Computers & Geosciences Vol. 16, No. 3, pp. 379–384, 1990
Printed in Great Britain.

0098-3004/90 $3.00 + 0.00

SHORT NOTE

A FORTRAN 77 SUBROUTINE FOR DETERMINING THE FRACTIONAL AREA OF RECTANGULAR GRID BLOCKS WITHIN A POLYGON

CLAYTON DEUTSCH
Computer Services, Placer Dome Inc., 1600–1055 Dunsmuir Street, Vancouver, B.C., Canada V7X 1P1

(*Received* 21 *July* 1989; *accepted* 29 *August* 1989)

INTRODUCTION

This Short Note presents a subroutine that calculates the fractional area of rectangular grid blocks that fall with a polygon. One application of this routine is to compute the average attribute of a polgon when the attribute is defined for a block model.

DESCRIPTION OF THE ALGORITHM

A rectangular grid network can be described by four parameters:

(1) *ymx* — the maximum *y* coordinate which corresponds to the upper boundary of the block with *y* index equal to 1.

(2) *ysz* — the size of the rectangular blocks in the *y* coordinate direction.

(3) *xmn* — the minimum *x* coordinate which corresponds to the lower boundary of the block with *x* index equal to 1.

(4) *xsz* — the size of the rectangular blocks in the *x* coordinate direction.

With this description of a grid network the *y* index increases with decreasing *y* and the *x* index increases with increasing *x*. Any coordinate point must fall in some block with a corresponding *x* and *y* index. The indices arbitrarily can be large or small depending on the coordinate, that is the *y* index is the integer portion of $(ymx - y)/ysz + 1$ and the *x* index is the integer portion of $(x - xmn)/xsz + 1$.

Figure 1 shows an example grid network, polygon, and the corresponding grid blocks of interest. The algorithm described here will determine what fraction of each block in the area of interest falls within the polygon. The polygon vertices may be ordered clockwise or counter-clockwise. The polygon should be closed (i.e. the first and last point should be the same).

If there are *np* vertices defining the polygon there are $np - 1$ vectors defining the sides of the polygon (a polygon must have at least three sides).The idea is to traverse around the outside of the polygon and accumulate all the area of grid blocks to the right of the vector. If the vector is pointing up (increasing *y*), then the area is positive and if the vector is pointing down (decreasing *y*) the area is accumulated as negative. Of course, if the vector is parallel to the *x* axis then no area is accumulated. Figure 2 shows an example of the areas that are added/subtracted for one vector. After completing the circuit around the polygon the intersection area of grid blocks within the polygon will be known.

The accumulation of the area to the right of a vector must be done in a number of swaths. If the vector crosses either an *x* or a *y* block boundary a new swath must be started. The different shaded areas on Figure 2 demonstrate the swaths for one vector. Special care is required when coding the algorithm to handle the vertices which exactly match block boundaries.

The polygon boundaries may cross but the user should note the following rules for accumulating the fractional areas:

(1) An area enclosed by vertices ordered clockwise will yield fractional blocks with positive areas.

(2) An area enclosed by vertices ordered counter-clockwise will yield fractional blocks with negative areas

(3) Any area that is enclosed more than once by vertices ordered clockwise will be accounted for twice (could yield blocks with an area > 1.0).

(4) Any block that is enclosed more than once by vertices ordered counter-clockwise also will be accounted for twice (could yield blocks with an area < -1.0).

(5) A block that has been enclosed both by clockwise and counter-clockwise vertices will have the sum of the areas.

The coding of the algorithm presented here will indicate whether there are both positive and negative

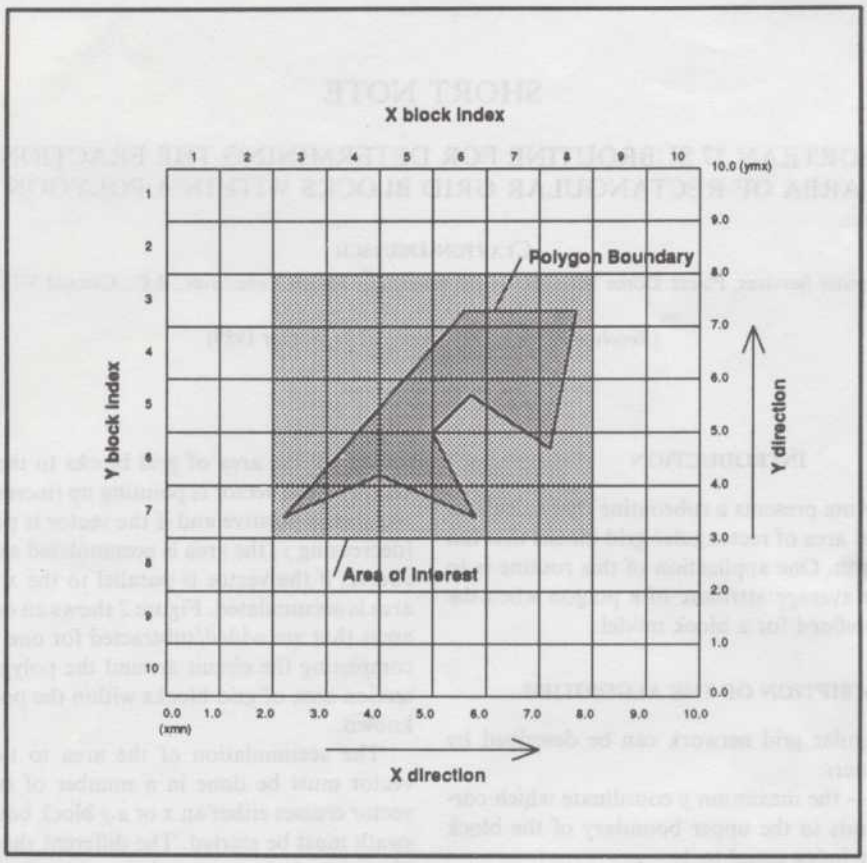

Figure 1. Example grid network, polygon, and corresponding grid blocks of interest.

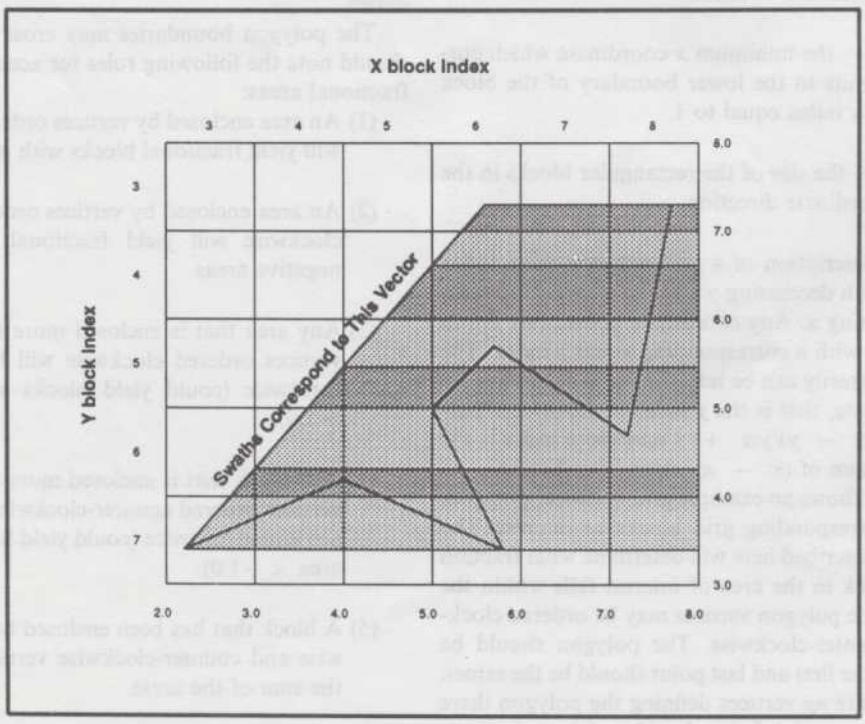

Figure 2. Example of areas that are added/subtracted for one vector.

blocks and whether there are blocks that have been enclosed more than once.

SUBROUTINE DESCRIPTION

The subroutine is written in standard FORTRAN 77 and does not require any external subroutines or functions. The calling arguments are as follows:

(1) *ymx* — the maximum *y* coordinate (as described).
(2) *ysz* — the size of the rectangular blocks in the *y* coordinate direction.
(3) *xmn* — the minimum *x* coordinate (as described).
(4) *xsz* — the size of the rectangular blocks in the *x* coordinate direction.
(5) *np* — the number of vertices describing the polygon.
(6) *xp* — a real (floating point) array containing the *x* coordinate of each vertex.
(7) *yp* — a real (floating point) array containing the *y* coordinate of each vertex.
(8) *maxgb* — this parameter is used to make sure that the size of output arrays *ix*, *iy*, and *fwb* are not exceeded.

Parameters 9–13 are output from the subroutine:

(9) *ngb* — the number of grid blocks that are in the area of interest (rectangular bounds of the polygon). The following three arrays will have *ngb* values ($ngb \leqslant maxgb$).
(10) *ix* — an integer array of the *x* block indices for the *ngb* blocks of interest.
(11) *iy* — an integer array of the *y* block indices for the *ngb* blocks of interest.
(12) *fwb* — a real (floating point) array of the fractional area within the *ngb* blocks of interest.
(13) *irt* — an error return code: = 1 the number of grid blocks in the area of interest exceeds max*gb* — ERROR SITUATION!; = 2 all fractional areas are positive (clockwise) and < 1 (nonoverlapping); = 3 all fractional areas are negative (counter-clockwise) and > -1 (nonoverlapping); = 4 fraction areas are positive and negative but nonoverlapping; = 5 fractional areas outside of [-1.0, 1.0] exist (overlapping).

After calling the subroutine with a description of a polygon the five output variables completely describe the grid blocks within the polygon. Negative areas indicate that the polygon was ordered counter-clockwise (take the absolute value of the area). If a grid block is completely within the polygon then *fwb*(*i*) will be 1.0, half in the polygon then *fwb*(*i*) will be 0.5, and completely outside the polygon then *fwb* (*i*) will be 0.0. The block location of *i* (where *i* is between 1 and *ngb*) is given by the *x* and *y* indices *ix* (*i*) and *iy* (*i*). The total area enclosed by the polygon is given by the sum of *fwb*(*i*) from $i = 1$ to $i = ngb$.

Table 1. Fractional areas of grid blocks within polygon shown on Figure 1

i	*ix*(*i*)	*iy*(*i*)	*fwb*(*i*)	*i*	*ix*(*i*)	*iy*(*i*)	*fwb*(*i*)
1	3	3	0.000	16	6	5	0.575
2	4	3	0.000	17	7	5	0.813
3	5	3	0.000	18	8	5	0.354
4	6	3	0.159	19	3	6	0.044
5	7	3	0.300	20	4	6	0.752
6	8	3	0.201	21	5	6	0.955
7	3	4	0.000	22	6	6	0.250
8	4	4	0.000	23	7	6	0.021
9	5	4	0.163	24	8	6	0.055
10	6	4	0.934	25	3	7	0.181
11	7	4	1.000	26	4	7	0.067
12	8	4	0.546	27	5	7	0.067
13	3	5	0.000	28	6	7	0.248
14	4	5	0.094	29	7	7	0.000
15	5	5	0.875	30	8	7	0.000

The subroutine assumes the polygon is closed. Erroneous results will be obtained for polygons that are not closed. Depending on *xp* and *yp* the indices of the blocks in the area of interest may be zero, negative, or large. A quick check in *ix* (1), *iy* (1), and *ix* (*ngb*), *iy* (*ngb*) will indicate the bounds of the polygon. A number of comments in the code indicate places to improve the efficiency of the subroutine.

EXAMPLE

The polygon shown in Figure 1 can be described by $np = 9$ with the following coordinates for *xp* and *yp*:

i	*xp*(*i*)	*yp*(*i*)
1	2.2	3.4
2	5.6	7.3
3	7.7	7.3
4	7.2	4.7
5	5.7	5.7
6	5.0	5.0
7	5.8	3.4
8	4.0	4.2
9	2.2	3.4

The subroutine *farea* will return $ngb = 30$, $irt = 2$, and the values shown in Table 1. A mathematical check of the areas calculated by *farea* confirms that the result is correct.

Acknowledgment—The author would like to thank the management of Placer Dome Inc. for allowing publication of this subroutine.

Appendix overleaf

APPENDIX

Program Listing

```
      subroutine farea(ymx,ysz,xmn,xsz,np,xp,yp,maxgb,ngb,ix,iy,fwb,irt)
c-----------------------------------------------------------------------
c           Fractional Area of Grid Blocks within Polygons
c           **********************************************
c
c This subroutine computes the intersection of a grid block network with
c  a polygon. The fraction of each grid block within the polygon is
c  computed exactly.
c
c INPUT:   ymx      the Y origin of the grid system. This is the
c                   maximum (upper boundary) of the Y direction
c          ysz      the size of the grid blocks in the Y direction
c          xmn      the X origin of the grid system. This is the
c                   minimum (left hand boundary) of the X direction
c          xsz      the size of the grid blocks in the X direction
c          np       the number of points defining the polygon
c          xp()     the array of the X coordinates of the vertices
c          yp()     the array of the Y coordinates of the vertices
c          maxgb    the maximum allowable number of elements that
c                   output arrays ix,iy,fwb are dimensioned for.
c
c
c OUTPUT:  ngb      the number of grid blocks in the area of interest
c          ix()     the X index of the blocks in the area of interest
c          iy()     the Y index of the blocks in the area of interest
c          fwb()    the fraction of the grid block within the polygon
c          irt      an error return code:
c                    =1  the number of grid blocks in the area of
c                        interest exceeds maxgb - ERROR SITUATION!
c                    =2  all fractional areas are positive (clockwise)
c                        and less than 1 (non-overlapping)
c                    =3  all fractional areas are negative (counter-
c                        clockwise) and greater than -1 (non-overlapping)
c                    =4  fractional areas are positive and negative
c                        but non-overlapping [-1.0,1]
c                    =5  fractional areas outside of [-1.0,1.0] exist
c
c NOTES:
c
c       1. The grid network is defined by the maximum Y coordinate and
c          the minimum X coordinate. The X index increases with
c          increasing X and the Y index increases with decresing Y.
c
c       2. This routine does not check for negative grid block indices.
c
c       3. The polygon must be closed. A modification should be added
c          to the code if non-closed polygons are to be considered.
c
c       4. The parameter EPSLON is used to test for equality on block
c          boundaries and to avoid very small denominators.
c
c-----------------------------------------------------------------------
      parameter(EPSLON=0.00001)
      real      xp(*),yp(*),fwb(*)
      integer   ix(*),iy(*)
      logical   pos,neg,one
c
c Determine the extent of the grid area covered by the polygon:
c
      ypmin = yp(1)
      ypmax = yp(1)
      xpmin = xp(1)
      xpmax = xp(1)
      do 1 i=2,np-1
            if(xp(i).lt.xpmin) xpmin = xp(i)
            if(xp(i).gt.xpmax) xpmax = xp(i)
            if(yp(i).lt.ypmin) ypmin = yp(i)
            if(yp(i).gt.ypmax) ypmax = yp(i)
 1    continue
      nlx   = int(((xpmin-xmn)/xsz)) + 1
      nux   = int(((xpmax-xmn)/xsz)) + 1
      nly   = int(((ymx-ypmin)/ysz)) + 1
      nuy   = int(((ymx-ypmax)/ysz)) + 1
      ngb   = (nux - nlx + 1 ) * (nly - nuy + 1 )
      if(ngb.gt.maxgb) then
            write(*,*) 'ERROR: require larger arrays for FAREA'
            write(*,*) '       require: ',ngb,'available: ',maxgb
```

```
                irt = 1
                return
        endif
c
c Initialize the grid area used. NOTE: The parameters nlx,nux,nly,nuy
c  completely define the arrays ix and iy. One place where storage
c  space could be saved is to pass nlx,nux,nly,nuy to the calling
c  program and perform the following indexing in the calling program.
c
        igb = 0
        do 2 j=nuy,nly
                do 3 i=nlx,nux
                        igb      = igb + 1
                        ix(igb)  = i
                        iy(igb)  = j
                        fwb(igb) = 0.0
 3              continue
 2      continue
c
c MAIN LOOP over the "np" vectors:
c
        do 4 ip=1,np-1
c
c Determine whether going up or down (relative to Y direction):
c
        if((yp(ip+1)-yp(ip)).ge.0.0) then
                sign =  1.0
                xpt  = xp(ip)
                ypt  = yp(ip)
                xe   = xp(ip+1)
                ye   = yp(ip+1)
        else
                sign = -1.0
                xpt  = xp(ip+1)
                ypt  = yp(ip+1)
                xe   = xp(ip)
                ye   = yp(ip)
        endif
c
c SECONDARY LOOP along vector (if no Y change then go to next vector):
c
 5      continue
        if((abs(ye-ypt)).lt.EPSLON) go to 4
c
c       try to go up to next Y grid line without passing end of vector:
c
        ypt2 = ymx - real(int((ymx-ypt)/ysz-EPSLON))*ysz
        if(ypt2.gt.ye) ypt2 = ye
        xpt2 = xpt + ((ypt2-ypt)/(ye-ypt))*(xe-xpt)
c
c       make sure that an X grid line is not crossed:
c
        isp = int((xpt -xmn)/xsz+EPSLON) + 1
        isn = int((xpt -xmn)/xsz-EPSLON) + 1
        iep = int((xpt2-xmn)/xsz+EPSLON) + 1
        ien = int((xpt2-xmn)/xsz-EPSLON) + 1
        if((xpt.lt.xpt2.and.isp.ne.ien).or.
     +     (xpt.gt.xpt2.and.isn.ne.iep)) then
                if(xpt.lt.xpt2) xpt2 = xmn+real((isp  ))*xsz
                if(xpt.gt.xpt2) xpt2 = xmn+real((isn-1))*xsz
                ypt2 = ypt + ((xpt2-xpt)/(xe-xpt))*(ye-ypt)
        endif
c
c       add (or subtract) partial block contribution to total grade:
c
        xmd  = 0.5*(xpt+xpt2)
        xcor = ((xmn+real(int((xmd+xsz-xmn)/xsz))*xsz)-xmd)/xsz
        ycor = abs((ypt2-ypt)/ysz)
        area = sign*xcor*ycor
        ixbl = int((xmd-xmn)/xsz) + 1
        iybl = int((ymx-ypt)/ysz-EPSLON) + 1
        ind  = (ixbl-nlx+1) + (iybl-nuy)*(nux-nlx+1)
        fwb(ind) = fwb(ind) + area
c
c       add (or subtract) full blocks up to eastern boundary:
c
        if(ixbl.lt.nux) then
                area = sign*ycor
                do 6 j=1,(nux-ixbl)
                        xmd  = xmd + xsz
                        ixbl = int((xmd-xmn)/xsz) + 1
                        iybl = int((ymx-ypt)/ysz-EPSLON) + 1
```

```
                        ind  = (ixbl-nlx+1) + (iybl-nuy)*(nux-nlx+1)
                        fwb(ind) = fwb(ind) + area
 6              continue
        endif
c
c       if current vector is not finished - continue to step along it:
c
        if(abs(xpt2-xe).gt.EPSLON.or.abs(ypt2-ye).gt.EPSLON) then
                xpt = xpt2
                ypt = ypt2
                go to 5
        endif
c
c END MAIN LOOP over the "np" vectors:
c
 4      continue
c
c Determine the return code. NOTE: for specific applications the user
c  may not care that the polygon is clockwise or counter-clockwise in
c  which case one could take the absolute value of fwb here and also
c  ensure that the fwb(i) is between 0.0 and 1.0:
c
        pos = .false.
        neg = .false.
        one = .false.
        do 7 i = 1,ngb
                if(fwb(i).gt.0.0)       pos = .true.
                if(fwb(i).lt.0.0)       neg = .true.
                if(abs(fwb(i)).gt.1.0) one = .true.
 7      continue
                        irt = 2
        if(neg)         irt = 3
        if(neg.and.pos) irt = 4
        if(one)         irt = 5
        return
        end
```