RP

A PACKAGE FOR EFFICIENT CALCULATION OF SPARSE JACOBIAN MATRIX FOR NONLINEAR SYSTEMS OF EQUATIONS USING FINITE DIFFERENCES

Neculai Andrei

Research Institute for Informatics
Bucharest - Romania

ABSTRACT

In this paper we present a Fortran package for Jacobian matrix determination of a given large nonlinear system of equations, in a given point. The package contains two subroutines. The first one determines the pattern of the Jacobian matrix in sparse matrix representation, i.e. the row index of nonzero elements on columns and the point where the columns start. The second one computes the nonzero elements of the Jacobian matrix in a given point. Some numerical examples illustrate the running of the RP package.

Technical Report
Bucharest
April 15, 1983

CONTENTS

1. Introduction

Sparsity plays an important role in the computational methods in large scale nonlinear optimization. Both for solving large-scale nonlinear systems of equations and for large-scale nonlinear constrained optimization, it is necessary to compute the Jacobian of the nonlinear functions defining the equations of the system or the constraints of the optimization problem.

In this paper we confine our attention to the generation of sparse Jacobian of a system of nonlinear equations. We are interested to compute the numerical value of the Jacobian in a given point. This is an important step in implementing the algorithms for nonlinear optimization, or in the solution of nonlinear algebraic systems of equations.

Let us consider the solution of a nonlinear algebraic system of equation

$$F(x) = 0, \tag{1}$$

where $F : \mathbb{R}^n \to \mathbb{R}^m$, and $F(x) = [\, f_1(x) \cdots f_m(x)]^T$, every function $f_i(x)$, $i = 1, \ldots, m$, is assumed to be continuous differentiable on $\mathbb{R}^n$.

For solving this nonlinear system, the Newton method involves the computation of the Jacobian

$$J = \nabla F(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{m \times n}. \tag{2}$$

If n is enough large, then the Jacobian matrix J is usually sparse. The sparsity is useful in reducing the number of function evaluations necessary to estimate J through finite differences.

In this paper we describe the algorithm by Curtis, Powell and Reid [1974] for an approximation of the $m \times n$-Jacobian J of function $F(x)$, by finite differences. The simplest way to approximate J is to use the forward differences:

$$\frac{\partial f_i}{\partial x_j} = \frac{f_i(x + h_j e_j) - f_i(x)}{h_j}, \tag{3}$$

or the central differences:

$$\frac{\partial f_i}{\partial x_j} = \frac{f_i(x + h_j e_j) - f_i(x - h_j e_j)}{2h_j}, \tag{4}$$

where e_j is the column j of the identity matrix and h_j is a suitable step-length (increment). Computing derivatives by differences has the advantage that only the function is needed, but the accuracy of such derivative approximations is hard to assess. The choice of the difference parameter h_j can be a source of difficulties for many problems, in particular if the functions f_i are highly nonlinear or if it is noisy. A small step size h_j is needed to get a suitable approximate of the derivatives, but this may lead to numerical cancellation and the loss of the accuracy (see [Curtis and Reid, 1974]).

A straightforward implementation of (3) involves computing $F(x + h_j e_j)$ for each j, that is n evaluations of function $F(x)$ at displacements from the current point x. However, because

of sparsity, each function $f_i(x)$ depends on only a few components x_j. This can be used to reduce the number of function evaluations.

For the very beginning let us consider the case where the following relations

$$\frac{\partial f_i(x)}{\partial x_j} = 0, \text{ for } |i-j| \geq 2, \tag{5}$$

that is the Jacobian matrix J is tridiagonal. In this case we see that we may compute the approximations to columns 1, 4, 7, … simultaneously by using the displacements $h_1 e_1 + h_4 e_4 + h_7 e_7 + \cdots$. Similarly, the columns 2, 5, 8, … and 3, 6, 9, … can be found together. Therefore, only three function evaluations of F at displacements from the current point x must be considered. This is a great improvement in estimation of the Jacobian matrix.

From this example we see that for more general sparse case we need to find a set of columns of the Jacobian matrix J which are such that no two have entries in the same row. A set of such columns of J can be computed simultaneously with a single function evaluation. However, finding on optimal set of such of columns is an NP-complete problem (see [McCormick, 1983]), but Curtis, Powell and Reid [1974] suggested a heuristic for grouping columns of J for a given sparsity pattern.

The idea of this heuristic is as follows. The first group is formed by starting with the first column of J and then combining with it the next column that has no entries in common with it. The pattern of this next column is merged with the first, and other columns from J are examined in turn seeking columns with no entries in common with the merged pattern. The first group is established when the merged pattern is full, or all columns of J have been examined. The second and the successive groups are formed exactly in the same way as above from the remaining columns.

2. Fortran subroutines

```
************************************************************
*  Main Program for computation of the sparse Jacobian
*  with minimization of the number of function evaluations.
*
*  The structure of the package is as follows:
*  MAIN ------ RP01A --- FUNC
*         ---- RP01B --- FUNC
*
*  MAIN is the main program (this program)
*  RP01A is the subroutine for sparsity pattern determination
*        of the Jacobian.
*  RP01B is the subroutine for computing the approximation
*        of the nonzero elements of the Jacobian.
*  FUNC  is the subroutine with functions we want to compute
*        the Jacobian.
*
*
*                                              Neculai Andrei
*                                                April 1983
************************************************************

          IMPLICIT DOUBLE PRECISION (A-H,O-Z)
          dimension a(100), x(50), f(50)
          dimension h(50), hmax(50), zl(100), z(100)
```

```
          integer indl(100), ipec(100), izl(100)
          logical band

          dimension ajac(3,5), rjac(3,5)

* The output file
          open(unit=2,file='jacoby.out',status='unknown')

* Introduce n (number of variables) and m (number of constraints)
          n=5
          m=3
          ia=100
          npm=n+m
          n2=2*n+1

* Introduce the current point x in which we want to compute an
* approximation of the Jacobian
          x(1)=1.d0
          x(2)=1.d0
          x(3)=2.d0
          x(4)=1.d0
          x(5)=3.d0

* Define the step length corresponding for each variable
          epsmch=1.d0
1         epsmch=epsmch/2.d0
          if(1.d0+epsmch .ne. 1.d0) go to 1
          epsmch=epsmch*2.d0
          do i=1,n
            h(i) = sqrt(epsmch)
          end do

          band=.false.

* The sparsity pattern determination of the Jacobian matrix
          call rp01a(n,m,nz,ia,x,f,h,indl,ipec,band,
     *               mesaj,npm,zl)

          write(2,10)
10        format(/4x,'Pattern of the Jacobian: INDL and IPEC arrays')
          write(2,11)
11        format(4x,'INDL array:')
          write(2,12) (k,indl(k),k=1,nz)
12        format(4x,i4,4x,i4)
          write(2,13)
13        format(4x,'IPEC array:')
          write(2,12) (k,ipec(k),k=1,n)
          write(2,14) nz
14        format(4x,'Number of nonzero elements in Jacobian=',i4)

* Define the upper bounds of the step lengths
          do i=1,n
            hmax(i)=0.01d0
          end do

* Computation of an approximation of the Jacobian in point x
          call rp01b(n,m,nz,x,f,h,hmax,indl,ipec,a,mesaj,
     *               npm,n2,izl,zl,z)

          write(2,24)
24        format(4x,'The numerical values of the Jacobian nonzeros')
```

```
         write(2,25) (i,a(i),i=1,nz)
25       format(4x,i4,3x,f6.2)

         write(2,26) ifix(zl(1))
26       format(/4x,'Number of calls of FUNC =',i4)

* End of the main program in which RP01A and RP01B subroutines
* are used for an approximation of Jacobian.
*-----------------------------------------------------------------


* This part of the main program is for verification of the
* subroutines RP01A and RP01B.
*
* We compare the approximation of the Jacobian matrix given by
* RP01A and RP01B subroutines with the real Jacobian given by
* the subroutine JFUNC in which the elements of the Jacobian are
* analytically computed.
*
* 1) The approximation of Jacobian computed by RO01A and RP01B
*    subroutines is assembled from the arrays: INDL, IPEC and A
*    into the matrix AJAC.
*
* 2) The real Jacobian computed by the JFUNC subroutine is in
*    array RJAC.
*
         do i=1,m
           do j=1,n
             ajac(i,j)=0.d0
           end do
         end do

         ipec(n+1)=nz+1
         do i=1,n
           kp=ipec(i)
           ku=ipec(i+1)-1
           if(ipec(i) .ne. ipec(i+1)) then
             do k=kp,ku
               j=indl(k)
               ajac(j,i)=a(k)
             end do
           end if
         end do

         write(2,30)
30       format(/4x,'Jacobian matrix computed by RP01A & RP01B')
         write(2,31)((ajac(i,j),j=1,5),i=1,3)
31       format(5(4x,f6.2))
*
         write(2,40)
40       format(/4x,'Jacobian matrix analytically computed by JFUNC')
         call jfunc(n,x,m,rjac)
         write(2,31)((rjac(i,j),j=1,5),i=1,3)

         stop
         end
*****************************************************************
```

```
*                                RP01A
*                               =======
*
****************************************************************
C0 Purpose
C  =======
C  This subroutine automatically generates the sparsity pattern
C  of a sparse Jacobian matrix with an arbitrary or band structure.
C  This subroutine must be used in conjunction with RP01B to evaluate
C  an approximation of the Jacobian matrix for given functions in
C  a current point.
C  For more detalils please see:
C  1. J.K. Reid, Fortran subroutines for the solution of sparse
C     systems of nonlinear equations.
C     Report AERE - R.7293, Harwell (1972).
C  2. A.R. Curtis, M.J.D. Powell, J.K. Reid, On the estimation of
C     sparse Jacobian matrices.
C     J. Inst. Math. Applics. vol.13, (1974), pp.117-126.
C  3. N. Andrei, C. Rasturnoiu, Matrice Rare si Aplicatiile lor.
C     Editura Tehnica, Bucuresti, 1983.
C
C1 Argument list
C  =============
C  CALL RP01A(N,M,NZ,IA,X,F,H,INDL,IPEC,BAND,MESAJ,NPM,ZL)
C
C  Inputs:
C  -------
C  N      integer must be set by the user to the number of variables
C         i.e. the number of columns in the Jacobian matrix.
C         It is not altered by RP01A.
C  M      integer must be set by the user to the number of functions
C         i.e. the number of rows in the Jacobian matrix.
C         It is not altered by RP01A.
C  IA     integer must be set by the user to indicate the size of
C         array INDL.
C         It is not altered by RP01A.
C  X      real array of length N. On entry to RP01A it must be set
C         by the user to the current point in which the Jacobian
C         matrix followed to be evaluated.
C         It is not altered by RP01A.
C  H      real array of length N. On entry to RP01A it contains the
C         step lengths to be used when estimating the derivatives by
C         finite difference.
C         It is not altered by RP01A.
C
C  Outputs:
C  --------
C  INDL   integer array of length IA. On exit from RP01A it contains
C         the row number of non-zero derivatives in column order.
C  IPEC   integer array of length N. On exit from RP01A, IPEC(i)
C         contains the point to the start of i-th column of the
C         Jacobian matrix, unless this column is null in which case
C         IPEC(i)=IPEC(i+1).
C  NZ     on exit from RP01A it records the number of non-zero
C         elements of the Jacobian matrix.
C  F      real array of length M. On exit from RP01A it contains the
C         value of functions in the current point X.
C
C  Parameters:
C  -----------
C  BAND   logical variable. If the Jacobian matrix is known to be
```

```
C         band, than BAND is set to .TRUE. and in this case M must
C         be set by the user to the semi-bandwidth of the Jacobian.
C         For an unknown sparsity structure of the Jacobian, BAND
C         is set to .FALSE..
C  MESAJ  integer used to transmit the error diagnostics. A non-
C         negative value on exit indicate success of RP01A. After
C         an unsuccessful entry a mesage is issued in the output
C         file and MESAJ is set negative to indicate:
C         -1  IA is too small --- Error in RP01A.
C  NPM    = N + M. Integer specifying the length of ZL area.
C  ZL     real working area of length NPM.
C
C2 Use of COMMON
C  =============
C  RP01A contains the following COMMON BLOCK:
C  COMMON/RP01C/UMIN,UMAX,U,EPS,AJUSTA
C  (Please see the RP01B Subroutine)
C
C3 Other Subroutines
C  =================
C  The following soubroutine is called by RP01A:
C  FUNC    subroutine written by the user in which the functions
C          are defined.
C
C
C                                                          April 1983
*
*****************************************************************

         subroutine rp01a(n,m,nz,ia,x,f,h,indl,ipec,band,mesaj,
     *                    npm,zl)

         IMPLICIT DOUBLE PRECISION (A-H,O-Z)
         dimension x(n),f(m),zl(npm),h(n)
         integer indl(ia),ipec(ia)
         logical band

         common/rp01c/umin,umax,u,eps,ajusta

         mesaj=0
         k=1
         if(band) go to 2

* The particular structure (pattern) determination of the Jacobian

         call func(n,x,m,f)

         do 1 j=1,n
           xj=x(j)
           x(j)=xj+h(j)
           call func(n,x,m,zl)
           x(j)=xj
           ipec(j)=k
           do 1 i=1,m
             if(zl(i) .eq. f(i)) go to 1
             if(k .gt. ia) go to 5
             indl(k)=i
             k=k+1
1        continue
         go to 4
```

```
* The band pattern determination of the Jacobian

2        call func(n,x,n,f)
         do 3 i=1,n
           xi=x(i)
           x(i)=xi+h(i)
           call func(n,x,n,zl)
           x(i)=xi
           ipec(i)=k
           jp=max0(1,i-m+1)
           ju=min0(n,i+m-1)
           do 3 j=jp,ju
             if(zl(j) .eq. f(j)) go to 3
             if(k .gt. ia) go to 5
             indl(k)=j
             k=k+1
3        continue
4        nz=k-1
         return
5        write(2,6)
6        format(5x,'IA is too small - ERROR in RP01A')
         mesaj=-1
         return
         end
*****************************************************************
```

```
*                              RP01B
*                            =========
*
*****************************************************************
C0 Purpose
C  =======
C  This subroutine uses the sparsity pattern given by RP01A to
C  evaluate an approximation to the sparse Jacobian matrix of a set
C  of non-linear functions, using the finite differences and taking
C  advantage of derivatives which are known to be zero and
C  minimizing the number of evaluations of the functions.
C  The steps used are adjusted automatically within bounds set by
C  the user.
C  For more details please see:
C  1. N. Andrei, C. Rasturnoiu, Matrice Rare si Aplicatiile lor.
C     Editura Tehnica, Bucuresti, 1983.
C
C1 Argument list
C  =============
C
C  CALL RP01B(N,M,NZ,X,F,H,HMAX,INDL,IPEC,A,MESAJ,NPM,N2,IZL,ZL,Z)
C
C  Inputs:
C  -------
C  N      integer must be set by the user to the number of variables
C         i.e. the number of columns in the Jacobian matrix.
C         It is not altered by RP01B.
C  M      integer must be set by the user to the number of functions
C         i.e. the number of rows in the Jacobian matrix.
C         It is not altered by RP01B.
C  NZ     integer must be set by the user, or given by RP01A
C         subroutine to the number of non-zero elements of the
```

```
C         Jacobian matrix.
C         It is not altered by RP01B.
C  X      real array of length N. On entry to RP01B it must be set by
C         the user to the current point in which the Jacobian follows
C         to be evaluated.
C         It is not altered by RP01B.
C  F      real array of length M. On entry to RP01B it must be set by
C         the user to the value of functions in the current point X.
C         It is not altered by RP01B.
C  H      real array of length N. On entry to RP01B it contains the
C         step lengths to be used when estimating the derivatives by
C         finite-difference.
C         It is not altered by RP01B.
C  HMAX   real array of length N. On entry to RP01B it contains the
C         upper bounds for the step lengths. If HMAX(1)<0, then all
C         bounds are taken as modulus of HMAX(1) and the dimension
C         may be one. Lower bounds for the step lengths are taken as
C         EPS times the corresponding upper bound, where EPS is the
C         relative machine accuracy.
C  INDL   integer array of length NZ. On entry to and exit from RP01B
C         it must be set to contains the row number of non-zero
C         derivatives in column order, given by the RP01A subroutine.
C  IPEC   integer array of length N. On entry to and exit from RP01B
C         IPEC(I) contains the point to the start of the I-th column
C         of the Jacobian matrix, unless this column is null in which
C         case IPEC(I)=IPEC(I+1).
C
C  Outputs:
C  --------
C  A      real array of length NZ. On exit from RP01B it contains the
C         non-zero derivatives of the Jacobian matrix in column order
C         according to INDL and IPEC arrays.
C
C  Parameters:
C  -----------
C  MESAJ  integer to transmit the error diagnostics. A non-negative
C         value on entry to RP01B determines the running of the
C         subroutine.
C         In a string of subroutines if a previous subroutine (i.e.
C         RP01A) gives a negative value for MESAJ, than RP01B is
C         suppressed.
C  MPN    = N + M. Integer specifying the length of ZL array.
C  N2     = 2*N + 1. Integer specifying the length of IZL array.
C  IZL    integer working array of length N2.
C  ZL     real working array of length NPM. On exit to RP01B, ZL(1)
C         records the number of calls of FUNC made, and ZL(2) records
C         the number of groups used.
C  Z      real working array of length N.
C
C2 Use of COMMON
C  =============
C  RP01B contains the following COMMON BLOCK:
C  COMMON/RP01C/UMIN,UMAX,U,EPS,AJUSTA
C         Each step-length is adjusted so that, the greatest ratio of
C         roundoff error to truncation error for a column ofg the
C         Jacobian is near U and within the range (UMIN,UMAX).
C  EPS    is the relative accuracy of the floating-point computation.
C  AJUSTA logical variable that allows the user by setting AJUSTA to
C         .FALSE. to economise the number function calls made if he
C         does not want any estimation of errors or adjustments of
C         step sizes using the one-side difference approximation
```

```
C         formula.
C         If ADJUSTA is set to .TRUE. then the derivatives are
C         estimated by the centred difference formula.
C
C3 Other Subroutines
C  =================
C  The following soubroutine is called by RP01A:
C  FUNC    subroutine written by the user in which the functions
C          are defined.
C
C
C                                                          April 1983
C
*****************************************************************


         subroutine rp01b(n,m,nz,x,f,h,hmax,indl,ipec,a,mesaj,
     *                    npm,n2,izl,zl,z)

         IMPLICIT DOUBLE PRECISION (A-H,O-Z)
         dimension a(nz),x(n),f(m),h(n),hmax(n),z(n),zl(npm)
         integer indl(nz),ipec(n),izl(n2)
         logical ajusta,repeta

         common/rp01c/umin,umax,u,eps,ajusta

         if(mesaj .lt. 0) go to 31
         nf=0
         ic=1
         n1=n+1

* izl(j) points to the first entry of group j.
* izl(n1+j), j=1,2,... are the column numbers for group 1,
* than group 2, etc.

         do 1 j=1,n
           izl(j+1)=0
1        zl(m+j)=0.d0
         do 6 nc=1,n1

* zl(m+j) is 0.d0 if column j has not been included in a group yet.
* zl(i), i=1,2,...,m holds the boolean pattern of the union of the
*     columns so far included in group nc.

           izl(nc)=ic
           do 2 i=1,m
             zl(i)=0.d0
2          continue
           do 5 j=1,n
             if(zl(j+m) .ne. 0.d0) go to 5
             kp=ipec(j)
             ku=nz
             if(j .lt. n) ku=ipec(j+1)-1
             if(ku .lt. kp) go to 5
             do 3 k=kp,ku
               i=indl(k)
               if(zl(i) .ne. 0.d0) go to 5
3            continue
             do 4 k=kp,ku
               i=indl(k)
               zl(i)=1.d0
```

```
4            continue
             izl(n1+ic)=j
             ic=ic+1
             zl(j+m)=1.d0
5          continue
           if(ic .eq. izl(nc)) go to 7
6        continue
7        do 8 j=1,n
           hm=-hmax(1)
           if(hm .lt. 0.d0) hm=hmax(j)
           xj=dabs(x(j))
           t1=dmax1(eps*xj, dmin1(h(j),hm), eps*hm)
           if(ajusta) h(j)=(xj+t1)-xj
           zl(j+m)=x(j)
8        continue

*  z(j) holds maximum of estimated ratio of truncation error
*  to roundoff error in column j, j=1,2,...,n.

9        do 10 j=1,n
           z(j)=1.d0
10       continue

*  Find the initial approximate derivatives.

         do 16 ng=1,n
           kp=izl(ng)
           ku=izl(ng+1)-1
           if(ku .lt. kp) go to 17
           do 11 k=kp,ku
             j=izl(n1+k)
             if(h(j) .gt. 0.d0) go to 12
11         continue
           go to 16
12         do 13 k=kp,ku
             j=izl(n1+k)
             h(j)=dabs(h(j))
13         x(j)=x(j)+h(j)
           call func(n,x,m,zl)
           nf=nf+1
           do 15 k=kp,ku
             j=izl(n1+k)
             x(j)=zl(j+m)
             jp=ipec(j)
             ju=nz
             if(j .lt. n) ju=ipec(j+1)-1
             if(ju .lt. jp) go to 15
             do 14 jj=jp,ju
               i=indl(jj)
14           a(jj)=(zl(i)-f(i))/h(j)
15         continue
16       continue

*  Estimate errors and improved step-lengths.

17       if(.not. ajusta) go to 30
         do 23 ng=1,n
           kp=izl(ng)
           ku=izl(ng+1)-1
           if(ku .lt. kp) go to 24
           do 18 k=kp,ku
```

```
              j=izl(n1+k)
              if(h(j) .gt. 0.d0) go to 19
18          continue
            go to 23
19          do 20 k=kp,ku
              j=izl(n1+k)
20          x(j)=x(j)-h(j)
            call func(n,x,m,zl)
            nf=nf+1
            do 22 k=kp,ku
              j=izl(n1+k)
              x(j)=zl(j+m)
              jp=ipec(j)
              ju=nz
              if(j .lt. n) ju=ipec(j+1)-1
              if(ju .lt. jp) go to 22
              do 21 jj=jp,ju
                i=indl(jj)
                der=(f(i)-zl(i))/h(j)
                t1=dabs(zl(i))+dabs(a(jj)*h(j)+f(i))
                t2=dmax1(dabs(a(jj)), dabs(der))*(dabs(x(j))+h(j))
                r0=eps*(0.5d0*t1+t2)/h(j)
                a(jj)=(a(jj)+der)*0.5d0
                tr=der-a(jj)
                if(r0 .eq. 0.d0) r0=1.d0
                z(j)=dmax1(z(j), dabs(tr/r0))
21            continue
22          continue
23        continue

* Find improved steps and decide whether further sweep is needed.

24        repeta=.false.
          do 28 j=1,n
            if(h(j) .lt. 0.d0) go to 28
            hj=h(j)
            xj=dabs(x(j))
            hm=-hmax(1)
            if(hm .lt. 0.d0) hm=hmax(j)
            if(z(j) .lt. umin) go to 26
            t1=h(j)*sqrt(u/z(j))
            h(j)=(dmax1(t1, eps*hm, eps*xj)+xj)-xj
            if(z(j) .gt. umax) go to 27
25          h(j)=-h(j)
            go to 28
26          t1=h(j)*sqrt(u/z(j))
            h(j)=(dmin1(t1, hm)+xj)-xj
27          if(dabs(hj/h(j)-1.d0) .le. 0.001d0) go to 25
            repeta=.true.
28        continue
          if(repeta) go to 9
          do 29 j=1,n
29        h(j)=dabs(h(j))
30        zl(1)=nf
          zl(2)=nc-1
          return
31        write(2,32)
32        format(4x,'Error in RP01B because the previously',
     *             'entry gave error')
          return
          end
```

```
*************************************************************

      BLOCK DATA
      common/rp01c/umin,umax,u,eps,ajusta
      logical ajusta
      data umin,umax,u,eps,ajusta/
     *     10.d0, 1000.d0, 100.d0, 0.00000000001d0,.false./
      end
*************************************************************



      subroutine func(n,x,m,f)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      dimension x(n), f(m)

      f(1)=2.d0*x(1) + x(2) + x(3)**2 + x(5)
      f(2)=x(2) + x(3)**2 + x(4)**2
      f(3)=x(1) + 3.d0*x(4)**2 + x(5)

      return
      end
*************************************************************



      subroutine jfunc(n,x,m,rjac)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      dimension x(n), rjac(m,n)

      rjac(1,1)=2.d0
      rjac(1,2)=1.d0
      rjac(1,3)=2.d0*x(3)
      rjac(1,4)=0.d0
      rjac(1,5)=1.d0

      rjac(2,1)=0.d0
      rjac(2,2)=1.d0
      rjac(2,3)=2.d0*x(3)
      rjac(2,4)=2.d0*x(4)
      rjac(2,5)=0.d0

      rjac(3,1)=1.d0
      rjac(3,2)=0.d0
      rjac(3,3)=0.d0
      rjac(3,4)=6.d0*x(4)
      rjac(3,5)=1.d0

      return
      end
```

3. Numerical examples

In the following we shall present some numerical examples which illustrate the running of the RP01A and RP01B subroutines.

Example 1 (please see the program JACOBY.FOR)
Let us consider the functions:

$$f_1(x) = 2x_1 + x_2 + x_3^2 + x_5,$$
$$f_2(x) = x_2 + x_3^2 + x_4^2,$$
$$f_3(x) = x_1 + 3x_4^2 + x_5.$$

In this example $m = 3$ and $n = 5$. We compute an approximation of the Jacobian of these functions in point $x = [1 \quad 1 \quad 2 \quad 1 \quad 3]$. The results given by the RP01A and RP01B subroutines are as follows:

```
Pattern of the Jacobian: INDL and IPEC arrays
INDL array:
   1       1
   2       3
   3       1
   4       2
   5       1
   6       2
   7       2
   8       3
   9       1
  10       3
IPEC array:
   1       1
   2       3
   3       5
   4       7
   5       9

Number of nonzero elements in Jacobian=  10

The numerical values of the Jacobian nonzeros
   1     2.00
   2     1.00
   3     1.00
   4     1.00
   5     4.00
   6     4.00
   7     2.00
   8     6.00
   9     1.00
  10     1.00

Number of calls of FUNC =   5
```

```
Jacobian matrix computed by RP01A & RP01B
  2.00      1.00      4.00      0.00      1.00
  0.00      1.00      4.00      2.00      0.00
  1.00      0.00      0.00      6.00      1.00

Jacobian matrix analytically computed by JFUNC
  2.00      1.00      4.00      0.00      1.00
  0.00      1.00      4.00      2.00      0.00
  1.00      0.00      0.00      6.00      1.00
```

Example 2 (please see the program JACOBY1.FOR)
Consider the following functions:

$$
\begin{aligned}
f_1(x) &= 2x_1 + 3x_2^2 + x_3 + x_5 + x_6, \\
f_2(x) &= x_1 + x_2^3 + x_6^2, \\
f_3(x) &= x_2 + \sin(x_3) + \exp(x_5) + x_4, \\
f_4(x) &= x_1 - x_3 + \cos(x_4).
\end{aligned}
$$

In this example $m = 4$ and $n = 6$. The point in which the approximation of the Jacobian of these function is computed is $x = [1 \quad 1 \quad 2 \quad 1 \quad 3 \quad 5]$. The results of the RRP01A and RP01B subroutines are as follows:

```
Pattern of the Jacobian: INDL and IPEC arrays
INDL array:
   1       1
   2       2
   3       4
   4       1
   5       2
   6       3
   7       1
   8       3
   9       4
  10       3
  11       4
  12       1
  13       3
  14       1
  15       2
IPEC array:
   1       1
   2       4
   3       7
   4      10
   5      12
   6      14
Number of nonzero elements in Jacobian=  15
The numerical values of the Jacobian nonzeros
   1     2.00
   2     1.00
   3     1.00
   4     6.00
   5     3.00
```

```
   6     1.00
   7     1.00
   8    -0.42
   9    -1.00
  10     1.00
  11    -0.84
  12     1.00
  13    20.09
  14     1.00
  15    10.00

Number of calls of FUNC =   5

Jacobian matrix computed by RP01A & RP01B
 2.00     6.00     1.00     0.00     1.00     1.00
 1.00     3.00     0.00     0.00     0.00    10.00
 0.00     1.00    -0.42     1.00    20.09     0.00
 1.00     0.00    -1.00    -0.84     0.00     0.00

Jacobian matrix analytically computed by JFUNC
 2.00     6.00     1.00     0.00     1.00     1.00
 1.00     3.00     0.00     0.00     0.00    10.00
 0.00     1.00    -0.42     1.00    20.09     0.00
 1.00     0.00    -1.00    -0.84     0.00     0.00
```

Example 3 (please see the program JACOBY2.FOR)
Consider the following functions:

$$
\begin{aligned}
f_1(x) &= x_1 x_2 - 1, \\
f_2(x) &= x_1 x_2 x_3 x_4 - 1, \\
f_3(x) &= x_1 x_2 x_3 x_4 x_5 x_6 - 1, \\
f_4(x) &= x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 - 1, \\
f_5(x) &= x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} - 1.
\end{aligned}
$$

In this example $m = 5$ and $n = 10$. The point in which we approximate the Jacobian is $x = [1\ 1\ 2\ 1\ 3\ 5\ 2\ 1\ 3\ 1]$. The results issued by RP01A and RP01B subroutines are as follows:

```
Pattern of the Jacobian: INDL and IPEC arrays
INDL array:
   1       1
   2       2
   3       3
   4       4
   5       5
   6       1
   7       2
   8       3
   9       4
  10       5
  11       2
  12       3
```

```
  13       4
  14       5
  15       2
  16       3
  17       4
  18       5
  19       3
  20       4
  21       5
  22       3
  23       4
  24       5
  25       4
  26       5
  27       4
  28       5
  29       5
  30       5
IPEC array:
   1       1
   2       6
   3      11
   4      15
   5      19
   6      22
   7      25
   8      27
   9      29
  10      30
Number of nonzero elements in Jacobian=  30
The numerical values of the Jacobian nonzeros
   1     1.00
   2     2.00
   3    30.00
   4    60.00
   5   180.00
   6     1.00
   7     2.00
   8    30.00
   9    60.00
  10   180.00
  11     1.00
  12    15.00
  13    30.00
  14    90.00
  15     2.00
  16    30.00
  17    60.00
  18   180.00
  19    10.00
  20    20.00
  21    60.00
  22     6.00
  23    12.00
  24    36.00
  25    30.00
```

```
   26    90.00
   27    60.00
   28   180.00
   29    60.00
   30   180.00

  Number of calls of FUNC =  10

  Jacobian matrix computed by RP01A & RP01B
  1.00    1.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
  2.00    2.00    1.00    2.00    0.00    0.00    0.00    0.00    0.00    0.00
 30.00   30.00   15.00   30.00   10.00    6.00    0.00    0.00    0.00    0.00
 60.00   60.00   30.00   60.00   20.00   12.00   30.00   60.00    0.00    0.00
180.00  180.00   90.00  180.00   60.00   36.00   90.00  180.00   60.00  180.00

  Jacobian matrix analytically computed by JFUNC
  1.00    1.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
  2.00    2.00    1.00    2.00    0.00    0.00    0.00    0.00    0.00    0.00
 30.00   30.00   15.00   30.00   10.00    6.00    0.00    0.00    0.00    0.00
 60.00   60.00   30.00   60.00   20.00   12.00   30.00   60.00    0.00    0.00
180.00  180.00   90.00  180.00   60.00   36.00   90.00  180.00   60.00  180.00
```

Example 4 (please see the program JACOBY3.FOR) (null column in Jacobian)
Consider the following functions:

$$f_1(x) = 2x_1 + x_2 + x_3^2 + x_5,$$
$$f_2(x) = x_2 + x_3^2 + x_5^2,$$
$$f_3(x) = x_1 + 3x_2^2 + x_5.$$

In this case $m = 3$ and $n = 5$. Observe that the elements of column 4 of the Jacobian are all zero. The point in which the Jacobian is computed is $x = [1\ 1\ 2\ 1\ 3]$. The results given by RP01A and RP01B subroutines are as follows:

```
  Pattern of the Jacobian: INDL and IPEC arrays
  INDL array:
     1       1
     2       3
     3       1
     4       2
     5       3
     6       1
     7       2
     8       1
     9       2
    10       3
  IPEC array:
     1       1
     2       3
     3       6
     4       8
     5       8
  Number of nonzero elements in Jacobian=  10
  The numerical values of the Jacobian nonzeros
     1     2.00
     2     1.00
```

```
   3     1.00
   4     1.00
   5     6.00
   6     4.00
   7     4.00
   8     1.00
   9     6.00
  10     1.00

Number of calls of FUNC =   4

Jacobian matrix computed by RP01A & RP01B
  2.00      1.00      4.00      0.00      1.00
  0.00      1.00      4.00      0.00      6.00
  1.00      6.00      0.00      0.00      1.00

Jacobian matrix analytically computed by JFUNC
  2.00      1.00      4.00      0.00      1.00
  0.00      1.00      4.00      0.00      6.00
  1.00      6.00      0.00      0.00      1.00
```

Example 5 (please see the program JACOBY4.FOR) (two null columns in Jacobian)
Now, let us consider the following functions:

$$f_1(x) = x_1 + x_2^3 + x_4^2 + 2x_5 + x_7^3 + x_8,$$
$$f_2(x) = 2x_1^2 + 3x_2 + x_4 + x_5 + 3x_7 + 2x_8,$$
$$f_3(x) = 3x_1^3 + 2x_2^2 + x_4 + x_7,$$
$$f_4(x) = 3x_2 + x_5 + x_8.$$

In this case $m = 4$ and $n = 8$. Observe that the Jacobian of this system has two zero columns. Considering the point $x = [1\ 1\ 2\ 1\ 3\ 2\ 4\ 1]$, then the RP01A and RP01B subroutines give the following results:

```
Pattern of the Jacobian: INDL and IPEC arrays
INDL array:
   1       1
   2       2
   3       3
   4       1
   5       2
   6       3
   7       4
   8       1
   9       2
  10       3
  11       1
  12       2
  13       4
  14       1
  15       2
  16       3
  17       1
  18       2
```

```
  19       4
IPEC array:
   1       1
   2       4
   3       8
   4       8
   5      11
   6      14
   7      14
   8      17
Number of nonzero elements in Jacobian=  19
The numerical values of the Jacobian nonzeros
   1     1.00
   2     4.00
   3     9.00
   4     3.00
   5     3.00
   6     4.00
   7     3.00
   8     2.00
   9     1.00
  10     1.00
  11     2.00
  12     1.00
  13     1.00
  14    48.00
  15     3.00
  16     1.00
  17     1.00
  18     2.00
  19     1.00

Number of calls of FUNC =   6

Jacobian matrix computed by RP01A & RP01B
1.00    3.00    0.00    2.00    2.00    0.00   48.00    1.00
4.00    3.00    0.00    1.00    1.00    0.00    3.00    2.00
9.00    4.00    0.00    1.00    0.00    0.00    1.00    0.00
0.00    3.00    0.00    0.00    1.00    0.00    0.00    1.00

Jacobian matrix analytically computed by JFUNC
1.00    3.00    0.00    2.00    2.00    0.00   48.00    1.00
4.00    3.00    0.00    1.00    1.00    0.00    3.00    2.00
9.00    4.00    0.00    1.00    0.00    0.00    1.00    0.00
0.00    3.00    0.00    0.00    1.00    0.00    0.00    1.00
```

Example 6 (please see JACOBY5.FOR)
Let us consider the following system of functions:

$$
\begin{aligned}
f_1(x) &= 3x_1 + \sin(x_2) + 2\cos(x_3) - x_4 + x_5, \\
f_2(x) &= x_1^3 - \exp(x_2 + 2x_3) + x_4 - x_6\cos(x_7) - x_8\sin(x_7), \\
f_3(x) &= x_2 + x_7^3 - 1, \\
f_4(x) &= \sin(x_3) + \frac{3}{x_4 + 1} + x_3 x_8^2,
\end{aligned}
$$

$$f_5 = 3x_1^3 - 6x_2 + \cos^2(x_3) - \sin(x_4 + x_8).$$

In this case $m = 5$ and $n = 8$. Consider the point: $x = [1\ 1\ 2\ 1\ 3\ 2\ 1\ 4]$. The RP01A and RP01B subroutines issued the following results:

```
Pattern of the Jacobian: INDL and IPEC arrays
INDL array:
   1       1
   2       2
   3       5
   4       1
   5       2
   6       3
   7       5
   8       1
   9       2
  10       4
  11       5
  12       1
  13       2
  14       4
  15       5
  16       1
  17       2
  18       2
  19       3
  20       2
  21       4
  22       5
IPEC array:
   1       1
   2       4
   3       8
   4      12
   5      16
   6      17
   7      18
   8      20
Number of nonzero elements in Jacobian=  22
The numerical values of the Jacobian nonzeros
   1         3.00
   2         3.00
   3         9.00
   4         0.54
   5      -148.41
   6         1.00
   7        -6.00
   8        -1.82
   9      -296.83
  10        15.58
  11         0.76
  12        -1.00
  13         1.00
  14        -0.75
```

```
    15         -0.28
    16          1.00
    17         -0.54
    18         -0.48
    19          3.00
    20         -0.84
    21         16.00
    22         -0.28

  Number of calls of FUNC =   7

Jacobian matrix computed by RP01A & RP01B
    3.00        0.54       -1.82       -1.00        1.00        0.00        0.00        0.00
    3.00     -148.41     -296.83        1.00        0.00       -0.54       -0.48       -0.84
    0.00        1.00        0.00        0.00        0.00        0.00        3.00        0.00
    0.00        0.00       15.58       -0.75        0.00        0.00        0.00       16.00
    9.00       -6.00        0.76       -0.28        0.00        0.00        0.00       -0.28

Jacobian matrix analytically computed by JFUNC
    3.00        0.54       -1.82       -1.00        1.00        0.00        0.00        0.00
    3.00     -148.41     -296.83        1.00        0.00       -0.54       -0.48       -0.84
    0.00        1.00        0.00        0.00        0.00        0.00        3.00        0.00
    0.00        0.00       15.58       -0.75        0.00        0.00        0.00       16.00
    9.00       -6.00        0.76       -0.28        0.00        0.00        0.00       -0.28
```

Example 7 (please see JACOBY6.FOR)
Consider the following system of functions:

$$
\begin{aligned}
&f_1(x) = 2x_1 + x_2,\\
&f_i(x) = x_{i-1} + 2x_i + x_{i+1}, \quad i = 1,\ldots,n-1,\\
&f_n = x_{n-1} + 2x_n.
\end{aligned}
$$

Observe that the Jacobian associated to these function is an $n \times n$-tridiagonal constant matrix. For $n = m = 8$, in point $x = [1 \ldots 1] \in \mathbb{R}^8$, RP01A and RP01B subroutines produce the following results:

```
  Pattern of the Jacobian: INDL and IPEC arrays
  INDL array:
     1        1
     2        2
     3        1
     4        2
     5        3
     6        2
     7        3
     8        4
     9        3
    10        4
    11        5
    12        4
    13        5
    14        6
    15        5
    16        6
    17        7
    18        6
    19        7
    20        8
```

```
  21       7
  22       8
IPEC array:
   1       1
   2       3
   3       6
   4       9
   5      12
   6      15
   7      18
   8      21
Number of nonzero elements in Jacobian=  22
The numerical values of the Jacobian nonzeros
   1     2.00
   2     1.00
   3     1.00
   4     2.00
   5     1.00
   6     1.00
   7     2.00
   8     1.00
   9     1.00
  10     2.00
  11     1.00
  12     1.00
  13     2.00
  14     1.00
  15     1.00
  16     2.00
  17     1.00
  18     1.00
  19     2.00
  20     1.00
  21     1.00
  22     2.00

Number of calls of FUNC =   3

Jacobian matrix computed by RP01A & RP01B
2.00    1.00    0.00    0.00    0.00    0.00    0.00    0.00
1.00    2.00    1.00    0.00    0.00    0.00    0.00    0.00
0.00    1.00    2.00    1.00    0.00    0.00    0.00    0.00
0.00    0.00    1.00    2.00    1.00    0.00    0.00    0.00
0.00    0.00    0.00    1.00    2.00    1.00    0.00    0.00
0.00    0.00    0.00    0.00    1.00    2.00    1.00    0.00
0.00    0.00    0.00    0.00    0.00    1.00    2.00    1.00
0.00    0.00    0.00    0.00    0.00    0.00    1.00    2.00

Jacobian matrix analytically computed by JFUNC
2.00    1.00    0.00    0.00    0.00    0.00    0.00    0.00
1.00    2.00    1.00    0.00    0.00    0.00    0.00    0.00
0.00    1.00    2.00    1.00    0.00    0.00    0.00    0.00
0.00    0.00    1.00    2.00    1.00    0.00    0.00    0.00
0.00    0.00    0.00    1.00    2.00    1.00    0.00    0.00
0.00    0.00    0.00    0.00    1.00    2.00    1.00    0.00
0.00    0.00    0.00    0.00    0.00    1.00    2.00    1.00
0.00    0.00    0.00    0.00    0.00    0.00    1.00    2.00
```

Example 8 (please see JACOBY7.FOR)
Let us consider the following system of functions:

$$f_1(x) = \frac{x_1 x_3}{2.6058 x_2} - x_4,$$

$$f_2(x) = \frac{400 x_1 x_4^3}{178370 x_3} - x_5,$$

$$f_3(x) = \frac{2}{x_3 + x_4 + 2x_2} - x_7,$$

$$f_4(x) = x_7 \left(0.5(x_1 + x_3) + x_2\right) - x_6,$$

$$f_5(x) = x_1 + x_2 + x_5 - \frac{1}{x_7},$$

$$f_6 = -28837 x_1 - 139009 x_2 - 78213 x_3 + 18927 x_4 + 8427 x_5 + \frac{13492 - 10690 x_6}{x_7},$$

$$f_7(x) = x_1 + x_2 + x_3 + x_4 + x_5 - 1.$$

In this case $m = 7$ and $n = 7$. Consider the point $x = [0.0022, 0.0075, 0.0001, 1, 3, 2, 1]$. The RP01A and RP01B subroutines give the following estimation of the Jacobian in the above point.

```
Pattern of the Jacobian: INDL and IPEC arrays
INDL array:
   1       1
   2       2
   3       4
   4       5
   5       6
   6       7
   7       1
   8       4
   9       5
  10       6
  11       7
  12       1
  13       2
  14       3
  15       4
  16       6
  17       7
  18       1
  19       2
  20       3
  21       6
  22       7
  23       2
  24       3
  25       5
  26       6
  27       7
  28       4
  29       6
  30       3
  31       4
  32       5
  33       6
```

```
 IPEC array:
    1       1
    2       7
    3      12
    4      18
    5      23
    6      28
    7      30
 Number of nonzero elements in Jacobian=  33
 The numerical values of the Jacobian nonzeros
    1          0.01
    2         22.43
    3          0.50
    4          1.00
    5     -28837.01
    6          1.00
    7          0.00
    8          1.00
    9          1.00
   10    -139009.02
   11          1.00
   12          0.11
   13       -493.35
   14         -0.04
   15          0.50
   16     -78213.01
   17          1.00
   18         -1.00
   19          0.15
   20         -0.04
   21      18926.99
   22          1.00
   23         -1.00
   24         -0.08
   25          1.00
   26       8426.99
   27          1.00
   28         -1.00
   29     -10690.00
   30         -1.00
   31          0.01
   32          1.00
   33       7888.00

 Number of calls of FUNC =   7

 Jacobian matrix computed by RP01A & RP01B
    0.01       0.00       0.11      -1.00       0.00       0.00       0.00
   22.43       0.00    -493.35       0.15      -1.00       0.00       0.00
    0.00       0.00      -0.04      -0.04      -0.08       0.00      -1.00
    0.50       1.00       0.50       0.00       0.00      -1.00       0.01
    1.00       1.00       0.00       0.00       1.00       0.00       1.00
-28837.01  -139009.02  -78213.01   18926.99    8426.99  -10690.00    7888.00
    1.00       1.00       1.00       1.00       1.00       0.00       0.00

 Jacobian matrix analytically computed by JFUNC
    0.01       0.00       0.11      -1.00       0.00       0.00       0.00
   22.43       0.00    -493.36       0.15      -1.00       0.00       0.00
    0.00       0.00      -0.04      -0.04      -0.08       0.00      -1.00
    0.50       1.00       0.50       0.00       0.00      -1.00       0.01
    1.00       1.00       0.00       0.00       1.00       0.00       1.00
-28837.00  -139009.00  -78213.00   18927.00    8427.00  -10690.00    7888.00
    1.00       1.00       1.00       1.00       1.00       0.00       0.00
```

It is interesting to see the Frobenius norm of the difference $J_e - J_a$ matrix, where J_e is an estimation of the Jacobian computed by means of RP01A and RP01B, and J_a is the value of the Jacobian analytically computed. For example 8, $\left|J_e - J_a\right|_F = 0.2749415468629\text{E-}01$. On the other hand, for example 7, $\left|J_e - J_a\right|_F = 0.4054841211547\text{E-}05$.

A conclusion is clear.

Computing the derivatives using finite differences sometimes produce acceptable approximations. However, they may lead to poor results in the presence of bad scaling or non-typical starting point. Although no procedure is guaranteed for every function and every point, the method presented in this technical report can lead to substantial performance of the associated minimization algorithms or nonlinear systems of equations algorithms solvers.

It is worth emphasizing that RP package (RP01A and RP01B subroutines) is complicated by tacking into account the sparsity of the system of function $F(x)$. An extremely simple procedure for approximation of Jacobian may be described in the following Fortran program, which uses the forward or central finite difference formulae, applied for **example 8** above.

```
* Program NEWJACOBY7.FOR

************************************************************
*  Main Program for the approximation of the Jacobian using
*  forward or central difference formulae.
*
*  The method is very simple and uses the finite difference
*  formulae for approximation of the Jacobian of a given set
*  of functions.
*  The sparsity of the Jacobian is not considered.
*
*  Example 8
*
*                                                Neculai Andrei
*                                                  April 1983
************************************************************

         IMPLICIT DOUBLE PRECISION (A-H,O-Z)
         dimension x(50), f(50), fi(50), fip(50),fin(50)
         logical forward, central

         dimension ajac(7,7), rjac(7,7)

* The output file
         open(unit=2,file='newjacoby.out',status='unknown')

* Introduce n (number of variables) and m (number of constraints)
         n=7
         m=7

* Select the formula for difference computation
         forward=.true.
         central=.false.

* Introduce the current point x in which we want to compute an
* approximation of the Jacobian
         x(1)=0.0022d0
```

```
         x(2)=0.0075d0
         x(3)=0.0001d0
         x(4)=1.d0
         x(5)=3.d0
         x(6)=2.d0
         x(7)=1.d0

* Define the step length h for modification of variables
         epsmch=1.d0
1        epsmch=epsmch/2.d0
         if(1.d0+epsmch .ne. 1.d0) go to 1
         epsmch=epsmch*2.d0
         h = sqrt(epsmch)

* Approximation of the Jacobian - forward difference formula
         if(forward) then
           write(2,10)
10         format(/4x,'Forward difference formula')
           call func(n,x,m,f)
           do j=1,m
             do i=1,n
               xi=x(i)
               x(i)=x(i)+h
               call func(n,x,m,fi)
               x(i)=xi
               ajac(j,i)=(fi(j)-f(j))/h
             end do
           end do
         end if

* Approximation of the Jacobian - central difference formula
         if(central)then
           write(2,20)
20         format(/4x,'Central difference formula')
           do j=1,m
             do i=1,n
               xi=x(i)
               x(i)=x(i)+h
               call func(n,x,m,fip)
               x(i)=x(i)-2.d0*h
               call func(n,x,m,fin)
               x(i)=xi
               ajac(j,i)=(fip(j)-fin(j))/(2.d0*h)
             end do
           end do
         end if
*-----------------------------------------------------------------

         write(2,30)
30       format(/4x,'Approximation of Jacobian matrix')
         write(2,31)((ajac(i,j),j=1,7),i=1,7)
31       format(7(2x,f10.2))
*

* Compute the Jacobian using its analytical expression
         call jfunc(n,x,m,rjac)
         write(2,40)
40       format(/4x,'Jacobian matrix analytically computed by JFUNC')
         call jfunc(n,x,m,rjac)
         write(2,31)((rjac(i,j),j=1,7),i=1,7)
```

```
* Frobenius norm of ||ajac-rjac||
         fnorm=0.d0
         do i=1,m
           do j=1,n
             fnorm = fnorm + (ajac(i,j)-rjac(i,j))**2
           end do
         end do
         fnorm=sqrt(fnorm)
         write(2,50) fnorm
50       format(/4x,'Frobenius norm of ||ajac-rjac||=',e20.13)

         stop
         end
*****************************************************************



c     This subroutine computes the function and Jacobian matrix of
c     the - chemical equilibrium resulting from partial oxidation
c     of methane with oxygen - problem.
c
c       n is an integer variable.
c         On entry n is the number of variables. n = 7.
c
      subroutine func(n,x,m,fx)
      real*8 x(n), fx(m)

      do i=2,3
        if (x(i).eq.0.0d0) then
          write(*,*) 'x(2), x(3) = 0.'
          return
        endif
      enddo
      if (x(7).eq.0.0d0) then
        write(*,*) 'x(7) = 0.'
        return
      endif

        fx(1) = x(1)*x(3)/(2.6058d0*x(2))-x(4)
        fx(2) = 400.0d0*x(1)*x(4)**3/(1.7837d5*x(3))-x(5)
        fx(3) = 2.0d0/(x(3)+x(4)+2.0d0*x(5))-x(7)
        fx(4) = x(7)*(0.5d0*(x(1)+x(3))+x(2))-x(6)
        fx(5) = x(1)+x(2)+x(5)-1.0d0/x(7)
        fx(6) = -28837.0d0*x(1)-139009.0d0*x(2)-78213.0d0*x(3)
     $            +18927.0d0*x(4)+8427.0d0*x(5)
     $            +(13492.0d0-10690.0d0*x(6))/x(7)
        fx(7) = x(1)+x(2)+x(3)+x(4)+x(5)-1.0d0

      return
      end
*-------------------------------------------------------------

      subroutine jfunc(n,x,m,rjac)
      real*8 x(n), rjac(m,n)

      do i=1,7
        do j=1,7
          rjac(i,j) = 0.0d0
        end do
      end do
```

```
        rjac(1,1) = x(3)/(2.6058d0*x(2))
        rjac(1,3) = x(1)/(2.6058d0*x(2))
        rjac(1,2) = x(1)*x(3)/(2.6058d0*x(2))**2*2.6058d0
        rjac(1,4) = -1.0d0
        rjac(2,1) = 400.0d0*x(4)**3/(1.7837d5*x(3))
        rjac(2,4) = 400.0d0*x(1)*3.0d0*x(4)**2/(1.7837d5*x(3))
        rjac(2,3) = -400.0d0*x(1)*x(4)**3/(1.7837d5*x(3))**2*1.7837d5
        rjac(2,5) = -1.0d0
        rjac(3,3) = -2.0d0/(x(3)+x(4)+2.0d0*x(5))**2
        rjac(3,4) = -2.0d0/(x(3)+x(4)+2.0d0*x(5))**2
        rjac(3,5) = -4.0d0/(x(3)+x(4)+2.0d0*x(5))**2
        rjac(3,7) = -1.0d0
        rjac(4,7) = 0.5d0*(x(1)+x(3))+x(2)
        rjac(4,1) = 0.5d0*x(7)
        rjac(4,3) = 0.5d0*x(7)
        rjac(4,2) = x(7)
        rjac(4,6) = -1.0d0
        rjac(5,1) = 1.0d0
        rjac(5,2) = 1.0d0
        rjac(5,5) = 1.0d0
        rjac(5,7) = 1.0d0/x(7)**2
        rjac(6,1) = -28837.0d0
        rjac(6,2) = -139009.0d0
        rjac(6,3) = -78213.0d0
        rjac(6,4) = 18927.0d0
        rjac(6,5) = 8427.0d0
        rjac(6,6) = -10690.0d0/x(7)
        rjac(6,7) = -(13492.0d0-10690.0d0*x(6))/x(7)**2
        do i=1,5
          rjac(7,i) = 1.0d0
        end do

      return
      end
********************************************** Last line
```

The results of this program is as follows:

```
 Forward difference formula

 Approximation of Jacobian matrix
     0.01        0.00        0.11       -1.00        0.00        0.00        0.00
    22.43        0.00     -493.35        0.15       -1.00        0.00        0.00
     0.00        0.00       -0.04       -0.04       -0.08        0.00       -1.00
     0.50        1.00        0.50        0.00        0.00       -1.00        0.01
     1.00        1.00        0.00        0.00        1.00        0.00        1.00
-28837.01  -139009.02   -78213.01    18926.99     8426.99   -10690.00     7888.00
     1.00        1.00        1.00        1.00        1.00        0.00        0.00

 Jacobian matrix analytically computed by JFUNC
     0.01        0.00        0.11       -1.00        0.00        0.00        0.00
    22.43        0.00     -493.36        0.15       -1.00        0.00        0.00
     0.00        0.00       -0.04       -0.04       -0.08        0.00       -1.00
     0.50        1.00        0.50        0.00        0.00       -1.00        0.01
     1.00        1.00        0.00        0.00        1.00        0.00        1.00
-28837.00  -139009.00   -78213.00    18927.00     8427.00   -10690.00     7888.00
     1.00        1.00        1.00        1.00        1.00        0.00        0.00

 Frobenius norm of ||ajac-rjac||= 0.2749415468629E-01
```

```
 Central difference formula

 Approximation of Jacobian matrix
     0.01        0.00        0.11       -1.00        0.00        0.00        0.00
    22.43        0.00     -493.36        0.15       -1.00        0.00        0.00
     0.00        0.00       -0.04       -0.04       -0.08        0.00       -1.00
     0.50        1.00        0.50        0.00        0.00       -1.00        0.01
     1.00        1.00        0.00        0.00        1.00        0.00        1.00
-28837.00  -139009.01   -78213.00    18927.01     8427.00   -10690.00     7888.01
     1.00        1.00        1.00        1.00        1.00        0.00        0.00

 Jacobian matrix analytically computed by JFUNC
     0.01        0.00        0.11       -1.00        0.00        0.00        0.00
    22.43        0.00     -493.36        0.15       -1.00        0.00        0.00
     0.00        0.00       -0.04       -0.04       -0.08        0.00       -1.00
     0.50        1.00        0.50        0.00        0.00       -1.00        0.01
     1.00        1.00        0.00        0.00        1.00        0.00        1.00
-28837.00  -139009.00   -78213.00    18927.00     8427.00   -10690.00     7888.00
     1.00        1.00        1.00        1.00        1.00        0.00        0.00

 Frobenius norm of ||ajac-rjac||= 0.1364421568137E-01
```

Observe that the Frobenius norm of the difference $J_e - J_a$ matrix is exactly the same given by RP package. The advantage of RP package is that of reducing the number of function call. For large-scale nonlinear algebraic systems this is an important point which must be taken into consideration. Otherwise, the above NEWJACOBY program is enough good.

References

Curtis, A.R., Reid, J.K., (1974) *The choice of step lengths when using differences to approximate Jacobian matrices.* J. Inst. Math. Applics., 13, 1974, pp.121-126.

Curtis, A.R., Powell, M.J.D., Reid, J.K., (1974) *On the estimation of sparse Jacobian matrices.* J. Inst. Maths. Applics., 13, 1974, pp.117-120.

McCormick, S.T., (1983) *Optimal approximation of sparse Hessians and its equivalence to a graph coloring problem.* Mathematical Programming, 26, 1983, pp. 153-171.

Powell, M.J.D., Toint, Ph,L., (1979) *On the estimation of sparse Hessian matrices.* SIAM J. Numer. Anal., 16, December 1979, pp.1060-1074.

ICI-Bucharest, Mathematical Models for Informatic Systems Laboratory

♦

----------oooooOooooo----------