = timer ()
      do 990 iiiiii = 1, 50

      NEQ = 3
      Y(1) = 1.0D0
      Y(2) = 0.0D0
      Y(3) = 0.0D0
      T = 0.0D0
      TOUT = 0.4D0
      ITOL = 2
      RTOL = 1.D-4
      ATOL(1) = 1.D-8
      ATOL(2) = 1.D-14
      ATOL(3) = 1.D-6
      ITASK = 1
      ISTATE = 1
      IOPT = 0
      LRW = 67
      LIW = 33
      MF = 21
      DO 40 IOUT = 1,12
        CALL VODE(FEX,NEQ,Y,T,TOUT,ITOL,RTOL,ATOL,ITASK,ISTATE,
     1            IOPT,RWORK,LRW,IWORK,LIW,VODJAC,MF,RPAR,IPAR)
c        WRITE(6,20)T,Y(1),Y(2),Y(3)
  20    FORMAT(7H At t =,E12.4,6H   y =,3E14.6)
        IF (ISTATE .LT. 0) GO TO 80
  40    TOUT = TOUT*10.
c GFC modification for timing:
 990  continue
      ElapsedTime = timer () - StartTime

      WRITE(6,60) IWORK(11),IWORK(12),IWORK(13),IWORK(19),
     1            IWORK(20),IWORK(21),IWORK(22)
  60  FORMAT(/12H No. steps =,I4,12H   No. f-s =,I4,
     1       12H   No. J-s =,I4,13H   No. LU-s =,I4/
     2       28H  No. nonlinear iterations =,I4/
     3       38H  No. nonlinear convergence failures =,I4/
     4       27H  No. error test failures =,I4/)
c GFC modification for timing:
      write (6, 1050) ElapsedTime / 50.0
```

```
 1050 format (12X, 1pE10.2,
     +            '  Elapsed CPU seconds for solution' / )

      STOP
  80  WRITE(6,90)ISTATE
  90  FORMAT(///22H Error halt.. ISTATE =,I3)
      STOP
      END
```

Listing 6. Example main program

5 ADIFOR-Generated Derivatives

We took the subroutine that defines the RHS of the ODE given in the VODE documentation

```
      SUBROUTINE FEX (NEQ, T, Y, YDOT, RPAR, IPAR)
      DOUBLE PRECISION RPAR, T, Y, YDOT
      DIMENSION Y(NEQ), YDOT(NEQ)
      YDOT(1) = -.04D0*Y(1) + 1.D4*Y(2)*Y(3)
      YDOT(3) = 3.D7*Y(2)*Y(2)
      YDOT(2) = -YDOT(1) - YDOT(3)
      RETURN
      END
```

and applied ADIFOR (see the makefile below). `Y` is independent, and `YDOT` is dependent. ADIFOR generated the following code:

```
      subroutine g$fex$c(g$p$, neq, t, y, g$y, ldg$y, ydot, g$ydot, ldg$
     *ydot, rpar, ipar)
C
C       Formal ydot is active.
C       Formal y is active.
C
        integer g$p$
        integer g$pmax$
        parameter (g$pmax$ = 3)
        integer g$i$
        double precision d$5
        double precision d$4
        double precision d$3
        double precision d$3bar
        double precision d$2
        double precision d$1bar
C
        integer ipar
        integer neq
        double precision rpar, t, y, ydot
        double precision g$y(ldg$y, neq), g$ydot(ldg$ydot, neq)
        dimension y(neq), ydot(neq)
        integer ldg$y
        integer ldg$ydot
        if (g$p$ .gt. g$pmax$) then
          print *, 'Parameter g$p is greater than g$pmax.'
          stop
        endif
C       ydot(1) = -.04d0 * y(1) + 1.d4 * y(2) * y(3)
        d$4 = 1.d4 * y(2)
        d$5 = y(3)
        d$3bar = d$5 * 1.d4
        do 99999 g$i$ = 1, g$p$
```

```
          g$ydot(g$i$, 1) = -.04d0 * g$y(g$i$, 1) + d$3bar * g$y(g$i$, 2
     *) + d$4 * g$y(g$i$, 3)
99999   continue
        ydot(1) = -.04d0 * y(1) + d$4 * d$5
C       ydot(3) = 3.d7 * y(2) * y(2)
        d$2 = 3.d7 * y(2)
        d$3 = y(2)
        d$1bar = d$3 * 3.d7
        do 99998 g$i$ = 1, g$p$
          g$ydot(g$i$, 3) = d$1bar * g$y(g$i$, 2) + d$2 * g$y(g$i$, 2)
99998   continue
        ydot(3) = d$2 * d$3
C       ydot(2) = -ydot(1) - ydot(3)
        do 99997 g$i$ = 1, g$p$
          g$ydot(g$i$, 2) = -g$ydot(g$i$, 1) + (-g$ydot(g$i$, 3))
99997   continue
        ydot(2) = -ydot(1) - ydot(3)
        return
      end
```

Listing 7. ADIFOR-generated `g$fex$c`

This subroutine `g$fex$c` does not have the interface expected by VODE (see subroutine `JEX` in Listing 8).

```
      SUBROUTINE JEX (NEQ, T, Y, ML, MU, PD, NRPD, RPAR, IPAR)
      DOUBLE PRECISION PD, RPAR, T, Y
      DIMENSION Y(NEQ), PD(NRPD,NEQ)

      PD(1,1) = -.04D0
      PD(1,2) = 1.D4*Y(3)
      PD(1,3) = 1.D4*Y(2)
      PD(2,1) = .04D0
      PD(2,3) = -PD(1,3)
      PD(3,2) = 6.E7*Y(2)
      PD(2,2) = -PD(1,2) - PD(3,2)
      RETURN
      END
```

Listing 8. Subroutine `JEX` supplied with VODE

Surprisingly, the ADIFOR-generated code in Listing 7 executes about 10% faster than the code in Listing 8.

We could modify VODE so that it *does* expect the interface of `g$fex$c`, but it is much easier and safer to use the subroutine `JEX` provided and modify it to call the ADIFOR-generated routine. We provide a template subroutine `VODJAC` to serve that function (see Listing 9). Inside `VODJAC`, we need to

1. declare variables,

2. initialize `gy`, the Jacobian of `Y` with respect to itself,

3. call `g$fex$c`, and

4. transform the Jacobian into the format expected by VODE.

```
      SUBROUTINE VODJAC (NEQ, T, Y, ML, MU, PD, NRPD, RPAR, IPAR)

c  Purpose: Template to supply ADIFOR-generated Jacobian to VODE
c  Author:  George Corliss, 13-JUL-1992
c  Usage:
c     VODE is a Variable-coefficient Ordinary Differential Equation
c     solver, with fixed-leading coefficient implementation.
c     The use of VODE must provide a subroutine of the form..
c
c           SUBROUTINE FEX (NEQ, T, Y, YDOT, RPAR, IPAR)
c           DOUBLE PRECISION T, Y, YDOT, RPAR
c           DIMENSION Y(NEQ), YDOT(NEQ), RPAR(*), IPAR(*)
c
c     which supplies the vector function f by loading YDOT(i) with f(i).
c     The use of VODE must also provide a subroutine for the Jacobian in
c     the form of the current subroutine.
c
c     ADIFOR can generate a subroutine for computing the Jacobian using
c     a dummy main program of the form..
c
c           integer NEQ, IPAR
c           parameter (NEQ = 3)
c           DOUBLE PRECISION RPAR, T, Y(NEQ), YDOT(NEQ)
c           call FEX (NEQ, T, Y, YDOT, RPAR, IPAR)
c           STOP
c           END
c
c     and a file.adf of the form..
c
c           TOP fex
c           PMAX 3   (whatever is the maximum value of NEQ)
c           IVARS Y
c           OVARS YDOT
c
c     The subroutine VODJAC calls the ADIFOR-generated routine
c     g$fec$c and returns the Jacobian in the format expected by VODE.
c
c     The user of VODE might have to make two changes in VODJAC..
c     1. Replace the value of NEQMAX by the maximum value of NEQ.
c     2. Replace the call g$fex$c by the one of the user-supplied
c        subroutine defining the right-hand side of the ODE.
c
c     Then VODE is called:
c
c          EXTERNAL FEX, VODJAC
c            . . .
c          CALL VODE(FEX,NEQ,Y,T,TOUT,ITOL,RTOL,ATOL,ITASK,ISTATE,
c       1            IOPT,RWORK,LRW,IWORK,LIW,VODJAC,MF,RPAR,IPAR)
c                                             ======
c
c  References:
c  1. P. N. Brown, G. D. Byrne, and A. C. Hindmarsh, "VODE: A Variable
c     Coefficient ODE Solver," SIAM J. Sci. Stat. Comput., 10 (1989),
c     pp. 1038-1051.  Also, LLNL Report UCRL-98412, June 1988.
c  2. C. Bischof, A. Carle, G. Corliss, A. Griewank, and P. Hovland,
c     "Generating Derivative Codes from Fortran Programs," Scientific
c     Computing, to appear.  Also Argonne NL Preprint MCS-P263-0991,
c     September 1991, and as CRCP Technical Report 91185.
c  3. G. Corliss, "VODE + ADIFOR," Argonne NL Technical Memorandum
c     ANL/MCS-TM-168, July 1992.
c
c--------------------------------------------------------------------

      DOUBLE PRECISION PD, RPAR, T, Y
```

```
      DIMENSION Y(NEQ), PD(NRPD,NEQ)

      integer NEQMAX, i, j

C=================================================================
C
C  USER MODIFICATION MAY BE NEEDED HERE
C
C     Replace NEQMAX by the maximum value of NEQ desired.
C
C                          VV
      parameter (NEQMAX = 10)
      double precision gy(NEQMAX,NEQMAX), ydot(NEQMAX), temp

      if (NEQ .gt. NEQMAX) then
          print *, 'The system dimension NEQ is larger than NEQMAX.'
          stop
      end if
      do 20 j = 1, NEQ
         do 10 i = 1, NEQ
            gy(i,j) = 0.0
 10      continue
         gy(j,j) = 1.0
 20   continue

C=================================================================
C
C  USER MODIFICATION MAY BE NEEDED HERE
C
C     Replace "fex" by the name of the user-supplied subroutine
C     that defines the right-hand side of the differential equation.
C
C             VVV
      call g$fex$c(NEQ, neq, t, y, gy, NEQMAX, ydot, PD, NRPD,
     +             rpar, ipar)

      do 30 i = 1, NEQ
         do 30 j = i+1, NEQ
            temp   = PD(i,j)
            PD(i,j) = PD(j,i)
            PD(j,i) = temp
   30 continue

      RETURN
      END
```

Listing 9. Template to interface ADIFOR-generated Jacobian with VODE

Here is the makefile to control the entire process:

```
#  File:  Adifor/Examples/Vode/Makefile

FFLAGS = -O
AD_TOPLEVEL = fex

ADbyrne : byrne.o vodjac.o fex.c.o fex.o timer.o \
vode.o dgefa.o dgesl.o dgbfa.o dgbsl.o daxpy.o \
dcopy.o ddot.o idamax.o dscal.o
f77 $(FFLAGS) -o ADbyrne byrne.o vodjac.o fex.c.o fex.o timer.o \
vode.o dgefa.o dgesl.o dgbfa.o dgbsl.o daxpy.o \
dcopy.o ddot.o idamax.o dscal.o

adifor : $(AD_TOPLEVEL).adf $(AD_TOPLEVEL).comp
```

```
adifor.new $(AD_TOPLEVEL).adf $(AD_TOPLEVEL).comp
make -f ADMakefile

vodedemo: vodedemo.o vode.o jac1.o jac2.o dgefa.o dgesl.o dgbfa.o \
dgbsl.o daxpy.o dcopy.o ddot.o idamax.o dscal.o
f77 -o vodedemo  vodedemo.o vode.o jac1.o jac2.o dgefa.o \
dgesl.o dgbfa.o dgbsl.o daxpy.o dcopy.o ddot.o idamax.o dscal.o

byrne : byrne.o jacb.o timer.o\
fex.o vode.o dgefa.o dgesl.o dgbfa.o dgbsl.o daxpy.o \
dcopy.o ddot.o idamax.o dscal.o
f77 $(FFLAGS) -o byrne byrne.o jacb.o timer.o\
fex.o vode.o dgefa.o dgesl.o dgbfa.o dgbsl.o daxpy.o \
dcopy.o ddot.o idamax.o dscal.o
```

6 Conclusions

The lesson here is that ADIFOR is a very useful tool which can relieve a user of VODE from the task of hand-coding a routine for computing the Jacobian. This simple test shows that the correct values can sometimes be computed faster than hand-written code. The ADIFOR-generated code almost always beats difference quotient approximations, although it did not do so in this example. Further, the ADIFOR-generated code can take advantage of sparsity in the Jacobian, although we have not illustrated that here.

Acknowledgments

We thank George Byrne for suggestions and the explanation of the example system of ODEs. The ADIFOR project team includes Christian Bischof, Alan Carle, George Corliss, Andreas Griewank, Paul Hovland, and Moe El-Khadiri.

References

[1] Christian Bischof, Alan Carle, George Corliss, Moe El-Khadiri, Paul Hovland, and Andreas Griewank. Getting started with ADIFOR. Technical Memorandum ANL/MCS–TM–164, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1992. ADIFOR Working Note # 9.

[2] Christian Bischof, Alan Carle, George Corliss, Andreas Griewank, and Paul Hovland. Generating derivative codes from Fortran programs. *Scientific Computing*, to appear. ADIFOR Working Note # 1. Also appeared as Preprint MCS–P263–0991, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., September 1991, and as Technical Report 91185, Center for Research in Parallel Computation, Rice University, Houston, Tex., 1991.

[3] Christian Bischof and Paul Hovland. Using ADIFOR to compute dense and sparse Jacobians. Technical Memorandum ANL/MCS–TM–158, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., October 1991. ADIFOR Working Note # 2.

[4] P. N. Brown, George D. Byrne, and Alan C. Hindmarsh. VODE: A variable coefficient ODE solver. *SIAM J. Sci. Stat. Comput.*, 10:1038–1051, 1989. Also appeared as Lawrence Livermore National Laboratory Report UCL-98412, June 1988.