stichting
mathematisch
centrum

MC

AFDELING NUMERIEKE WISKUNDE NW 94/80 NOVEMBER
(DEPARTMENT OF NUMERICAL MATHEMATICS)

J. KOK

AN ALGOL 68 ROUTINE FOR THE APPROXIMATION OF PARTIAL DERIVATIVES ON A TWO-DIMENSIONAL GRID

kruislaan 413 1098 SJ amsterdam

Printed at the Mathematical Centre, 413 Kruislaan, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

1980 Mathematics subject classification: 65-04, 65M20

An ALGOL 68 routine for the approximation of partial derivatives on a two-dimensional grid*)

by

J. Kok

ABSTRACT

The documentation is given of an ALGOL 68 routine. This routine computes weights matrices for the finite difference approximation of first and second order partial derivatives on a specified, two-dimensional grid.

KEY WORDS & PHRASES: *Partial differential equations, Finite difference methods, Software*

---

*) This paper consists of the proposal for a contribution to the NAG-ALGOL 68 library.

CONTENTS

DOCUMENTATION OF THE ALGOL68 ROUTINE D03ZZB

1. Purpose

The routine d03zzb computes weights matrices for approximating first and second order partial derivatives of 2-dimensional functions using finite difference formulas on a specified, not necessarily rectangular grid.

IMPORTANT : before using this routine, read the appropriate implementation document to check implementation-dependent details.

2. Specification (Algol 68)

```
MODE VEC = REF [ ] REAL ;

MODE MAT = REF [ , ] REAL ;

MODE WMAT = REF [ , ] MAT ;

MODE POINT = STRUCT ( REAL xc, yc);

MODE TRIO = STRUCT ( POINT p00, p10, p01);

MODE DEFGRID = STRUCT (

      UNION ( REF [,] POINT , PROC ( INT , INT ) POINT , TRIO ) gr,

      REF [ ] INT ex, ey );

MODE DSCRR = STRUCT (

         BOOL uniform, INT numgp,

         REF [ , ] INT position,

         REF [ , ] POINT grid,

         WMAT mastor, snstor,

         REF FILE dataf, REF [ , ] INT cposmas, cpossn

       );

MODE DISCARR = REF DSCRR ;

PROC d03zzb = (DEFGRID dg, STRING lfn, PROC INT available,

                 NAGFAIL fail) DISCARR :
```

3. Description

a) Statement of the problem.

The routine d03zzb computes weights matrices which can be used for the finite difference approximation of two-dimensional space derivatives appearing in partial differential equations (PDEs) and in accessory boundary conditions. By use of these weights the PDE can be transformed to a system of algebraic equations or a system of ODEs (see ref. [2]).
To this aim the domain of the solution of the PDE must be replaced by a grid consisting of rows of grid points in two directions.
The domain D is a two-dimensional connected set, and its boundary dD consists of one or more closed curves.

b) Properties of the grid

A grid R must be imposed on the domain D in such a way, that R can be mapped on a rectangular grid (or: each grid point lies on exactly one "horizontal" and one "vertical", but not necessarily straight, grid line).
R is allowed to be non-uniform, which means that the elementary quadrangles formed by 4 grid points may possess any shape and that they need not to be congruent. The grid may have any number of holes provided that it does not consist of boundary points only.
(A grid is defined to be uniform, if all elementary quadrangles are congruent and equally oriented parallelograms).
The boundary dD is replaced by the boundary of R. In this way the boundary dR of R consists of one or more closed polygons.

c) Method

The particular method used for the discretization of space derivatives is described in [1]. In summary, a derivative at a certain point is approximated using a general 9 point discretization.
Let x and y be the space variables, and let u = u(x, y) be a given function. For the approximation of the derivatives ux, uy, uxx, uxy and uyy at an interior point, weights are delivered using the nine points of the 3 * 3 - subgrid with this point as its center.
For the approximation of the derivatives ux and uy at a boundary grid point, weights are delivered for each 3 * 3 - subgrid with an interior grid point as its center and containing the boundary point. Thus, derivatives can be approximated using several sets of grid points.

4. References

[1] DEKKER, K.
Semi-discretization methods for partial differential equations on non-rectangular grids.
Int. J. Num. Math. Engng, Vol. 15, pp. 405 - 419, 1980.

[2] KOK, J.
A package for the solution of initial-boundary value problems on a two-dimensional domain.
Mathematisch Centrum, Amsterdam (to appear).

5. Parameters

dg - a DEFGRID value.
The STRUCT dg serves to define the grid. Its components have to be used to define the set of all grid points and to indicate the subset of boundary grid points.
dg is unchanged on exit.
Its components have the following meaning:

gr - a UNION(REF[,]POINT, PROC(INT, INT)POINT, TRIO) value,
defines the coordinates of all grid points.
Let x(k,r), y(k,r) denote the x- and y-coordinates of the [k, r]-th grid point. The coordinates of all grid points can be given in 3 ways:
1) in an array [kmin : kmax, rmin : rmax] POINT gr, where gr[k, r] contains (x(k,r), y(k,r)),
2) by a routine PROC gr = (INT k, r) POINT : <unit>, where gr(k, r) delivers (x(k,r), y(k,r)),
3) in case that the grid is <u>uniform</u> by a TRIO of POINTs:

```
TRIO( (x(kmin, rmin), y(kmin, rmin)),
      (x(kmin+1, rmin), y(kmin+1, rmin)),
      (x(kmin, rmin+1), y(kmin, rmin+1))).
```

(These three POINTs should be the points with subscripts [kmin, rmin], [kmin+1, rmin] and [kmin, rmin+1], respectively. They define one elementary parallelogram, and by translation the whole uniform grid.)

ex - a REF[ ]INT array variable,
contains the k-indices (first subscripts) of consecutive end points of the grid lines forming the boundary polygon(s) (see description of ey).

ey - a REF[ ]INT array variable,
contains the r-indices (second subscripts) of consecutive end points of the grid lines forming the boundary polygon(s) (in the same order as in ex).
A pair (ex[i], ey[i]) contains the pair of subscripts of a corner of the boundary polygon.
The sequence of pairs (k, r) is such that one or more closed polygons are formed along grid lines. The polygon closes when a new pair equals the first pair of the polygon, the following pair (if any) begins another polygon. Except for this first point a polygon may not intersect itself or another polygon.

Additional description.

It is not necessary that the bounds of a given array of grid points are equal to the minimum and maximum of the indices given in the arrays ex OF dg and ey OF dg. Actually, the lower bounds of the array of grid points are allowed to be less, the upper bounds are allowed to be greater than the corresponding minima and maxima.
When a PROC or a TRIO is used for the definition of the grid point coordinates, the index bounds kmin, kmax, rmin and rmax will be the minima and maxima of the values given in ex OF dg and ey OF dg.

Examples:

1) The definition of a full rectangle with straight equidistant grid lines with subscript bounds [kmin : kmax, rmin : rmax] (fig. 1.A) :
The grid point coordinates are delivered by a PROC :

```
   DEFGRID dg =
# gr #( (INT k, r)POINT : (k * delta, r * delta),
# ex #  HEAP[1:5]INT := (kmin, kmax, kmax, kmin, kmin),
# ey #  HEAP[1:5]INT := (rmin, rmin, rmax, rmax, rmin)
      ) .
```

2) A better way for the definition of this particular grid is by defining the grid point coordinates by a TRIO, since in that case the grid is recognized to be uniform, thus allowing more efficient computation and storing of the weights matrices.
A possible definition is :

```
PROC grd = ( INT k, r) POINT : (k * delta, r * delta);

DEFGRID dg =
   ( TRIO (grd(kmin, rmin), grd(kmin+1, rmin),
           grd(kmin, rmin+1) ),
        HEAP[1 : 5]INT := (kmin, kmax, kmax, kmin, kmin),
        HEAP[1 : 5]INT := (rmin, rmin, rmax, rmax, rmin)
      ) .
```

3) A grid over the semi-ring
   [y >= 0, 0 < r1 <= sqrt(x**2 + y**2) <= r2]
   with an equidistant subdivision of [r1, r2] and also of every arc (see fig. 1.B), is defined by:

```
   DEFGRID dg =
      ( (INT i, j)POINT :
        (  REAL arc = pi * j / m,
                r   = r1 + i * (r2 - r1) / n;
           ( r * cos(arc), r * sin(arc) )
        ),
        HEAP[1 : 5]INT := (0, n, n, 0, 0),
        HEAP[1 : 5]INT := (0, 0, m, m, 0)
      ) .
```

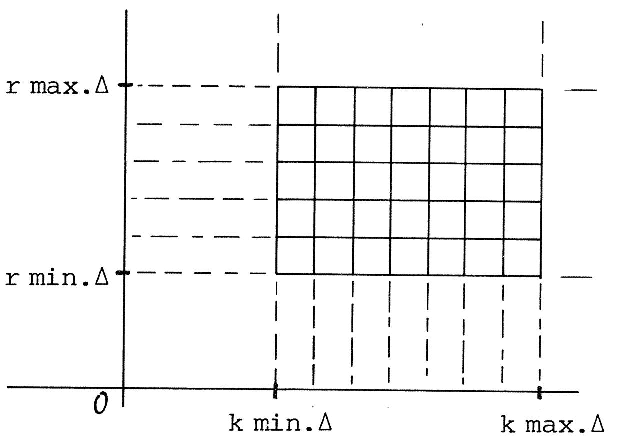


fig. 1.A

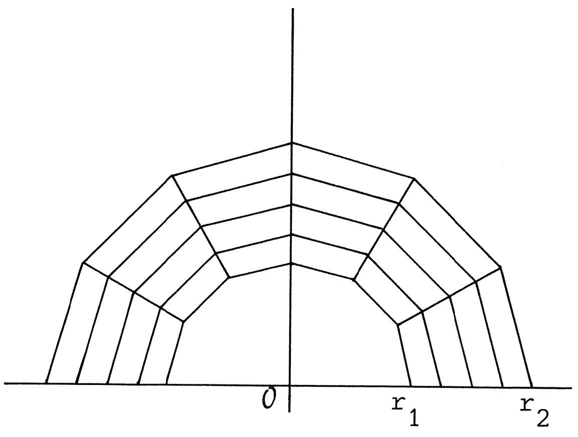


fig. 1.B

lfn - a character string, viz. the identifying string of a FILE variable.

(i) If the file, identified by lfn, is empty, then d03zzb will write on this file information about the grid (coordinates of each point and indication of the subset of boundary grid points) and all computed weights. After termination this file will still exist for further handling by the user (the file will have been opened as standback channel).

(ii) If the file with identification string lfn is not empty, then it must contain all information about a grid including the discretization weights. This file can only have been created by an earlier call of d03zzb. In this case the weights are read from the file without further computing effort and a grid definition by the parameter dg is ignored.

(iii) If the character string is empty (= ""), no file is supplied nor used.

available - a routine supplied by the user with specification
INT : # an integral value # .
The routine must deliver the value of the amount of central memory available for the declaration of REAL variables. If a weights matrix has been computed and the amount of its components exceeds available, then it will not be stored in central memory (see section 11.i.).

fail - the failure routine (see section 6). Users unfamiliar with the use of this parameter should use naghard.

d03zzb delivers an object of mode DISCARR containing all weights matrices. It consists of the following fields:

uniform - a BOOL variable containing TRUE if the grid is uniform, otherwise FALSE.

numgp - an INT variable, containing the number of (interior and boundary) grid points.

position - an array [kmin : kmax, rmin : rmax]INT, containing the representation of the state of each grid point, viz. inside not near border, inside near border, normal border point, corner point of border, or outside (values are 1, 2, 0, -3, -1 respectively).

grid - an array [kmin : kmax, rmin : rmax]POINT, containing all grid point coordinates.

mastor - an array [kmin+1 : kmax-1, rmin+1 : rmax-1]MAT, contains all weights matrices for approximating partial derivatives at interior grid points.
For an interior grid point grid[i, j], the corresponding array element mastor[i, j] refers to a 5 * 8 - matrix of weights. This matrix contains the weights for the approximation of the derivatives ux, uy, ux, uxy and uyy (rows 1 to 5 of the matrix, respectively) at the (i, j)-th grid point using the subgrid grid[i-1 : i+1, j-1 : j+1]. The 8 weights in each row correspond with the grid points

```
grid[i-1, j+1], grid[ i , j+1], grid[i+1, j+1],
grid[i-1,  j ],                 grid[i+1,  j ],
grid[i-1, j-1], grid[ i , j-1], grid[i+1, j-1],
```

respectively.
Let function values in these grid points be given by u[1] , ... , u[8], assuming the same correspondence, and let u0 be the function value at grid[i, j]. Then a derivative approximation is obtained by
SUM (weight[i] * (u[i] - u0) , i = 1 .. 8).
See the description of dataf if mastor[i, j] delivers NIL for an interior grid point (for boundary points and exterior points mastor[i, j] is always NIL).

snstor - an array [kmin+1 : kmax-1, rmin+1 : rmax-1]MAT, contains all weights matrices for approximating boundary derivatives.
For an interior grid point grid[i, j] with bp neighbouring boundary grid points, the corresponding array element snstor[i, j] refers to a (2 bp) * 9 - matrix of weights. This matrix contains the weights for the approximation of the derivatives ux and uy at these boundary points. The 9 weights in each row correspond with the grid points

```
grid[i-1, j+1], grid[ i , j+1], grid[i+1, j+1],
grid[i-1,  j ], grid[ i ,  j ], grid[i+1,  j ],
grid[i-1, j-1], grid[ i , j-1], grid[i+1, j-1],
```

respectively.
The rows with indices (2*k-1) and (2*k) contain the weights for approximating the derivatives ux and uy, respectively, at the k-th boundary point among the 9 grid points, counted in the above used order.
For example, when grid[i-1, j+1], grid[i, j+1] and grid[i+1, j+1] are the boundary points of these 9 grid points, then grid[i+1, j+1] is the third boundary point, and the weights for approximating ux and uy at this point are found in the 5-th and 6-th row of snstor[i, j].
Let function values in the 9 grid points be given by u[1] , ... , u[9], assuming the above used order.
Then a derivative approximation is obtained by
SUM (weight[i] * u[i] , i = 1 .. 9).
See also section 11.ii. for some details.
See the description of dataf if snstor[i, j] delivers NIL for an interior point near the boundary (for other grid points snstor[i, j] is always NIL).

data f - a REF FILE variable, referencing the external file containing the weights matrices. If the weights matrices are not available in central memory, i.e. if for the (i, j)-th grid point mastor[i, j] or snstor[i, j] contains NIL, then a weights matrix is obtained in the following way:

if (mastor[i, j] IS NIL) for an interior point:

```
( set(data f, 1, 1, cposmas[i, j]);
  MAT w = HEAP[1 : 5, 1 : 8]REAL;
  getbin(data f, w); w
)
```

delivers the intended weights matrix,

if (snstor[i, j] IS NIL) for an interior point near the boundary:

```
( set(data f, 1, 1, cpossn[i, j]);
  INT upb; getbin(data f, upb);
  MAT sxy = HEAP[1 : upb, 1 : 9]REAL;
  getbin(data f, sxy); sxy
)
```

delivers the matrix of weights for approximating the first order derivatives at the boundary points neighbouring the (i, j)-th grid point.

cposmas - NIL if no file is supplied, otherwise an array [kmin+1 : kmax-1, rmin+1 : rmax-1]INT, containing keys for finding the weights matrices in the file (see data f for use of these keys).

cpossn - NIL if no file is supplied, otherwise an array [kmin+1 : kmax-1, rmin+1 : rmax-1]INT, containing keys for finding the matrices of weights for the boundary point derivatives in the file (see data f for use of these keys).

The routine d03zzb delivers NIL if an error is detected.

6. Error Indicators

In the event of an error condition being detected, the error routine is called with the parameters listed below. These are printed and execution terminated if the standard failure routine naghard is used (see the document on the Algol 68 error mechanism).

parameter	message
1	INSUFFICIENT CENTRAL MEMORY

2	BACKGROUND MEMORY EXHAUSTED
40	DATA FILE NOT CORRECTLY AVAILABLE
42	PREMATURE END OF DATA FILE
43	NO GRID DEFINITION AND NO DATA FILE
44	NO GRID DEFINITION WHILE DATA FILE GIVEN IS EMPTY
151, 152	SINGULAR MATRIX In the subgrid of three rows and three columns more than 3 grid points are collinear, or 2 points coincide.
201	UNEQUAL LENGTHS OF EX AND EY OF DEFINED GRID ex OF dg and ey OF dg must have corresponding lower and upper bounds.
202	SUCCESSIVE (EX[I], EY[I]) NOT ALONG GRID LINE Two successive corners (of the same boundary polygon) should lie either on the same row or on the same column
205	INTERSECTING BORDER LINES
206	NON-CLOSING BORDER
207	GRID DOES NOT CONTAIN INTERIOR POINTS This error is only signaled if $kmax < kmin + 2$ or $rmax < rmin + 2$.
208	SUCCESSIVE CORNERS COINCIDE
209	SUCCESSIVE EDGES IN SAME DIRECTION
220	BOUNDARY GIVEN DOES NOT FIT IN ARRAY OF GRID POINTS A subscript given in ex OF dg or ey OF dg exceeds the bounds of the given array of grid points.
231, 232, 233, 234	ILLEGAL POSITION OF GRID POINT IN ELEMENTARY QUADRILATERAL The orientation of the vertices of some elementary quadrilateral is differing (should be either clockwise or counterclock-wise for all quadrilaterals). This occurs when two parallel rows or columns of the grid intersect each other.

Since computations cannot proceed if errors in the parameters of d03zzb are met, in most cases the standard failure routine naghard is used.

7. Auxiliary Routines None.

8. Timing

For a first call of the routine d03zzb the computation time depends linearly upon the number of grid points.

9. Storage

The number of memory places required by internally declared arrays, including those of auxiliary routines, is (approximately):

```
   If the grid is uniform :
      8 * ( # of columns of grid ) * ( # of rows of grid ) + 500
   REAL variables, else :
      8 * ( # of columns of grid ) * ( # of rows of grid )
         + 40 * ( # of interior grid points )
         + 60 * ( # of boundary grid points )
   REAL variables.
```

(It depends upon the availability of storage in central memory whether this space is used as direct access storage or only on the supplied FILE, see section 11.i.).

10. Accuracy

The order of accuracy of the underlying finite difference technique equals 2. For details see ref. [1].

11. Further Comments

(i) Use of d03zzb with a file has the following side effect: If space is lacking in central memory for storing all weights matrices, then the part that could not be stored can be retrieved from the given file each time it is needed for discretization. This will slow up the computations of a process for solving a discretized PDE, in which the discretization weights are used at all grid points, alternatingly. Otherwise, if no data file were given and space was lacking, then the calculation of discretization weights is stopped.

(ii) Usually, weights for approximating boundary derivatives at a given point occur in the weights matrices of several interior points near the boundary, viz. for all overlapping 3 * 3 -

subgrids that contain the given boundary grid point. Thus, when a boundary derivative is required, one can still choose which set of 9 grid points is used for the approximation. In this way the coupling of unknowns appearing in the boundary conditions can be controlled.

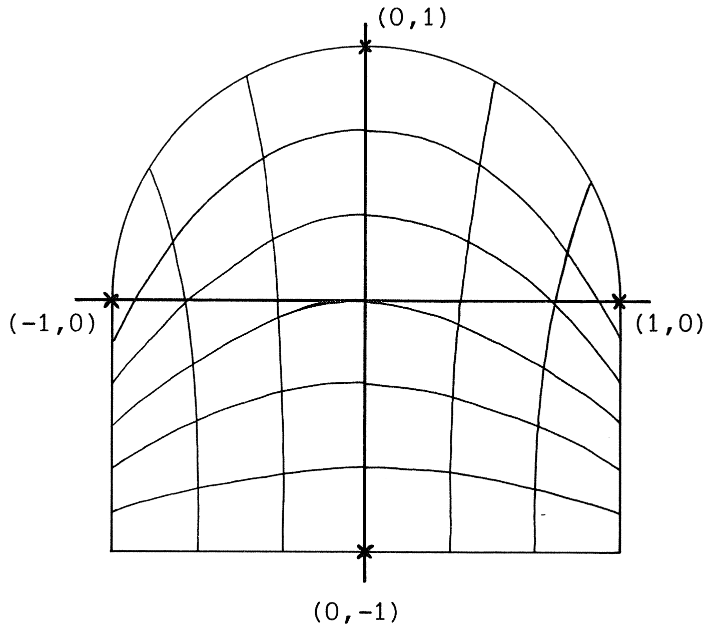
12. Keywords

Finite Difference Methods,
Partial Differential Equations.

13. Example

This program may require amendment before it can be used in some implementations. The results produced may differ slightly.

13.1. Program text



(fig. 2)

#
The curvilinear grid inside [-1, +1] * [-1, +1] (fig. 2), example in [1], is defined by the DEFGRID parameter of d03zzb.

```
The discretization weights for the approximation of uxx (= second
order partial derivative of u w.r.t. the first space variable)
at the POINT (0, 2/3) are found in the third row of the
corresponding weights matrix.
These weights are used for calculating the value of uxx for a given
function u = exp(x + y) at (0, 2/3).
The discretization weights for the approximation of ux and uy at
the boundary point (0, 1) using the subgrid around the point
(-0.472, +0.555) are found in the two rows of the array weights.
These weights are used for calculating the values of ux and uy for
the given function u = exp(x + y) at (0, 1) :                      #

BEGIN INT n = 6;

  PROC gr = ( INT k, r) POINT :
  (  INT i = k, j = n - r;
     ( (j * (2 * i - n) / n - (n - j) * cos (i * pi / n) ) / n,
       ((n - j) * sin(i * pi / n) - j ) / n )
  );

  DISCARR dc =
     d03zzb ( (  gr,
                 HEAP [1 : 5] INT := ( 0, n, n, 0, 0 ),
                 HEAP [1 : 5] INT := ( 0, 0, n, n, 0 )
              ),
              "myfile",
              INT : 100000,
              nagfail
            );

  PROC u = ( REAL x, y) REAL : exp(x + y);

  PROC uu = ( INT i, j) REAL :
  ( POINT g = gr(i, j); u(xc OF g, yc OF g) );

  VEC xxwghts =
     (  MAT wm = (mastor OF dc)[3, 5];
        wm ISNT NIL
     !  wm
     !  REF FILE locf = data f OF dc,
        MAT w = HEAP [1 : 5, 1 : 8] REAL ;
        set(locf, 1, 1, (cposmas OF dc)[3, 5]);
        getbin(locf, w); w
     ) [3, ];

  INT k:= 0, REAL uxx:= 0.0, REAL u35 = uu(3, 5);
  print((newline, " xxwghts are :", newline));
  FOR i FROM 6 BY -1 TO 4
  DO FOR j FROM 2 TO 4
     DO print( IF i = 5 AND j = 3 THEN 12 * " "
            ELSE k +:= 1; uxx +:= xxwghts[k] * (uu(j, i) - u35);
               fixed(xxwghts[k], -12, 6)
            FI )
      OD ;
      print(newline)
   OD ;
   print((newline, " uxx at (", fixed(xc OF gr(3, 5), -6, 3),
          ", ", fixed(yc OF gr(3, 5), -6, 3), "): ",
          float(uxx, 16, 10, 2), newline, " ( exp(0.667) = ",
          float(exp(2/3), 16, 10, 2), " )", newline));

   MAT weights =
      (  MAT sxy = (snstor OF dc)[2, 5];
         sxy ISNT NIL
      !  sxy
      !  REF FILE locf = data f OF dc, INT upb;
         set(locf, 1, 1, (cpossn OF dc)[2, 5]);
         getbin(locf, upb); MAT sn = HEAP [1:upb, 1:9] REAL ;
         getbin(locf, sn); sn
      )
      # point [3, 6] is third boundary point of the 3*3-subgrid,
        so it corresponds with rows 5 and 6 of weights matrix : #
      [5 : 6, ];

   REAL ux := 0.0, uy := 0.0;
   print((newline, " weights around (", fixed(xc OF gr(2, 5), -6,3),
      ", ", fixed(yc OF gr(2, 5), -6, 3), ") are :", newline));
   FOR der TO 2
   DO VEC wghts = weights[der, ], INT k := 0;
      FOR i FROM 6 BY -1 TO 4
      DO FOR j FROM 1 TO 3
         DO print( ( k +:= 1;
                     IF der = 1 THEN ux ELSE uy FI +:=
                     wghts[k] * uu(j, i);
                     fixed(wghts[k], -12, 6)
                   ) )
         OD ;
         print(newline)
      OD ; print(newline)
   OD ;
   print((newline, " ux at (0, 1):  ", float(ux, 16, 10, 2),
          newline, " uy at (0, 1):  ", float(uy, 16, 10, 2),
      newline, " ( exp(1.0) = ",
   float(exp(1.0), 16, 10, 2), " )", newline))
END
```

13.2. Data for program

None.

13.3. Results

If myfile was empty, the DEFGRID parameter of d03zzb is analysed and all discretization weights for the approximation of the derivatives are stored in the DISCARR yield of d03zzb, and written on myfile.
The VEC xxwghts declaration finds the 5 * 8 - matrix of weights, its third row is delivered. xxwghts[1 : 8] contains the weights for uxx for the 8 surrounding points. The program prints:

```
xxwghts are :
   1.107726   -0.153416    1.107726
   3.544425                3.544425
  -0.340789   -0.338515   -0.340789

uxx at (-0.000,  0.667):  +2.0071863574e+0
( exp(0.667) = +1.9477340411e+0 )
```

The weights matrix found for the approximation of derivatives at the boundary grid point is a 6 * 9 - matrix (because the subgrid contains 3 boundary points). The MAT weights contains a 2 * 9 - matrix corresponding with the boundary point indicated by (3, 6). The vector wghts refers to the two rows of this submatrix, successively. The program prints:

```
weights around (-0.472,  0.555) are :
   3.937076   -3.229303    1.311667
  -2.939585   -3.702999    3.316885
   0.338155    2.545619   -1.577515

  -1.198204    1.395958    3.950936
   2.399324   -2.821029   -4.875084
  -1.198449    1.421869    0.924679


ux at (0, 1):  +2.4693365991e+0
uy at (0, 1):  +2.6168837904e+0
( exp(1.0) = +2.7182818285e+0 )
```

Source Text:

```
BEGIN
INT inside= 1, innearb= 2, border= 0, corner= -3, outside= -1;


PROC d03zzb = ( DEFGRID dg, STRING lfn,
                    PROC INT available, NAGFAIL nfail) DISCARR :
BEGIN
   MODE LSQEPS = STRUCT ( REAL prec, max, INT rnk);
   BOOL erron:= FALSE ;

   NAGFAIL fail = ( INT rn, STRING txt) VOID :
   BEGIN erron:= TRUE ; nfail(rn, "d03zzb : " + txt) END ;

   PRIO +< = 1;

   # the declarations for genvec, genmat, COPY , * and +< can be
     deleted when torrix is used #

   PROC genvec = ( INT u) VEC : HEAP [1 : u] REAL ,

   PROC genmat = ( INT m, n) MAT : HEAP [1 : m, 1 : n] REAL ,

   OP COPY = ( VEC u) VEC :
   IF u IS NIL THEN NIL
   ELSE INT l = LWB u; genvec( UPB u - l + 1)[ AT l]:= u
   FI ,

   OP * = ( REAL a, VEC b) VEC :
    ( VEC c = COPY b;
      FOR i FROM LWB b TO UPB b DO c[i] *:= a OD ; c
    ),

   OP * = ( VEC a, b) REAL :
    ( INT l = LWB a; INT lb = LWB b - l, REAL s:= 0.0;
      FOR i FROM l TO UPB a DO s+:=a[i] * b[i + lb] OD ; s
    ),

   OP +< = ( VEC a, b) VEC :
   ( FOR i FROM LWB a TO UPB a DO a[i] +:= b[i] OD ; a ),


   PROC lsqdec = ( MAT a, VEC aid, REF [] INT ci,
                     REF LSQEPS aux) INT :
   IF INT n = 1 UPB a, m = 2 UPB a,
      REF INT r = rnk OF aux:= -1;
      UPB aid /= m OR UPB ci /= m THEN r
   ELSE INT pk:= 1, INT minmn= (m < n ! m ! n),
      REAL sigma:= 0.0,
      VEC sum = genvec(m); r:= 0;

      FOR k TO m
      DO IF REAL w= (sum[k]:= ( VEC ak = a[ ,k]; ak * ak) );
            w > sigma THEN sigma:= w; pk:= k FI
      OD ;

      REAL w:= max OF aux:= sqrt(sigma);
      REAL eps= (prec OF aux) * w;
      FOR k TO minmn WHILE w > eps
      DO VEC ak = a[k : , k], REAL akk = a[k,pk]; r:= k;
         IF INT lpk = (ci[k]:= pk); lpk /= k
         THEN VEC colk = a[ ,k], colpk = a[ , lpk];
            VEC h = COPY colk; colk:= colpk; colpk:= h;
            sum[lpk]:= sum[k]
         FI ;
         REAL aidk= (aid[k]:= (akk < 0.0 ! w ! - w) );
         ak[1]:= akk - aidk; REAL beta= - 1.0 / (sigma - akk * aidk);
         pk:= k; sigma:= 0.0;
         FOR j FROM k + 1 TO m
         DO VEC colj = a[k : ,j]; colj +< beta * (ak * colj) * ak;
            IF REAL locw= (sum[j] -:= colj[1] ** 2); locw > sigma
            THEN pk:= j; sigma:= locw FI
         OD ;
         w:= sqrt( sigma := ( VEC ak1 = a[k+1 : , pk]; ak1 * ak1) )
      OD ;
      r
   FI # end of householder triangularization #,

   PROC lsqsol = ( MAT a, VEC aid, REF [] INT ci, VEC b) VEC :
   BEGIN INT n = 1 UPB a, m = 2 UPB a, VEC bb = COPY b;
      IF m <= n
      THEN FOR k TO m
         DO VEC colk = a[k : , k];
            bb[k: ] +< colk * bb[k: ] / (aid[k] * colk[1]) * colk
         OD ;
         FOR k FROM m BY - 1 TO 1 DO bb[k] :=
            (bb[k] - a[k,k+1: ] * bb[k+1:m]) / aid[k] OD ;
         FOR k FROM m - 1 BY - 1 TO 1
         DO IF INT cik= ci[k]; cik /= k
            THEN REAL w= bb[k]; bb[k]:= bb[cik]; bb[cik]:= w FI
         OD
      FI ;
      bb
   END # of computation of least squares solution #,


   CO optimal inverse of non-square matrix routine
   using least squares solution routines.
   part 6 of library of numerical algebra routines. CO
```

```
PROC mininverse = ( MAT a, INT l) MAT :
BEGIN INT m = 1 UPB a, n = 2 UPB a;

   # compute w with l rows :
     w * a = ( i (l * l) !  minimal (l * (n - l)) matrix ) #

   MAT u = genmat(m, m),
   VEC diag = genvec(m),
   LSQEPS aux:= (1.0e-12, 0.0, 0),
   [1 : m] INT piv;

   u[ , 1 : l]:= a[, 1 : l];
   MAT a2 = a[ , l + 1 : n];

   IF lsqdec(u[ , : l], diag[:l], piv[:l], aux) /= l
   THEN fail(151, "singular matrix") FI ;

   # form r(inv) in upper triangle, mind diag #
   FOR i FROM l - 1 BY -1 TO 1
   DO REAL xii = 1 / diag[i], VEC ai = u[i, i + 1 : l];
      FOR j FROM l - i BY -1 TO 1
      DO ai[j]:= - (ai[ : j - 1] * u[i + 1 : j + i - 1, j + i]
                        + ai[j] / diag[j + i] ) * xii
      OD
   OD ;

   # compute r(inv)(m * m) * q(transp)
     = r(inv) * q(l) * q(l-1) * ... * q(2) * q(1) #
   VEC v = genvec(m); VEC vl = v[l : m] := u[l : m, l];
   REAL s= 1.0 / (diag[l] * vl[1]);

   FOR i TO l
   DO REAL ail = ( i = l ! 1.0 / diag[l] ! u[i, l] );
      u[i, l : m]:= ( vl[1] * ail * s ) * vl; u[i, l]+:= ail
   OD ;
   FOR i FROM l + 1 TO m
   DO u[i, l : m]:= s * v[i] * vl; u[i, i] +:= 1.0 OD ;

   FOR k FROM l - 1 BY -1 TO 1
   DO VEC vk = v[k : m] := u[k : m, k];
      u[k, k]:= 1.0 / diag[k];
      FOR i FROM k + 1 TO m DO u[i, k]:= 0.0 OD ;
      REAL s= 1.0 / (diag[k] * vk[1]);
      FOR i TO m
      DO VEC ui = u[i, k : m]; ui  +< vk * ui * s * vk OD
   OD ;

   # back changes (using piv) of first l rows #
   FOR k FROM l - 1 BY -1 TO 1
   DO IF INT cik = piv[k]; cik /= k
      THEN VEC uk = u[k, ], ucik = u[cik, ]; VEC h= COPY uk;
```

```
          uk:= ucik; ucik:= h

        FI
      OD ;

      MAT a1inv = u[ : l, ], a1orthtrp = u[l + 1 : m, ],
          h = genmat(n - l, m);
      MAT h1 = h[ , l + 1 : m], h2 = h[ , : l];

      FOR i TO n - l
      DO FOR j TO m DO h[i, j]:= a2[, i] * u[j, ] OD OD ;
      # h1, h2 formed inside h #

      IF lsqdec(h1, diag[ : m-l], piv[ : m-l], aux) /= m - l
      THEN fail(152, "singular matrix") FI ;
      FOR j TO l
      DO h2[,j]:= lsqsol(h1, diag[ : m-l], piv[ : m-l], h2[, j]) OD ;

      MAT x = h2[ : m - l, ], w = a1inv;

      FOR i TO l
      DO FOR j TO m DO w[i, j] -:= x[, i] * a1orthtrp[, j] OD
      OD ;

      w
    END # min inverse #,


    PROC locd03zzb = ( DEFGRID dg, STRING lfn) DISCARR :
    BEGIN DISCARR dcr = HEAP DSCRR :=
            ( TRUE , SKIP ,
              NIL , NIL , NIL , NIL , NIL , NIL , NIL
            ),
       BOOL parfile = lfn /= "";
       BOOL get data:= parfile, put data:= FALSE ;
       BOOL pargrid =
          IF ex OF dg IS REF [ ] INT ( SKIP ) THEN FALSE
          ELSE CASE gr OF dg
             IN ( REF [,] POINT ) : TRUE ,
                  ( PROC ( INT , INT ) POINT ) : TRUE ,
                  ( TRIO ) : TRUE
             OUT FALSE # in this case SKIP given #
             ESAC
          FI ;

       IF parfile
       THEN INT kmin, kmax, rmin, rmax,
          REF FILE locfile = data f OF dcr:= HEAP FILE ;
          IF IF open(locfile, lfn, stand back channel) /= 0
             THEN INT es= establish(locfile, lfn, stand back channel,
                          1, 1, 131071); NOT (es = 0 OR es = 2)
             ELSE FALSE FI # error = file was already opened #
        THEN fail(40, "data file not correctly available");fin FI ;
        # test if file is not empty #
        on logical file end(locfile,
           ( REF FILE f) BOOL :
              ( IF NOT pargrid
                THEN fail(44, "empty data file given"); fin FI ;
                get data:= FALSE ; put data:= TRUE ; continue
              )            );
        getbin(locfile, numgp OF dcr); getbin(locfile, kmin);
        getbin(locfile, kmax); getbin(locfile, rmin);
        getbin(locfile, rmax);

        position OF dcr:= HEAP [kmin : kmax, rmin : rmax] INT ;
        grid OF dcr:= HEAP [kmin : kmax, rmin : rmax] POINT ;
        INT aid; IF getbin(locfile, aid); aid < 0
        THEN uniform OF dcr := FALSE FI ;
     continue : SKIP
     ELIF NOT pargrid
     THEN fail(43, "both no definition of grid and no data file"
              " given"); fin
     FI ;

     IF pargrid AND NOT getdata
     THEN tfm grid(dg, numgp OF dcr, uniform OF dcr,
             position OF dcr, grid OF dcr, fail);
        IF erron THEN fin FI
     FI ;

     REF [,] INT positn = position OF dcr,
     REF [,] POINT grid = grid OF dcr;

     INT kmin = 1 LWB positn, kmax = 1 UPB positn,
            rmin = 2 LWB positn, rmax = 2 UPB positn,
            numgp = numgp OF dcr,
     BOOL uniform = uniform OF dcr,
     REF FILE locfile = data f OF dcr;
     INT lw1mas = kmin + 1, up1mas = kmax - 1,
            lw2mas = rmin + 1, up2mas = rmax - 1;
     HEAP [lw1mas : up1mas, lw2mas : up2mas] MAT mastor, snstor;
     mastor OF dcr:= mastor; snstor OF dcr:= snstor;
     FOR i FROM lw1mas TO up1mas
     DO FOR j FROM lw2mas TO up2mas
        DO mastor[i, j]:= NIL OD
     OD ;
     snstor:= mastor;

  # the discretization weights are either computed by  compute data
    and possibly written (with other information) to a data file,
    or read from a data file. depending on the available space the
    weights can be kept in central memory or left on the data file
    till they are needed for actual discretization.
```

```
#

    INT cpos cposmas;

    IF get data OR put data
    THEN cposmas OF dcr:= HEAP [lw1mas:up1mas,lw2mas:up2mas] INT ;
       cpossn OF dcr:= HEAP [lw1mas:up1mas,lw2mas:up2mas] INT
    FI ;
    REF [,] INT cposmas = cposmas OF dcr,
       cpossn = cpossn OF dcr;
    INT avail := available;
    IF INT border = (kmax - kmin + rmax - rmin + 2) * 2;
       INT needed = ( uniform ! 1 ! numgp - border )
          * 44 + (border - 4) * 59 + 100 - avail; needed > 0
    THEN print((newline, " ====== d03zzb : insufficient field len"
          "gth, needed about ", whole(needed OVER 100 + 1, -5),
          "00 (decimal) words more.", newline));
       IF get data OR put data
       THEN print((8*" ", "data kept on file, no abort.", newline))
       ELSE fail(1, "insufficient central memory"); fin
       FI
    FI ;
    IF put data
    THEN putbin(locfile, numgp); putbin(locfile, kmin);
       putbin(locfile, kmax); putbin(locfile, rmin);
       putbin(locfile, rmax);
       putbin(locfile, INT (uniform ! 1 ! -1) );
       putbin(locfile, positn); putbin(locfile, grid);
       cpos cposmas:= char number(locfile); putbin(locfile, cposmas);
       putbin(locfile, cpossn)#to reserve space for cposmas and /sn#
    FI ;
    IF NOT get data
    THEN compute data(mastor, snstor, grid, positn, uniform,
        put data, locfile, cposmas, cpossn, avail);
       IF erron THEN fin FI ;
       IF put data
       THEN set(locfile, 1, 1, cpos cposmas);
          putbin(locfile, cposmas); putbin(locfile, cpossn)
       FI
    ELSE BOOL start:= TRUE , getthem:= TRUE , MAT w,
       INT nsit:= 0, [1 : 60] INT situation, ksit, rsit,
       [1 : 60] MAT sxy;
       on logical file end(locfile,
             ( REF FILE f) BOOL :
             ( fail(42, "premature end of data file"); fin )
                         );
       getbin(locfile, positn); getbin(locfile, grid);
       cpos cposmas:= char number(locfile);
       getbin(locfile, cposmas); getbin(locfile, cpossn);
       FOR i FROM lw1mas TO up1mas
       DO REF [] INT positni= positn[i, ];
```

```
            FOR j FROM lw2mas TO up2mas
            DO IF positni[j] >= inside
               THEN IF start
                  THEN IF get them
                     THEN mastor[i, j]:= genmat(5, 8); avail -:= 44;
                        getbin(locfile, mastor[i, j]);
                        get them:= avail > 0
                     FI ;
                     IF uniform
                     THEN w:= mastor[i, j]; start:= FALSE FI
                  ELSE mastor[i, j]:= w
                  FI ;
                  IF positni[j] = in nearb AND get them
                  THEN snstor[i, j]:=
                     IF INT recog =
                        IF uniform THEN recognize sit(
                           positn[i-1:i+1,j-1:j+1], situation, nsit)
                        ELSE -1 FI ;
                        IF recog = 0
                        THEN ksit[nsit]:= i; rsit[nsit]:= j FI ;
                        recog > 0
                     THEN sxy[recog]
                     ELSE INT nrow; getbin(locfile, nrow);
                        MAT locsxy = genmat(nrow, 9);
                        avail -:= nrow * 9 + 4; get them:= avail > 0;
                        getbin(locfile, locsxy);
                        IF recog = 0 THEN sxy[nsit]:= locsxy FI ;
                        locsxy
                     FI
                  FI
               FI
            OD
         OD
      FI ;
      dcr
   END # end of generation weights by locd03zzb #,


   #  external for computation of grid from the user supplied
      information in dgrid. #

   PROC check coord = ( REF [ , ] POINT grid, REF [ , ] INT pos,
                        NAGFAIL fail) VOID :
   # mln 790521 #
   BEGIN INT sign;

      PROC det = ( REF POINT p1, p2, p3) REAL :
      ((xc OF p2 - xc OF p1) * (yc OF p3 - yc OF p1) -
       (xc OF p3 - xc OF p1) * (yc OF p2 - yc OF p1)
      );
```

```
PROC check two lines = ( REF [ ] POINT gr1, gr2,
   REF [ ] INT pos1, pos2) VOID :
BEGIN
   INT nsucc := 0, REF POINT p11, p12, p21, p22;

   PROC check orientation = VOID :
   BEGIN
      IF REAL area1 = det (p21, p12, p11);
        SIGN (area1) /= sign
      THEN fail(231, "grid point out of position")
      ELIF REAL area2 = det (p21, p22, p12);
         SIGN (area2) /= sign
      THEN fail(232, "grid point out of position")
      ELIF REAL area3 = det (p11, p21, p22);
         SIGN (area3) /= sign
      THEN fail(233, "grid point out of position")
      ELIF SIGN (area1 + area2 - area3) /= sign
      THEN fail(234, "grid point out of position")
      FI
   END # check orientation #;

   FOR j TO UPB gr1
   DO
      IF pos1[j] /= outside AND pos2[j] /= outside
      THEN nsucc +:= 1
      ELSE nsucc := 0
      FI ;
      CASE nsucc
      IN (p11 := gr1[j]; p21 := gr2[j]),
           (p12 := gr1[j]; p22 := gr2[j];
            check orientation),
           (p11 := p12; p21 := p22;
            p12 := gr1[j]; p22 := gr2[j];
            check orientation; nsucc :=2)
      OUT SKIP
      ESAC
   OD
END # check two lines #;

REF [ ] POINT gr1, REF [ ] INT pos1;
REF [ ] POINT gr2 := grid[1, ];
REF [ ] INT pos2 := pos[1, ];
sign := ( INT j := 0;
         WHILE pos2[j+:=1] /= corner DO SKIP OD ;
         SIGN (det (grid[2,j], gr2[j+1], gr2[j])) );
FOR i FROM 2 TO 1 UPB grid
DO gr1 := gr2; pos1 := pos2;
   gr2 := grid[i, ]; pos2 := pos[i, ];
   check two lines (gr1, gr2, pos1, pos2)
OD
```

```
END # check coord #,


PROC tfm grid = ( DEFGRID dgrid, REF INT numgp,
      REF BOOL uniform, REF REF [,] INT position,
      REF REF [,] POINT grid, NAGFAIL fail) VOID :
   CO values of positn[i, j] signify :
        -1 (= outside) : outside grid,
         0 (= border)  : point on boundary grid,
        -3 (= corner)  : corner point of boundary grid,
         1 (= inside)  : point lying inside grid,
         2 (= innearb) : point lying inside grid but neighbouring
                         boundary point(s).                    CO
BEGIN REF [ ] INT ex = (ex OF dgrid)[ AT 0],
      ey = (ey OF dgrid)[ AT 0];
   INT upb ex = UPB ex;
   IF upb ex /= UPB ey
   THEN fail(201, "unequal lengths of ex and ey of defgrid"); fin
   FI ;
   INT kmin:= ex[0], rmin:= ey[0]; INT kmax:= kmin, rmax:= rmin;
   FOR i TO upb ex
   DO IF INT exi = ex[i]; kmin > exi THEN kmin:= exi
      ELIF kmax < exi THEN kmax:= exi
      FI ;
      IF INT eyi = ey[i]; rmin > eyi THEN rmin:= eyi
      ELIF rmax < eyi THEN rmax:= eyi
      FI
   OD ;
   IF INT imax = kmax - kmin + 1, jmax = rmax - rmin + 1;
      numgp:= imax * jmax; imax < 3 OR jmax < 3
   THEN fail(207, "grid does not contain interior grid points");
      fin
   FI ;
   position:= HEAP [kmin : kmax, rmin : rmax] INT ;
   REF [,] INT positn = position;
   FOR i FROM kmin TO kmax
   DO FOR j FROM rmin TO rmax DO positn[i, j]:= inside OD
   OD ;                                # grid preset on inside #

   INT i0:= ex[0], j0:= ey[0]; INT outi= kmin-1, outj= rmin-1;
   INT dir:= 0, in:= i0, jn:= j0, BOOL erron:= FALSE ;

   NAGFAIL nf = ( INT rn, STRING txt) VOID :
   BEGIN erron:= TRUE ; fail(rn, txt) END ;

   FOR i TO upb ex           # fill border elements of grid #
   DO INT i1 = ex[i], j1 = ey[i];

      PROC trace = ( REF [] INT locp) VOID :
      FOR k TO UPB locp
      DO IF locp[k] = border
      THEN nf(205,"intersecting border lines")
      ELSE locp[k]:= border
      FI
   OD ;

   IF i1 = i0 AND j1 /= j0
   THEN
      IF dir = 1
      THEN nf(209, "successive edges in same direction") FI ;
      trace( positn[i0, ( j1 > j0 ! j0 ! j1 + 1 ) :
         ( j1 > j0 ! j1 - 1 ! j0 ) ] ); dir:= 1
   ELIF i1 /= i0 AND j1 = j0
   THEN
      IF dir = -1
      THEN nf(209, "successive edges in same direction") FI ;
      trace( positn[ ( i1 > i0 ! i0 ! i1 + 1 ) :
         ( i1 > i0 ! i1 - 1 ! i0 ), j0] ); dir:= -1
   ELIF dir:= 0; i0 /= outi AND j0 /= outj
   THEN ( i1 = i0 AND j1 = j0
        ! nf(208, "successive corners coincide")
        ! nf(202, "successive corners not along grid line")
        )
   ELIF positn[i1, j1] = border
   THEN nf(208,"successive corners coincide")
   ELSE in:= i1; jn:= j1
   FI ;
   IF positn[i1, j1] = border
   THEN IF i1 = in AND j1 = jn THEN i0:= outi; j0:= outj
      ELSE nf(206, "non-closing border") FI
   ELIF i = upb ex
   THEN nf(206, "non-closing border")
   ELSE i0:= i1; j0:= j1
   FI
OD ;
IF erron THEN fin FI ;

FOR i FROM kmin TO kmax        # compute outside elements #
DO INT last:= outside, allast:= outside,
   REF [ ] INT locp = positn[i, ];
   FOR j FROM rmin TO rmax
   DO REF INT present = locp[j];
      CASE allast + 2
      IN IF last = outside THEN present:= - present FI ,
         CASE last + 2
         IN present:= - present,
            IF present /= border
            THEN IF i = kmin THEN present:= outside
               ELIF REF [ ] INT lcg = positn[i-1,j-2 : j];
                  INT temp = lcg[3];
                  temp /= border THEN present:= temp
               ELIF INT tmp1 = lcg[2];
```

```
                    tmp1 /= border THEN present:= tmp1
                  ELIF lcg[1] = inside THEN present:= outside
                  FI
                FI ,
                SKIP
              ESAC ,
              IF last = border THEN present:= - present FI
            ESAC ;
            allast:= last; last:= present;
            IF present = outside THEN numgp -:= 1 FI
          OD
        OD ;
                                                    # copy coordinates #
        CASE gr OF dgrid IN
        ( REF [ , ] POINT ar ):
          IF 1 LWB ar > kmin OR 2 LWB ar > rmin OR
             1 UPB ar < kmax OR 2 UPB ar < rmax
          THEN fail(220, "array bounds for grid do not fit in"
             " boundary given"); fin
          ELSE grid:= ar[kmin : kmax AT kmin,rmin : rmax AT rmin];
             uniform:= FALSE
          FI ,
        ( PROC ( INT , INT ) POINT pr ):
          BEGIN grid:= HEAP [kmin : kmax, rmin : rmax] POINT ;
             uniform:= FALSE ;
             FOR i FROM kmin TO kmax
             DO REF [ ] POINT locg = grid[i, ],
                REF [ ] INT locp = positn[i, ];
                FOR j FROM rmin TO rmax
                DO IF locp[j] >= border THEN locg[j]:= pr(i, j) FI
                OD
             OD
          END ,
        ( TRIO tr ) :
          BEGIN grid:= HEAP [kmin : kmax, rmin : rmax] POINT ;
             POINT p00= p00 OF tr, p10= p10 OF tr,
                p01= p01 OF tr;
             REAL ox:= xc OF p00, oy:= yc OF p00;
             REAL dxk = xc OF p10 - ox, dyk = yc OF p10 - oy,
                dxr = xc OF p01 - ox, dyr = yc OF p01 - oy;
             grid[kmin, rmin]:= p00;
             FOR i FROM kmin TO kmax
             DO REF [ ] POINT locg = grid[i, : ];
                REF POINT p1 = locg[1];
                IF i > kmin
                THEN xc OF p1:= (ox +:= dxk);
                   yc OF p1:= (oy +:= dyk)
                FI ;
                REAL px:= xc OF p1, py:= yc OF p1;
                FOR j FROM 2 TO UPB locg
                DO xc OF locg[j]:= ( px +:= dxr);
```

```
                    yc OF locg[j]:= ( py +:= dyr)
                 OD
              OD
           END
        ESAC ;

                                        # compute near border elements of grid #
        i0:= ex[0]; j0:= ey[0]; positn[i0, j0]:= -2;
        FOR i TO upb ex
        DO INT i1 = ex[i], j1 = ey[i];

           PROC trace = ( REF [] INT locp) VOID :
           FOR k TO UPB locp
           DO IF locp[k] = inside THEN locp[k]:= innearb FI OD ;

           IF i1 = i0 AND j1 /= j0
           THEN IF j1 > j0
              THEN trace( positn[
                        ( i0 = kmin ! i0+1 !: positn[i0-1,j1-1] <
                                     inside ! i0 + 1 ! i0 - 1 ),
                        ( j0 > rmin + 1 ! j0 - 1 ! rmin+1 ) : j1] )
              ELSE trace( positn[
                        ( i0 = kmin ! i0+1 !: positn[i0-1,j1+1] <
                                     inside ! i0 + 1 ! i0 - 1 ),
                          j1 : ( j0 < rmax - 1 ! j0 + 1 ! rmax - 1 ) ] )
              FI
           ELIF j1 = j0 AND i1 /= i0
           THEN IF i1 > i0
              THEN trace( positn[
                    ( i0 > kmin+1 ! i0 - 1 ! kmin+1 ) : i1,
                    ( j0 = rmin ! j0+1 !: positn[i1-1,j0-1] < inside
                              ! j0 + 1 ! j0 - 1 ) ] )
              ELSE trace( positn[i1 : ( i0 < kmax-1 ! i0+1 ! kmax-1),
                    ( j0 = rmin ! j0+1 !: positn[i1+1,j0-1] < inside
                              ! j0 + 1 ! j0 - 1 ) ] )
              FI
           FI ;
           IF positn[i1, j1] = -2
           THEN positn[i1, j1]:= corner; i0:= outi; j0:= outj
           ELSE positn[i1, j1]:= ( i0 = outi ! -2 ! corner );
              i0:= i1; j0:= j1
           FI
        OD ;
        check coord(grid[:,:], positn[:,:], nf)
     END # of tfm grid #,


     PROC recognize sit = ( REF [,] INT pos, REF [] INT situation,
           REF INT nsit) INT :
     # yield is index such that sxy[ind] is sxy with same situation,
       else store new sxy-weights and nsit +:= 1 #
```

```
BEGIN INT val:= 0, pow:= 1;
   FOR i TO 3 DO FOR j TO 3
      DO IF pos[i, j] <= border THEN val+:= pow FI ;
         IF pos[i, j] = -3 THEN val+:= pow FI ;
         pow *:= 4
   OD OD ;
   pow:= 0;
   FOR j TO nsit WHILE pow = 0
   DO IF situation[j] = val THEN pow:= j FI OD ;
   IF pow = 0 AND nsit < 60
   THEN situation[ nsit +:= 1 ]:= val FI ;
   pow
END # recognize situation #,


COMMENT the interface for semidiscretization of initial
       boundary value problems. this part by : p.h.m. wolkenfelt
       (ordinary points) and j. kok (near-boundary points),
       using the minimal-inverse-method by k. dekker.
                                                      COMMENT

# ==================== begin of discretizer ======================#


PROC compute data = ( WMAT mastor, snstore, REF [,] POINT grid,
                 REF [ , ] INT position, BOOL uniform, put data,
                 REF FILE data, REF [ , ] INT cposmas, cpossn,
                 INT available) VOID :
BEGIN INT kmin= 1 LWB position, rmin= 2 LWB position,
          kmax= 1 UPB position, rmax= 2 UPB position,
   REAL  sqrt2 = 1.414 21356 23731, sqrt6 = 2.449 48974 27832;

   PROC generate wghts = ( INT k, r, REF [,] POINT grid
                         ) MAT :
   BEGIN [1 : 8] REF POINT p, INT ind:= 0;
      FOR r1 FROM 3 BY -1 TO 1
      DO FOR k1 TO 3
         DO IF r1 /= r OR k1 /= k
              THEN p[ ind+:= 1 ] := grid[k1, r1] FI
         OD
      OD ;
      #  [ 1, 3 ]   [ 2, 3 ]   [ 3, 3 ]       p1   p2   p3

         [ 1, 2 ]   [ 2, 2 ]   [ 3, 2 ]       p4   pc   p5

         [ 1, 1 ]   [ 2, 1 ]   [ 3, 1 ]       p6   p7   p8

     provided that [k, r] indicates the centre of the nine points #
     REF POINT pc = grid [ k , r ];

     REAL xcentre = xc OF pc, ycentre = yc OF pc;
   MAT m = genmat(8, 14),
   REAL delta:= 0.0;

   FOR i TO 8
   DO REAL xi= xc OF p[i] - xcentre,
           yi= yc OF p[i] - ycentre;
       REAL xi2= xi*xi, yi2= yi*yi;
       m[ i, ] :=  ( xi, yi, xi2/sqrt2, xi*yi, yi2/sqrt2,
                     xi2*xi/sqrt6, xi2*yi/sqrt2,
                     xi*yi2/sqrt2, yi2*yi/sqrt6,
                     xi2*xi2/(2*sqrt6), xi*xi2*yi/sqrt6,
                     xi2*yi2/2, xi*yi*yi2/sqrt6, yi2*yi2/(2*sqrt6)
                   );
       IF ABS xi > delta THEN delta:= ABS xi FI ;
       IF ABS yi > delta THEN delta:= ABS yi FI
   OD ;

# scale factors #
   REAL d1= 1/delta; REAL d2= d1*d1; REAL d3= d1*d2,
      d4= d2*d2;
   [] REAL scale= (d1,d1,d2,d2,d2,d3,d3,d3,d3,d4,d4,d4,d4,d4);
# scaling the columns of m #
   FOR j TO 14
   DO REAL s= scale[j];
        FOR i TO 8 DO m[i,j]*:= s OD
   OD ;

# computation of the minimal inverse #
   MAT w = min inverse(m,5);

# scaling back the rows of w #
   FOR j TO 8 DO w[3,j]*:= sqrt2; w[5,j]*:= sqrt2 OD ;
   FOR i TO 5
   DO REAL s= scale[i];
        FOR j TO 8 DO w[i,j]*:= s OD
   OD ;

  # send data to mass storage #  w
 END # of generate wghts # ;


 PROC generate sn = ( REF [,] POINT grid, REF [,] INT pos
                    ) MAT :
 BEGIN INT neq:= 0, MAT skn = genmat(16, 9);
    FOR row FROM 3 BY -1 TO 1
    DO FOR col TO 3
       DO IF pos[col, row] <= border
          THEN MAT ws = generate wghts(col, row, grid)[:2,];
             neq +:= 2;
             VEC sk1 = skn[neq - 1, ], sk2 = skn[neq, ],
                ws1 = ws[1, ], ws2 = ws[2, ],
            INT notj = (3 - row) * 3 + col, INT j:= 0;
            REF REAL sumx = sk1[notj]:= 0.0,
                        sumy = sk2[notj]:= 0.0;
            FOR ii TO 8
            DO j+:= 1; IF ii = notj THEN j+:= 1 FI ;
               sumx -:= (sk1[j]:=  ws1[ii]);
               sumy -:= (sk2[j]:=  ws2[ii])
            OD
         FI
      OD
   OD ;

   skn[ : neq, ]
END # generate sn #;


INT nsit := 0, avail := available,
[1 : 60] INT situation, ksit, rsit, [1 : 60] MAT sxy,
BOOL start:= TRUE , keep them:= TRUE , MAT w;
IF put data
THEN on physical file end(data,
      ( REF FILE f) BOOL :
      ( fail(2, "back ground memory exhausted"); fin )   )
FI ;
FOR k FROM kmin + 1 TO kmax - 1
DO REF [] INT postk = position [k, ],
      cposmk = ( put data ! cposmas[k, ] ! NIL ),
      cpossnk = ( put data ! cpossn[k, ] ! NIL );
   FOR r FROM rmin + 1 TO rmax - 1
   DO IF put data THEN cposmk[r]:= cpossnk[r]:= 0 FI ;
      IF postk[r] >= inside
      THEN IF start
         THEN MAT rm =
            generate wghts(2, 2, grid[k-1 : k+1, r-1 : r+1]);
            IF put data
            THEN cposmk[r]:= char number(data);
               putbin(data, rm)
            FI ;
            IF keepthem
            THEN mastor[k, r]:= rm; avail -:= 44;
               keepthem:= avail > 0;
               IF NOT (keepthem OR put data)
               THEN fail(1, "insufficient central memory"); fin
               FI
            FI ;
            IF uniform THEN w:= rm; start:= FALSE FI
         ELSE mastor[k,r]:= w
         FI ;
         IF postk[r] = in near b
         THEN REF [,] INT pos = position[k-1:k+1, r-1:r+1];
```

```
                    INT recogn = IF uniform
                       THEN recognize sit(pos, situation, nsit)
                       ELSE -1
                       FI ;
                    snstore[k, r]:=
                    IF recogn <= 0
                    THEN MAT rems =
                       generate sn(grid[k-1:k+1,r-1:r+1], pos);
                       IF recogn = 0
                       THEN ksit[nsit]:= k; rsit[nsit]:= r;
                          sxy[nsit]:= rems
                       FI ;
                       IF put data
                       THEN cpossnk[r]:= char number(data);
                          putbin(data, 1 UPB rems); putbin(data, rems)
                       FI ;
                       IF keep them THEN avail -:= (1 UPB rems) * 9
                             + 4; keep them:= avail > 0; rems
                       ELSE NIL FI
                    ELSE INT ks = ksit[recogn], rs = rsit[recogn];
                       IF put data
                       THEN cpossnk[r]:= cpossn[ks, rs] FI ;
                       sxy[recogn]
                    FI
                 FI
              FI
           OD
        OD
     END # compute data #;


     DISCARR dc = locd03zzb(dg, lfn); fin : SKIP ;
     IF erron THEN NIL ELSE dc FI
  END # of d03zzb # ;


     SKIP
  END # of Source Text #
```