

Cranfield Institute of Technology

Department of Applied Computing and Mathematics

Ph.D. THESIS

Academic Year 1992-1993

Karam Ali Fayed

Computational Methods for Image Rotation and Computer Aided Tomography

Volume II: Computer Software

Supervisor:

Dr. J. M. Blackledge

January 1993



IMAGING SERVICES NORTH

Boston Spa, Wetherby
West Yorkshire, LS23 7BQ
www.bl.uk

**TEXT CUT OFF IN THE
ORIGINAL**

Software Package for Image Rotation and Reconstructions

A1. Introduction

The aim of this software package is to use a set of routines as the basis for the development of image rotation and reconstruction package which is both user friendly and versatile in the sense that it can be expanded as more digital image processing and reconstruction routines become available.

Figure A shows a tree structure of the software package. This package is divided into four appendices (Appendix A, B, C, D) which contains the FORTRAN source code for different application, i.e. rotation operation, filtering operations, fast Radon transform and 1D process, and reconstruction operations.

Appendix A lists the rotation routines which rotate the image by an angle through 360° . It also lists a set of digital filter routines used to remove the noise from the rotated image and to enhance the quality of the image.

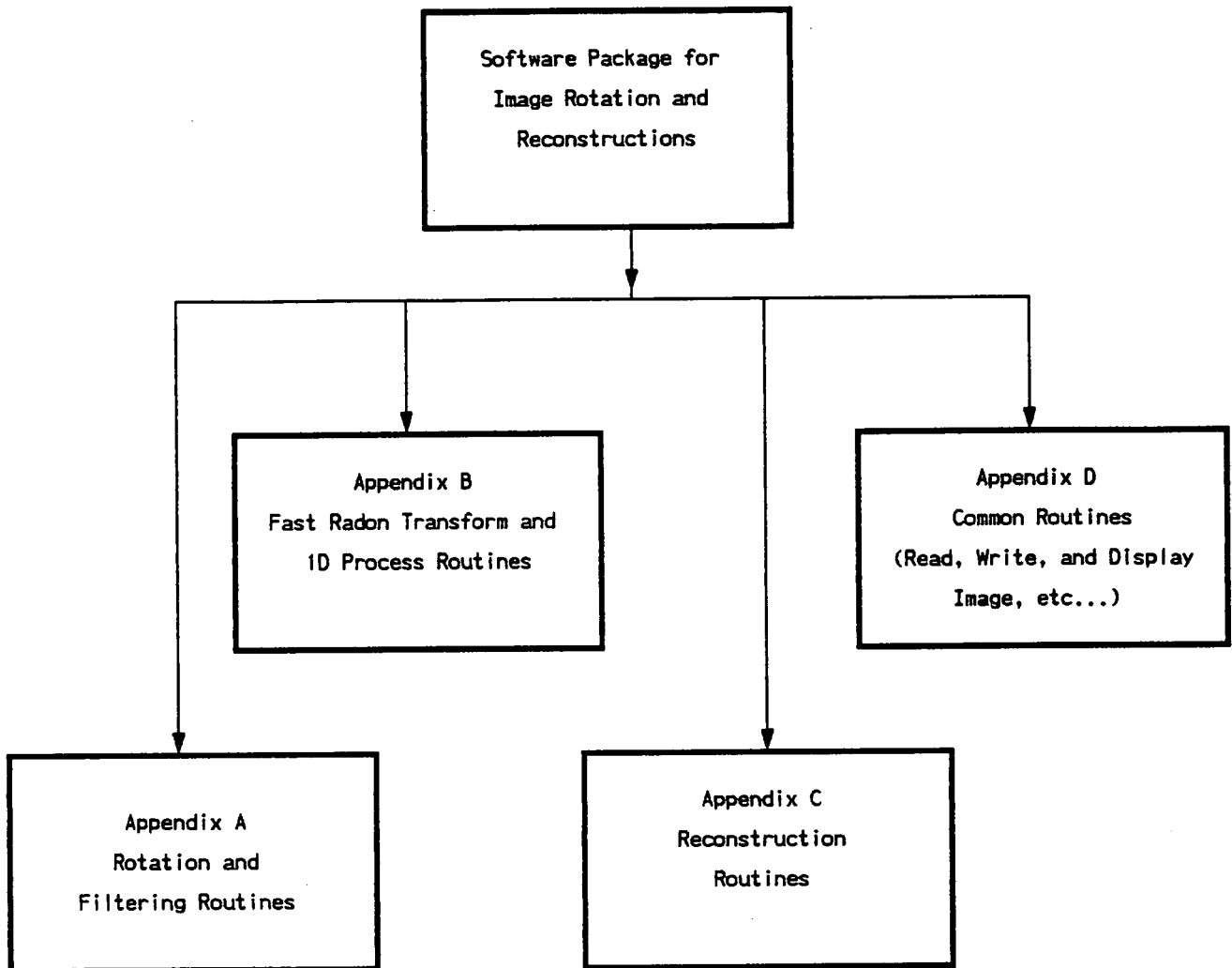
Appendix B lists the subroutines used for computing fast Radon transform and 1D process operation.

Appendix C lists the subroutines used for image reconstruction.

Appendix D lists the common subroutines used for the aforementioned appendices (A-C). These subroutines are the digital image processing routines. The routines designed for reading, writing and displaying images.

A2. Software Used

Figure A The tree structure of the software package



The software used was written in DEC FORTRAN using standard F77. Extensive use was made of UNIRAS-GIMAGE graphics library (Version 6.3) giving a 4-bit image.

APPENDIX A

ROTATION AND FILTERING ROUTINES

```

program Fayed Rotate.for
real a(256,256),xy(256,256),xx(256,256),f(256,256)
real yy(256,256),f1(256,256)
character*6 answer

```

FUNCTION: Generates an image on a DEC digital workstation using data from a specified data file or produces a plot file.Also, rotate the object at different angles using 30 techniques

EXTERNAL SUBROUTINES: UNIRAS Library - groute
gopen
gclose

DIPLIB - read_image, timer_start, time_elapsed,
display_image, 30:Rotation_Techniques

```

real p(256,256)
Select route for gimage.

```

```

write(6,10)
format(1x,'View image (1) or generate plotfile (2)?'$)
read(5,*)id
if(id.eq.1)then
call groute('s mx11; e') !for VAX workstations 2000 or 3200
endif

```

```

if(id.eq.2)then
write(6,11)
format(1x,'Black and white (1) or colour (2) plotfile?'$)
read(6,*)ic
if(ic.eq.1)then
call groute('s hposta41; e')
endif
if(ic.eq.2)then
call groute('s ht4692; e')
call groute('s hqcs100a4; e')
endif
if((ic.ne.1).and.(ic.ne.2))goto 2
endif

```

```

if((id.ne.1).and.(id.ne.2))goto 1

```

```

call gopen !open UNIRAS window

```

Ask user for image parameters:

1. size of image
2. colour coding with options

continue !for looping back to examine another data file.

Open data file and read contents by calling
subroutine read_image.

```

write(6,*)'enter your choice:'
write(6,*)'1. read_image'
write(6,*)'2. Generate_rectangle'
write(6,*)'3. Generate_circle'
write(6,*)'4. Generate_var_rectangle'
read(5,*) k1
if (k1.eq.1) then
call read_image(a,n)
endif
if (k1.eq.2) then

```

```

write(6,*)'enter size of square image'
read(5,*)n
call rectangle(a,n)
call write_image(a,n)
endif
if (kl.eq.3) then
write(6,*)'enter size of square image'
read(5,*)n
call circle(a,n)
call write_image(a,n)
endif
if (kl.eq.4) then
write(6,*)'enter size of square image'
read(5,*)n
call var_rectangle(a,n)
call write_image(a,n)
endif

```

continue !for looping back to examine a different colour code.

```

write(6,*)'      UNIRAS 6 colour scales'
write(6,*)
write(6,*)'Default colour scale 1..... (1)'
write(6,*)'Default colour scale 2..... (2)'
write(6,*)'Grey scale..... (3)'
write(6,*)'Topographic colour scale..... (4)'
write(6,*)'Seismic colour scale..... (5)'
write(6,*)
write(6,80)
format(1x,'Scale (1-5)?'$)
read(5,*)icol
write(6,90)
format(1x,'Positive (1) or negative (-1)?'$)
read(5,*)ipol

```

Generate image using subroutine display_image.

```

call display_image_grey(a,n,icol,ipol)
call greset
call gclear

m=nint(1.414213562*n)
write(6,*)'enter theta value in degree = '
read(5,*)theta
write(6,*)'enter your choise for rotation'
write(6,*)' 1. Nearest neighbours rotation algorithm'
write(6,*)' 2. Interpolation algorithm '
write(6,*)' 3. Clipping Nearest neighbours '
write(6,*)' 4. Milt_neighbour interpolation algorithm '
read(5,*) ix
if((ix.ne.1).and.(ix.ne.2).and.(ix.ne.3).and.(ix.ne.4))goto 20
if (ix.eq.1) then
write(6,*)' Optimum rotation algorithm :'
write(6,*)'1. Optimum nearest neighbor pixel'
write(6,*)'8. Optimum 2 row neighbor pixel'
write(6,*)'10. Optimum one row neighbor pixel'
write(6,*)'2. Optimum four_ neighbor pixels'
write(6,*)'3. Optimum eight_ neighbor pixels '
write(6,*)' Normal rotation algorithm :'
write(6,*)'4. Normal nearest neighbor pixel'
write(6,*)'5. Normal four_ neighbor pixels'
write(6,*)'6. Normal eight_ neighbor pixels '
write(6,*)'7. Optimum four neighbors modify '
write(6,*)'9. Three pass rotation '
read(5,*) nfn
if (nfn.eq.1) then

```

Send message to user 'Optimum nearest neighbour rotation...'

```

write(6,21)
1  format(1x,'Optimum nearest neighbour rotation...')
    call timer_start(context)
    call rotate_nearest_neig(a,n,xy,m,theta)
    call time_elapsed(context)
    call write_image(xy,m)
    open (unit=1,file='rn.dat',status='new')
2  format(4(1x,f14.4))
    write(1,42)((xy(j,i),j=1,181),i=1,181)
    close(1)
    endif
    if (nfn.eq.2) then
Send message to user 'Optimum four neighbours rotation...'

    write(6,22)
2  format(1x,'Optimum four neighbours rotation...')
    call timer_start(context)
    call rotate_four_neig(a,n,xy,m,theta)
    call time_elapsed(context)
    call write_image(xy,m)
    endif
    if (nfn.eq.3) then
Send message to user 'Optimum eight neighbours rotation...'

    write(6,23)
3  format(1x,'Optimum eight neighbours rotation...')
    call timer_start(context)
    call rotate_eight_neig(a,n,xy,m,theta)
    call time_elapsed(context)
    call write_image(xy,m)
    endif
    if (nfn.eq.4) then
Send message to user 'Normal nearest neighbour rotation...'

    write(6,24)
4  format(1x,'Normal nearest neighbour rotation...')
    call timer_start(context)
    call rotate_nearest(a,n,xy,m,theta)
    call time_elapsed(context)
    call write_image(xy,m)
    endif
    if (nfn.eq.5) then
Send message to user 'Normal four neighbours rotation...'

    write(6,25)
5  format(1x,'Normal four neighbours rotation...')
    call timer_start(context)
    call rotate_four(a,n,xy,m,theta)
    call time_elapsed(context)
    call write_image(xy,m)
    endif
    if (nfn.eq.6) then
Send message to user 'Normal eight neighbours rotation...'

    write(6,26)
6  format(1x,'Normal eight neighbours rotation...')
    call timer_start(context)
    call rotate_eight(a,n,xy,m,theta)
    call time_elapsed(context)
    call write_image(xy,m)
    endif
    if (nfn.eq.7) then
Send message to user 'Optimum four neighbours modify...'

    write(6,47)
47  format(1x,'Optimum four neighbours modify...')
    call timer_start(context)
    call rotate_four_modify(a,n,xy,m,theta)

```

```

        call time_elapsed(context)
        call write_image(xy,m)
    endif
    if (nfn.eq.8) then
Send message to user 'Optimum 2 row neighbours ...'

        write(6,46)
6        format(1x,'Optimum 2 row neighbours ...'$)
            call timer_start(context)
            call rotate_2rp_neig(a,n,xy,m,theta)
            call time_elapsed(context)
            call write_image(xy,m)
            endif
            if (nfn.eq.9) then
Send message to user 'Three pass rotation ...'

            write(6,48)
8            format(1x,'Three pass rotation ...'$)
                call timer_start(context)
                call rotate_three_pass(a,n,xy,m,theta)
                call time_elapsed(context)
                call write_image(xy,m)
                endif
                if (nfn.eq.10) then
Send message to user 'Optimum one row neighbours ...'

                write(6,49)
9                format(1x,'Optimum one row neighbours ...'$)
                    call timer_start(context)
                    call rotate_lrp_neig(a,n,xy,m,theta)
                    call time_elapsed(context)
                    call write_image(xy,m)
                    endif
                    elseif (ix.eq.2) then
                        write(6,*) ' Forward interpolation algorithm : '
                        write(6,*) ' 1. forward bilinear interpolation '
                        write(6,*) ' 2. Forward triangle interpolation '
                        write(6,*) ' 3. Forward 2 Dim.linear interpolation '
                        write(6,*) ' Reverse interpolation algorithm : '
                        write(6,*) ' 4. Reverse bilinear interpolation '
                        write(6,*) ' 5. Reverse triangle interpolation '
                        write(6,*) ' 6. Reverse 2dim. linear interpolation '
                        read(5,*) nfn
                        if (nfn.eq.1) then
Send message to user 'Forward bilinear interpolation...'

                            write(6,27)
7                            format(1x,'Forward bilinear interpolation...'$)
                                call timer_start(context)
                                call rotate_rectangle_sum(a,n,xy,m,theta)
                                call time_elapsed(context)
                                call write_image(xy,m)
                                endif
                                if (nfn.eq.2) then
Send message to user 'Forward triangle interpolation...'

                                    write(6,28)
28                                    format(1x,'Forward triangle interpolation...'$)
                                        call timer_start(context)
                                        call rotate_triangle_sum(a,n,xy,m,theta)
                                        call time_elapsed(context)
                                        call write_image(xy,m)
                                        endif
                                        if (nfn.eq.3) then
Send message to user 'Forward 2D linear interpolation...'

                                            write(6,29)
29                                            format(1x,'Forward 2D linear interpolation...'$)

```

```

        call timer_start(context)
        call rotate_2dlinear_sum(a,n,xy,m,theta)
        call time_elapsed(context)
        call write_image(xy,m)
    endif
    if (nfn.eq.4) then
Send message to user 'Reverse bilinear interpolation...'

        write(6,30)
        format(1x,'Reverse bilinear interpolation...'$)
        call timer_start(context)
        call rotate_divide_rectangle_sum(a,n,xy,m,theta)
        call time_elapsed(context)
        call write_image(xy,m)
    endif
    if (nfn.eq.5) then
Send message to user 'Reverse triangle interpolation...'

        write(6,31)
        format(1x,'Reverse triangle interpolation...'$)
        call timer_start(context)
        call rotate_divide_triangle_sum(a,n,xy,m,theta)
        call time_elapsed(context)
        call write_image(xy,m)
    endif
    if (nfn.eq.6) then
Send message to user 'Reverse 2D linear interpolation...'

        write(6,32)
        format(1x,'Reverse 2D linear interpolation...'$)
        call timer_start(context)
        call rotate_divide_2dlinear_sum(a,n,xy,m,theta)
        call time_elapsed(context)
        call write_image(xy,m)
    endif
    elseif(ix.eq.3) then
        write(6,*) '1. Clipping Optimum nearest neighbour '
        read(5,*) nfn
        if (nfn.eq.1) then
Send message to user 'Clipping Optimum nearest neighbour...'

            write(6,45)
            format(1x,'Clipping Optimum nearest neighbour...'$)
            call timer_start(context)
            call rotate_no(a,n,theta)
            call time_elapsed(context)
            call display_image_grey(a,n,icol,ipol)
            call write_image(a,n)
        endif
        elseif (ix.eq.4) then
            write(6,*) ' Multi neighbours Interpolation :'
            write(6,*) ' 1. Forward Multi neighbours Interpolation'
            write(6,*) ' 2. Reverse Multi neighbours Interpolation'
            read(5,*) fr
            if (fr.eq.1) then
                write(6,*) '1. Forward rectangle_12 neighbours '
                write(6,*) '2. Forward triangle_12 neighbors '
                write(6,*) '3. Forward 2d linear_12 neighbours'
                write(6,*) '4. Forward rectangle_16 neighbours '
                write(6,*) '5. Forward triangle_16 neighbors '
                write(6,*) '6. Forward 2dlinear_16 neighbours'
                write(6,*) '7. Forward triangle_16 neighbors(aother)'
                read(5,*) nfn
                if (nfn.eq.1) then
Send message to user 'Farward rectangle 12 neigh. interpolation...'

                    write(6,33)
                    format(1x,'Forward rectangle 12 neigh. interpolation...'$)

```

```

        call timer_start(context)
        call rotate_rectangle_12neigh_sum(a,n,xy,m,theta)
        call time_elapsed(context)
        call write_image(xy,m)
    endif
    if (nfn.eq.2) then
Send message to user 'Farward triangle 12 neigh. interpolation...'

        write(6,34)
4        format(1x,'Forward triangle 12 neigh. interpolation...'$)
            call timer_start(context)
            call rotate_triangle_12neigh_sum(a,n,xy,m,theta)
            call time_elapsed(context)
            call write_image(xy,m)
        endif
        if (nfn.eq.3) then
Send message to user 'Farward linear 12 neigh. interpolation...'

        write(6,134)
34        format(1x,'Forward linear 12 neigh. interpolation...'$)
            call timer_start(context)
            call rotate_linear_12neigh_sum(a,n,xy,m,theta)
            call time_elapsed(context)
            call write_image(xy,m)
        endif
        if (nfn.eq.4) then
Send message to user 'Farward rectangle 16 neigh. interpolation...'

        write(6,35)
5        format(1x,'Forward rectangle 16 neigh. interpolation...'$)
            call timer_start(context)
            call rotate_rectangle_16neigh_sum(a,n,xy,m,theta)
            call time_elapsed(context)
            call write_image(xy,m)
        endif
        if (nfn.eq.5) then
Send message to user 'Farward triangle 16 neigh. interpolation...'

        write(6,36)
6        format(1x,'Forward triangle 16 neigh. interpolation...'$)
            call timer_start(context)
            call rotate_triangle_16neigh_sum(a,n,xy,m,theta)
            call time_elapsed(context)
            call write_image(xy,m)
        endif
        if (nfn.eq.6) then
Send message to user 'Farward linear 16 neigh. interpolation...'

        write(6,135)
35        format(1x,'Forward linear 16 neigh. interpolation...'$)
            call timer_start(context)
            call rotate_linear_16neigh_sum(a,n,xy,m,theta)
            call time_elapsed(context)
            call write_image(xy,m)
        endif
        if (nfn.eq.7) then
Send message to user 'Farward triangle 16 neigh. interpolation...'

        write(6,37)
37        format(1x,'Forward triangle 16 neigh.(other) interpo...'$)
            call timer_start(context)
            call rotate_triangle_16neigh_sum1(a,n,xy,m,theta)
            call time_elapsed(context)
            call write_image(xy,m)
        endif
        elseif (fr.eq.2) then
            write(6,*) ' Reverse Multi neighbours Interpolation :'
            write(6,*) ' 1. Reverse rectangle_12neighbours '

```



```

        write(6,*) '2. Reverse triangle_12 neighbor '
        write(6,*) '3. Reverse 2d linear_12neighbours'
        write(6,*) '4. Reverse rectangle_16neighbours '
        write(6,*) '5. Reverse triangle_16 neighbors'
        write(6,*) '6. Reverse 2dlinear_16neighbours'
        write(6,*) '7. Reverse triangle_16 neighbor(another) '
        write(6,*) '8. Quasi_nearest_rotation'
        read (5,*) nfn
        if (nfn.eq.1) then
Send message to user 'Reverse rectangle 12 neigh. interpolation...'

        write(6,38)
8        format(1x,'Reverse rectangle 12 neigh. interpolation...'$)
            call timer_start(context)
            call rotate_divide_rectangle12n_sum(a,n,xy,m,theta)
            call time_elapsed(context)
            call write_image(xy,m)
        endif
        if (nfn.eq.2) then
Send message to user 'Reverse triangle 12 neigh. interpolation...'

        write(6,39)
9        format(1x,'Reverse triangle 12 neigh. interpolation...'$)
            call timer_start(context)
            call rotate_divide_triangle12n_sum(a,n,xy,m,theta)
            call time_elapsed(context)
            call write_image(xy,m)
        endif
        if (nfn.eq.3) then
Send message to user 'Reverse linear 12 neigh. interpolation...'

        write(6,139)
39       format(1x,'Reverse linear 12 neigh. interpolation...'$)
            call timer_start(context)
            call rotate_divide_linear12n_sum(a,n,xy,m,theta)
            call time_elapsed(context)
            call write_image(xy,m)
        endif
        if (nfn.eq.4) then
Send message to user 'Reverse rectangular 16 neigh. interpolation...'

        write(6,141)
41       format(1x,'Reverse rectangular 16 neigh. interpolation...'$)
            call timer_start(context)
            call rotate_divide_rectangle16n_sum(a,n,xy,m,theta)
            call time_elapsed(context)
            call write_image(xy,m)
        endif
        if (nfn.eq.5) then
Send message to user 'Reverse triangle 16 neigh. interpolation...'

        write(6,40)
0        format(1x,'Reverse triangle 16 neigh. interpolation...'$)
            call timer_start(context)
            call rotate_divide_triangle16n_sum(a,n,xy,m,theta)
            call time_elapsed(context)
            call write_image(xy,m)
        endif
        if (nfn.eq.6) then
Send message to user 'Reverse linear 16 neigh. interpolation...'

        write(6,140)
40       format(1x,'Reverse linear 16 neigh. interpolation...'$)
            call timer_start(context)
            call rotate_divide_linear16n_sum(a,n,xy,m,theta)
            call time_elapsed(context)
            call write_image(xy,m)
        endif

```

```

        if (nfn.eq.7) then
Send message to user 'Reverse triangle 16 neigh.(oth) interpolation...'

        write(6,41)
1        format(1x,'Reverse triangle 16 neigh.(other) interpol...'$)
            call timer_start(context)
            call rotate_divide_triangle16n_sum1(a,n,xy,m,theta)
            call time_elapsed(context)
            call write_image(xy,m)
        endif
        if (nfn.eq.8) then
Send message to user 'quasi_nearest_neig.rotation...'

        write(6,97)
7        format(1x,'Quasi_nearest_rotation...'$)
            call timer_start(context)
            call rotate_nearest_quasi(a,n,xy,m,theta)
            call time_elapsed(context)
            call write_image(xy,m)
        endif
        else
        endif
    else
    end if

    call display_image_grey(xy,m,icol,ipol)
    call compare_rotate_image(xy,m)

Ask user if another colour code is required.

        write(6,104)
104        format(1x,'Choose another colour code? <n>'$)
            read(5,990) answer
990        format(a6)
            if((answer.eq.'yes').or.(answer.eq.'y')) goto 4
ask user if image filtering is required.
        write(6,*) 'filter the image <Y/N>?'
            read(5,1000) r
000        format(a1)
            if ((r.eq.'Y') .or. (r.eq.'y')) then
93        write(6,*) 'window size'
            read(5,*) nw
            do i=1,nw
            do j=1,nw
            p(j,i)=1.
            enddo
            enddo

                open (unit=1,file='test.128',status='new')
1140                format(4(1x,f14.4))
                write(1,1140)((a(j,i),j=1,128),i=1,128)
                close(1)

                open (unit=1,file='rotate.181',status='new')
1141                format(4(1x,f14.4))
                write(1,1141)((xy(j,i),j=1,181),i=1,181)
                close(1)

        write(6,*) '1. Fir convolution filter image'
        write(6,*) '2. Moving average filter image'
        write(6,*) '3. Median filter image'
        write(6,*) '4. Threshold filter image'
        read(5,*) ty
        if (ty.eq.1) then
            call timer_start(context)
            call fir_convolution_filter_image(xy,p,f,m,nw)
            call time_elapsed(context)
                open (unit=1,file='fir.181',status='new')
1142                format(4(1x,f14.4))

```

```

        write(1,1142)((f(j,i),j=1,181),i=1,181)
        close(1)
    call greset
    call gclear
    call display_image_grey(f,m,icol,ipol)
    call compare_rotate_image(f,m)
endif
if (ty.eq.2) then
    call timer_start(context)
    call moving_average_filter_image(xy,m,f1,nw)
    call time_elapsed(context)
1143        open (unit=1,file='maf.181',status='new')
            format(4(1x,f14.4))
            write(1,1143)((f1(j,i),j=1,181),i=1,181)
            close(1)
    call greset
    call gclear
    call display_image_grey(f1,m,icol,ipol)
    call compare_rotate_image(f1,m)
endif
if (ty.eq.3) then
    call timer_start(context)
    call median_filter_image(xy,m,f1,nw)
    call time_elapsed(context)
1144        open (unit=1,file='mf.181',status='new')
            format(4(1x,f14.4))
            write(1,1144)((f1(j,i),j=1,181),i=1,181)
            close(1)
    call greset
    call gclear
    call display_image_grey(f1,m,icol,ipol)
    call compare_rotate_image(f1,m)
endif
if (ty.eq.4) then
    call timer_start(context)
    call threshold_filter_image(xy,m,f1)
    call time_elapsed(context)
    call greset
    call gclear
    call display_image_grey(f1,m,icol,ipol)
    call compare_rotate_image(f1,m)
endif
else
endif
write(6,*)'Another filter <Y/N>?'
read(5,992) r
92    format(a1)
    if ((r.eq.'Y') .or. (r.eq.'y')) then
        goto 993
    else
    endif
Ask user if another rotation is required.
    write(6,*)'rotate again <Y/N>?'
    read(5,991) r
91    format(a1)
    if ((r.eq.'Y') .or. (r.eq.'y')) then
        goto 100
    else
    endif
Ask user if another display is required.
    write(6,1001)
001    format(1x,'Display another? <n>'$)
    read(5,1002) answer
002    format(a6)
    if((answer.eq.'yes').or.(answer.eq.'y')) goto 5
c
c Terminate uniras.

```

Write out response to user indicating next stage for producing
a hardcopy (in the case of option for generating a plotfile).

```
if(ic.eq.1)then  
write(6,*)'Your plot file is called lps40.plot'  
endif
```

```
if(ic.eq.2)then  
write(6,*)'Your plot file is called t4510.plot'  
endif
```

```
call greset  
call gclear  
call gclose  
stop  
end
```

subroutine rotate nearest(a,n,b,m,theta)

real a(256,256),b(256,256),w(256,256),xy(256,256)

FUNCTION : This subroutine rotates an n X n array a
through an angle theta (given in degree)
anticlockwise placing the result at the
center of an array mxm b

PARAMETERS: a _ Input image of n X n size
b _ Rotated image of m X m size

EXTERNAL SUBROUTINE :None

INTERNAL SUBROUTINE :None

convert theta into radians

pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)

initial b

do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) = 0.
enddo
enddo

compute new coordinates (xd,yd) of rotate array

nn = n/2 + 1
mm = m/2 + 1

start process

do j = 1,n
do i = 1,n

generate 4-quadrant cartesian plan

y = float(j-nn)
x = float(i-nn)
xd = x * costh - y * sinth
yd = x * sinth + y * costh

compute both real and integer coordinates of array b
associated with the above transformation for later use

kx = mm+ nint(xd)
ky = mm+ nint(yd)

assign value of a at (j,i) to the nearest_point cell
of array b in wich the transformed coordinates
lie.

b(ky,kx) = a(j,i)

enddo
enddo
return
end

```

subroutine rotate nearest neig (a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
integer xu,yu,xd,yd,dul,eul,du2,eu2,vu(256),hu(256),ku(256)
integer gu(256)

```

Function: This subroutine rotates an nXn array a through an angle theta (given in degrees) placing the result at the center of an mXm array b.

Convert theta into radians.

```

pi=4.*atan(1.)
theta=theta*(pi/180.)

```

initialize b

```

do i=1,m
do j=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo

```

Compute centers of array a and b

```

nn=n/2+1
mm=m/2+1

```

Compute the integer of horizontal and vertical displacement along the x and y-axis which corresponds to n-pixel movement along the u-axis.

```

xu=nint(nn*cos(theta))
yu=nint(nn*sin(theta))
if (xu.lt.0) then
ml=-1
else
ml=1
endif
if (yu.lt.0) then
nl=1
else
nl=-1
endif

```

Compute the rotation parameters

```

dul=2*abs(yu)
du2=2*(abs(yu)-nn)
eul=2*abs(xu)
eu2=2*(abs(xu)-nn)
vu(1)=2*abs(yu)-nn
hu(1)=2*abs(xu)-nn
ku(0)=0
gu(0)=0
do i=1,nn
if (hu(i).ge.0) then
hu(i+1)=hu(i)+eu2
gu(i)=gu(i-1)+1
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
else
hu(i+1)=hu(i)+eul
gu(i)=gu(i-1)
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
endif
enddo

```

Compute new coordinates (xd,yd) of rotated array and assign the value of a at (i,j) to the appropriate cell of b which encloses these coordinates.

Generate 4-quadrant cartesian plane.

```
do i=1,nn
ku(-i)=-ku(i)
gu(-i)=-gu(i)
enddo
```

Start process

```
do j=1,n
do i=1,n
```

Employ coordinate transformation for rotation using the integer parameters.

```
yd=m1*gu(i-nn)-n1*ku(j-nn)
xd=m1*gu(j-nn)+n1*ku(i-nn)
```

Compute the integer coordinates of array b associated with the above transformation.

```
kx=mm+xd
ky=mm+yd
```

Assign value of a at (i,j) to the nearest cell of array b in which the transformed coordinates lie.

```
b(ky+1,kx-1)=a(i,j)
b(ky,kx-1)=a(i,j)
b(ky-1,kx-1)=a(i,j)
b(ky+1,kx)=a(i,j)
b(ky,kx)=a(i,j)
b(ky-1,kx)=a(i,j)
b(ky+1,kx+1)=a(i,j)
b(ky,kx+1)=a(i,j)
b(ky-1,kx+1)=a(i,j)
```

```
enddo
enddo
return
end
```

subroutine rotate_nearest_quasi(a,n,b,m,theta)

real a(256,256),b(256,256),w(256,256)

FUNCTION : This subroutine rotates an n X n array a
through an angle theta (given in degree)
anticlockwise placing the result at the
center of an array mxm b

PARAMETERS: a - Input image of n X n size
b - Rotated image of m X m size

EXTERNAL SUBROUTINE :None

INTERNAL SUBROUTINE :None

convert theta into radians

pi = 4. * atan(1.)
theta = theta * (pi/180.)
tanth = tan(theta)

initial b

do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) = 0.
enddo
enddo

compute new coordinates (xd,yd) of rotate array

nn = n/2 + 1
mm = m/2 + 1

start process

do j = 1,n
do i = 1,n

generate 4-quadrant cartesian plan

y = float(j-nn)
x = float(i-nn)
xd = x - y * tanth
yd = y + x * tanth

compute both real and integer coordinates of array b
associated with the above transformation for later use

kx = mm+ nint(xd)
ky = mm+ nint(yd)

assign value of a at (j,i) to the nearest_point cell
of array b in wich the transformed coordinates
lie.

b(ky,kx) = a(j,i)

enddo
enddo
return
end


```

subroutine rotate_lrp neig (a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
integer xu,yu,xd,yd,dul,eul,du2,eu2,vu(256),hu(256),ku(256)
integer gu(256)

```

Function: This subroutine rotates an nXn array a through an angle theta (given in degrees) placing the result at the center of an mXm array b.

Convert theta into radians.

```

pi=4.*atan(1.)
theta=theta*(pi/180.)

```

initialize b

```

do i=1,m
do j=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo

```

Compute centers of array a and b

```

nn=n/2+1
mm=m/2+1

```

Compute the integer of horizontal and vertical displacement along the x and y-axis which corresponds to n-pixel movement along the u-axis.

```

xu=nint(nn*cos(theta))
yu=nint(nn*sin(theta))
if (xu.lt.0) then
ml=-1
else
ml=1
endif
if (yu.lt.0) then
nl=1
else
nl=-1
endif

```

Compute the rotation parameters

```

dul=2*abs(yu)
du2=2*(abs(yu)-nn)
eul=2*abs(xu)
eu2=2*(abs(xu)-nn)
vu(1)=2*abs(yu)-nn
hu(1)=2*abs(xu)-nn
ku(0)=0
gu(0)=0
do i=1,nn
if (hu(i).ge.0) then
hu(i+1)=hu(i)+eu2
gu(i)=gu(i-1)+1
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
else
hu(i+1)=hu(i)+eul
gu(i)=gu(i-1)
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
endif
enddo

```

Compute new coordinates (xd,yd) of rotated array and assign the value of a at (i,j) to the appropriate cell of b which encloses these coordinates.

Generate 4-quadrant cartesian plane.

```
do i=1,nn
ku(-i)=-ku(i)
gu(-i)=-gu(i)
enddo
```

Start process

```
do j=1,n
do i=1,n
```

Employ coordinate transformation for rotation using the integer parameters.

```
yd=m1*gu(i-nn)-n1*ku(j-nn)
xd=m1*gu(j-nn)+n1*ku(i-nn)
```

Compute the integer coordinates of array b associated with the above transformation.

```
kx=mm+xd
ky=mm+yd
```

Assign value of a at (i,j) to the nearest cell of array b in which the transformed coordinates lie.

```
b(ky,kx-1)=a(i,j)
b(ky+1,kx)=a(i,j)
b(ky,kx)=a(i,j)
b(ky-1,kx)=a(i,j)
b(ky,kx+1)=a(i,j)
```

```
enddo
enddo
return
end
```

```

subroutine rotate 2rp(a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)

```

```

FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
           THROUGH AN ANGLE THETA (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b

```

```

PARAMETERS: a  _ Input image of n X n size
              b  _ Rotated image of m X m size

```

```

EXTERNAL SUBROUTINE :None

```

```

INTERNAL SUBROUTINE :None

```

```

convert theta into radians

```

```

pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)

```

```

initial b

```

```

do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) = 0.
enddo
enddo

```

```

compute new coordinates (xd,yd) of rotate array

```

```

nn = n/2 + 1
mm = m/2 + 1

```

```

Compensation of the center position for rotation.

```

```

xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)

```

```

start process

```

```

do j = 1,n
do i = 1,n

```

```

generate 4-quadrant cartesian plan

```

```

y = float(j-nn)
x = float(i-nn)
xd = x * costh - y * sinth
yd = x * sinth + y * costh

```

```

compute both real and integer coordinates of array b
associated with the above transformation for later use

```

```

kx = mm+ nint(xd)
ky = mm+ nint(yd)

```

```

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.

```

```

b(ky,kx-1) = a(j,i)
b(ky+1,kx) = a(j,i)
b(ky,kx) = a(j,i)
b(ky-1,kx) = a(j,i)
b(ky,kx+1) = a(j,i)
enddo
enddo
return
end

```

```

subroutine rotate 2rp neig (a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
integer xu,yu,xd,yd,dul,eul,du2,eu2,vu(256),hu(256),ku(256)
integer gu(256)

```

Function: This subroutine rotates an nXn array a through an angle theta (given in degrees) placing the result at the center of an mXm array b.

Convert theta into radians.

```

pi=4.*atan(1.)
theta=theta*(pi/180.)

```

initialize b

```

do i=1,m
do j=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo

```

Compute centers of array a and b

```

nn=n/2+1
mm=m/2+1

```

Compute the integer of horizontal and vertical displacement along the x and y-axis which corresponds to n-pixel movement along the u-axis.

```

xu=nint(nn*cos(theta))
yu=nint(nn*sin(theta))
if (xu.lt.0) then
ml=-1
else
ml=1
endif
if (yu.lt.0) then
nl=1
else
nl=-1
endif

```

Compute the rotation parameters

```

dul=2*abs(yu)
du2=2*(abs(yu)-nn)
eul=2*abs(xu)
eu2=2*(abs(xu)-nn)
vu(1)=2*abs(yu)-nn
hu(1)=2*abs(xu)-nn
ku(0)=0
gu(0)=0
do i=1,nn
if (hu(i).ge.0) then
hu(i+1)=hu(i)+eu2
gu(i)=gu(i-1)+1
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
else
hu(i+1)=hu(i)+eul
gu(i)=gu(i-1)
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
endif
enddo

```

Compute new coordinates (xd,yd) of rotated array and assign the value of a at (i,j) to the appropriate cell of b which encloses these coordinates.

```

Generate 4-quadrant cartesian plane.
    do i=1,nn
        ku(-i)=-ku(i)
        gu(-i)=-gu(i)
    enddo

Start process
    do j=1,n
        do i=1,n
Employ coordinate transformation for rotation using the integer parameters.
            yd=m1*gu(i-nn)-n1*ku(j-nn)
            xd=m1*gu(j-nn)+n1*ku(i-nn)
Compute the integer coordinates of array b associated with the above
transformation.
            kx=mm+xd
            ky=mm+yd
Assign value of a at (i,j) to the nearest cell of array b in which the
transformed coordinates lie.
            b(ky,kx-1)=a(i,j)
            b(ky+1,kx)=a(i,j)
            b(ky,kx)=a(i,j)
            b(ky-1,kx)=a(i,j)
            b(ky,kx+1)=a(i,j)

        enddo
    enddo
return
end

```

```
subroutine rotate_four(a,n,b,m,theta)
  real a(256,256),b(256,256),w(256,256)
```

```
*****
```

```
FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
           THROUGH AN ANGLE THETA (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b
```

```
PARAMETERS: a _ Input image of n X n size
             b _ Rotated image of m X m size
```

```
EXTERNAL SUBROUTINE :None
```

```
INTERNAL SUBROUTINE :None
```

```
*****
```

```
convert theta into radians
```

```
pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)
```

```
initial b
```

```
do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) = 0.
enddo
enddo
```

```
compute new coordinates (xd,yd) of rotate array
```

```
nn = n/2 + 1
mm = m/2 + 1
```

```
Compensation of the center position for rotation.
```

```
xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)
```

```
start process
```

```
do j = 1,n
do i = 1,n
```

```
generate 4-quadrant cartesian plan
```

```
y = float(j-nn)
x = float(i-nn)
xd = x * costh - y * sinth
yd = x * sinth + y * costh
```

```
compute both real and integer coordinates of array b
associated with the above transformation for later use
```

```
kx = mm+ nint(xd)
ky = mm+ nint(yd)
```

```
assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.
```

```
b(ky,kx-1) = a(j,i)
b(ky+1,kx) = a(j,i)
b(ky,kx) = a(j,i)
b(ky-1,kx) = a(j,i)
b(ky,kx+1) = a(j,i)
enddo
enddo
return
end
```

```

subroutine rotate four modify (a,n,b,m,theta)
real a(256,256),b(256,256)
integer xu,yu,xd,yd,dul,eul,du2,eu2,vu(256),hu(256),ku(256)
integer gu(256)

```

```

*****
Function: This subroutine rotates an nXn array a through an angle theta
(given in degrees ) placing the result at the center of an mXm
array b.

```

```

PARAMETERS: a _ Input image of n X n size
             b _ Rotated image of m X m size

```

```

EXTERNAL SUBROUTINE :None

```

```

INTERNAL SUBROUTINE :None

```

```

*****

```

```

Convert theta into radiants.

```

```

pi=4.*atan(1.)
theta=theta*(pi/180.)

```

```

initialize b

```

```

do i=1,m
do j=1,m
b(j,i)=0.
enddo
enddo

```

```

Compute centers of array a and b

```

```

nn=n/2+1
mm=m/2+1

```

```

Compute the integer of horizontal and vertical displacement along the x and
y-axis which corresponds to n-pixel movement along the u-axis.

```

```

xu=nint(n*cos(theta))
yu=nint(n*sin(theta))

```

```

Compute the rotation parameters

```

```

dul=abs(yu)
du2=abs(yu)-n
eul=abs(xu)
eu2=abs(xu)-n
vu(1)=abs(yu)-nn
hu(1)=abs(xu)-nn
ku(0)=0
gu(0)=0
do i=1,nn
if (hu(i).ge.0) then
hu(i+1)=hu(i)+eu2
gu(i)=gu(i-1)+1
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
else
hu(i+1)=hu(i)+eul
gu(i)=gu(i-1)
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
endif
enddo

```

```

Indicate the direction of the rotated axes

```

```

if (xu.lt.0) then
do i=1,nn
gu(i)=-gu(i)
enddo
endif

```

```

    if (yu.gt.0) then
    do i=1,nn
    ku(i)=-ku(i)
    enddo
    endif

```

Compute new coordinates (xd,yd) of rotated array and assign the value of a at (i,j) to the 4 appropriate cells of b which encloses these coordinates.
 Generate 4-quadrant cartesian plane.

```

    do i=1,nn
    ku(-i)=-ku(i)
    gu(-i)=-gu(i)
    enddo

```

Start process

```

    do j=1,n
    do i=1,n

```

Employ coordinate transformation for rotation using the integer parameters.

```

        yd=gu(i-nn)-ku(j-nn)
        xd=gu(j-nn)+ku(i-nn)

```

Compute the integer coordinates of array b associated with the above transformation.

```

        kx=mm+xd
        ky=mm+yd

```

Assign value of a at (i,j) to the 4 cells of array b in which the transformed coordinates lie.

```

        b(ky,kx-1)=a(i,j)
        b(ky+1,kx)=a(i,j)
        b(ky,kx)=a(i,j)
        b(ky-1,kx)=a(i,j)
        b(ky,kx+1)=a(i,j)

```

```

    enddo
    enddo
    return
end

```



```

subroutine rotate four neig (a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
integer xu,yu,xd,yd,dul,eul,du2,eu2,vu(256),hu(256),ku(256)
integer gu(256)

```

Function: This subroutine rotates an nXn array a through an angle theta (given in degrees) placing the result at the center of an mXm array b.

Convert theta into radiants.

```

pi=4.*atan(1.)
theta=theta*(pi/180.)

```

initialize b

```

do i=1,m
do j=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo

```

Compute centers of array a and b

```

nn=n/2+1
mm=m/2+1

```

Compute the integer of horizontal and vertical displacement along the x and y-axis which corresponds to n-pixel monement along the u-axis.

```

xu=nint(nn*cos(theta))
yu=nint(nn*sin(theta))
if (xu.lt.0) then
ml=-1
else
ml=1
endif
if (yu.lt.0) then
nl=1
else
nl=-1
endif

```

Compute the rotation parameters

```

dul=2*abs(yu)
du2=2*(abs(yu)-nn)
eul=2*abs(xu)
eu2=2*(abs(xu)-nn)
vu(1)=2*abs(yu)-nn
hu(1)=2*abs(xu)-nn
ku(0)=0
gu(0)=0
do i=1,nn
if (hu(i).ge.0) then
hu(i+1)=hu(i)+eu2
gu(i)=gu(i-1)+1
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
else
hu(i+1)=hu(i)+eul
gu(i)=gu(i-1)
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
endif
enddo

```

Compute new coordinates (xd,yd) of rotated array and assign the value of a at (i,j) to the appropriate cell of b which encloses these coordinates.

Generate 4-quadrant cartesian plane.

```
do i=1,nn
ku(-i)=-ku(i)
gu(-i)=-gu(i)
enddo
```

Start process

```
do j=1,n
do i=1,n
```

Employ coordinate transformation for rotation using the integer parameters.

```
yd=m1*gu(i-nn)-n1*ku(j-nn)
xd=m1*gu(j-nn)+n1*ku(i-nn)
```

Compute the integer coordinates of array b associated with the above transformation.

```
kx=mm+xd
ky=mm+yd
```

Assign value of a at (i,j) to the nearest cell of array b in which the transformed coordinates lie.

```
b(ky+1,kx-1)=a(i,j)
b(ky,kx-1)=a(i,j)
b(ky-1,kx-1)=a(i,j)
b(ky+1,kx)=a(i,j)
b(ky,kx)=a(i,j)
b(ky-1,kx)=a(i,j)
b(ky+1,kx+1)=a(i,j)
b(ky,kx+1)=a(i,j)
b(ky-1,kx+1)=a(i,j)
```

```
enddo
enddo
return
end
```

```
subroutine rotate eight(a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
```

```
*****
```

```
FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
           THROUGH AN ANGLE THETA (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b
```

```
PARAMETERS: a _ Input image of n X n size
             b _ Rotated image of m X m size
```

```
EXTERNAL SUBROUTINE :None
```

```
INTERNAL SUBROUTINE :None
```

```
*****
```

```
convert theta into radians
```

```
pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)
```

```
initial b
```

```
do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) = 0.
enddo
enddo
```

```
compute new coordinates (xd,yd) of rotate array
```

```
nn = n/2 + 1
mm = m/2 + 1
```

```
start process
```

```
do j = 1,n
do i = 1,n
```

```
generate 4-quadrant cartesian plan
```

```
y = float(j-nn)
x = float(i-nn)
xd = x * costh - y * sinth
yd = x * sinth + y * costh
```

```
compute both real and integer coordinates of array b
associated with the above transformation for later use
```

```
kx = mm+ nint(xd)
ky = mm+ nint(yd)
```

```
assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.
```

```
b(ky+1,kx-1) = a(j,i)
b(ky,kx-1) = a(j,i)
b(ky-1,kx-1) = a(j,i)
b(ky+1,kx) = a(j,i)
b(ky,kx) = a(j,i)
b(ky-1,kx) = a(j,i)
b(ky+1,kx+1) = a(j,i)
b(ky,kx+1) = a(j,i)
b(ky-1,kx+1) = a(j,i)
enddo
enddo
return
end
```

```

subroutine rotate eight neig (a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
integer xu,yu,xd,yd,dul,eul,du2,eu2,vu(256),hu(256),ku(256)
integer gu(256)

```

Function: This subroutine rotates an nXn array a through an angle theta (given in degrees) placing the result at the center of an mXm array b.

Convert theta into radiants.

```

pi=4.*atan(1.)
theta=theta*(pi/180.)

```

initialize b

```

do i=1,m
do j=1,m
b(j,i)=0
enddo
enddo

```

Compute centers of array a and b

```

nn=n/2+1
mm=m/2+1

```

Compute the integer of horizontal and vertical displacement along the x and y-axis which corresponds to n-pixel monement along the u-axis.

```

xu=nint(nn*cos(theta))
yu=nint(nn*sin(theta))
if (xu.lt.0) then
ml=-1
else
ml=1
endif
if (yu.lt.0) then
nl=1
else
nl=-1
endif

```

Compute the rotation parameters

```

dul=2*abs(yu)
du2=2*(abs(yu)-nn)
eul=2*abs(xu)
eu2=2*(abs(xu)-nn)
vu(1)=2*abs(yu)-nn
hu(1)=2*abs(xu)-nn
ku(0)=0
gu(0)=0
do i=1,nn
if (hu(i).ge.0) then
hu(i+1)=hu(i)+eu2
gu(i)=gu(i-1)+1
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
else
hu(i+1)=hu(i)+eul
gu(i)=gu(i-1)
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
endif
enddo

```

Compute new coordinates (xd,yd) of rotated array and assign the value of a at (i,j) to the appropriate cell of b which encloses these coordinates.
Generate 4-quadrant cartesian plane.

```
do i=1,nn
ku(-i)=-ku(i)
gu(-i)=-gu(i)
enddo
```

Start process

```
do j=1,n
do i=1,n
```

Employ coordinate transformation for rotation using the integer parameters.

```
yd=m1*gu(i-nn)-n1*ku(j-nn)
xd=m1*gu(j-nn)+n1*ku(i-nn)
```

Compute the integer coordinates of array b associated with the above transformation.

```
kx=mm+xd
ky=mm+yd
```

Assign value of a at (j,i) to the nearest cell of array b in which the transformed coordinates lie.

```
b(ky+1,kx-1)=a(i,j)
b(ky,kx-1)=a(i,j)
b(ky-1,kx-1)=a(i,j)
b(ky+1,kx)=a(i,j)
b(ky,kx)=a(i,j)
b(ky-1,kx)=a(i,j)
b(ky+1,kx+1)=a(i,j)
b(ky,kx+1)=a(i,j)
b(ky-1,kx+1)=a(i,j)
```

```
enddo
enddo
return
end
```

subroutine rotate three pass(a,n,b,m,theta)

real a(256,256),b(256,256),w(256,256)

FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
THROUGH AN ANGLE THETA (given in degree)
anticlockwise placing the result at the
center of an array mxm b

convert theta into radians

pi = 4. * atan(1.)
theta = theta * (pi/180.)
tanth = tan(theta/2.)
sinth = sin(theta)

initial b

do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) = 0.
enddo
enddo

compute new coordinates (xd,yd) of rotate array

nn = n/2 + 1
mm = m/2 + 1

start process

do j = 1,m
do i = 1,m

generate 4-quadrant cartesian plan

y = float(j-nn)
x = float(i-nn)

First pass

x1=x-y*tanth
y1=y

Second pass

x2=x1
y2=x1*sinth+y1

Third pass

xd=x2-y2*tanth
yd=y2

compute both real and integer coordinates of array b
associated with the above transformation for later use

ax =mm+ xd
ay = mm+ yd

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.

kx=nint(ax)
ky=nint(ay)
b(ky,kx)=a(j,i)

enddo
enddo
return
end

```
subroutine rotate_rectangle_sum(a,n,b,m,theta)
real a(256,256),b(256,256),ss(256,256),w(256,256)
```

FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
THROUGH AN ANGLE THETA (given in degree)
anticlockwise placing the result at the
center of an array mxm b

convert theta into radians

```
pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)
```

Zero pad image a with anint((m-n+1)/2) storing the result in the array ss.
The function anint is used in the case of (m-n+1) is even

```
nw=m-n+1
ml=m+nw-1
iframe=anint(float(m-n)/2.)
```

initial ss

```
do i=1,ml
doj=1,ml
ss(j,i)=0.
enddo
enddo
do i=1,n
do j=1,n
ss(j+iframe,i+iframe)=a(j,i)
enddo
enddo
```

initial b

```
do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) = 0.
enddo
enddo
```

compute new coordinates (xd,yd) of rotate array

```
nn = m/2 + 1
mm = ml/2 + 1
```

Compensation of the center position for rotation.

```
xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)
```

start process

```
do j = 1,m
do i = 1,m
```

generate 4-quadrant cartesian plan

```
y = float(j-nn)
x = float(i-nn)
xd = x * costh + y * sinth
yd = -x * sinth + y * costh
```

compute both real and integer coordinates of array b
associated with the above transformation for later use

```
ax = nn+ xd
ay = nn+ yd
```

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.

```
fx=(ax-aint(ax))
fy=(ay-aint(ay))
kx=aint(ax)
ky=aint(ay)
dl=(1.-fx)*(1.-fy)*ss(ky,kx)
```

```
d2=fx*fy* ss(ky+1,kx+1)
d3=fy*(1.-fx)*ss(ky+1,kx)
d4=fx*(1.-fy)*ss(ky,kx+1)
```

```
b(j,i)=d1+d2+d3+d4
```

```
enddo
enddo
return
end
```



```

subroutine rotate_triangle_sum(a,n,b,m,theta)
real a(256,256),b(256,256),ss(256,256),w(256,256)

```

FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
THROUGH AN ANGLE THETA (given in degree)
anticlockwise placing the result at the
center of an array mxm b

convert theta into radians

```

pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)

```

Zero pad image a with aint((m-n+1)/2) storing the result in the array ss.
The function aint is used in the case of (m-n+1) is even

```

nw=m-n+1
ml=m+nw-1
iframe=aint(float(m-n)/2.)

```

initial ss

```

do i=1,ml
doj=1,ml
ss(j,i)=0.
enddo
enddo
do i=1,n
do j=1,n
ss(j+iframe,i+iframe)=a(j,i)
enddo
enddo

```

initial b

```

do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) = 0.
enddo
enddo

```

compute new coordinates (xd,yd) of rotate array

```

nn = m/2 + 1
mm = ml/2 + 1

```

Compensation of the center position for rotation.

```

xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)

```

start process

```

do j = 1,m
do i = 1,m

```

generate 4-quadrant cartesian plan

```

y = float(j-nn)
x = float(i-nn)
xd = x * costh + y * sinth
yd = -x * sinth + y * costh

```

compute both real and integer coordinates of array b
associated with the above transformation for later use

```

ax =nn+ xd
ay = nn+ yd

```

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.

```

fx=ax-aint(ax)
fy=ay-aint(ay)
kx=aint(ax)
ky=aint(ay)

```

```
d1=(1.-fy)*ss(ky,kx)
d2=fy* ss(ky+1,kx+1)
d3=(1.-fx)*ss(ky+1,kx)
d4=fx*ss(ky,kx+1)

b(j,i)=(d1+d2+d3+d4)*.5

enddo
enddo
return
end
```

subroutine rotate divide rectangle sum(a,n,b,m,theta)

real a(256,256),b(256,256),w(256,256)

FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
THROUGH AN ANGLE THETA (given in degree)
anticlockwise placing the result at the
center of an array mxm b

convert theta into radians

```
pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)
```

initial b

```
do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) = 0.
enddo
enddo
```

compute new coordinates (xd,yd) of rotate array

```
nn = n/2 + 1
mm = m/2 + 1
```

Compensation of the center position for rotation.

```
xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)
```

start process

```
do j = 1,m
do i = 1,m
```

generate 4-quadrant cartesian plan

```
y = float(j-nn)
x = float(i-nn)
xd = x * costh - y * sinth
yd = x * sinth + y * costh
```

compute both real and integer coordinates of array b
associated with the above transformation for later use

```
ax =mm+ xd
ay = mm+ yd
```

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.

```
fx=(ax-aint(ax))
fy=(ay-aint(ay))
kx=aint(ax)
ky=aint(ay)
d1=(1.-fx)*(1.-fy)*a(j,i)
d2=fy*(1.-fx)*a(j,i)
d3=fx*(1.-fy)*a(j,i)
d4=fx*fy*a(j,i)

b(ky,kx)=b(ky,kx)+d1
b(ky+1,kx)=b(ky+1,kx)+d2
b(ky,kx+1)=b(ky,kx+1)+d3
b(ky+1,kx+1)=b(ky+1,kx+1)+d4

enddo
enddo
return
end
```

subroutine divide triangle sum(a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)

FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
THROUGH AN ANGLE THETA (given in degree)
anticlockwise placing the result at the
center of an array mxm b

convert theta into radians

```
pi = 4. * atan(1.)  
theta = theta * (pi/180.)  
costh = cos(theta)  
sinth = sin(theta)
```

initial b

```
do i = 1,m  
do j = 1,m  
b(j,i) = 0.  
w(j,i) = 0.  
enddo  
enddo
```

compute new coordinates (xd,yd) of rotate array

```
nn = n/2 + 1  
mm = m/2 + 1
```

Compensation of the center position for rotation.

```
xcomp=nn+nn*(sinth - costh)  
ycomp=nn-nn*(sinth + costh)
```

start process

```
do j = 1,m  
do i = 1,m
```

generate 4-quadrant cartesian plan

```
y = float(j-nn)  
x = float(i-nn)  
xd = x * costh - y * sinth  
yd = x * sinth + y * costh
```

compute both real and integer coordinates of array b
associated with the above transformation for later use

```
ax = mm+ xd  
ay = mm+ yd
```

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.

```
fx=.5*(ax-aint(ax))  
fy=.5*(ay-aint(ay))  
kx=aint(ax)  
ky=aint(ay)  
d1=(.5-fy)*a(j,i)  
d2=(.5-fx)*a(j,i)  
d3=fx*a(j,i)  
d4=fy*a(j,i)
```

```
b(ky,kx)=b(ky,kx)+d1  
b(ky+1,kx)=b(ky+1,kx)+d2  
b(ky,kx+1)=b(ky,kx+1)+d3  
b(ky+1,kx+1)=b(ky+1,kx+1)+d4
```

```
enddo  
enddo  
return  
end
```

```

subroutine rotate 2dlinear sum(a,n,b,m,theta)
real a(256,256),b(256,256),ss(256,256),w(256,256)

```

FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
THROUGH AN ANGLE THETA (given in degree)
anticlockwise placing the result at the
center of an array mxm b

convert theta into radians

```

pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)

```

Zero pad image a with anint((m-n+1)/2) storing the result in the array ss.
The function anint is used in the case of (m-n+1) is even

```

nw=m-n+1
ml=m+nw-1
iframe=anint(float(m-n)/2.)

```

initial ss

```

do i=1,ml
doj=1,ml
ss(j,i)=0.
enddo
enddo
do i=1,n
do j=1,n
ss(j+iframe,i+iframe)=a(j,i)
enddo
enddo

```

initial b

```

do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) =0.
enddo
enddo

```

compute new coordinates (xd,yd) of rotate array

```

nn = m/2 + 1
mm = ml/2 + 1

```

Compensation of the center position for rotation.

```

xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)

```

start process

```

do j = 1,m
do i = 1,m

```

generate 4-quadrant cartesian plan

```

y = float(j-nn)
x = float(i-nn)
xd = x * costh + y * sinth
yd = -x * sinth + y * costh

```

compute both real and integer coordinates of array b
associated with the above transformation for later use

```

ax =nn+ xd
ay = nn+ yd

```

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.

```

kx=aint(ax)
ky=aint(ay)
xm1=sqrt((ky-ay)**2.+(kx-ax)**2.)
xm4=sqrt((ky+1.-ay)**2.+(kx+1.-ax)**2.)
xm2=sqrt((ky+1.-ay)**2.+(kx-ax)**2.)

```

```
xm3=sqrt((ky-ay)**2.+(kx+1.-ax)**2.)
d1=xm4*ss(ky,kx)
d2=xm1* ss(ky+1,kx+1)
d3=xm3*ss(ky+1,kx)
d4=xm2*ss(ky,kx+1)

b(j,i)=d1+d2+d3+d4

enddo
enddo
return
end
```

subroutine rotate divide 2dlinear sum(a,n,b,m,theta)

real a(256,256),b(256,256),w(256,256)

FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
THROUGH AN ANGLE THETA (given in degree)
anticlockwise placing the result at the
center of an array mxm b

convert theta into radians

pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)

initial b

do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) = 0.
enddo
enddo
sqrt2=sqrt(2.)

compute new coordinates (xd,yd) of rotate array

nn = n/2 + 1
mm = m/2 + 1

Compensation of the center position for rotation.

xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)

start process

do j = 1,m
do i = 1,m

generate 4-quadrant cartesian plan

y = float(j-nn)
x = float(i-nn)
xd = x * costh - y * sinth
yd = x * sinth + y * costh

compute both real and integer coordinates of array b
associated with the above transformation for later use

ax =mm+ xd
ay = mm+ yd

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.

kx=aint(ax)
ky=aint(ay)
xm1=sqrt((ky-ay)**2.+(kx-ax)**2.)
xm4=sqrt((ky+1.-ay)**2.+(kx+1.-ax)**2.)
xm2=sqrt((ky+1.-ay)**2.+(kx-ax)**2.)
xm3=sqrt((ky-ay)**2.+(kx+1.-ax)**2.)
d1=xm4*a(j,i)
d2=xm3*a(j,i)
d3=xm2*a(j,i)
d4=xm1*a(j,i)

b(ky,kx)=b(ky,kx)+d1
b(ky+1,kx)=b(ky+1,kx)+d2
b(ky,kx+1)=b(ky,kx+1)+d3
b(ky+1,kx+1)=b(ky+1,kx+1)+d4

enddo
enddo
return
end

subroutine rotate_rectangle 12neigh sum(a,n,b,m,theta)

real a(256,256),b(256,256),ss(256,256),w(256,256)

FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
THROUGH AN ANGLE THETA (given in degree)
anticlockwise placing the result at the
center of an array mxm b

convert theta into radians

pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)

Zero pad image a with anint((m-n+1)/2) storing the result in the array ss.
The function anint is used in the case of (m-n+1) is even

nw=m-n+1
ml=m+nw-1
iframe=anint(float(m-n)/2.)

initial ss

do i=1,ml
do j=1,ml
ss(j,i)=0.
enddo
enddo
do i=1,n
do j=1,n
ss(j+iframe,i+iframe)=a(j,i)
enddo
enddo

initial b

do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) = 0.
enddo
enddo

compute new coordinates (xd,yd) of rotate array

nn = m/2 + 1
mm = ml/2 + 1

Compensation of the center position for rotation.

xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)

start process

do j = 1,m
do i = 1,m

generate 4-quadrant cartesian plan

y = float(j-nn)
x = float(i-nn)
xd = x * costh + y * sinth
yd = -x * sinth + y * costh

compute both real and integer coordinates of array b
associated with the above transformation for later use

ax = nn+ xd
ay = nn+ yd

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.

fx=(ax-aint(ax))
fy=(ay-aint(ay))
kx=aint(ax)
ky=aint(ay)
dl=(1.+fy)*(1.-fx)/7.


```
d2=(2.-fx)*(1.-fy)/7.  
d3=(2.-fy)*fx/7.  
d4=fy*(1.+fx)/7.  
d5=(1.-fy)*(1.-fx)/7.  
d6=(1.-fy)*fx/7.  
d7=fy*fx/7.  
d8=fy*(1.-fx)/7.  
d9=fy*(2.-fx)/7.  
d10=(2.-fy)*(1.-fx)/7.  
d11=(1.+fx)*(1.-fy)/7.  
d12=(1.+fy)*fx/7.
```

```
e1=d1*ss(ky+1,kx)  
e2=d2* ss(ky,kx)  
e3=d3*ss(ky,kx+1)  
e4=d4*ss(ky+1,kx+1)  
e5=d5*ss(ky-1,kx)  
e6=d6* ss(ky,kx+2)  
e7=d7*ss(ky+2,kx+1)  
e8=d8*ss(ky+1,kx-1)  
e9=d9*ss(ky+2,kx)  
e10=d10* ss(ky,kx-1)  
e11=d11*ss(ky-1,kx+1)  
e12=d12*ss(ky+1,kx+2)
```

```
b(j,i)=e1+e2+e3+e4+e5+e6+e7+e8+e9+e10+e11+e12
```

```
enddo  
enddo  
return  
end
```

```
subroutine rotate_rectangle 16neigh sum(a,n,b,m,theta)
real a(256,256),b(256,256),ss(256,256),w(256,256)
```

```
FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
           THROUGH AN ANGLE THETA (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b
```

```
convert theta into radians
```

```
pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)
```

```
Zero pad image a with aint((m-n+1)/2) storing the result in the array ss.
The function aint is used in the case of (m-n+1) is even
```

```
nw=m-n+1
ml=m+nw-1
iframe=aint(float(m-n)/2.)
```

```
initial ss
```

```
do i=1,ml
doj=1,ml
ss(j,i)=0.
enddo
enddo
do i=1,n
do j=1,n
ss(j+iframe,i+iframe)=a(j,i)
enddo
enddo
```

```
initial b
```

```
do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) = 0.
enddo
enddo
```

```
compute new coordinates (xd,yd) of rotate array
```

```
nn = m/2 + 1
mm = ml/2 + 1
```

```
Compensation of the center position for rotation.
```

```
xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)
```

```
start process
```

```
do j = 1,m
do i = 1,m
```

```
generate 4-quadrant cartesian plan
```

```
y = float(j-nn)
x = float(i-nn)
xd = x * costh + y * sinth
yd = -x * sinth + y * costh
```

```
compute both real and integer coordinates of array b
associated with the above transformation for later use
```

```
ax =nn+ xd
ay = nn+ yd
```

```
assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.
```

```
fx=(ax-aint(ax))
fy=(ay-aint(ay))
kx=aint(ax)
ky=aint(ay)
dl=(1.+fy)*(2.-fx)/16.
```

```

d2=(2.-fx)*(2.-fy)/16.
d3=(2.-fy)*(1.+fx)/16.
d4=(1.+fy)*(1.+fx)/16.
d5=(1.-fy)*(1.-fx)/16.
d6=fx*(1.-fy)/16.
d7=fy*fx/16.
d8=fy*(1.-fx)/16.
d9=fy*(2.-fx)/16.
d10=(2.-fy)*(1.-fx)/16.
d11=(1.+fx)*(1.-fy)/16.
d12=(1.+fy)*fx/16.
d13=(1.-fx)*(1.+fy)/16.
d14=(2.-fx)*(1.-fy)/16.
d15=fx*(2.-fy)/16.
d16=(1.+fx)*fy/16.

```

```

e1=d1*ss(ky+1,kx)
e2=d2* ss(ky,kx)
e3=d3*ss(ky,kx+1)
e4=d4*ss(ky+1,kx+1)
e5=d5*ss(ky-1,kx-1)
e6=d6* ss(ky-1,kx+2)
e7=d7*ss(ky+2,kx+2)
e8=d8*ss(ky+2,kx-1)
e9=d9*ss(ky+1,kx-1)
e10=d10* ss(ky-1,kx)
e11=d11*ss(ky,kx+2)
e12=d12*ss(ky+2,kx+1)
e13=d13*ss(ky+2,kx)
e14=d14*ss(ky,kx-1)
e15=d15*ss(ky-1,kx+1)
e16=d16*ss(ky+1,kx+2)

```

```

b(j,i)=e1+e2+e3+e4+e5+e6+e7+e8+e9+e10+e11+e12+e13+e14+e15+e16

```

```

enddo
enddo
return
end

```

```
subroutine rotate triangle 12neigh sum(a,n,b,m,theta)
real a(256,256),b(256,256),ss(256,256),w(256,256)
```

FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
THROUGH AN ANGLE THETA (given in degree)
anticlockwise placing the result at the
center of an array mxm b

convert theta into radians

```
pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)
```

Zero pad image a with anint((m-n+1)/2) storing the result in the array ss.
The function anint is used in the case of (m-n+1) is even

```
nw=m-n+1
ml=m+nw-1
iframe=anint(float(m-n)/2.)
```

initial ss

```
do i=1,ml
doj=1,ml
ss(j,i)=0.
enddo
enddo
do i=1,n
do j=1,n
ss(j+iframe,i+iframe)=a(j,i)
enddo
enddo
```

initial b

```
do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) = 0.
enddo
enddo
```

compute new coordinates (xd,yd) of rotate array

```
nn = m/2 + 1
mm = ml/2 + 1
```

Compensation of the center position for rotation.

```
xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)
```

start process

```
do j = 1,m
do i = 1,m
```

generate 4-quadrant cartesian plan

```
y = float(j-nn)
x = float(i-nn)
xd = x * costh + y * sinth
yd = -x * sinth + y * costh
```

compute both real and integer coordinates of array b
associated with the above transformation for later use

```
ax = nn+ xd
ay = nn+ yd
```

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.

```
fx=(ax-aint(ax))
fy=(ay-aint(ay))
kx=aint(ax)
ky=aint(ay)
dl=.1*(2.-fy)
```

```
d2=.1*(1.+fx)
d3=.1*(1.+fy)
d4=.1*(2.-fx)
d5=.1*(1.-fx)
d6=.1*(1.-fy)
d7=.1*fx
d8=.1*fy
```

```
e1=d1*ss(ky,kx)
e2=d2* ss(ky,kx+1)
e3=d3*ss(ky+1,kx+1)
e4=d4*ss(ky+1,kx)
e5=d5*ss(ky+2,kx)
e6=d6* ss(ky,kx-1)
e7=d7*ss(ky-1,kx+1)
e8=d8*ss(ky+1,kx+2)
e9=d6*ss(ky,kx+2)
e10=d7* ss(ky+2,kx+1)
e11=d8*ss(ky+1,kx-1)
e12=d5*ss(ky-1,kx)
```

```
b(j,i)=e1+e2+e3+e4+e5+e6+e7+e8+e9+e10+e11+e12
```

```
enddo
enddo
return
end
```

```
subroutine rotate_triangle_16neigh sum(a,n,b,m,theta)
real a(256,256),b(256,256),ss(256,256),w(256,256)
```

```
FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
           THROUGH AN ANGLE THETA (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b
```

```
convert theta into radians
```

```
pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)
```

```
Zero pad image a with anint((m-n+1)/2) storing the result in the array ss.
The function anint is used in the case of (m-n+1) is even
```

```
nw=m-n+1
ml=m+nw-1
iframe=anint(float(m-n)/2.)
```

```
initial ss
```

```
do i=1,ml
doj=1,ml
ss(j,i)=0.
enddo
enddo
do i=1,n
do j=1,n
ss(j+iframe,i+iframe)=a(j,i)
enddo
enddo
```

```
initial b
```

```
do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) = 0.
enddo
enddo
```

```
compute new coordinates (xd,yd) of rotate array
```

```
nn = m/2 + 1
mm = ml/2 + 1
```

```
Compensation of the center position for rotation.
```

```
xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)
```

```
start process
```

```
do j = 1,m
do i = 1,m
```

```
generate 4-quadrant cartesian plan
```

```
y = float(j-nn)
x = float(i-nn)
xd = x * costh + y * sinth
yd = -x * sinth + y * costh
```

```
compute both real and integer coordinates of array b
associated with the above transformation for later use
```

```
ax = nn+ xd
ay = nn+ yd
```

```
assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.
```

```
fx=(ax-aint(ax))
fy=(ay-aint(ay))
kx=aint(ax)
ky=aint(ay)
dl=.05*(2.-fy)
```

```
d2=.05*(1.+fx)
d3=.05*(1.+fy)
d4=.05*(2.-fx)
d13=.05*fy
d14=.05*(1.-fx)
d15=.05*(1.-fy)
d16=.05*fx
```

```
e1=d1*ss(ky,kx)
e2=d2*ss(ky,kx+1)
e3=d3*ss(ky+1,kx+1)
e4=d4*ss(ky+1,kx)
e5=d1*ss(ky-1,kx)
e6=d2* ss(ky,kx+2)
e7=d3*ss(ky+2,kx+1)
e8=d4*ss(ky+1,kx-1)
e9=d1*ss(ky-1,kx+1)
e10=d2* ss(ky+1,kx+2)
e11=d3*ss(ky+2,kx)
e12=d4*ss(ky,kx-1)
e13=d13*ss(ky+2,kx+2)
e14=d14*ss(ky+2,kx-1)
e15=d15* ss(ky-1,kx-1)
e16=d16*ss(ky-1,kx+2)
```

```
b(j,i)=e1+e2+e3+e4+e5+e6+e7+e8+e9+e10+e11+e12+e13+e14+e15+e16
```

```
enddo
enddo
return
end
```

```
subroutine rotate triangle 16neigh sum1(a,n,b,m,theta)
real a(256,256),b(256,256),ss(256,256),w(256,256)
```

FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
THROUGH AN ANGLE THETA (given in degree)
anticlockwise placing the result at the
center of an array mxm b

convert theta into radians

```
pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)
```

Zero pad image a with anint((m-n+1)/2) storing the result in the array ss.
The function anint is used in the case of (m-n+1) is even

```
nw=m-n+1
ml=m+nw-1
iframe=anint(float(m-n)/2.)
```

initial ss

```
do i=1,ml
doj=1,ml
ss(j,i)=0.
enddo
enddo
do i=1,n
do j=1,n
ss(j+iframe,i+iframe)=a(j,i)
enddo
enddo
```

initial b

```
do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) =0.
enddo
enddo
```

compute new coordinates (xd,yd) of rotate array

```
nn = m/2 + 1
mm = ml/2 + 1
```

Compensation of the center position for rotation.

```
xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)
```

start process

```
do j = 1,m
do i = 1,m
```

generate 4-quadrant cartesian plan

```
y = float(j-nn)
x = float(i-nn)
xd = x * costh + y * sinth
yd = -x * sinth + y * costh
```

compute both real and integer coordinates of array b
associated with the above transformation for later use

```
ax =nn+ xd
ay = nn+ yd
```

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates

lie.

```
fx=(ax-aint(ax))
fy=(ay-aint(ay))
kx=aint(ax)
ky=aint(ay)
dl=(1.-fx)/32.
```



```
d2=(1.-fy)/32.  
d3=fx/32.  
d4=fy/32.  
d5=(2.-fy)/32.  
d6=(1.+fy)/32.  
d7=(2.-fx)/32.  
d8=(1.+fx)/32.
```

```
e1=d1*ss(ky+2,kx-1)  
e2=d2*ss(ky-1,kx-1)  
e3=d3*ss(ky-1,kx+2)  
e4=d4*ss(ky+2,kx+2)  
e5=d5*ss(ky-1,kx)  
e6=d6* ss(ky+2,kx+1)  
e7=d7*ss(ky+1,kx-1)  
e8=d8*ss(ky,kx+2)  
e9=3.*d3*ss(ky-1,kx+1)  
e10=3.*d2* ss(ky,kx-1)  
e11=3.*d4*ss(ky+1,kx+2)  
e12=3.*d1*ss(ky+2,kx)  
e13=3.*d8*ss(ky,kx+1)  
e14=3.*d6*ss(ky+1,kx+1)  
e15=3.*d7* ss(ky+1,kx)  
e16=3.*d5*ss(ky,kx)
```

```
b(j,i)=e1+e2+e3+e4+e5+e6+e7+e8+e9+e10+e11+e12+e13+e14+e15+e16
```

```
enddo  
enddo  
return  
end
```

```
subroutine rotate divide rectangle12n sum(a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
```

FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
THROUGH AN ANGLE THETA (given in degree)
anticlockwise placing the result at the
center of an array mxm b

convert theta into radians

```
pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)
```

initial ss

```
do i=1,m
doj=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo
```

compute new coordinates (xd,yd) of rotate array

```
nn = n/2 + 1
mm = m/2 + 1
```

Compensation of the center position for rotation.

```
xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)
```

start process

```
do j = 1,m
do i = 1,m
```

generate 4-quadrant cartesian plan

```
y = float(j-nn)
x = float(i-nn)
xd = x * costh - y * sinth
yd = x * sinth + y * costh
```

compute both real and integer coordinates of array b
associated with the above transformation for later use

```
ax =mm+ xd
ay = mm+ yd
```

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.

```
fx=(ax-aint(ax))
fy=(ay-aint(ay))
kx=aint(ax)
ky=aint(ay)
d1=(1.+fy)*(1.-fx)/7.
d2=(2.-fx)*(1.-fy)/7.
d3=(2.-fy)*fx/7.
d4=fy*(1.+fx)/7.
d5=(1.-fy)*(1.-fx)/7.
d6=(1.-fy)*fx/7.
d7=fy*fx/7.
d8=fy*(1.-fx)/7.
d9=fy*(2.-fx)/7.
d10=(2.-fy)*(1.-fx)/7.
d11=(1.+fx)*(1.-fy)/7.
d12=(1.+fy)*fx/7.
```

```
e1=d1*a(j,i)
e2=d2*a(j,i)
e3=d3*a(j,i)
```

```
e4=d4*a(j,i)
e5=d1*a(j,i)
e6=d2* a(j,i)
e7=d3*a(j,i)
e8=d4*a(j,i)
e9=d1*a(j,i)
e10=d2* a(j,i)
e11=d3*a(j,i)
e12=d4*a(j,i)
e13=d13*a(j,i)
e14=d14*a(j,i)
e15=d15* a(j,i)
e16=d16*a(j,i)
```

```
b(ky,kx)=b(ky,kx)+e2
b(ky,kx+1)=b(ky,kx+1)+e3
b(ky+1,kx+1)=b(ky+1,kx+1)+e4
b(ky+1,kx)=b(ky+1,kx)+e1
b(ky-1,kx)=b(ky-1,kx)+e5
b(ky,kx+2)=b(ky,kx+2)+e6
b(ky+2,kx+1)=b(ky+2,kx+1)+e7
b(ky+1,kx-1)=b(ky+1,kx-1)+e8
b(ky-1,kx+1)=b(ky-1,kx+1)+e11
b(ky+1,kx+2)=b(ky+1,kx+2)+e12
b(ky+2,kx)=b(ky+2,kx)+e9
b(ky,kx-1)=b(ky,kx-1)+e10
```

```
enddo
enddo
return
end
```

```

subroutine rotate divide rectangle16n sum(a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)

```

FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
THROUGH AN ANGLE THETA (given in degree)
anticlockwise placing the result at the
center of an array mxm b

convert theta into radians

```

pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)

```

initial ss

```

do i=1,m
doj=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo

```

compute new coordinates (xd,yd) of rotate array

```

nn = n/2 + 1
mm = m/2 + 1

```

Compensation of the center position for rotation.

```

xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)

```

start process

```

do j = 1,m
do i = 1,m

```

generate 4-quadrant cartesian plan

```

y = float(j-nn)
x = float(i-nn)
xd = x * costh - y * sinth
yd = x * sinth + y * costh

```

compute both real and integer coordinates of array b
associated with the above transformation for later use

```

ax =mm+ xd
ay = mm+ yd

```

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.

```

fx=(ax-aint(ax))
fy=(ay-aint(ay))
kx=aint(ax)
ky=aint(ay)
d1=(1.+fy)*(2.-fx)/16.
d2=(2.-fx)*(2.-fy)/16.
d3=(2.-fy)*(1.+fx)/16.
d4=(1.+fy)*(1.+fx)/16.
d5=(1.-fy)*(1.-fx)/16.
d6=(1.-fy)*fx/16.
d7=fy*fx/16.
d8=fy*(1.-fx)/16.
d9=fy*(2.-fx)/16.
d10=(2.-fy)*(1.-fx)/16.
d11=(1.+fx)*(1.-fy)/16.
d12=(1.+fy)*fx/16.
d13=(1.+fy)*(1.-fx)/16.
d14=(1.-fy)*(2.-fx)/16.
d15=fx*(2.-fy)/16.
d16=fy*(1.+fx)/16.

```

```
e1=d1*a(j,i)
e2=d2*a(j,i)
e3=d3*a(j,i)
e4=d4*a(j,i)
e5=d5*a(j,i)
e6=d6* a(j,i)
e7=d7*a(j,i)
e8=d8*a(j,i)
e9=d9*a(j,i)
e10=d10* a(j,i)
e11=d11*a(j,i)
e12=d12*a(j,i)
e13=d13*a(j,i)
e14=d14*a(j,i)
e15=d15* a(j,i)
e16=d16*a(j,i)
```

```
b(ky+1,kx)=b(ky+1,kx)+e1
b(ky,kx)=b(ky,kx)+e2
b(ky,kx+1)=b(ky,kx+1)+e3
b(ky+1,kx+1)=b(ky+1,kx+1)+e4
b(ky-1,kx-1)=b(ky-1,kx-1)+e5
b(ky-1,kx+2)=b(ky-1,kx+2)+e6
b(ky+2,kx+2)=b(ky+2,kx+2)+e7
b(ky+2,kx-1)=b(ky+2,kx-1)+e8
b(ky+1,kx-1)=b(ky+1,kx-1)+e9
b(ky-1,kx)=b(ky-1,kx)+e10
b(ky,kx+2)=b(ky,kx+2)+e11
b(ky+2,kx+1)=b(ky+2,kx+1)+e12
b(ky+2,kx)=b(ky+2,kx)+e13
b(ky,kx-1)=b(ky,kx-1)+e14
b(ky-1,kx+1)=b(ky-1,kx+1)+e15
b(ky+1,kx+2)=b(ky+1,kx+2)+e16
```

```
enddo
enddo
return
end
```

```

subroutine divide triangle12n sum(a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)

```

FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
THROUGH AN ANGLE THETA (given in degree)
anticlockwise placing the result at the
center of an array mxm b

convert theta into radians

```

pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)

```

initial ss

```

do i=1,m
doj=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo

```

compute new coordinates (xd,yd) of rotate array

```

nn = n/2 + 1
mm = m/2 + 1

```

Compensation of the center position for rotation.

```

xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)

```

start process

```

do j = 1,m
do i = 1,m

```

generate 4-quadrant cartesian plan

```

y = float(j-nn)
x = float(i-nn)
xd = x * costh - y * sinth
yd = x * sinth + y * costh

```

compute both real and integer coordinates of array b
associated with the above transformation for later use

```

ax =mm+ xd
ay = mm+ yd

```

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.

```

fx=(ax-aint(ax))
fy=(ay-aint(ay))
kx=aint(ax)
ky=aint(ay)
d1=.1*(2.-fy)
d2=.1*(1.+fx)
d3=.1*(1.+fy)
d4=.1*(2.-fx)
d5=.1*(1.-fx)
d6=.1*(1.-fy)
d7=.1*fx
d8=.1*fy

```

```

e1=d1*a(j,i)
e2=d2*a(j,i)
e3=d3*a(j,i)
e4=d4*a(j,i)
e5=d5*a(j,i)
e6=d6*a(j,i)
e7=d7*a(j,i)

```

```
e8=d8*a(j,i)
e9=d6*a(j,i)
e10=d7* a(j,i)
e11=d8*a(j,i)
e12=d5*a(j,i)

b(ky,kx)=b(ky,kx)+e1
b(ky,kx+1)=b(ky,kx+1)+e2
b(ky+1,kx+1)=b(ky+1,kx+1)+e3
b(ky+1,kx)=b(ky+1,kx)+e4
b(ky+2,kx)=b(ky+2,kx)+e5
b(ky,kx-1)=b(ky,kx-1)+e6
b(ky-1,kx+1)=b(ky-1,kx+1)+e7
b(ky+1,kx+2)=b(ky+1,kx+2)+e8
b(ky,kx+2)=b(ky,kx+2)+e9
b(ky+2,kx+1)=b(ky+2,kx+1)+e10
b(ky+1,kx-1)=b(ky+1,kx-1)+e11
b(ky-1,kx)=b(ky-1,kx)+e12

enddo
enddo
return
end
```

```
subroutine divide triangle16n sum(a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
```

```
FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
           THROUGH AN ANGLE THETA (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b
```

```
convert theta into radians
```

```
pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)
```

```
initial ss
```

```
do i=1,m
doj=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo
```

```
compute new coordinates (xd,yd) of rotate array
```

```
nn = n/2 + 1
mm = m/2 + 1
```

```
Compensation of the center position for rotation.
```

```
xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)
```

```
start process
```

```
do j = 1,m
do i = 1,m
```

```
generate 4-quadrant cartesian plan
```

```
y = float(j-nn)
x = float(i-nn)
xd = x * costh - y * sinth
yd = x * sinth + y * costh
```

```
compute both real and integer coordinates of array b
associated with the above transformation for later use
```

```
ax =mm+ xd
ay = mm+ yd
```

```
assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.
```

```
fx=(ax-aint(ax))
fy=(ay-aint(ay))
kx=aint(ax)
ky=aint(ay)
d1=.05*(2.-fy)
d2=.05*(1.+fx)
d3=.05*(1.+fy)
d4=.05*(2.-fx)
d13=.05*fy
d14=.05*(1.-fx)
d15=.05*(1.-fy)
d16=.05*fx
```

```
e1=d1*a(j,i)
e2=d2*a(j,i)
e3=d3*a(j,i)
e4=d4*a(j,i)
e5=d1*a(j,i)
e6=d2* a(j,i)
e7=d3*a(j,i)
```



```
e8=d4*a(j,i)
e9=d1*a(j,i)
e10=d2* a(j,i)
e11=d3*a(j,i)
e12=d4*a(j,i)
e13=d13*a(j,i)
e14=d14*a(j,i)
e15=d15* a(j,i)
e16=d16*a(j,i)
```

```
b(ky,kx)=b(ky,kx)+e1
b(ky,kx+1)=b(ky,kx+1)+e2
b(ky+1,kx+1)=b(ky+1,kx+1)+e3
b(ky+1,kx)=b(ky+1,kx)+e4
b(ky-1,kx)=b(ky-1,kx)+e5
b(ky,kx+2)=b(ky,kx+2)+e6
b(ky+2,kx+1)=b(ky+2,kx+1)+e7
b(ky+1,kx-1)=b(ky+1,kx-1)+e8
b(ky-1,kx+1)=b(ky-1,kx+1)+e9
b(ky+1,kx+2)=b(ky+1,kx+2)+e10
b(ky+2,kx)=b(ky+2,kx)+e11
b(ky,kx-1)=b(ky,kx-1)+e12
b(ky+2,kx+2)=b(ky+2,kx+2)+e13
b(ky+2,kx-1)=b(ky+2,kx-1)+e14
b(ky-1,kx-1)=b(ky-1,kx-1)+e15
b(ky-1,kx+2)=b(ky-1,kx+2)+e16
```

```
enddo
enddo
return
end
```

```
subroutine divide triangle16n sum1(a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
```

FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
THROUGH AN ANGLE THETA (given in degree)
anticlockwise placing the result at the
center of an array mxm b

convert theta into radians

```
pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)
```

initial ss

```
do i=1,m
doj=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo
```

compute new coordinates (xd,yd) of rotate array

```
nn = n/2 + 1
mm = m/2 + 1
```

Compensation of the center position for rotation.

```
xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)
```

start process

```
do j = 1,m
do i = 1,m
```

generate 4-quadrant cartesian plan

```
y = float(j-nn)
x = float(i-nn)
xd = x * costh - y * sinth
yd = x * sinth + y * costh
```

compute both real and integer coordinates of array b
associated with the above transformation for later use

```
ax =mm+ xd
ay = mm+ yd
```

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.

```
fx=(ax-aint(ax))
fy=(ay-aint(ay))
kx=aint(ax)
ky=aint(ay)
d1=(1.-fx)/32.
d2=(1.-fy)/32.
d3=fx/32.
d4=fy/32.
d5=(2.-fy)/32.
d6=(1.+fy)/32.
d7=(2.-fx)/32.
d8=(1.+fx)/32.
```

```
e1=d1*a(j,i)
e2=d2*a(j,i)
e3=d3*a(j,i)
e4=d4*a(j,i)
e5=d5*a(j,i)
e6=d6* a(j,i)
e7=d7*a(j,i)
```

```

e8=d8*a(j,i)
e9=3.*d3*a(j,i)
e10=3.*d2* a(j,i)
e11=3.*d4*a(j,i)
e12=3.*d1*a(j,i)
e13=3.*d8*a(j,i)
e14=3.*d6*a(j,i)
e15=3.*d7* a(j,i)
e16=3.*d5*a(j,i)

b(ky,kx)=b(ky,kx)+e16
b(ky,kx+1)=b(ky,kx+1)+e13
b(ky+1,kx+1)=b(ky+1,kx+1)+e14
b(ky+1,kx)=b(ky+1,kx)+e15
b(ky-1,kx)=b(ky-1,kx)+e5
b(ky,kx+2)=b(ky,kx+2)+e8
b(ky+2,kx+1)=b(ky+2,kx+1)+e6
b(ky+1,kx-1)=b(ky+1,kx-1)+e7
b(ky-1,kx+1)=b(ky-1,kx+1)+e9
b(ky+1,kx+2)=b(ky+1,kx+2)+e11
b(ky+2,kx)=b(ky+2,kx)+e12
b(ky,kx-1)=b(ky,kx-1)+e10
b(ky+2,kx+2)=b(ky+2,kx+2)+e4
b(ky+2,kx-1)=b(ky+2,kx-1)+e1
b(ky-1,kx-1)=b(ky-1,kx-1)+e2
b(ky-1,kx+2)=b(ky-1,kx+2)+e3

enddo
enddo
return
end

```

```
subroutine rotate linear 12neigh sum(a,n,b,m,theta)
real a(256,256),b(256,256),ss(256,256),w(256,256)
```

FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
THROUGH AN ANGLE THETA (given in degree)
anticlockwise placing the result at the
center of an array mxm b

convert theta into radians

```
pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)
```

Zero pad image a with anint((m-n+1)/2) storing the result in the array ss.
The function anint is used in the case of (m-n+1) is even

```
nw=m-n+1
ml=m+nw-1
iframe=anint(float(m-n)/2.)
```

initial ss

```
do i=1,ml
doj=1,ml
ss(j,i)=0.
enddo
enddo
do i=1,n
do j=1,n
ss(j+iframe,i+iframe)=a(j,i)
enddo
enddo
```

initial b

```
do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) = 0.
enddo
enddo
```

compute new coordinates (xd,yd) of rotate array

```
nn = m/2 + 1
mm = ml/2 + 1
```

Compensation of the center position for rotation.

```
xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)
```

start process

```
do j = 1,m
do i = 1,m
```

generate 4-quadrant cartesian plan

```
y = float(j-nn)
x = float(i-nn)
xd = x * costh + y * sinth
yd = -x * sinth + y * costh
```

compute both real and integer coordinates of array b
associated with the above transformation for later use

```
ax = nn+ xd
ay = nn+ yd
```

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.

```
fx=(ax-aint(ax))
fy=(ay-aint(ay))
kx=aint(ax)
ky=aint(ay)
```

```
d1=sqrt(((ay-float(ky-1))**2.+(ax-float(kx+1))**2.)/5.)
```

```

d2=sqrt(((ay-float(ky+1))**2.+(ax-float(kx+2))**2.)/5.)
d3=sqrt(((ay-float(ky+2))**2.+(ax-float(kx))**2.)/5.)
d4=sqrt(((ay-float(ky))**2.+(ax-float(kx-1))**2.)/5.)
d5=sqrt(((ay-float(ky+1))**2.+(ax-float(kx+1))**2.)/5.)
d6=sqrt(((ay-float(ky+1))**2.+(ax-float(kx))**2.)/5.)
d7=sqrt(((ay-float(ky))**2.+(ax-float(kx))**2.)/5.)
d8=sqrt(((ay-float(ky))**2.+(ax-float(kx+1))**2.)/5.)
d9=sqrt(((ay-float(ky))**2.+(ax-float(kx+2))**2.)/5.)
d10=sqrt(((ay-float(ky+2))**2.+(ax-float(kx+1))**2.)/5.)
d11=sqrt(((ay-float(ky+1))**2.+(ax-float(kx-1))**2.)/5.)
d12=sqrt(((ay-float(ky-1))**2.+(ax-float(kx))**2.)/5.)

```

```

e1=d1*ss(ky-1,kx+1)
e2=d2* ss(ky+1,kx+2)
e3=d3*ss(ky+2,kx)
e4=d4*ss(ky,kx-1)
e5=d5*ss(ky+1,kx+1)
e6=d6* ss(ky+1,kx)
e7=d7*ss(ky,kx)
e8=d8*ss(ky,kx+1)
e9=d9*ss(ky,kx+2)
e10=d10* ss(ky+2,kx+1)
e11=d11*ss(ky+1,kx-1)
e12=d12*ss(ky-1,kx)

```

```

b(j,i)=e1+e2+e3+e4+e5+e6+e7+e8+e9+e10+e11+e12

```

```

enddo
enddo
return
end

```

```

subroutine rotate linear 16neigh sum(a,n,b,m,theta)
real a(256,256),b(256,256),ss(256,256),w(256,256)
FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
           THROUGH AN ANGLE THETA (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b

convert theta into radians

pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)
Zero pad image a with anint((m-n+1)/2) storing the result in the array ss.
The function anint is used in the case of (m-n+1) is even
nw=m-n+1
ml=m+nw-1
iframe=anint(float(m-n)/2.)

initial ss
do i=1,ml
doj=1,ml
ss(j,i)=0.
enddo
enddo
do i=1,n
do j=1,n
ss(j+iframe,i+iframe)=a(j,i)
enddo
enddo

initial b
do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) = 0.
enddo
enddo

compute new coordinates (xd,yd) of rotate array

nn = m/2 + 1
mm = ml/2 + 1
Compensation of the center position for rotation.
xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)

start process

do j = 1,m
do i = 1,m

generate 4-quadrant cartesian plan
y = float(j-nn)
x = float(i-nn)
xd = x * costh + y * sinth
yd = -x * sinth + y * costh

compute both real and integer coordinates of array b
associated with the above transformation for later use
ax =nn+ xd
ay = nn+ yd
assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.
fx=(ax-aint(ax))
fy=(ay-aint(ay))
kx=aint(ax)
ky=aint(ay)
dl=sqrt(((ay-float(ky+2))**2.+(ax-float(kx+2))**2.)/9.)

```

```

d2=sqrt(((ay-float(ky+2))**2.+(ax-float(kx-1))**2.)/9.)
d3=sqrt(((ay-float(ky-1))**2.+(ax-float(kx-1))**2.)/9.)
d4=sqrt(((ay-float(ky-1))**2.+(ax-float(kx+2))**2.)/9.)
d5=sqrt(((ay-float(ky+2))**2.+(ax-float(kx+1))**2.)/9.)
d6=sqrt(((ay-float(ky+2))**2.+(ax-float(kx))**2.)/9.)
d7=sqrt(((ay-float(ky-1))**2.+(ax-float(kx))**2.)/9.)
d8=sqrt(((ay-float(ky-1))**2.+(ax-float(kx+1))**2.)/9.)
d9=sqrt(((ay-float(ky+1))**2.+(ax-float(kx+2))**2.)/9.)
d10=sqrt(((ay-float(ky+1))**2.+(ax-float(kx-1))**2.)/9.)
d11=sqrt(((ay-float(ky))**2.+(ax-float(kx-1))**2.)/9.)
d12=sqrt(((ay-float(ky))**2.+(ax-float(kx+2))**2.)/9.)
d13=sqrt(((ay-float(ky+1))**2.+(ax-float(kx+1))**2.)/9.)
d14=sqrt(((ay-float(ky+1))**2.+(ax-float(kx))**2.)/9.)
d15=sqrt(((ay-float(ky))**2.+(ax-float(kx))**2.)/9.)
d16=sqrt(((ay-float(ky))**2.+(ax-float(kx+1))**2.)/9.)

```

```

e1=d1*ss(ky+2,kx+2)
e2=d2* ss(ky+2,kx-1)
e3=d3*ss(ky-1,kx-1)
e4=d4*ss(ky-1,kx+2)
e5=d5*ss(ky+2,kx+1)
e6=d6* ss(ky+2,kx)
e7=d7*ss(ky-1,kx)
e8=d8*ss(ky-1,kx+1)
e9=d9*ss(ky+1,kx+2)
e10=d10* ss(ky+1,kx-1)
e11=d11*ss(ky,kx-1)
e12=d12*ss(ky,kx+2)
e13=d13*ss(ky+1,kx+1)
e14=d14* ss(ky+1,kx)
e15=d15*ss(ky,kx)
e16=d16*ss(ky,kx+1)

```

```

b(j,i)=e1+e2+e3+e4+e5+e6+e7+e8+e9+e10+e11+e12+e13+e14+e15+e16

```

```

enddo
enddo
return
end

```

subroutine divide linear12n sum(a,n,b,m,theta)
real a(256,256),b(256,256),ss(256,256),w(256,256)

FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
THROUGH AN ANGLE THETA (given in degree)
anticlockwise placing the result at the
center of an array mxm b

convert theta into radians

pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)

initial ss

do i=1,m
do j=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo

compute new coordinates (xd,yd) of rotate array

nn = n/2 + 1
mm = m/2 + 1

Compensation of the center position for rotation.

xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)

start process

do j = 1,m
do i = 1,m

generate 4-quadrant cartesian plan

y = float(j-nn)
x = float(i-nn)
xd = x * costh - y * sinth
yd = x * sinth + y * costh

compute both real and integer coordinates of array b
associated with the above transformation for later use

ax =mm+ xd
ay = mm+ yd

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.

fx=(ax-aint(ax))
fy=(ay-aint(ay))
kx=aint(ax)
ky=aint(ay)

d1=sqrt(((ay-float(ky-1))**2.+(ax-float(kx+1))**2.)/5.)
d2=sqrt(((ay-float(ky+1))**2.+(ax-float(kx+2))**2.)/5.)
d3=sqrt(((ay-float(ky+2))**2.+(ax-float(kx))**2.)/5.)
d4=sqrt(((ay-float(ky))**2.+(ax-float(kx-1))**2.)/5.)
d5=sqrt(((ay-float(ky+1))**2.+(ax-float(kx+1))**2.)/5.)
d6=sqrt(((ay-float(ky+1))**2.+(ax-float(kx))**2.)/5.)
d7=sqrt(((ay-float(ky))**2.+(ax-float(kx))**2.)/5.)
d8=sqrt(((ay-float(ky))**2.+(ax-float(kx+1))**2.)/5.)
d9=sqrt(((ay-float(ky))**2.+(ax-float(kx+2))**2.)/5.)
d10=sqrt(((ay-float(ky+2))**2.+(ax-float(kx+1))**2.)/5.)
d11=sqrt(((ay-float(ky+1))**2.+(ax-float(kx-1))**2.)/5.)
d12=sqrt(((ay-float(ky-1))**2.+(ax-float(kx))**2.)/5.)

e1=d1*a(j,i)
e2=d2*a(j,i)
e3=d3*a(j,i)
e4=d4*a(j,i)
e5=d5*a(j,i)


```
e6=d6* a(j,i)
e7=d7*a(j,i)
e8=d8*a(j,i)
e9=d9*a(j,i)
e10=d10* a(j,i)
e11=d11*a(j,i)
e12=d11*a(j,i)
```

```
b(ky-1,kx+1)=b(ky-1,kx+1)+e1
b(ky+1,kx+2)=b(ky+1,kx+2)+e2
b(ky+2,kx)=b(ky+2,kx)+e3
b(ky,kx-1)=b(ky,kx-1) +e4
b(ky+1,kx+1)=b(ky+1,kx+1)+e5
b(ky+1,kx)=b(ky+1,kx)+e6
b(ky,kx)=b(ky,kx)+e7
b(ky,kx+1)=b(ky,kx+1)+e8
b(ky,kx+2)=b(ky,kx+2)+e9
b(ky+2,kx+1)=b(ky+2,kx+1)+e10
b(ky+1,kx-1)=b(ky+1,kx-1)+e11
b(ky-1,kx)=b(ky-1,kx)+e12
```

```
enddo
enddo
return
end
```

```

subroutine rotate divide linear16n sum(a,n,b,m,theta)
real a(256,256),b(256,256),ss(256,256),w(256,256)

```

```

FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
           THROUGH AN ANGLE THETA (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b

```

```

convert theta into radians

```

```

pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)

```

```

initial ss

```

```

do i=1,m
doj=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo

```

```

compute new coordinates (xd,yd) of rotate array

```

```

nn = n/2 + 1
mm = m/2 + 1

```

```

Compensation of the center position for rotation.

```

```

xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)

```

```

start process

```

```

do j = 1,m
do i = 1,m

```

```

generate 4-quadrant cartesian plan

```

```

y = float(j-nn)
x = float(i-nn)
xd = x * costh - y * sinth
yd = x * sinth + y * costh

```

```

compute both real and integer coordinates of array b
associated with the above transformation for later use

```

```

ax =mm+ xd
ay = mm+ yd

```

```

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.

```

```

fx=(ax-aint(ax))
fy=(ay-aint(ay))
kx=aint(ax)
ky=aint(ay)

```

```

d1=sqrt(((ay-float(ky+2))**2.+(ax-float(kx+2))**2.)/9.)
d2=sqrt(((ay-float(ky+2))**2.+(ax-float(kx-1))**2.)/9.)
d3=sqrt(((ay-float(ky-1))**2.+(ax-float(kx-1))**2.)/9.)
d4=sqrt(((ay-float(ky-1))**2.+(ax-float(kx+2))**2.)/9.)
d5=sqrt(((ay-float(ky+2))**2.+(ax-float(kx+1))**2.)/9.)
d6=sqrt(((ay-float(ky+2))**2.+(ax-float(kx))**2.)/9.)
d7=sqrt(((ay-float(ky-1))**2.+(ax-float(kx))**2.)/9.)
d8=sqrt(((ay-float(ky-1))**2.+(ax-float(kx+1))**2.)/9.)
d9=sqrt(((ay-float(ky+1))**2.+(ax-float(kx+2))**2.)/9.)
d10=sqrt(((ay-float(ky+1))**2.+(ax-float(kx-1))**2.)/9.)
d11=sqrt(((ay-float(ky))**2.+(ax-float(kx-1))**2.)/9.)
d12=sqrt(((ay-float(ky))**2.+(ax-float(kx+2))**2.)/9.)
d13=sqrt(((ay-float(ky+1))**2.+(ax-float(kx+1))**2.)/9.)
d14=sqrt(((ay-float(ky+1))**2.+(ax-float(kx))**2.)/9.)
d15=sqrt(((ay-float(ky))**2.+(ax-float(kx))**2.)/9.)
d16=sqrt(((ay-float(ky))**2.+(ax-float(kx+1))**2.)/9.)

```

```

el=d1*a(j,i)

```

```
e2=d2*a(j,i)
e3=d3*a(j,i)
e4=d4*a(j,i)
e5=d5*a(j,i)
e6=d6* a(j,i)
e7=d7*a(j,i)
e8=d8*a(j,i)
e9=d9*a(j,i)
e10=d10* a(j,i)
e11=d11*a(j,i)
e12=d11*a(j,i)
e13=d13*a(j,i)
e14=d14* a(j,i)
e15=d15*a(j,i)
e16=d16*a(j,i)
```

```
b(ky+2,kx+2)=b(ky+2,kx+2)+e1
b(ky+2,kx-1)=b(ky+2,kx-1)+e2
b(ky-1,kx-1)=b(ky-1,kx-1)+e3
b(ky-1,kx+2)=b(ky-1,kx+2) +e4
b(ky+2,kx+1)=b(ky+2,kx+1)+e5
b(ky+2,kx)=b(ky+2,kx)+e6
b(ky-1,kx)=b(ky-1,kx)+e7
b(ky-1,kx+1)=b(ky-1,kx+1)+e8
b(ky+1,kx+2)=b(ky+1,kx+2)+e9
b(ky+1,kx-1)=b(ky+1,kx-1)+e10
b(ky,kx-1)=b(ky,kx-1)+e11
b(ky,kx+2)=b(ky,kx+2)+e12
b(ky+1,kx+1)=b(ky+1,kx+1)+e13
b(ky+1,kx)=b(ky+1,kx)+e14
b(ky,kx)=b(ky,kx)+e15
b(ky,kx+1)=b(ky,kx+1)+e16
```

```
enddo
enddo
return
end
```

```
subroutine moving average filter image(x,n,fl,msize)
real x(256,256),fl(256,256),xy(256,256),a(256,256)
```

FUNCTION: Returns a digital image, filtered using the
neighbourhood averaging method.

AUTHOR: J M Blackledge

PARAMETERS

Input: x - digital image
n - array size
msize - size of mask or window over which neighbourhood
of pixels is selected.

Output: fl - filtered image

EXTERNAL SUBROUTINES: None

INTERNAL SUBROUTINES: average

Declare internal workspace.

```
real y(256,256)
real al(256,256)
```

First, frame the image x with an array of zeros
anint((msize-1)/2 pixels wide. The framed image
is stored in the m x m array y.

Note: function anint is used in case msize is even;
msize = 2,4,6...

```
m=n+msize-1
```

```
do i=1,m
do j=1,m
y(j,i)=0. !initialise
enddo
enddo
```

```
iframe=anint((float(msize)-1.)/2.)
do i=1,n
do j=1,n
y(j+iframe,i+iframe)=x(j,i)
enddo
enddo
```

Start the process.

```
do i=1,n
do j=1,n
```

Find the msize x msize neighbourhood of pixels at centre (j,i)
and store the result in the array a.

```
do k=1,msize
do kk=1,msize
al(kk,k)=y(j-1+kk,i-1+k)
enddo
enddo
```

Compute the average value of a.

```
call average(a1,av,msize)
```

Store the average value of the neighbourhood in f1(j,i)

```
f1(j,i)=av
```

repeat the process , i.e., move on to the next centre (j,i).

```
enddo
```

```
enddo
```

```
return
```

```
end
```

```
subroutine average(a1,av,msize)
```

```
real a1(256,256)
```

FUNCTION: Computes the average value av of a 2D array a
size msize x msize.

Sum pixels in array a.

```
sum=0.
```

```
do i=1,msize
```

```
do j=1,msize
```

```
sum=sum+a1(j,i)
```

```
enddo
```

```
enddo
```

Compute the average value.

```
mask=msize*msize
```

```
av=sum/float(mask)
```

```
return
```

```
end
```

```

subroutine fir_convolution_filter_image (s,p,f,n,nw)
  real s(256,256),p(256,256),f(256,256),xy(256,256)
*****
function: Computes the fir(convolution) filter (local weighted average) of
a digital images s, i.e. computes the discrete convolution of s
with p.
Parameters:
Input:   s - digital image
         p - fir filter or 'kernel'
         n - array size of image
         nw - array size of kernel (must be <=n)
Output:  f - filtered image
external subroutine :none
internal subroutine :convolve
*****
Declare internal workspace.
      real ss(256,256),x(256,256),y(256,256)
First, zero pad image s with anint((nw-1)/2) zeros storing the result in
array ss. The function anint is used in case the value of nw is even.
      mm=n+nw-1
      do i=1,mm
      do j=1,mm
      ss(j,i)=0. !initialise
      enddo
      enddo
      iframe=anint((float(nw)-1.)/2.)
      do i=1,n
      do j=1,n
      ss(j+iframe,i+iframe)=s(j,i)
      enddo
      enddo
Start the process
      do i=1,n
      do j=1,n
Write the neighbourhood of samples in window nw to array x.
      do k=1,nw
      do kk=1,nw
      x(kk,k)=ss(j-1+kk,i-1+k)
      enddo
      enddo
Compute convolution of x with kernel p.
      call convolve(x,p,nw,con)
Store result of convolution in f.
      f(j,i)=con
Repeat the process, i.e. move window to the next point.
      enddo
      enddo
      return
      end
*****
      subroutine convolve(x,p,nw,con)
      real x(256,256),p(256,256)
      sum=0.
      do i=1,nw
      do j=1,nw
      sum=sum+x(j,i)*p(nw-j+1,nw-i+1)
      enddo
      enddo
      con=sum
      return
      end

```

```
subroutine median_filter_image(x,n,f2,msize)
real x(256,256),f2(256,256),f1(256,256),xy(256,256)
```

FUNCTION: Returns a median filtered digital image.

AUTHOR: J M Blackledge

PARAMETERS

Input: x - digital image
n - array size
msize - size of mask or window over which neighbourhood
of pixels is selected.

Output: f2 - filtered image

EXTERNAL SUBROUTINES: None

INTERNAL SUBROUTINES: median

Declare internal workspace.

```
real y(256,256)
real a(256,256)
```

First, frame the image x with an array of zeros
anint((msize-1)/2 pixels wide. The framed image
is stored in the m x m array y.

Note: function anint is used in case msize is even;
msize = 2,4,6...

```
m=n+msize-1
```

```
do i=1,m
do j=1,m
y(j,i)=0. !initialise
enddo
enddo
```

```
iframe=anint((float(msize)-1.)/2.)
do i=1,n
do j=1,n
y(j+iframe,i+iframe)=x(j,i)
enddo
enddo
```

Start the process.

```
do i=1,n
do j=1,n
```

Find the msize x msize neighbourhood of pixels at centre (j,i)
and store the result in the array a.

```
do k=1,msize
do kk=1,msize
a(kk,k)=y(j-1+kk,i-1+k)
enddo
enddo
```

Compute the median value of a.

```
call median(a,xmed,msize)
```

Store the median of the neighbourhood in f2(j,i)

```
f2(j,i)=xmed
```

repeat the process , i.e., move on to the next centre (j,i).

```
enddo
```

```
enddo
```

```
return
```

```
end
```

```
subroutine median(a,xmed,msize)
```

```
real a(256,256),b(256)
```

FUNCTION: Computes the median of a 2D array a
size msize x msize.

Convert array a into row vector b.

```
kkk=1
```

```
do k=1,msize
```

```
do kk=1,msize
```

```
b(kkk)=a(kk,k)
```

```
kkk=kkk+1
```

```
enddo
```

```
enddo
```

Compute the median value of b.

```
mask=msize*msize
```

```
call mdian2(b,mask,xmed)
```

```
return
```

```
end
```

```
SUBROUTINE MDIAN2(X,N,XMED)
```

FUNCTION: Given an array X of N numbers, returns their median value
XMED. The array X is not modified, and is accessed
sequentially in each consecutive pass.

```
dim X(N)
```

```
real x(256)
```

```
LOGICAL SORTED
```

```
SORTED = .FALSE.
```

```
IF (.NOT.SORTED) THEN
```

```
SORTED = .TRUE.
```

```
DO I=1,N-1
```

```
IF (X(I).LT.X(I+1)) THEN
```

```
TEMP=X(I)
```

```
X(I)=X(I+1)
```

```
X(I+1)=TEMP
```

```
SORTED = .FALSE.
```

```
ENDIF
```

```
enddo
```

```
GOTO 15
```

```
ENDIF
```

Comput the median

```
if (mod(n,2).eq.0) then
```

```
nn=n/2
```



```
xmed=.5*(x(nn)+x(nn+1))  
else  
nn=n/2+1  
xmed=x(nn)  
endif  
RETURN  
END
```

subroutine threshold_filter_image(x,n,f)

FUNCTION: returns the rotated image with removing spikes.

PARAMETERS

Input: x -image

n - size of image = n x n

Output: f - Threshold image

INTERNAL SUBROUTINES: None

EXTERNAL SUBROUTINES: None

real x(256,256),f(256,256)

real max

Find the absolute maximum value of the image.

max=0.

do i=1,n

do j=1,n

x1=x(j,i)

if(x1.gt.max) max=x1

enddo

enddo

write(6,*)'max=',x1

thr=max/2.

Exclude the spike value of array x

f(0,0)=f(1,1)

do i=1,n

do j=1,n

if (x(j,i).gt.thr) then

f(j,i) = x(j,i)

else

assign value of f at (j-1,i-1) to the nearest_point cell
of array f

lie.

f(j,i) = f(j-1,i-1)

endif

enddo

enddo

return

end

APPENDIX B

FAST RADON TRANSFORM AND 1D PROCESS ROUTINES

```

program Fayed 1d process.for
real a(256,256),xy(256,256),xx(256,256),ff(256,256)
real h(256),h1(256),h2(256),h3(256),h4(256),s(256)
real h5(256),b(256,256),yy(256,256),abser(256,256)
character*6 answer

```

FUNCTION: Generates an image on a DEC digital workstation using data from a specified data file or produces a plot file. Also, reconstruct the object using different rotation techniques and apply the 1D & 2D median filter.

EXTERNAL SUBROUTINES: UNIRAS Library - groute
gopen
gclose

COMMON - read_image, timer_start, time_elapsed,
display_image, 30:Rotation_Techniques

Select route for gimage.

```

write(6,10)
format(1x,'View image (1) or generate plotfile (2)?'$)
read(5,*)id
if(id.eq.1)then
call groute('s mx11; e') !for VAX workstations 2000 or 3200
endif

if(id.eq.2)then
write(6,11)
format(1x,'Black and white (1) or colour (2) plotfile?'$)
read(6,*)ic
if(ic.eq.1)then
call groute('s hposta4; e')
endif
if(ic.eq.2)then
call groute('s hqcs100a4; e')
endif
if((ic.ne.1).and.(ic.ne.2))goto 2
endif

if((id.ne.1).and.(id.ne.2))goto 1

call gopen !open UNIRAS window
call grpsiz(xsiz,ysiz)

```

Ask user for image parameters:

1. size of image
2. colour coding with options

continue !for looping back to examine another data file.

Open data file and read contents by calling
subroutine read_image.

```

write(6,*)'enter your choice:'
write(6,*)'1. read_image'
write(6,*)'2. Generate_rectangle'
write(6,*)'3. Generate_circle'
write(6,*)'4. Generate_var_rectangle'
write(6,*)'5. Head_phantom'
read(5,*) kl
if (kl.eq.1) then
call read_image(a,n)
endif

```

```

if (kl.eq.2) then
write(6,*)'enter size of square image'
read(5,*)n
call rectangle(a,n)
call write_image(a,n)
endif
if (kl.eq.3) then
write(6,*)'enter size of square image'
read(5,*)n
call circle(a,n)
call write_image(a,n)
endif
if (kl.eq.4) then
write(6,*)'enter size of square image'
read(5,*)n
call var_rectangle(a,n)
call write_image(a,n)
endif
if (kl.eq.5) then
write(6,*)'enter size of square image'
read(5,*)n
call head_phantom(a,n)
call write_image(a,n)
endif
continue !for looping back to examine a different colour code.

write(6,*)'      UNIRAS 6 colour scales'
write(6,*)
write(6,*)'Default colour scale 1..... (1)'
write(6,*)'Default colour scale 2..... (2)'
write(6,*)'Grey scale..... (3)'
write(6,*)'Topographic colour scale..... (4)'
write(6,*)'Seismic colour scale..... (5)'
write(6,*)
write(6,80)
format(1x,'Scale (1-5)?'$)
read(5,*)icol
write(6,90)
format(1x,'Positive (1) or negative (-1)?'$)
read(5,*)ipol

```

Generate image using subroutine display_image.

```

size=min(((0.90*xsiz)/2.), (0.90*ysiz))
xpos=(xsiz-2.*size)/3.
ypos=(ysiz-size)/2.0
call gimori(xpos,ypos)
call gimsiz(size,size)
call disp_image(a,n,icol,ipol,size,xpos,ypos)
01      open (unit=1,file='spikel.128',status='new')
        format(4(1x,f14.4))
        write(1,101)((a(j,i),j=1,128),i=1,128)
        close(1)
0      call greset
        call gclear

m=nint(1.414213562*n)

write(6,*) '1. Implementing 1D median filter process'
write(6,*) '2. Implementing 2d median filter process'
write(6,*) '3. Reconstruction using central slice'
xpos=2.*(xsiz-2.*size)/3. + size
call gimori(xpos,ypos)
call gimsiz(size,size)

read(5,*) nfn
write(6,*) 'window size'
read(5,*)msize

```

```

if (nfn.eq.1) then
    call timer_start(context)
    call rtr_nearest_neig(a,n,xy,m)
    do i=1,m
    do j=1,m
    s(j)=xy(j,i)
    enddo
    call median_filter_signal(s,m,msize)
    do k=1,m
    b(k,i)=s(k)
    enddo
    enddo
    call ift_nearest_neig(b,m,xx,n)
    call time_elapsed(context)
call disp_image(xx,n,icol,ipol,size,xpos,ypos)
    open (unit=1,file='median1d.128',status='new')
    format(4(1x,f14.4))
    write(1,117)((xx(j,i),j=1,128),i=1,128)
    close(1)
    call write_image(xy,m)
endif
if (nfn.eq.2) then
    call timer_start(context)
    call median_filter_image(a,n,ff,msize)
    call time_elapsed(context)
call disp_image(ff,n,icol,ipol,size,xpos,ypos)
    open (unit=1,file='median2d.128',status='new')
    format(4(1x,f14.4))
    write(1,118)((ff(j,i),j=1,128),i=1,128)
    close(1)
    call write_image(xy,m)
endif
if (nfn.eq.3) then
    call timer_start(context)
    call rtr_nearest_neig(a,n,xy,m)
    call time_elapsed(context)
size=min(((0.90*xsiz)/2.), (0.90*ysiz))
xpos=(xsiz-2.*size)/3.
ypos=(ysiz-size)/2.0
call gimori(xpos,ypos)
call gimsiz(size,size)
call disp_image(xy,m,icol,ipol,size,xpos,ypos)

    call timer_start(context)
    call ift_nearest_neig(xy,m,xx,n)
    call time_elapsed(context)
xpos=2.*(xsiz-2.*size)/3. + size
call gimori(xpos,ypos)
call gimsiz(size,size)
call disp_image(xx,n,icol,ipol,size,xpos,ypos)
    call write_image(xx,m)
endif
call greset
call gclear
call histogram_image(xx,n,h2,lev,gsum)
call display_histogram(h2,lev)
call greset
call gclear

```

Ask user if another is required.

```

    write(6,111)
1   format(1x,'Select another? <n>'$)
    read(5,222) answer
2   format(a6)
    if((answer.eq.'yes').or.(answer.eq.'y')) goto 16

```

Ask user if another display is required.

```
01      write(6,1001)
      format(1x,'Display another? <n>'$)
      read(5,1002)answer
02      format(a6)
      if((answer.eq.'yes').or.(answer.eq.'y'))goto 5
```

Terminate uniras.

Write out response to user indicating next stage for producing a hardcopy (in the case of option for generating a plotfile).

```
      if(ic.eq.1)then
      write(6,*)'Your plot file is called post.dat'
      endif
```

```
      if(ic.eq.2)then
      write(6,*)'Your plot file is called post.dat'
      endif
```

```
      call greset
      call gclear
      call gclose
      stop
      end
```


APPENDIX C

RECONSTRUCTION

ROUTINES


```

program Fayed_nearest.for
real a(256,256),xy(256,256),xx(256,256)
real h(256),h1(256),h2(256),h3(256),h4(256)
real h5(256),b(256,256),yy(256,256)
real abser(256,256),difer(256,256),per(256)
character*6 answer

```

FUNCTION: Generates an image on a DEC digital workstation using data from a specified data file or produces a plot file. Also, reconstruct the object using the nearest neighbour rotation techniques and 1D inverse filter.

EXTERNAL SUBROUTINES: UNIRAS Library - groute
gopen
gclose

DIPLIB - read_image, timer_start, time_elapsed,
display_image, 30:Rotation_Techniques

Select route for gimage.

```

write(6,10)
format(1x,'View image (1) or generate plotfile (2)?'$)
read(5,*)id
if(id.eq.1)then
call groute('s mx11; e') !for VAX workstations 2000 or 3200
endif

if(id.eq.2)then
write(6,11)
format(1x,'Black and white (1) or colour (2) plotfile?')
read(6,*)ic
if(ic.eq.1)then
call groute('s hposta4; e')
endif
if(ic.eq.2)then
call groute('s hqcs1004a; e')
endif
if((ic.ne.1).and.(ic.ne.2))goto 2
endif

if((id.ne.1).and.(id.ne.2))goto 1

call gopen !open UNIRAS window

```

Ask user for image parameters:

1. size of image
2. colour coding with options

continue !for looping back to examine another data file.

Open data file and read contents by calling
subroutine read_image.

```

write(6,*)'enter your choice:'
write(6,*)'1. read_image'
write(6,*)'2. Generate_rectangle'
write(6,*)'3. Generate_circle'
write(6,*)'4. Generate_var_rectangle'
write(6,*)'5. Generate_exp_2dimension'
write(6,*)'6. Generate_multi_circle'
read(5,*) k1
if (k1.eq.1) then
call read_image(a,n)

```

```

endif
if (kl.eq.2) then
write(6,*)'enter size of square image'
read(5,*)n
call rectangle(a,n)
call write_image(a,n)
endif
if (kl.eq.3) then
write(6,*)'enter size of square image'
read(5,*)n
call circle(a,n)
call write_image(a,n)
endif
if (kl.eq.4) then
write(6,*)'enter size of square image'
read(5,*)n
call var_rectangle(a,n)
call write_image(a,n)
endif
if (kl.eq.5) then
write(6,*)'enter size of square image'
read(5,*)n
call exp_2d(a,n)
call write_image(a,n)
endif
if (kl.eq.6) then
write(6,*)'enter size of square image'
read(5,*)n
call multi_circle(a,n)
call write_image(a,n)
endif
continue !for looping back to examine a different colour code.

write(6,*)'      UNIRAS 6 colour scales'
write(6,*)
write(6,*)'Default colour scale 1..... (1)'
write(6,*)'Default colour scale 2..... (2)'
write(6,*)'Grey scale..... (3)'
write(6,*)'Topographic colour scale..... (4)'
write(6,*)'Seismic colour scale..... (5)'
write(6,*)
write(6,80)
format(1x,'Scale (1-5)?'$)
read(5,*)icol
write(6,90)
format(1x,'Positive (1) or negative (-1)?'$)
read(5,*)ipol

```

Generate image using subroutine display_image.

```

call counter(a,n,necount,nzcount,npocount)
write(6,*)('necount= ',necount)
write(6,*)('zcount= ',nzcount)
write(6,*)('pocount= ',npocount)
call GSEGCR (1)
call display_image_grey(a,n,icol,ipol)
call GSEGCL (1)
      open (unit=1,file='data1.dat',status='new')
      format(4(1x,f14.4))
      write(1,101)((a(j,i),j=1,128),i=1,128)
      close(1)
01
0 call greset
call gclear
write(6,*)'enter grey_level='
read(5,*)lev
call histogram_image(a,n,h,lev,gsum)
write(6,*)'grey_sum9_16=',gsum
call GSEGCR (2)

```

```

call display_histogram(h,lev)
call GSEGCL (2)
  call greset
  call gclear
  call norm_histogram(h,lev,n,h1,dev,area)
write(6,*) 'deviation=',dev
write(6,*) 'Histogram_Area=',area
call GSEGCR (3)
call display_histogram_norm(h1,lev)
call GSEGCL (3)
  call greset
  call gclear
m=nint(1.414213562*n)

write (6,*) 'Computing reconstruction using nearest_neig.:'
write(6,*) ' 1. optimum rotation algorithm'
write(6,*) ' 2. normal rotation algorithm '
read (5,*) ix
if ((ix.ne.1) .and. (ix.ne.2)) goto 20
if (ix.eq.1) then
  write(6,*) '1. Optimum nearest neighbor pixel'
  write(6,*) '2. Optimum four_neighbor pixels'
  write(6,*) '3. Optimum eight neighbor pixels '
  write(6,*) '4. Optimum 2 row neighbor pixels '
  write(6,*) '5. Optimum one row neighbor pixel'
  read(5,*) nfn
  if (nfn.eq.1) then
    call timer_start(context)
    call rtr_nearest_neig(a,n,xy,m)
    call time_elapsed(context)
    open (unit=1,file='onnp.181',status='new')
    format(4(1x,f14.4))
    write(1,116)((xy(j,i),j=1,181),i=1,181)
    close(1)
  call GSEGCR (4)
  call display_image_grey(xy,m,icol,ipol)
  call GSEGCL (4)
  call radon_gauss_noise (xy,m)
    open (unit=1,file='rgnon.181',status='new')
    format(4(1x,f14.4))
    write(1,116)((xy(j,i),j=1,181),i=1,181)
    close(1)
  call greset
  call gclear
  call GSEGCR (5)
    call display_image_grey(xy,m,icol,ipol)
  call GSEGCL (5)
    call write_image(xy,m)
    call timer_start(context)
    call irtr_nearest_neig(xy,m,xx,n)
    call time_elapsed(context)
    open (unit=1,file='onnr.128',status='new')
    format(4(1x,f14.4))
    write(1,117)((xx(j,i),j=1,128),i=1,128)
    close(1)
    call write_image(xy,m)
  endif
  if (nfn.eq.2) then
    call timer_start(context)
    call rtr_four_neig(a,n,xy,m)
    call time_elapsed(context)
    open (unit=1,file='ofnp.181',status='new')
    format(4(1x,f14.4))
    write(1,115)((xy(j,i),j=1,181),i=1,181)
    close(1)
  call GSEGCR (6)
  call display_image_grey(xy,m,icol,ipol)
  call GSEGCL (6)

```

18

```

        call write_image(xy,m)
        call timer_start(context)
        call irtr_four_neig(xy,m,xx,n)
        call time_elapsed(context)
        open (unit=1,file='ofnr.128',status='new')
        format(4(1x,f14.4))
        write(1,118)((xx(j,i),j=1,128),i=1,128)
        close(1)
        call write_image(xy,m)
    endif
    if (nfn.eq.3) then
        call timer_start(context)
        call rtr_eight_neig(a,n,xy,m)
        call time_elapsed(context)
call GSEGCR (7)
call display_image_grey(xy,m,icol,ipol)
call GSEGCL (7)
        call write_image(xy,m)
        call timer_start(context)
        call irtr_eight_neig(xy,m,xx,n)
        call time_elapsed(context)
        call write_image(xy,m)
    endif
    if (nfn.eq.4) then
        call timer_start(context)
        call rtr_2rp_neig(a,n,xy,m)
        call time_elapsed(context)
call GSEGCR (8)
call display_image_grey(xy,m,icol,ipol)
call GSEGCL (8)
        call write_image(xy,m)
        call timer_start(context)
        call irtr_2rp_neig(xy,m,xx,n)
        call time_elapsed(context)
        call write_image(xy,m)
    endif
    if (nfn.eq.5) then
        call timer_start(context)
        call rtr_lrp_neig(a,n,xy,m)
        call time_elapsed(context)
call GSEGCR (9)
call display_image_grey(xy,m,icol,ipol)
call GSEGCL (9)
        call write_image(xy,m)
        call timer_start(context)
        call irtr_lrp_neig(xy,m,xx,n)
        call time_elapsed(context)
        call write_image(xy,m)
    endif
elseif (ix.eq.2) then
    write(6,*) '1. Normal nearest neighbor pixel'
    write(6,*) '2. Normal four_neighbor pixels'
    write(6,*) '3. Normal eight_neighbor pixels '
    read(5,*) nfn
    if (nfn.eq.1) then
        call timer_start(context)
        call rtr_nearest(a,n,xy,m)
        call time_elapsed(context)
        open (unit=1,file='nnnp.181',status='new')
        format(4(1x,f14.4))
        write(1,123)((xy(j,i),j=1,181),i=1,181)
        close(1)
call GSEGCR (10)
call display_image_grey(xy,m,icol,ipol)
call GSEGCL (10)
        call write_image(xy,m)
        call timer_start(context)
        call irtr_nearest(xy,m,xx,n)

```

23

```

14      call time_elapsed(context)
        open (unit=1,file='nnnr.128',status='new')
        format(4(1x,f14.4))
        write(1,114)((xx(j,i),j=1,128),i=1,128)
        close(1)
        call write_image(xy,m)
    endif
    if (nfn.eq.2) then
        call timer_start(context)
        call rtr_four(a,n,xy,m)
        call time_elapsed(context)
        open (unit=1,file='nfnr.181',status='new')
22      format(4(1x,f14.4))
        write(1,122)((xy(j,i),j=1,181),i=1,181)
        close(1)
    call GSEGCR (11)
    call display_image_grey(xy,m,icol,ipol)
    call GSEGCL (11)
        call write_image(xy,m)
        call timer_start(context)
        call irtr_four(xy,m,xx,n)
        call time_elapsed(context)
        open (unit=1,file='nfnr.128',status='new')
21      format(4(1x,f14.4))
        write(1,121)((xx(j,i),j=1,128),i=1,128)
        close(1)
        call write_image(xy,m)
    endif
    if (nfn.eq.3) then
        call timer_start(context)
        call rtr_eight(a,n,xy,m)
        call time_elapsed(context)
    call GSEGCR (12)
    call display_image_grey(xy,m,icol,ipol)
    call GSEGCL (12)
        call write_image(xy,m)
        call timer_start(context)
        call irtr_eight(xy,m,xx,n)
        call time_elapsed(context)
        call write_image(xy,m)
    endif
else
endif
call greset
call gclear
call counter(xx,n,necount,nzecount,npocount)
write(6,*)('necount= ',necount)
write(6,*)('nzecount= ',nzecount)
write(6,*)('pocount= ',npocount)
call GSEGCR (13)
call display_image_grey(xx,n,icol,ipol)
call GSEGCL (13)
19      open (unit=1,file='datare.dat',status='new')
        format(4(1x,f14.4))
        write(1,119)((xx(j,i),j=1,128),i=1,128)
        close(1)
    call greset
    call gclear
    call histogram_image(xx,n,h2,lev,gsum)
    write(6,*)'grey_sum9_16=',gsum
    call GSEGCR (14)
    call display_histogram(h2,lev)
    call GSEGCL (14)
    call greset
    call gclear
    call norm_histogram(h2,lev,n,h3,dev,area)
    write(6,*)'deviation=',dev
    write(6,*)'Histogram_Area=',area

```

```

call GSEGCR (15)
call display_histogram_norm(h3,lev)
call GSEGCL (15)
  call greset
  call gclear
Compute least square error
  call least_square_error(a,xx,n,er,abser,std)
    open (unit=1,file='abseron.128',status='new')
61    format(4(1x,f14.4))
    write(1,161)((abser(j,i),j=1,128),i=1,128)
    close(1)

  write(6,*)'NMSE=',er
  write(6,*)'2D standard dev.=',std
Compute projection (Rotation) least square error
  call projection_square_error(a,xx,n,per)
  call display_signal(per,n)
  call mean_standard(per,n,xmean,st1)
  write(6,*)'eld=',xmean
  write(6,*)'1D standard dev.=',st1
  call write_signal(per,n)

1  write(6,*)'enter cutoff freq. in pixels:'
  read(5,*) icut
  write(6,*)'enter order of the filter : '
  read(5,*) iord
  call blpf_image(xx,n,icut,iord)
  call GSEGCR (16)
    call display_image_grey(xx,n,icol,ipol)
  call GSEGCL (16)
    call greset
    call gclear
  call histogram_image(xx,n,h4,lev,gsum)
  write(6,*)'grey_sum9_16=',gsum
  call GSEGCR (17)
  call display_histogram(h4,lev)
  call GSEGCL (17)
    call greset
  call gclear
  call norm_histogram(h4,lev,n,h5,dev,area)
  write(6,*)'deviation=',dev
  write(6,*)'Histogram_Area=',area
  call GSEGCR (18)
  call display_histogram_norm(h5,lev)
  call GSEGCL (18)
    call greset
  call gclear

Ask user if another filterig technique is required.

  write(6,110)
10  format(1x,'filter another? <n>'$)
  read(5,111)re
11  format(a6)
  if((re.eq.'y').or.(re.eq.'Y'))goto 21
  call compare_rotate_image(xy,m)
Ask user if another reconstruction technique is required.

  write(6,112)
2  format(1x,'reconstruct another? <n>'$)
  read(5,113)an
3  format(a6)
  if((an.eq.'y').or.(an.eq.'Y'))goto 20

Ask user if another display is required.

  write(6,1001)

```

```
01      format(1x,'Display another? <n>'$)  
      read(5,1002)answer  
02      format(a6)  
      if((answer.eq.'yes').or.(answer.eq.'y'))goto 5
```

Terminate uniras.

Write out response to user indicating next stage for producing
a hardcopy (in the case of option for generating a plotfile).

```
      if(ic.eq.1)then  
      write(6,*)'Your plot file is called post.dat'  
      endif
```

```
      if(ic.eq.2)then  
      write(6,*)'Your plot file is called post.dat'  
      endif
```

```
      call greset  
      call gclear  
      call gclose  
      stop  
      end
```

```

program Fayed_interpolate.for
real a(256,256),xy(256,256),xx(256,256),per(256)
real h(256),h1(256),h2(256),h3(256),h4(256)
real h5(256),b(256,256),yy(256,256),abser(256,256)
character*6 answer

```

FUNCTION: Generates an image on a DEC digital workstation using data from a specified data file or produces a plot file. Also, reconstruct the object using the interpolation techniques and 1D inverse filter.

EXTERNAL SUBROUTINES: UNIRAS Library - groute
gopen
gclose

DIPLIB - read_image, timer_start, time_elapsed,
display_image, 30:Rotation_Techniques

Select route for gimage.

```

write(6,10)
format(1x,'View image (1) or generate plotfile (2)?'$)
read(5,*)id
if(id.eq.1)then
call groute('s mx11; e') !for VAX workstations 2000 or 3200
endif

if(id.eq.2)then
write(6,11)
format(1x,'Black and white (1) or colour (2) plotfile?'$)
read(6,*)ic
if(ic.eq.1)then
call groute('s hposta4; e')
endif
if(ic.eq.2)then
call groute('s hqcs1004a; e')
endif
if((ic.ne.1).and.(ic.ne.2))goto 2
endif

if((id.ne.1).and.(id.ne.2))goto 1

call gopen !open UNIRAS window

```

Ask user for image parameters:

1. size of image
2. colour coding with options

continue !for looping back to examine another data file.

Open data file and read contents by calling
subroutine read_image.

```

write(6,*)'enter your choice:'
write(6,*)'1. read_image'
write(6,*)'2. Generate_rectangle'
write(6,*)'3. Generate_circle'
write(6,*)'4. Generate_var_rectangle'
read(5,*) k1
if (k1.eq.1) then
call read_image(a,n)
endif
if (k1.eq.2) then
write(6,*)'enter size of square image'

```



```

read(5,*)n
call rectangle(a,n)
call write_image(a,n)
endif
if (kl.eq.3) then
write(6,*)'enter size of square image'
read(5,*)n
call circle(a,n)
call write_image(a,n)
endif
if (kl.eq.4) then
write(6,*)'enter size of square image'
read(5,*)n
call var_rectangle(a,n)
call write_image(a,n)
endif
continue !for looping back to examine a different colour code.

```

```

write(6,*)'      UNIRAS 6 colour scales'
write(6,*)
write(6,*)'Default colour scale 1..... (1)'
write(6,*)'Default colour scale 2..... (2)'
write(6,*)'Grey scale..... (3)'
write(6,*)'Topographic colour scale..... (4)'
write(6,*)'Seismic colour scale..... (5)'
write(6,*)
write(6,80)
format(1x,'Scale (1-5)?'$)
read(5,*)icol
write(6,90)
format(1x,'Positive (1) or negative (-1)?'$)
read(5,*)ipol

```

Generate image using subroutine display_image.

```

call counter(a,n,necount,nzcount,npocount)
write(6,*)('necount= ',necount)
write(6,*)('zcount= ',nzcount)
write(6,*)('pocount= ',npocount)
call display_image_grey(a,n,icol,ipol)
      open (unit=1,file='data1.dat',status='new')
      format(4(1x,f14.4))
      write(1,101)((a(j,i),j=1,128),i=1,128)
      close(1)

```

```

call greset
call gclear
write(6,*)'enter grey_level='
read(5,*)lev
call histogram_image(a,n,h,lev,gsum)
write(6,*)'grey_sum9_16=',gsum
call display_histogram(h,lev)
call greset
call gclear
call norm_histogram(h,lev,n,h1,dev,area)
write(6,*)'deviation=',dev
write(6,*)'Histogram_Area=',area
call display_histogram_norm(h1,lev)
call greset
call gclear
m=nint(1.414213562*n)

```

```

write(6,*) 'Computing reconstruction using Interpolation:'
write(6,*) ' 1. Forward pixel mapping algorithm'
write(6,*) ' 2. Reverse pixel mapping algorithm '
read(5,*) ix
if ((ix.ne.1) .and. (ix.ne.2)) goto 20
if (ix.eq.1) then
write(6,*) '1. Bilinear (Rectangluar) interpolation'

```

```

write(6,*) '2. Triangle interpolation'
write(6,*) '3. Two_Dimension linear interpolation '
read(5,*) nfn
  if (nfn.eq.1) then
    call timer_start(context)
    call rtr_rectangle_sum(a,n,xy,m)
    call time_elapsed(context)
call display_image_grey(xy,m,icol,ipol)
    call write_image(xy,m)
    call timer_start(context)
    call irtr_rectangle_sum(xy,m,xx,n)
    call time_elapsed(context)
    call write_image(xy,m)
  endif
  if (nfn.eq.2) then
    call timer_start(context)
    call rtr_triangle_sum(a,n,xy,m)
    call time_elapsed(context)
call display_image_grey(xy,m,icol,ipol)
    open (unit=1,file='datar.dat',status='new')
    format(4(1x,f14.4))
    write(1,118)((xy(j,i),j=1,181),i=1,181)
    close(1)
    call write_image(xy,m)
    call timer_start(context)
    call irtr_triangle_sum(xy,m,xx,n)
    call time_elapsed(context)
    call write_image(xy,m)
  endif
  if (nfn.eq.3) then
    call timer_start(context)
    call rtr_2dlinear_sum(a,n,xy,m)
    call time_elapsed(context)
call display_image_grey(xy,m,icol,ipol)
    call write_image(xy,m)
    call timer_start(context)
    call irtr_2dlinear_sum(xy,m,xx,n)
    call time_elapsed(context)
    call write_image(xy,m)
  endif
elseif (ix.eq.2) then
  write(6,*) '1. Divide bilinear(rectangle) interpolation'
  write(6,*) '2. Divide triangle interpolation'
  write(6,*) '3. Divide two dimension linear interpolation'
  read(5,*) nfn
    if (nfn.eq.1) then
      call timer_start(context)
      call rtr_divide_rectangle_sum(a,n,xy,m)
      call time_elapsed(context)
call display_image_grey(xy,m,icol,ipol)
      call write_image(xy,m)
      call timer_start(context)
      call irtr_divide_rectangle_sum(xy,m,xx,n)
      call time_elapsed(context)
      call write_image(xy,m)
    endif
    if (nfn.eq.2) then
      call timer_start(context)
      call rtr_divide_triangle_sum(a,n,xy,m)
      call time_elapsed(context)
call display_image_grey(xy,m,icol,ipol)
      open (unit=1,file='datar.dat',status='new')
      format(4(1x,f14.4))
      write(1,118)((xy(j,i),j=1,181),i=1,181)
      close(1)
      call write_image(xy,m)
      call timer_start(context)
      call irtr_divide_triangle_sum(xy,m,xx,n)

```

```

        call time_elapsed(context)
        call write_image(xy,m)
    endif
    if (nfn.eq.3) then
        call timer_start(context)
        call rtr_divide_2dlinear_sum(a,n,xy,m)
        call time_elapsed(context)
    call display_image_grey(xy,m,icol,ipol)
        call write_image(xy,m)
        call timer_start(context)
        call irtr_divide_2dlinear_sum(xy,m,xx,n)
        call time_elapsed(context)
        call write_image(xy,m)
    endif
else
endif
call greset
call gclear
call display_image_grey(xx,n,icol,ipol)
    open (unit=1,file='datare.dat',status='new')
    format(4(1x,f14.4))
    write(1,119)((xx(j,i),j=1,128),i=1,128)
    close(1)
call greset
call gclear
call histogram_image(xx,n,h2,lev,gsum)
write(6,*)'grey_sum9_16=',gsum
call display_histogram(h2,lev)
    call greset
    call gclear
    call norm_histogram(h2,lev,n,h3,dev,area)
    write(6,*)'deviation=',dev
    write(6,*)'Histogram Area=',area
    call display_histogram_norm(h3,lev)
    call greset
    call gclear
Compute least square error
    call least_square_error(a,xx,n,er,abser,std)
    open (unit=1,file='abseron.128',status='new')
    format(4(1x,f14.4))
    write(1,161)((abser(j,i),j=1,128),i=1,128)
    close(1)

    write(6,*)'NMSE=',er
    write(6,*)'2D standard dev.=',std
Compute projection (Rotation) least square error
    call projection_square_error(a,xx,n,per)
    call display_signal(per,n)
    call mean_standard(per,n,xmean,stl)
    write(6,*)'eld=',xmean
    write(6,*)'1D standard dev.=',stl
    call write_signal(per,n)

write(6,*)'enter cutoff freq. in pixels:'
read(5,*) icut
write(6,*)'enter order of the filter :'
read(5,*) iord
    call blpf_image(xx,n,icut,iord)
    call display_image_grey(xx,n,icol,ipol)
    call greset
    call gclear
    call histogram_image(xx,n,h4,lev,gsum)
    write(6,*)'grey_sum9_16=',gsum
    call display_histogram(h4,lev)
    call greset
    call gclear

```

```

call norm_histogram(h4,lev,n,h5,dev,area)
write(6,*)'deviation=',dev
write(6,*)'Histogram_Area=',area
call display_histogram_norm(h5,lev)
call greset
call gclear

```

Ask user if another filterig technique is required.

```

10 write(6,110)
format(1x,'filter another? <n>','$)
read(5,111)re
11 format(a6)
if((re.eq.'y').or.(re.eq.'Y'))goto 21
call compare_rotate_image(xy,m)

```

Ask user if another reconstruction technique is required.

```

write(6,112)
2 format(1x,'reconstruct another? <n>','$)
read(5,113)an
3 format(a6)
if((an.eq.'y').or.(an.eq.'Y'))goto 20

```

Ask user if another display is required.

```

write(6,1001)
01 format(1x,'Display another? <n>','$)
read(5,1002)answer
02 format(a6)
if((answer.eq.'yes').or.(answer.eq.'y'))goto 5

```

Terminate uniras.

Write out response to user indicating next stage for producing a hardcopy (in the case of option for generating a plotfile).

```

if(ic.eq.1)then
write(6,*)'Your plot file is called post.dat'
endif

```

```

if(ic.eq.2)then
write(6,*)'Your plot file is called post.dat'
endif

```

```

call greset
call gclear
call gclose
stop
end

```

```

program Fayed Multi interpolation.for
real a(256,256),xy(256,256),xx(256,256)
real h(256),h1(256),h2(256),h3(256),h4(256)
real h5(256),b(256,256),yy(256,256),abser(256,256)
character*6 answer

```

FUNCTION: Generates an image on a DEC digital workstation using data from a specified data file or produces a plot file. Also, reconstruct the object using multi-interpolation techniques and 1D inverse filter

EXTERNAL SUBROUTINES: UNIRAS Library - groute
gopen
gclose

DIPLIB - read_image, timer_start, time_elapsed,
display_image, 30:Rotation_Techniques

Select route for gimage.

```

write(6,10)
format(1x,'View image (1) or generate plotfile (2)?'$)
read(5,*)id
if(id.eq.1)then
call groute('s mx11; e') !for VAX workstations 2000 or 3200
endif

```

```

if(id.eq.2)then
write(6,11)
format(1x,'Black and white (1) or colour (2) plotfile?'$)
read(6,*)ic
if(ic.eq.1)then
call groute('s hposta41; e')
endif
if(ic.eq.2)then
call groute('s ht4692; e')
endif
if((ic.ne.1).and.(ic.ne.2))goto 2
endif

```

```

if((id.ne.1).and.(id.ne.2))goto 1

```

```

call gopen !open UNIRAS window

```

Ask user for image parameters:

1. size of image
2. colour coding with options

continue !for looping back to examine another data file.

Open data file and read contents by calling
subroutine read_image.

```

write(6,*)'enter your choice:'
write(6,*)'1. read_image'
write(6,*)'2. Generate_rectangle'
write(6,*)'3. Generate_circle'
write(6,*)'4. Generate_var_rectangle'
read(5,*) k1
if (k1.eq.1) then
call read_image(a,n)
endif
if (k1.eq.2) then
write(6,*)'enter size of square image'

```

```

read(5,*)n
call rectangle(a,n)
call write_image(a,n)
endif
if (kl.eq.3) then
write(6,*)'enter size of square image'
read(5,*)n
call circle(a,n)
call write_image(a,n)
endif
if (kl.eq.4) then
write(6,*)'enter size of square image'
read(5,*)n
call var_rectangle(a,n)
call write_image(a,n)
endif
continue !for looping back to examine a different colour code.

```

```

write(6,*)'      UNIRAS 6 colour scales'
write(6,*)
write(6,*)'Default colour scale 1..... (1)'
write(6,*)'Default colour scale 2..... (2)'
write(6,*)'Grey scale..... (3)'
write(6,*)'Topographic colour scale..... (4)'
write(6,*)'Seismic colour scale..... (5)'
write(6,*)
write(6,80)
format(1x,'Scale (1-5)?'$)
read(5,*)icol
write(6,90)
format(1x,'Positive (1) or negative (-1)?'$)
read(5,*)ipol

```

Generate image using subroutine display_image.

```

call counter(a,n,necount,nzcount,npocount)
write(6,*)('necount= ',necount)
write(6,*)('zcount= ',nzcount)
write(6,*)('pocount= ',npocount)
call display_image_grey(a,n,icol,ipol)
      open (unit=1,file='data1.dat',status='new')
      format(4(1x,f14.4))
      write(1,101)((a(j,i),j=1,128),i=1,128)
      close(1)

```

```

call greset
call gclear
write(6,*)'enter grey_level='
read(5,*)lev
call histogram_image(a,n,h,lev,gsum)
write(6,*)'grey_sum9_16=',gsum
call display_histogram(h,lev)
call greset
call gclear
call norm_histogram(h,lev,n,h1,dev,area)
write(6,*)'deviation=',dev
write(6,*)'Histogram_Area=',area
call display_histogram_norm(h1,lev)
call greset
call gclear
m=nint(1.414213562*n)

```

```

write(6,*)'Computing reconstruction using multi_Interpolation:'
write(6,*)' 1. Forward pixel mapping algorithm'
write(6,*)' 2. Reverse pixel mapping algorithm'
read(5,*)ix
if ((ix.ne.1) .and. (ix.ne.2)) goto 20
if (ix.eq.1) then
  write(6,*)'1. Rectangle_12 neigh. interpolation'

```

```

write(6,*) '2. Triangle_12 neigh. interpolation'
write(6,*) '3. Rectangle_16 neigh. interpolation'
write(6,*) '4. Triangle_16 neigh. interpolation'
write(6,*) '5. Triangle_16 neigh.(another) interpolation'
read(5,*) nfn
  if (nfn.eq.1) then
    call timer_start(context)
    call rtr_rectangle_12neigh_sum(a,n,xy,m)
    call time_elapsed(context)
call display_image_grey(xy,m,icol,ipol)
    call write_image(xy,m)
    call timer_start(context)
    call irtr_rectangle_12neigh_sum(xy,m,xx,n)
    call time_elapsed(context)
    call write_image(xy,m)
  endif
  if (nfn.eq.2) then
    call timer_start(context)
    call rtr_triangle_12neigh_sum(a,n,xy,m)
    call time_elapsed(context)
call display_image_grey(xy,m,icol,ipol)
    open (unit=1,file='datar.dat',status='new')
    format(4(1x,f14.4))
    write(1,118)((xy(j,i),j=1,181),i=1,181)
    close(1)
    call write_image(xy,m)
    call timer_start(context)
    call irtr_triangle_12neigh_sum(xy,m,xx,n)
    call time_elapsed(context)
    call write_image(xy,m)
  endif
  if (nfn.eq.3) then
    call timer_start(context)
    call rtr_rectangle_16neigh_sum(a,n,xy,m)
    call time_elapsed(context)
call display_image_grey(xy,m,icol,ipol)
    call write_image(xy,m)
    call timer_start(context)
    call irtr_rectangle_16neigh_sum(xy,m,xx,n)
    call time_elapsed(context)
    call write_image(xy,m)
  endif
  if (nfn.eq.4) then
    call timer_start(context)
    call rtr_triangle_16neigh_sum(a,n,xy,m)
    call time_elapsed(context)
call display_image_grey(xy,m,icol,ipol)
    call write_image(xy,m)
    call timer_start(context)
    call irtr_triangle_16neigh_sum(xy,m,xx,n)
    call time_elapsed(context)
    call write_image(xy,m)
  endif
  if (nfn.eq.5) then
    call timer_start(context)
    call rtr_triangle_16neigh_sum1(a,n,xy,m)
    call time_elapsed(context)
call display_image_grey(xy,m,icol,ipol)
    call write_image(xy,m)
    call timer_start(context)
    call irtr_triangle_16neigh_sum1(xy,m,xx,n)
    call time_elapsed(context)
    call write_image(xy,m)
  endif
elseif (ix.eq.2) then
  write(6,*) '1. Divide_rectangle_12 neigh. interpolation'
  write(6,*) '2. Divide_triangle_12 neigh. interpolation'
  write(6,*) '3. Divide_rectangle_16 neigh. interpolation'

```

```

write(6,*) '4. Divide_triangle_16_neigh. interpolation'
write(6,*) '5. Divide_triangle_16_neigh.(another) inte.'
read(5,*) nfn
  if (nfn.eq.1) then
    call timer_start(context)
    call rtr_divide_rectangle12n_sum(a,n,xy,m)
    call time_elapsed(context)
call display_image_grey(xy,m,icol,ipol)
    call write_image(xy,m)
    call timer_start(context)
    call irtr_divide_rectangle12n_sum(xy,m,xx,n)
    call time_elapsed(context)
    call write_image(xy,m)
  endif
else
endif
call greset
call gclear
call counter(xx,n,necount,nzcount,npocount)
write(6,*) ('necount= ',necount)
write(6,*) ('nzcount= ',nzcount)
write(6,*) ('pocount= ',npocount)
call display_image_grey(xx,n,icol,ipol)
  open (unit=1,file='dataare.dat',status='new')
  format(4(1x,f14.4))
  write(1,119)((xx(j,i),j=1,128),i=1,128)
  close(1)
call greset
call gclear
call histogram_image(xx,n,h2,lev,gsum)
write(6,*) 'grey_sum9_16=',gsum
call display_histogram(h2,lev)
call greset
call gclear
call norm_histogram(h2,lev,n,h3,dev,area)
write(6,*) 'deviation=',dev
write(6,*) 'Histogram_Area=',area
call display_histogram_norm(h3,lev)
call greset
call gclear
Compute least square error
call least_square_error(a,xx,n,er,abser)
write(6,*) 'Least square error=',er

write(6,*) 'enter cutoff freq. in pixels:'
read(5,*) icut
write(6,*) 'enter order of the filter: '
read(5,*) iord
call blpf_image(xx,n,icut,iord)
call display_image_grey(xx,n,icol,ipol)
call greset
call gclear
call histogram_image(xx,n,h4,lev,gsum)
write(6,*) 'grey_sum9_16=',gsum
call display_histogram(h4,lev)
call greset
call gclear
call norm_histogram(h4,lev,n,h5,dev,area)
write(6,*) 'deviation=',dev
write(6,*) 'Histogram_Area=',area
call display_histogram_norm(h5,lev)
call greset
call gclear

```

Ask user if another filterig technique is required.

```

write(6,110)
format(1x,'filter another? <n>'$)

```



```

1      read(5,111)re
      format(a6)
      if((re.eq.'y').or.(re.eq.'Y'))goto 21
      call compare_rotate_image(xy,m)
Ask user if another reconstruction technique is required.

      write(6,112)
2      format(1x,'reconstruct another? <n>'$)
      read(5,113)an
3      format(a6)
      if((an.eq.'y').or.(an.eq.'Y'))goto 20

Ask user if another display is required.

      write(6,1001)
01     format(1x,'Display another? <n>'$)
      read(5,1002)answer
02     format(a6)
      if((answer.eq.'yes').or.(answer.eq.'y'))goto 5

Terminate uniras.

Write out response to user indicating next stage for producing
a hardcopy (in the case of option for generating a plotfile).

      if(ic.eq.1)then
      write(6,*)'Your plot file is called lps40.plot'
      endif

      if(ic.eq.2)then
      write(6,*)'Your plot file is called t4510.plot'
      endif


      call greset
      call gclear
      call gclose
      stop
      end
      subroutine counter(v,n,necount,nzecount,npocount)
      real v(256,256)
      necount=0
      nzecount=0
      npocount=0
      do i=1,n
      do j=1,n
      if (v(j,i).lt.0.) necount=necount+1
      if (v(j,i).eq.0.) nzecount=nzecount+1
      if (v(j,i).gt.0.) npocount=npocount+1
      enddo
      enddo
      return
      end

```

```

program Fayed Nearest Hilbert.for
real a(256,256),xy(256,256),xx(256,256)
real h(256),h1(256),h2(256),h3(256),h4(256)
real h5(256),b(256,256),yy(256,256),per(256),abser(256,256)
character*6 answer

```

FUNCTION: Generates an image on a DEC digital workstation using data from a specified data file or produces a plot file. Also, reconstruct the object using nearest neighbour techniques and Hilbert transform.

EXTERNAL SUBROUTINES: UNIRAS Library - groute
gopen
gclose

DIPLIB - read_image, timer_start, time_elapsed,
display_image, 30:Rotation_Techniques

Select route for gimage.

```

write(6,10)
format(1x,'View image (1) or generate plotfile (2)?'$)
read(5,*)id
if(id.eq.1)then
call groute('s mx11; e') !for VAX workstations 2000 or 3200
endif

if(id.eq.2)then
write(6,11)
format(1x,'Black and white (1) or colour (2) plotfile?'$)
read(6,*)ic
if(ic.eq.1)then
call groute('s hposta4; e')
endif
if(ic.eq.2)then
call groute('s hqcs1004a; e')
endif
if((ic.ne.1).and.(ic.ne.2))goto 2
endif

if((id.ne.1).and.(id.ne.2))goto 1

call gopen !open UNIRAS window

```

Ask user for image parameters:

1. size of image
2. colour coding with options

continue !for looping back to examine another data file.

Open data file and read contents by calling
subroutine read_image.

```

write(6,*)'enter your choice:'
write(6,*)'1. read_image'
write(6,*)'2. Generate_rectangle'
write(6,*)'3. Generate_circle'
write(6,*)'4. Generate_var_rectangle'
read(5,*) k1
if (k1.eq.1) then
call read_image(a,n)
endif
if (k1.eq.2) then
write(6,*)'enter size of square image'

```

```

read(5,*)n
call rectangle(a,n)
call write_image(a,n)
endif
if (kl.eq.3) then
write(6,*)'enter size of square image'
read(5,*)n
call circle(a,n)
call write_image(a,n)
endif
if (kl.eq.4) then
write(6,*)'enter size of square image'
read(5,*)n
call var_rectangle(a,n)
call write_image(a,n)
endif
continue !for looping back to examine a different colour code.

```

```

write(6,*)'      UNIRAS 6 colour scales'
write(6,*)
write(6,*)'Default colour scale 1..... (1)'
write(6,*)'Default colour scale 2..... (2)'
write(6,*)'Grey scale..... (3)'
write(6,*)'Topographic colour scale..... (4)'
write(6,*)'Seismic colour scale..... (5)'
write(6,*)
write(6,80)
format(1x,'Scale (1-5)?'$)
read(5,*)icol
write(6,90)
format(1x,'Positive (1) or negative (-1)?'$)
read(5,*)ipol

```

Generate image using subroutine display_image.

```

call display_image_grey(a,n,icol,ipol)
      open (unit=1,file='data1.dat',status='new')
01      format(4(1x,f14.4))
      write(1,101)((a(j,i),j=1,128),i=1,128)
      close(1)
0  call greset
   call gclear
   write(6,*)'enter grey_level='
   read(5,*)lev
   call histogram_image(a,n,h,lev,gsum)
   write(6,*)'grey_sum9_16=',gsum
   call display_histogram(h,lev)
   call greset
   call gclear
   call norm_histogram(h,lev,n,h1,dev,area)
   write(6,*)'deviation=',dev
   write(6,*)'Histogram Area=',area
   call display_histogram_norm(h1,lev)
   call greset
   call gclear
   m=nint(1.414213562*n)

   write(6,*) 'Computing reconstruction using nearest_neig.:'
   write(6,*) ' 1. optimum rotation algorithm'
   write(6,*) ' 2. normal rotation algorithm '
   read(5,*) ix
   if ((ix.ne.1) .and. (ix.ne.2)) goto 20
   if (ix.eq.1) then
     write(6,*) '1. Optimum nearest neighbor pixel'
     write(6,*) '2. Optimum four neighbor pixels'
     write(6,*) '3. Optimum eight neighbor pixels '
     read(5,*) nfn
     if (nfn.eq.1) then

```

```

        call timer_start(context)
        call rtr_nearest_neig(a,n,xy,m)
        call time_elapsed(context)
call display_image_grey(xy,m,icol,ipol)
call radon_gauss_noise (xy,m)
        open (unit=1,file='rgnon.181',status='new')
        format(4(1x,f14.4))
        write(1,116)((xy(j,i),j=1,181),i=1,181)
        close(1)
call greset
call gclear
        call display_image_grey(xy,m,icol,ipol)
        call write_image(xy,m)
        call timer_start(context)
        call irtr_nearest_neig(xy,m,xx,n)
        call time_elapsed(context)
        call write_image(xy,m)
    endif
    if (nfn.eq.2) then
        call timer_start(context)
        call rtr_four_neig(a,n,xy,m)
        call time_elapsed(context)
call display_image_grey(xy,m,icol,ipol)
        open (unit=1,file='datar.dat',status='new')
        format(4(1x,f14.4))
        write(1,118)((xy(j,i),j=1,181),i=1,181)
        close(1)
        call write_image(xy,m)
        call timer_start(context)
        call irtr_four_neig(xy,m,xx,n)
        call time_elapsed(context)
        call write_image(xy,m)
    endif
    if (nfn.eq.3) then
        call timer_start(context)
        call rtr_eight_neig(a,n,xy,m)
        call time_elapsed(context)
call display_image_grey(xy,m,icol,ipol)
        call write_image(xy,m)
        call timer_start(context)
        call irtr_eight_neig(xy,m,xx,n)
        call time_elapsed(context)
        call write_image(xy,m)
    endif
elseif (ix.eq.2) then
    write(6,*) '1. Normal nearest neighbor pixel'
    write(6,*) '2. Normal four neighbor pixels'
    write(6,*) '3. Normal eight neighbor pixels'
    read(5,*) nfn
    if (nfn.eq.1) then
        call timer_start(context)
        call rtr_nearest(a,n,xy,m)
        call time_elapsed(context)
call display_image_grey(xy,m,icol,ipol)
        call write_image(xy,m)
        call timer_start(context)
        call irtr_nearest(xy,m,xx,n)
        call time_elapsed(context)
        call write_image(xy,m)
    endif
    if (nfn.eq.2) then
        call timer_start(context)
        call rtr_four(a,n,xy,m)
        call time_elapsed(context)
call display_image_grey(xy,m,icol,ipol)
        open (unit=1,file='datar.dat',status='new')
        format(4(1x,f14.4))
        write(1,118)((xy(j,i),j=1,181),i=1,181)

```

```

        close(1)
        call write_image(xy,m)
        call timer_start(context)
        call irtr_four(xy,m,xx,n)
        call time_elapsed(context)
        call write_image(xy,m)
    endif
    if (nfn.eq.3) then
        call timer_start(context)
        call rtr_eight(a,n,xy,m)
        call time_elapsed(context)
19  call display_image_grey(xy,m,icol,ipol)
        call write_image(xy,m)
        call timer_start(context)
        call irtr_eight(xy,m,xx,n)
        call time_elapsed(context)
        call write_image(xy,m)
    endif
else
endif
call greset
call gclear
call counter(xx,n,necount,nzecount,npocount)
write(6,*)('necount= ',necount)
write(6,*)('zecount= ',nzecount)
write(6,*)('pocount= ',npocount)
call display_image_grey(xx,n,icol,ipol)
    open (unit=1,file='datare.dat',status='new')
    format(4(1x,f14.4))
    write(1,119)((xx(j,i),j=1,128),i=1,128)
    close(1)
call greset
call gclear
call histogram_image(xx,n,h2,lev,gsum)
write(6,*)'grey_sum9_16=',gsum
call display_histogram(h2,lev)
    call greset
    call gclear
    call norm_histogram(h2,lev,n,h3,dev,area)
write(6,*)'deviation=',dev
write(6,*)'Histogram Area=',area
call display_histogram_norm(h3,lev)
    call greset
    call gclear
Compute least square error
    call least_square_error(a,xx,n,er,abser,std)
    open (unit=1,file='abseron.128',status='new')
61    format(4(1x,f14.4))
    write(1,161)((abser(j,i),j=1,128),i=1,128)
    close(1)

write(6,*)'NMSE=',er
write(6,*)'2D standard dev.=',std
Compute projection (Rotation) least square error
call projection_square_error(a,xx,n,per)
call display_signal(per,n)
call mean_standard(per,n,xmean,st1)
write(6,*)'eld=',xmean
write(6,*)'1D standard dev.=',st1
call write_signal(per,n)

1  write(6,*)'enter cutoff freq. in pixels:'
    read(5,*) icut
    write(6,*)'enter order of the filter : '
    read(5,*) iord
    call blpf_image(xx,n,icut,iord)
    call display_image_grey(xx,n,icol,ipol)

```

```

call greset
call gclear
call histogram_image(xx,n,h4,lev,gsum)
write(6,*)'grey_sum9_16=',gsum
call display_histogram(h4,lev)
  call greset
call gclear
  call norm_histogram(h4,lev,n,h5,dev,area)
write(6,*)'deviation=',dev
write(6,*)'Histogram_Area=',area
call display_histogram_norm(h5,lev)
  call greset
call gclear

```

Ask user if another filterig technique is required.

```

10  write(6,110)
    format(1x,'filter another? <n>'$)
    read(5,111)re
11  format(a6)
    if((re.eq.'y').or.(re.eq.'Y'))goto 21
    call compare_rotate_image(xy,m)

```

Ask user if another reconstruction technique is required.

```

2  write(6,112)
    format(1x,'reconstruct another? <n>'$)
    read(5,113)an
3  format(a6)
    if((an.eq.'y').or.(an.eq.'Y'))goto 20

```

Ask user if another display is required.

```

01  write(6,1001)
    format(1x,'Display another? <n>'$)
    read(5,1002)answer
02  format(a6)
    if((answer.eq.'yes').or.(answer.eq.'y'))goto 5

```

Terminate uniras.

Write out response to user indicating next stage for producing a hardcopy (in the case of option for generating a plotfile).

```

if(ic.eq.1)then
write(6,*)'Your plot file is called post.dat'
endif

```

```

if(ic.eq.2)then
write(6,*)'Your plot file is called post.dat'
endif

```

```

call greset
call gclear
call gclose
stop
end

```

```

program Fayed_nearest_Central_slice.for
real a(256,256),xy(256,256),xx(256,256)
real h(256),h1(256),h2(256),h3(256),h4(256)
real h5(256),b(256,256),yy(256,256)
real abser(256,256),per(256)
character*6 answer

```

FUNCTION: Generates an image on a DEC digital workstation using data from a specified data file or produces a plot file. Also, reconstruct the object using nearest neighbour techniques and central slice theorem.

EXTERNAL SUBROUTINES: UNIRAS Library - groute
gopen
gclose

COMMON - read_image, timer_start, time_elapsed,
display_image, 30:Rotation_Techniques

Select route for gimage.

```

write(6,10)
format(1x,'View image (1) or generate plotfile (2)?'$)
read(5,*)id
if(id.eq.1)then
call groute('s mx11; e') !for VAX workstations 2000 or 3200
endif

```

```

if(id.eq.2)then
write(6,11)
format(1x,'Black and white (1) or colour (2) plotfile?'$)
read(6,*)ic
if(ic.eq.1)then
call groute('s hposta4; e')
endif
if(ic.eq.2)then
call groute('s hqcs1004a; e')
endif
if((ic.ne.1).and.(ic.ne.2))goto 2
endif

```

```

if((id.ne.1).and.(id.ne.2))goto 1

```

```

call gopen !open UNIRAS window

```

Ask user for image parameters:

1. size of image
2. colour coding with options

```

continue !for looping back to examine another data file.

```

Open data file and read contents by calling
subroutine read_image.

```

write(6,*)'enter your choice:'
write(6,*)'1. read_image'
write(6,*)'2. Generate_rectangle'
write(6,*)'3. Generate_circle'
write(6,*)'4. Generate_var_rectangle'
write(6,*)'5. Generate_exp_2dimension'
write(6,*)'6. Generate_multi_circle'
write(6,*)'7. Generate_vertical_lines '
read(5,*) k1
if (k1.eq.1) then

```

```

call read_image(a,n)
endif
if (kl.eq.2) then
write(6,*)'enter size of square image'
read(5,*)n
call rectangle(a,n)
call write_image(a,n)
endif
if (kl.eq.3) then
write(6,*)'enter size of square image'
read(5,*)n
call circle(a,n)
call write_image(a,n)
endif
if (kl.eq.4) then
write(6,*)'enter size of square image'
read(5,*)n
call var_rectangle(a,n)
call write_image(a,n)
endif
if (kl.eq.5) then
write(6,*)'enter size of square image'
read(5,*)n
call exp_2d(a,n)
call write_image(a,n)
endif
if (kl.eq.6) then
write(6,*)'enter size of square image'
read(5,*)n
call multi_circle(a,n)
call write_image(a,n)
endif
if (kl.eq.7) then
write(6,*)'enter size of square image'
read(5,*)n
call generate_vertical_lines(a,n)
call write_image(a,n)
endif
continue !for looping back to examine a different colour code.

```

```

write(6,*)'      UNIRAS 6 colour scales'
write(6,*)
write(6,*)'Default colour scale 1..... (1)'
write(6,*)'Default colour scale 2..... (2)'
write(6,*)'Grey scale..... (3)'
write(6,*)'Topographic colour scale..... (4)'
write(6,*)'Seismic colour scale..... (5)'
write(6,*)
write(6,80)
format(1x,'Scale (1-5)?'$)
read(5,*)icol
write(6,90)
format(1x,'Positive (1) or negative (-1)?'$)
read(5,*)ipol

```

Generate image using subroutine display_image.

```

call GSEGCR (1)
call display_image_grey(a,n,icol,ipol)
call GSEGCL (1)
      open (unit=1,file='data1.dat',status='new')
      format(4(1x,f14.4))
      write(1,101)((a(j,i),j=1,128),i=1,128)
      close(1)
call greset
call gclear
write(6,*)'enter grey_level='
read(5,*)lev
call histogram_image(a,n,h,lev,gsum)

```



```

write(6,*)'grey_sum9_16=',gsum
call GSEGCR (2)
call display_histogram(h,lev)
call GSEGCL (2)
  call greset
  call gclear
  call norm_histogram(h,lev,n,h1,dev,area)
write(6,*)'deviation=',dev
write(6,*)'Histogram_Area=',area
call GSEGCR (3)
call display_histogram_norm(h1,lev)
call GSEGCL (3)
  call greset
  call gclear
m=nint(1.414213562*n)

write (6,*) 'Computing reconstruction using nearest_neig.: '
write(6,*) ' 1. optimum rotation algorithm'
write(6,*) ' 2. normal rotation algorithm '
read (5,*) ix
if ((ix.ne.1) .and.(ix.ne.2)) goto 20
if (ix.eq.1) then
  write(6,*) '1. Optimum nearest neighbor pixel'
  write(6,*) '2. Optimum four_ neighbor pixels'
  write(6,*) '3. Optimum eight neighbor pixels '
  write(6,*) '4. Optimum 2 row neighbor pixels '
  write(6,*) '5. Optimum one row neighbor pixel'
  read(5,*) nfn
  if (nfn.eq.1) then
    call timer_start(context)
    call rtr_nearest_neig(a,n,xy,m)
    call time_elapsed(context)
    open (unit=1,file='onnp.181',status='new')
16    format(4(1x,f14.4))
    write(1,116)((xy(j,i),j=1,181),i=1,181)
    close(1)
  call GSEGCR (4)
  call display_image_grey(xy,m,icol,ipol)
  call GSEGCL (4)
  call radon_gauss_noise (xy,m)
    open (unit=1,file='rgnon.181',status='new')
16    format(4(1x,f14.4))
    write(1,116)((xy(j,i),j=1,181),i=1,181)
    close(1)
  call greset
  call gclear
  call GSEGCR (5)
    call display_image_grey(xy,m,icol,ipol)
  call GSEGCL (5)
    call write_image(xy,m)
    call timer_start(context)
    call ift_nearest_neig(xy,m,b,n)
    call time_elapsed(context)
    open (unit=1,file='onnr.128',status='new')
17    format(4(1x,f14.4))
    write(1,117)((xx(j,i),j=1,128),i=1,128)
    close(1)
    call write_image(xy,m)
  endif
  if (nfn.eq.2) then
    call timer_start(context)
    call rtr_four_neig(a,n,xy,m)
    call time_elapsed(context)
    open (unit=1,file='ofnp.181',status='new')
15    format(4(1x,f14.4))
    write(1,115)((xy(j,i),j=1,181),i=1,181)
    close(1)
  call GSEGCR (6)

```

```

call display_image_grey(xy,m,icol,ipol) .
call GSEGCL (6)
    call write_image(xy,m)
    call timer_start(context)
    call ift_four_neig(xy,m,b,n)
    call time_elapsed(context)
    open (unit=1,file='ofnr.128',status='new')
    format(4(1x,f14.4))
    write(1,118)((xx(j,i),j=1,128),i=1,128)
    close(1)
    call write_image(xy,m)
endif
if (nfn.eq.3) then
    call timer_start(context)
    call rtr_eight_neig(a,n,xy,m)
    call time_elapsed(context)
call GSEGCR (7)
call display_image_grey(xy,m,icol,ipol)
call GSEGCL (7)
    call write_image(xy,m)
    call timer_start(context)
    call ift_eight_neig(xy,m,b,n)
    call time_elapsed(context)
    call write_image(xy,m)
endif
if (nfn.eq.4) then
    call timer_start(context)
    call rtr_2rp_neig(a,n,xy,m)
    call time_elapsed(context)
call GSEGCR (8)
call display_image_grey(xy,m,icol,ipol)
call GSEGCL (8)
    call write_image(xy,m)
    call timer_start(context)
    call ift_2rp_neig(xy,m,b,n)
    call time_elapsed(context)
    call write_image(xy,m)
endif
if (nfn.eq.5) then
    call timer_start(context)
    call rtr_1rp_neig(a,n,xy,m)
    call time_elapsed(context)
call GSEGCR (9)
call display_image_grey(xy,m,icol,ipol)
call GSEGCL (9)
    call write_image(xy,m)
    call timer_start(context)
    call ift_1rp_neig(xy,m,b,n)
    call time_elapsed(context)
    call write_image(xy,m)
endif
elseif (ix.eq.2) then
    write(6,*) '1. Normal nearest neighbor pixel'
    write(6,*) '2. Normal four_ neighbor pixels'
    write(6,*) '3. Normal eight neighbor pixels '
    read(5,*) nfn
    if (nfn.eq.1) then
        call timer_start(context)
        call rtr_nearest(a,n,xy,m)
        call time_elapsed(context)
        open (unit=1,file='nnnp.181',status='new')
        format(4(1x,f14.4))
        write(1,123)((xy(j,i),j=1,181),i=1,181)
        close(1)
    call GSEGCR (10)
    call display_image_grey(xy,m,icol,ipol)
    call GSEGCL (10)
        call write_image(xy,m)

```

```

        call timer_start(context)
        call ift_nearest(xy,m,b,n)
        call time_elapsed(context)
call display_image_grey(b,n,icol,ipol)
        open (unit=1,file='nnnr.128',status='new')
14         format(4(1x,f14.4))
        write(1,114)((xx(j,i),j=1,128),i=1,128)
        close(1)
        call write_image(xy,m)
    endif
    if (nfn.eq.2) then
        call timer_start(context)
        call rtr_four(a,n,xy,m)
        call time_elapsed(context)
22         open (unit=1,file='nfnr.181',status='new')
        format(4(1x,f14.4))
        write(1,122)((xy(j,i),j=1,181),i=1,181)
        close(1)
    call GSEGCR (11)
    call display_image_grey(xy,m,icol,ipol)
    call GSEGCL (11)
        call write_image(xy,m)
        call timer_start(context)
        call ift_four(xy,m,b,n)
        call time_elapsed(context)
21         open (unit=1,file='nfnr.128',status='new')
        format(4(1x,f14.4))
        write(1,121)((xx(j,i),j=1,128),i=1,128)
        close(1)
        call write_image(xy,m)
    endif
    if (nfn.eq.3) then
        call timer_start(context)
        call rtr_eight(a,n,xy,m)
        call time_elapsed(context)
    call GSEGCR (12)
    call display_image_grey(xy,m,icol,ipol)
    call GSEGCL (12)
        call write_image(xy,m)
        call timer_start(context)
        call ift_eight(xy,m,b,n)
        call time_elapsed(context)
        call write_image(xy,m)
    endif
else
endif
call greset
call gclear
call counter(b,n,necount,nzcount,npocount)
write(6,*)( 'necount= ',necount)
write(6,*)( 'nzcount= ',nzcount)
write(6,*)( 'pocount= ',npocount)
call GSEGCR (13)
call display_image_grey(b,n,icol,ipol)
call GSEGCL (13)
19         open (unit=1,file='datare.dat',status='new')
        format(4(1x,f14.4))
        write(1,119)((b(j,i),j=1,128),i=1,128)
        close(1)
    call greset
    call gclear
    call histogram_image(b,n,h2,lev,gsum)
    write(6,*) 'grey_sum9_16=',gsum
    call GSEGCR (14)
    call display_histogram(h2,lev)
    call GSEGCL (14)
        call greset
        call gclear

```

```

call norm_histogram(h2,lev,n,h3,dev,area)
write(6,*)'deviation=',dev
write(6,*)'Histogram_Area=',area
call GSEGCR (15)
call display_histogram_norm(h3,lev)
call GSEGCL (15)
call greset
call gclear
Compute least square error
call least_square_error(a,b,n,er,abser,std)
open (unit=1,file='abseron.128',status='new')
format(4(1x,f14.4))
write(1,161)((abser(j,i),j=1,128),i=1,128)
close(1)

write(6,*)'NMSE=',er
write(6,*)'2D standard dev.=',std
Compute projection (Rotation) least square error
call projection_square_error(a,b,n,per)
call display_signal(per,n)
call mean_standard(per,n,xmean,st1)
write(6,*)'eld=',xmean
write(6,*)'1D standard dev.=',st1
call write_signal(per,n)

write(6,*)'enter cutoff freq. in pixels:'
read(5,*) icut
write(6,*)'enter order of the filter :'
read(5,*) iord
call blpf_image(b,n,icut,iord)
call GSEGCR (16)
call display_image_grey(b,n,icol,ipol)
call GSEGCL (16)
call greset
call gclear
call histogram_image(b,n,h4,lev,gsum)
write(6,*)'grey_sum9_16=',gsum
call GSEGCR (17)
call display_histogram(h4,lev)
call GSEGCL (17)
call greset
call gclear
call norm_histogram(h4,lev,n,h5,dev,area)
write(6,*)'deviation=',dev
write(6,*)'Histogram_Area=',area
call GSEGCR (18)
call display_histogram_norm(h5,lev)
call GSEGCL (18)
call greset
call gclear

Ask user if another filterig technique is required.

write(6,110)
format(1x,'filter another? <n>'$)
read(5,111)re
format(a6)
if((re.eq.'y').or.(re.eq.'Y'))goto 21
call compare_rotate_image(xy,m)

Ask user if another reconstruction technique is required.

write(6,112)
format(1x,'reconstruct another? <n>'$)
read(5,113)an
format(a6)
if((an.eq.'y').or.(an.eq.'Y'))goto 20

```

Ask user if another display is required.

```
01      write(6,1001)
      format(1x,'Display another? <n>'$)
      read(5,1002)answer
02      format(a6)
      if((answer.eq.'yes').or.(answer.eq.'y'))goto 5
```

Terminate uniras.

Write out response to user indicating next stage for producing a hardcopy (in the case of option for generating a plotfile).

```
      if(ic.eq.1)then
      write(6,*)'Your plot file is called post.dat'
      endif
```

```
      if(ic.eq.2)then
      write(6,*)'Your plot file is called post.dat'
      endif
```

```
      call greset
      call gclear
      call gclose
      stop
      end
```

```

program Fayed nearest convolution.for
real a(256,256),xy(256,256),xx(256,256)
real h(256),h1(256),h2(256),h3(256),h4(256)
real h5(256),b(256,256),yy(256,256)
real abser(256,256),per(256)
character*6 answer

```

FUNCTION: Generates an image on a DEC digital workstation using data from a specified data file or produces a plot file. Also, reconstruct the object using nearest neighbour techniques and back projection and deconvolution.

EXTERNAL SUBROUTINES: UNIRAS Library - groute
gopen
gclose

DIPLIB - read_image, timer_start, time_elapsed,
display_image, 30:Rotation_Techniques

Select route for gimage.

```

write(6,10)
format(1x,'View image (1) or generate plotfile (2)?'$)
read(5,*)id
if(id.eq.1)then
call groute('s mx11; e') !for VAX workstations 2000 or 3200
endif

```

```

if(id.eq.2)then
write(6,11)
format(1x,'Black and white (1) or colour (2) plotfile?'$)
read(6,*)ic
if(ic.eq.1)then
call groute('s hposta4; e')
endif
if(ic.eq.2)then
call groute('s hqcs1004a; e')
endif
if((ic.ne.1).and.(ic.ne.2))goto 2
endif

```

```

if((id.ne.1).and.(id.ne.2))goto 1

```

```

call gopen !open UNIRAS window

```

Ask user for image parameters:

1. size of image
2. colour coding with options

```

continue !for looping back to examine another data file.

```

Open data file and read contents by calling
subroutine read_image.

```

write(6,*)'enter your choice:'
write(6,*)'1. read_image'
write(6,*)'2. Generate_rectangle'
write(6,*)'3. Generate_circle'
write(6,*)'4. Generate_var_rectangle'
write(6,*)'5. Generate_exp_2dimension'
write(6,*)'6. Generate_multi_circle'
read(5,*) k1
if (k1.eq.1) then
call read_image(a,n)

```

```

endif
if (kl.eq.2) then
write(6,*)'enter size of square image'
read(5,*)n
call rectangle(a,n)
call write_image(a,n)
endif
if (kl.eq.3) then
write(6,*)'enter size of square image'
read(5,*)n
call circle(a,n)
call write_image(a,n)
endif
if (kl.eq.4) then
write(6,*)'enter size of square image'
read(5,*)n
call var_rectangle(a,n)
call write_image(a,n)
endif
if (kl.eq.5) then
write(6,*)'enter size of square image'
read(5,*)n
call exp_2d(a,n)
call write_image(a,n)
endif
if (kl.eq.6) then
write(6,*)'enter size of square image'
read(5,*)n
call multi_circle(a,n)
call write_image(a,n)
endif
continue !for looping back to examine a different colour code.

write(6,*)'      UNIRAS 6 colour scales'
write(6,*)
write(6,*)'Default colour scale 1..... (1)'
write(6,*)'Default colour scale 2..... (2)'
write(6,*)'Grey scale..... (3)'
write(6,*)'Topographic colour scale..... (4)'
write(6,*)'Seismic colour scale..... (5)'
write(6,*)
write(6,80)
format(1x,'Scale (1-5)?'$)
read(5,*)icol
write(6,90)
format(1x,'Positive (1) or negative (-1)?'$)
read(5,*)ipol

c Generate image using subroutine display_image.
call GSEGCR (1)
call display_image_grey(a,n,icol,ipol)
call GSEGCL (1)
      open (unit=1,file='data1.dat',status='new')
c101      format(4(1x,f14.4))
c      write(1,101)((a(j,i),j=1,128),i=1,128)
c      close(1)
100 call greset
call gclear
      write(6,*)'enter grey_level='
c      read(5,*)lev
c      call histogram_image(a,n,h,lev,gsum)
c      write(6,*)'grey_sum9_16=',gsum
c      call GSEGCR (2)
c      call display_histogram(h,lev)
c      call GSEGCL (2)
c      call greset
c      call gclear
c      call norm_histogram(h,lev,n,h1,dev,area)

```

```

c      write(6,*)'deviation=',dev
c      write(6,*)'Histogram_Area=',area
c      call GSEGCR (3)
c      call display_histogram_norm(h1,lev)
c      call GSEGCL (3)
c      call greset
c      call gclear
c      m=nint(1.414213562*n)
20  write (6,*) 'Computing reconstruction using nearest_neig.: '
c      write(6,*) ' 1. optimum rotation algorithm'
c      write(6,*) ' 2. normal rotation algorithm '
c      read (5,*) ix
c      if ((ix.ne.1) .and.(ix.ne.2)) goto 20
c      if (ix.eq.1) then
c          write(6,*) '1. Optimum nearest neighbor pixel'
c          write(6,*) '2. Optimum four_neighbor pixels'
c          write(6,*) '3. Optimum eight_neighbor pixels '
c          write(6,*) '4. Optimum 2 row neighbor pixels '
c          write(6,*) '5. Optimum one row neighbor pixel'
c          read(5,*) nfn
c          if (nfn.eq.1) then
c              call timer_start(context)
c              call rtr_nearest_neig(a,n,xy,m)
c              call time_elapsed(context)
c              open (unit=1,file='onnp.181',status='new')
c              format(4(1x,f14.4))
c              write(1,116)((xy(j,i),j=1,181),i=1,181)
c              close(1)
c              call GSEGCR (4)
c              call display_image_grey(xy,m,icol,ipol)
c              call GSEGCL (4)
c              call radon_gauss_noise (xy,m)
c              open (unit=1,file='rgnon.181',status='new')
c              format(4(1x,f14.4))
c              write(1,116)((xy(j,i),j=1,181),i=1,181)
c              close(1)
c              call greset
c              call gclear
c              call GSEGCR (5)
c              call display_image_grey(xy,m,icol,ipol)
c              call GSEGCL (5)
c              call write_image(xy,m)
c              call timer_start(context)
c              call irtr_nearest_neig(xy,m,xx,n)
c              call time_elapsed(context)
c              open (unit=1,file='onnr.128',status='new')
c              format(4(1x,f14.4))
c              write(1,117)((xx(j,i),j=1,128),i=1,128)
c              close(1)
c              call write_image(xy,m)
c          endif
c          if (nfn.eq.2) then
c              call timer_start(context)
c              call rtr_four_neig(a,n,xy,m)
c              call time_elapsed(context)
c              open (unit=1,file='ofnp.181',status='new')
c              format(4(1x,f14.4))
c              write(1,115)((xy(j,i),j=1,181),i=1,181)
c              close(1)
c              call GSEGCR (6)
c              call display_image_grey(xy,m,icol,ipol)
c              call GSEGCL (6)
c              call write_image(xy,m)
c              call timer_start(context)
c              call irtr_four_neig(xy,m,xx,n)
c              call time_elapsed(context)
c              open (unit=1,file='ofnr.128',status='new')

```



```

c118      format(4(1x,f14.4))
c          write(1,118)((xx(j,i),j=1,128),i=1,128)
c          close(1)
c          call write_image(xy,m)
c      endif
c      if (nfn.eq.3) then
c          call timer_start(context)
c          call rtr_eight_neig(a,n,xy,m)
c          call time_elapsed(context)
c      call GSEGCR (7)
c      call display_image_grey(xy,m,icol,ipol)
c      call GSEGCL (7)
c          call write_image(xy,m)
c          call timer_start(context)
c          call irtr_eight_neig(xy,m,xx,n)
c          call time_elapsed(context)
c          call write_image(xy,m)
c      endif
c      if (nfn.eq.4) then
c          call timer_start(context)
c          call rtr_2rp_neig(a,n,xy,m)
c          call time_elapsed(context)
c      call GSEGCR (8)
c      call display_image_grey(xy,m,icol,ipol)
c      call GSEGCL (8)
c          call write_image(xy,m)
c          call timer_start(context)
c          call irtr_2rp_neig(xy,m,xx,n)
c          call time_elapsed(context)
c          call write_image(xy,m)
c      endif
c      if (nfn.eq.5) then
c          call timer_start(context)
c          call rtr_lrp_neig(a,n,xy,m)
c          call time_elapsed(context)
c      call GSEGCR (9)
c      call display_image_grey(xy,m,icol,ipol)
c      call GSEGCL (9)
c          call write_image(xy,m)
c          call timer_start(context)
c          call irtr_lrp_neig(xy,m,xx,n)
c          call time_elapsed(context)
c          call write_image(xy,m)
c      endif
c      elseif (ix.eq.2) then
c          write(6,*) '1. Normal nearest neighbor pixel'
c          write(6,*) '2. Normal four neighbor pixels'
c          write(6,*) '3. Normal eight neighbor pixels'
c          read(5,*) nfn
c          if (nfn.eq.1) then
c              call timer_start(context)
c              call rtr_nearest(a,n,xy,m)
c              call time_elapsed(context)
c              open (unit=1,file='nnnp.181',status='new')
c123      format(4(1x,f14.4))
c              write(1,123)((xy(j,i),j=1,181),i=1,181)
c              close(1)
c          call GSEGCR (10)
c          call display_image_grey(xy,m,icol,ipol)
c          call GSEGCL (10)
c              call write_image(xy,m)
c              call timer_start(context)
c              call irtr_nearest(xy,m,xx,n)
c              call time_elapsed(context)
c              open (unit=1,file='nnnr.128',status='new')
c114      format(4(1x,f14.4))
c              write(1,114)((xx(j,i),j=1,128),i=1,128)
c              close(1)

```

```

        call write_image(xy,m)
    endif
    if (nfn.eq.2) then
        call timer_start(context)
        call rtr_four(a,n,xy,m)
        call time_elapsed(context)
        open (unit=1,file='nfnp.181',status='new')
c122        format(4(1x,f14.4))
        write(1,122)((xy(j,i),j=1,181),i=1,181)
        close(1)
    call GSEGCR (11)
    call display_image_grey(xy,m,icol,ipol)
    call GSEGCL (11)
        call write_image(xy,m)
        call timer_start(context)
        call irtr_four(xy,m,xx,n)
        call time_elapsed(context)
        open (unit=1,file='nfmr.128',status='new')
c121        format(4(1x,f14.4))
        write(1,121)((xx(j,i),j=1,128),i=1,128)
        close(1)
        call write_image(xy,m)
    endif
    if (nfn.eq.3) then
        call timer_start(context)
        call rtr_eight(a,n,xy,m)
        call time_elapsed(context)
    call GSEGCR (12)
    call display_image_grey(xy,m,icol,ipol)
    call GSEGCL (12)
        call write_image(xy,m)
        call timer_start(context)
        call irtr_eight(xy,m,xx,n)
        call time_elapsed(context)
        call write_image(xy,m)
    endif
else
endif
call greset
call gclear
call GSEGCR (13)
call display_image_grey(xx,n,icol,ipol)
call GSEGCL (13)
        open (unit=1,file='datarc.dat',status='new')
c119        format(4(1x,f14.4))
        write(1,119)((xx(j,i),j=1,128),i=1,128)
        close(1)
call greset
call gclear
call histogram_image(xx,n,h2,lev,gsum)
write(6,*)'grey_sum9_16=',gsum
call GSEGCR (14)
call display_histogram(h2,lev)
call GSEGCL (14)
    call greset
    call gclear
        call norm_histogram(h2,lev,n,h3,dev,area)
write(6,*)'deviation=',dev
write(6,*)'Histogram_Area=',area
call GSEGCR (15)
call display_histogram_norm(h3,lev)
call GSEGCL (15)
    call greset
    call gclear
c Compute least square error
    call least_square_error(a,xx,n,er,abser,std)
        open (unit=1,file='abseron.128',status='new')
c161        format(4(1x,f14.4))

```

```

c         write(1,161)((abser(j,i),j=1,128),i=1,128)
c         close(1)
c
c         write(6,*)'NMSE=',er
c         write(6,*)'2D standard dev.=',std
c Compute projection (Rotation) least square error
c         call projection_square_error(a,xx,n,per)
c         call display_signal(per,n)
c         call mean_standard(per,n,xmean,st1)
c         write(6,*)'eld=',xmean
c         write(6,*)'1D standard dev.=',st1
c         call write_signal(per,n)
c
c
c
c211 write(6,*)'enter cutoff freq. in pixels:'
c         read(5,*) icut
c         write(6,*)'enter order of the filter :'
c         read(5,*) iord
c         call blpf_image(xx,n,icut,iord)
c         call GSEGCR (16)
c             call display_image_grey(xx,n,icol,ipol)
c         call GSEGCL (16)
c         call greset
c         call gclear
c         call histogram_image(xx,n,h4,lev,gsum)
c         write(6,*)'grey_sum9_16=',gsum
c         call GSEGCR (17)
c         call display_histogram(h4,lev)
c         call GSEGCL (17)
c         call greset
c         call gclear
c         call norm_histogram(h4,lev,n,h5,dev,area)
c         write(6,*)'deviation=',dev
c         write(6,*)'Histogram_Area=',area
c         call GSEGCR (18)
c         call display_histogram_norm(h5,lev)
c         call GSEGCL (18)
c         call greset
c         call gclear
c Ask user if another filterig technique is required.
c
c         write(6,110)
c110     format(1x,'filter another? <n>'$)
c         read(5,111)re
c111     format(a6)
c         if((re.eq.'y').or.(re.eq.'Y'))goto 21
c         call compare_rotate_image(xy,m)
c Ask user if another reconstruction technique is required.
c
c         write(6,112)
c112     format(1x,'reconstruct another? <n>'$)
c         read(5,113)an
c113     format(a6)
c         if((an.eq.'y').or.(an.eq.'Y'))goto 20
c
c Ask user if another display is required.
c
c         write(6,1001)
c1001     format(1x,'Display another? <n>'$)
c         read(5,1002)answer
c1002     format(a6)
c         if((answer.eq.'yes').or.(answer.eq.'y'))goto 5
c
c Terminate uniras.
c

```

c Write out response to user indicating next stage for producing
c a hardcopy (in the case of option for generating a plotfile).
c

```
    if(ic.eq.1)then  
      write(6,*)'Your plot file is called lps40.plot'  
    endif
```

```
    if(ic.eq.2)then  
      write(6,*)'Your plot file is called t4510.plot'  
    endif
```

```
    call greset  
    call gclear  
    call gclose  
    stop  
    end  
    subroutine counter(v,n,necount,nzecount,npocount)  
    real v(256,256)  
    necount=0  
    nzecount=0  
    npocount=0  
    do i=1,n  
      do j=1,n  
        if (v(j,i).lt.0.) necount=necount+1  
        if (v(j,i).eq.0.) nzecount=nzecount+1  
        if (v(j,i).gt.0.) npocount=npocount+1  
      enddo  
    enddo  
    return  
    end
```

```

subroutine rtr_nearest(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)

```

```

*****

```

```

Function: This subroutine computes the Radon Transform of
         n X n image.

```

```

External subroutines: Rotate_nearest_neig
                     Project

```

```

Internal subroutines: None

```

```

Parameters: x - n X n input array

```

```

         y - m X m output array

```

```

*****

```

```

Send message to user 'Computing Radon Transform...'

```

```

write(6,1)
format(1x,'Computing Radon Transform...'$)

```

```

initialize array y

```

```

do j=1,m

```

```

do i=1,m

```

```

y(j,i)=0.

```

```

enddo

```

```

enddo

```

```

Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.

```

```

do i=1,m

```

```

theta=float(i-1)

```

```

call rotate_nearest(x,n,w,m,theta)

```

```

call project(w,z,m)

```

```

Stack projections together to form image of "Radon space"
data.

```

```

do j=1,m

```

```

y(j,i)=z(j)

```

```

enddo

```

```

enddo

```

```

return

```

```

end

```

```

subroutine rtr_nearest_neig(x,n,y,m)
  real x(256,256),y(256,256),xy(256,256)
  real w(256,256),z(256)
*****
Function: This subroutine computes the Radon Transform of
         n X n image.
External subroutines: Rotate_nearest_neig
                     Project
Internal subroutines: None
Parameters: x - n X n input array
            y - m X m output array
*****
Send message to user 'Computing Radon Transform...'

      write(6,1)
      format(1x,'Computing Radon Transform...'$)
c initialize array y
      do j=1,m
      do i=1,m
      y(j,i)=0.
      enddo
      enddo
c Compute angle of rotation in steps of 1 degree and rotate
c the object, take projection of rotated object.
      do i=1,m
      theta=float(i-1)
      call rotate_nearest_neig(x,n,w,m,theta)
      call project(w,z,m)
c Stack projections together to form image of "Radon space"
c data.
      do j=1,m
      y(j,i)=z(j)
      enddo
      enddo
      return
      end

```

```

      subroutine rtr_lrp_neig(x,n,y,m)
      real x(256,256),y(256,256),xy(256,256)
      real w(256,256),z(256)
      *****
Function: This subroutine computes the Radon Transform of
          n X n image.
External subroutines: Rotate_nearest_neig
                      Project
Internal subroutines: None
Parameters: x - n X n input array
            y - m X m output array
      *****
Send message to user 'Computing Radon Transform...'

      write(6,1)
      format(1x,'Computing Radon Transform...'$)
      initialize array y
      do j=1,m
      do i=1,m
      y(j,i)=0.
      enddo
      enddo
      Compute angle of rotation in steps of 1 degree and rotate
      the object, take projection of rotated object.
      do i=1,m
      theta=float(i-1)
      call rotate_lrp_neig(x,n,w,m,theta)
      call project(w,z,m)
      Stack projections together to form image of "Radon space"
      data.
      do j=1,m
      y(j,i)=z(j)
      enddo
      enddo
      return
      end

```

```

      subroutine rtr_2rp_neig(x,n,y,m)
      real x(256,256),y(256,256),xy(256,256)
      real w(256,256),z(256)
      *****
Function: This subroutine computes the Radon Transform of
      n X n image.
External subroutines: Rotate_nearest_neig
                     Project
Internal subroutines: None
Parameters: x - n X n input array
            y - m X m output array
      *****
Send message to user 'Computing Radon Transform...'

      write(6,1)
      format(1x,'Computing Radon Transform...'$)
      initialize array y
      do j=1,m
      do i=1,m
      y(j,i)=0.
      enddo
      enddo
      Compute angle of rotation in steps of 1 degree and rotate
      the object, take projection of rotated object.
      do i=1,m
      theta=float(i-1)
      call rotate_2rp_neig(x,n,w,m,theta)
      call project(w,z,m)
      Stack projections together to form image of "Radon space"
      data.
      do j=1,m
      y(j,i)=z(j)
      enddo
      enddo
      return
      end

```



```

subroutine rtr_four(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
*****
Function: This subroutine computes the Radon Transform of
         n X n image.
External subroutines: Rotate_nearest_neig
                     Project
Internal subroutines: None
Parameters: x - n X n input array
           y - m X m output array
*****
Send message to user 'Computing Radon Transform...'

        write(6,1)
        format(1x,'Computing Radon Transform...'$)
initialize array y
do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo
Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.
do i=1,m
theta=float(i-1)
call rotate_four(x,n,w,m,theta)
call project(w,z,m)
Stack projections together to form image of "Radon space"
data.
do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```

```

subroutine rtr_four_neig(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)

```

Function: This subroutine computes the Radon Transform of
n X n image.

External subroutines: Rotate_four_neig
Project

Internal subroutines: None

Parameters: x - n X n input array

y - m X m output array

Send message to user 'Computing Radon Transform...'

```

write(6,1)

```

```

format(1x,'Computing Radon Transform...'$)

```

initialize array y

```

do j=1,m

```

```

do i=1,m

```

```

y(j,i)=0.

```

```

enddo

```

```

enddo

```

Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.

```

do i=1,m

```

```

theta=float(i-1)

```

```

call rotate_four_neig(x,n,w,m,theta)

```

```

call project(w,z,m)

```

Stack projections together to form image of "Radon space"
data.

```

do j=1,m

```

```

y(j,i)=z(j)

```

```

enddo

```

```

enddo

```

```

return

```

```

end

```

```

subroutine rtr_eight(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)

```

```

*****

```

```

Function: This subroutine computes the Radon Transform of
n X n image.

```

```

External subroutines: Rotate_nearest_neig
Project

```

```

Internal subroutines: None

```

```

Parameters: x - n X n input array
y - m X m output array

```

```

*****

```

```

Send message to user 'Computing Radon Transform...'

```

```

write(6,1)
format(1x,'Computing Radon Transform...'$)

```

```

Initialize array y

```

```

do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo

```

```

Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.

```

```

do i=1,m
theta=float(i-1)
call rotate_eight(x,n,w,m,theta)
call project(w,z,m)

```

```

Stack projections together to form image of "Radon space"
data.

```

```

do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```

```

subroutine rtr_eight_neig(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)

```

Function: This subroutine computes the Radon Transform of
n X n image.

External subroutines: Rotate_eight_neig
Project

Internal subroutines: None

Parameters: x - n X n input array
y - m X m output array

Send message to user 'Computing Radon Transform...'

```

write(6,1)
format(1x,'Computing Radon Transform...'$)

```

initialize array y

```

do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo

```

Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.

```

do i=1,m
theta=float(i-1)
call rotate_eight_neig(x,n,w,m,theta)
call project(w,z,m)

```

Stack projections together to form image of "Radon space"
data.

```

do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```

subroutine rotate nearest(a,n,b,m,theta)

real a(256,256),b(256,256),w(256,256)

FUNCTION : This subroutine rotates an n X n array a
through an angle theta (given in degree)
anticlockwise placing the result at the
center of an array mxm b

PARAMETERS: a _ Input image of n X n size
b _ Rotated image of m X m size

EXTERNAL SUBROUTINE :None

INTERNAL SUBROUTINE :None

convert theta into radians

pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)

initial b

do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) = 0.
enddo
enddo

compute new coordinates (xd,yd) of rotate array

nn = n/2 + 1
mm = m/2 + 1

start process

do j = 1,n
do i = 1,n

generate 4-quadrant cartesian plan

y = float(j-nn)
x = float(i-nn)
xd = x * costh - y * sinth
yd = x * sinth + y * costh

compute both real and integer coordinates of array b
associated with the above transformation for later use

kx = mm+ nint(xd)
ky = mm+ nint(yd)

assign value of a at (j,i) to the nearest_point cell
of array b in wich the transformed coordinates
lie.

b(ky,kx) = a(j,i)

enddo
enddo
return
end

```

subroutine rotate nearest neig (a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
integer xu,yu,xd,yd,dul,eul,du2,eu2,vu(256),hu(256),ku(256)
integer gu(256)
Function: This subroutine rotates an nXn array a through an angle theta
          (given in degrees ) placing the result at the center of an mXm
          array b.
Convert theta into radiants.
pi=4.*atan(1.)
theta=theta*(pi/180.)
c initialize b
do i=1,m
do j=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo
c Compute centers of array a and b
nn=n/2+1
mm=m/2+1
c Compute the integer of horizontal and vertical displacement along the x and
y-axis which corresponds to n-pixel movement along the u-axis.
xu=nint(nn*cos(theta))
yu=nint(nn*sin(theta))
if (xu.lt.0) then
ml=-1
else
ml=1
endif
if (yu.lt.0) then
nl=1
else
nl=-1
endif
c Compute the rotation parameters
dul=2*abs(yu)
du2=2*(abs(yu)-nn)
eul=2*abs(xu)
eu2=2*(abs(xu)-nn)
vu(1)=2*abs(yu)-nn
hu(1)=2*abs(xu)-nn
ku(0)=0
gu(0)=0
do i=1,nn
if (hu(i).ge.0) then
hu(i+1)=hu(i)+eu2
gu(i)=gu(i-1)+1
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
else
hu(i+1)=hu(i)+eul
gu(i)=gu(i-1)
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
endif
endif
enddo
c Compute new coordinates (xd,yd) of rotated array and assign the value of a at
c (i,j) to the appropriate cell of b which encloses these coordinates.

```

```

Generate 4-quadrant cartesian plane.
    do i=1,nn
        ku(-i)=-ku(i)
        gu(-i)=-gu(i)
    enddo
Start process
    do j=1,n
        do i=1,n
Employ coordinate transformation for rotation using the integer parameters.
        yd=m1*gu(i-nn)-n1*ku(j-nn)
        xd=m1*gu(j-nn)+n1*ku(i-nn)
Compute the integer coordinates of array b associated with the above
transformation.
        kx=mm+xd
        ky=mm+yd
Assign value of a at (i,j) to the nearest cell of array b in which the
transformed coordinates lie.
        b(ky+1,kx-1)=a(i,j)
        b(ky,kx-1)=a(i,j)
        b(ky-1,kx-1)=a(i,j)
        b(ky+1,kx)=a(i,j)
        b(ky,kx)=a(i,j)
        b(ky-1,kx)=a(i,j)
        b(ky+1,kx+1)=a(i,j)
        b(ky,kx+1)=a(i,j)
        b(ky-1,kx+1)=a(i,j)

        enddo
        enddo
    return
end

```

```

        subroutine rotate_lrp_neig (a,n,b,m,theta)
        real a(256,256),b(256,256),w(256,256)
        integer xu,yu,xd,yd,dul,eul,du2,eu2,vu(256),hu(256),ku(256)
        integer gu(256)
Function: This subroutine rotates an nXn array a through an angle theta
        (given in degrees ) placing the result at the center of an mXm
        array b.
Convert theta into radiants.
        pi=4.*atan(1.)
        theta=theta*(pi/180.)
c initialize b
        do i=1,m
        do j=1,m
        b(j,i)=0.
        w(j,i)=0.
        enddo
        enddo
c Compute centers of array a and b
        nn=n/2+1
        mm=m/2+1
c Compute the integer of horizontal and vertical displacement along the x and
y-axis which corresponds to n-pixel monement along the u-axis.
        xu=nint(nn*cos(theta))
        yu=nint(nn*sin(theta))
        if (xu.lt.0) then
        m1=-1
        else
        m1=1
        endif
        if (yu.lt.0) then
        n1=1
        else
        n1=-1
        endif
c Compute the rotation parameters
        dul=2*abs(yu)
        du2=2*(abs(yu)-nn)
        eul=2*abs(xu)
        eu2=2*(abs(xu)-nn)
        vu(1)=2*abs(yu)-nn
        hu(1)=2*abs(xu)-nn
        ku(0)=0
        gu(0)=0
        do i=1,nn
        if (hu(i).ge.0) then
        hu(i+1)=hu(i)+eu2
        gu(i)=gu(i-1)+1
            if (vu(i).ge.0) then
            vu(i+1)=vu(i)+du2
            ku(i)=ku(i-1)+1
            else
            vu(i+1)=vu(i)+dul
            ku(i)=ku(i-1)
            endif
        else
        hu(i+1)=hu(i)+eul
        gu(i)=gu(i-1)
            if (vu(i).ge.0) then
            vu(i+1)=vu(i)+du2
            ku(i)=ku(i-1)+1
            else
            vu(i+1)=vu(i)+dul
            ku(i)=ku(i-1)
            endif
        endif
        enddo
c Compute new coordinates (xd,yd) of rotated array and assign the value of a at
c (i,j) to the appropriate cell of b which encloses these coordinates.

```



```

Generate 4-quadrant cartesian plane.
    do i=1,nn
        ku(-i)=-ku(i)
        gu(-i)=-gu(i)
    enddo

Start process
    do j=1,n
        do i=1,n
Employ coordinate transformation for rotation using the integer parameters.
            yd=m1*gu(i-nn)-n1*ku(j-nn)
            xd=m1*gu(j-nn)+n1*ku(i-nn)
Compute the integer coordinates of array b associated with the above
transformation.
            kx=mm+xd
            ky=mm+yd
Assign value of a at (i,j) to the nearest cell of array b in which the
transformed coordinates lie.
            b(ky,kx-1)=a(i,j)
            b(ky+1,kx)=a(i,j)
            b(ky,kx)=a(i,j)
            b(ky-1,kx)=a(i,j)
            b(ky,kx+1)=a(i,j)

        enddo
    enddo
return
end

```

```

subroutine rotate 2rp neig (a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
integer xu,yu,xd,yd,dul,eul,du2,eu2,vu(256),hu(256),ku(256)
integer gu(256)
c Function: This subroutine rotates an nXn array a through an angle theta
c (given in degrees ) placing the result at the center of an mXm
c array b.
c Convert theta into radiants.
pi=4.*atan(1.)
theta=theta*(pi/180.)
c initialize b
do i=1,m
do j=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo
c Compute centers of array a and b
nn=n/2+1
mm=m/2+1
c Compute the integer of horizontal and vertical displacement along the x and
c y-axis which corresponds to n-pixel monement along the u-axis.
xu=nint(nn*cos(theta))
yu=nint(nn*sin(theta))
if (xu.lt.0) then
ml=-1
else
ml=1
endif
if (yu.lt.0) then
nl=1
else
nl=-1
endif
c Compute the rotation parameters
dul=2*abs(yu)
du2=2*(abs(yu)-nn)
eul=2*abs(xu)
eu2=2*(abs(xu)-nn)
vu(1)=2*abs(yu)-nn
hu(1)=2*abs(xu)-nn
ku(0)=0
gu(0)=0
do i=1,nn
if (hu(i).ge.0) then
hu(i+1)=hu(i)+eu2
gu(i)=gu(i-1)+1
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
else
hu(i+1)=hu(i)+eul
gu(i)=gu(i-1)
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
endif
enddo
c Compute new coordinates (xd,yd) of rotated array and assign the value of a at
c (i,j) to the appropriate cell of b which encloses these coordinates.

```

```

c Generate 4-quadrant cartesian plane.
      do i=1,nn
        ku(-i)=-ku(i)
        gu(-i)=-gu(i)
      enddo
c Start process
      do j=1,n
        do i=1,n
c Employ coordinate transformation for rotation using the integer parameters.
          yd=m1*gu(i-nn)-n1*ku(j-nn)
          xd=m1*gu(j-nn)+n1*ku(i-nn)
c Compute the integer coordinates of array b associated with the above
c transformation.
          kx=mm+xd
          ky=mm+yd
c Assign value of a at (i,j) to the nearest cell of array b in which the
c transformed coordinates lie.
          b(ky,kx-1)=a(i,j)
          b(ky+1,kx)=a(i,j)
          b(ky,kx)=a(i,j)
          b(ky-1,kx)=a(i,j)
          b(ky,kx+1)=a(i,j)

          enddo
        enddo
      return
    end

```

```

subroutine rotate_four(a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)

```

```

*****

```

```

FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
           THROUGH AN ANGLE THETA (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b

```

```

PARAMETERS: a _ Input image of n X n size
             b _ Rotated image of m X m size

```

```

EXTERNAL SUBROUTINE :None

```

```

INTERNAL SUBROUTINE :None

```

```

*****

```

```

convert theta into radians

```

```

pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)

```

```

initial b

```

```

do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) = 0.
enddo
enddo

```

```

compute new coordinates (xd,yd) of rotate array

```

```

nn = n/2 + 1
mm = m/2 + 1

```

```

Compensation of the center position for rotation.

```

```

xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)

```

```

start process

```

```

do j = 1,n
do i = 1,n

```

```

generate 4-quadrant cartesian plan

```

```

y = float(j-nn)
x = float(i-nn)
xd = x * costh - y * sinth
yd = x * sinth + y * costh

```

```

compute both real and integer coordinates of array b
associated with the above transformation for later use

```

```

kx = mm+ nint(xd)
ky = mm+ nint(yd)

```

```

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.

```

```

b(ky,kx-1) = a(j,i)
b(ky+1,kx) = a(j,i)
b(ky,kx) = a(j,i)
b(ky-1,kx) = a(j,i)
b(ky,kx+1) = a(j,i)
enddo
enddo
return
end

```

```

subroutine rotate four neig (a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
integer xu,yu,xd,yd,dul,eul,du2,eu2,vu(256),hu(256),ku(256)
integer gu(256)

```

Function: This subroutine rotates an nXn array a through an angle theta (given in degrees) placing the result at the center of an mXm array b.

```

Convert theta into radiants.
pi=4.*atan(1.)
theta=theta*(pi/180.)

```

```

initialize b
do i=1,m
do j=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo

```

```

Compute centers of array a and b
nn=n/2+1
mm=m/2+1

```

Compute the integer of horizontal and vertical displacement along the x and y-axis which corresponds to n-pixel movement along the u-axis.

```

xu=nint(nn*cos(theta))
yu=nint(nn*sin(theta))
if (xu.lt.0) then
ml=-1
else
ml=1
endif
if (yu.lt.0) then
nl=1
else
nl=-1
endif

```

```

Compute the rotation parameters
dul=2*abs(yu)
du2=2*(abs(yu)-nn)
eul=2*abs(xu)
eu2=2*(abs(xu)-nn)
vu(1)=2*abs(yu)-nn
hu(1)=2*abs(xu)-nn
ku(0)=0
gu(0)=0
do i=1,nn
if (hu(i).ge.0) then
hu(i+1)=hu(i)+eu2
gu(i)=gu(i-1)+1
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
else
hu(i+1)=hu(i)+eul
gu(i)=gu(i-1)
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
endif
enddo

```

Compute new coordinates (xd,yd) of rotated array and assign the value of a at (i,j) to the appropriate cell of b which encloses these coordinates.

```

Generate 4-quadrant cartesian plane.
    do i=1,nn
        ku(-i)=-ku(i)
        gu(-i)=-gu(i)
    enddo

Start process
    do j=1,n
        do i=1,n
Employ coordinate transformation for rotation using the integer parameters.
            yd=m1*gu(i-nn)-n1*ku(j-nn)
            xd=m1*gu(j-nn)+n1*ku(i-nn)
Compute the integer coordinates of array b associated with the above
transformation.
            kx=mm+xd
            ky=mm+yd
Assign value of a at (i,j) to the nearest cell of array b in which the
transformed coordinates lie.
            b(ky+1,kx-1)=a(i,j)
            b(ky,kx-1)=a(i,j)
            b(ky-1,kx-1)=a(i,j)
            b(ky+1,kx)=a(i,j)
            b(ky,kx)=a(i,j)
            b(ky-1,kx)=a(i,j)
            b(ky+1,kx+1)=a(i,j)
            b(ky,kx+1)=a(i,j)
            b(ky-1,kx+1)=a(i,j)

        enddo
    enddo
return
end

```


subroutine rotate eight(a,n,b,m,theta)

real a(256,256),b(256,256),w(256,256)

FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
THROUGH AN ANGLE THETA (given in degree)
anticlockwise placing the result at the
center of an array mxm b

PARAMETERS: a _ Input image of n X n size
b _ Rotated image of m X m size

EXTERNAL SUBROUTINE :None

INTERNAL SUBROUTINE :None

convert theta into radians

pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)

initial b

do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) = 0.
enddo
enddo

compute new coordinates (xd,yd) of rotate array

nn = n/2 + 1
mm = m/2 + 1

start process

do j = 1,n
do i = 1,n

generate 4-quadrant cartesian plan

y = float(j-nn)
x = float(i-nn)
xd = x * costh - y * sinth
yd = x * sinth + y * costh

compute both real and integer coordinates of array b
associated with the above transformation for later use

kx = mm+ nint(xd)
ky = mm+ nint(yd)

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.

b(ky+1,kx-1) = a(j,i)
b(ky,kx-1) = a(j,i)
b(ky-1,kx-1) = a(j,i)
b(ky+1,kx) = a(j,i)
b(ky,kx) = a(j,i)
b(ky-1,kx) = a(j,i)
b(ky+1,kx+1) = a(j,i)
b(ky,kx+1) = a(j,i)
b(ky-1,kx+1) = a(j,i)
enddo
enddo
return
end

```

subroutine rotate eight neig (a,n,b,m,theta)
  real a(256,256),b(256,256),w(256,256)
  integer xu,yu,xd,yd,dul,eul,du2,eu2,vu(256),hu(256),ku(256)
  integer gu(256)
Function: This subroutine rotates an nXn array a through an angle theta
          (given in degrees ) placing the result at the center of an mXm
          array b.
Convert theta into radiants.
  pi=4.*atan(1.)
  theta=theta*(pi/180.)
initialize b
  do i=1,m
    do j=1,m
      b(j,i)=0
    enddo
  enddo
Compute centers of array a and b
  nn=n/2+1
  mm=m/2+1
Compute the integer of horizontal and vertical displacement along the x and
y-axis which corresponds to n-pixel movement along the u-axis.
  xu=nint(nn*cos(theta))
  yu=nint(nn*sin(theta))
  if (xu.lt.0) then
    ml=-1
  else
    ml=1
  endif
  if (yu.lt.0) then
    nl=1
  else
    nl=-1
  endif
Compute the rotation parameters
  dul=2*abs(yu)
  du2=2*(abs(yu)-nn)
  eul=2*abs(xu)
  eu2=2*(abs(xu)-nn)
  vu(1)=2*abs(yu)-nn
  hu(1)=2*abs(xu)-nn
  ku(0)=0
  gu(0)=0
  do i=1,nn
    if (hu(i).ge.0) then
      hu(i+1)=hu(i)+eu2
      gu(i)=gu(i-1)+1
      if (vu(i).ge.0) then
        vu(i+1)=vu(i)+du2
        ku(i)=ku(i-1)+1
      else
        vu(i+1)=vu(i)+dul
        ku(i)=ku(i-1)
      endif
    else
      hu(i+1)=hu(i)+eul
      gu(i)=gu(i-1)
      if (vu(i).ge.0) then
        vu(i+1)=vu(i)+du2
        ku(i)=ku(i-1)+1
      else
        vu(i+1)=vu(i)+dul
        ku(i)=ku(i-1)
      endif
    endif
  enddo
Compute new coordinates (xd,yd) of rotated array and assign the value of a at
(i,j) to the appropriate cell of b which encloses these coordinates.
Generate 4-quadrant cartesian plane.

```



```

do i=1,nn
ku(-i)=-ku(i)
gu(-i)=-gu(i)
enddo

```

Start process

```

do j=1,n
do i=1,n

```

Employ coordinate transformation for rotation using the integer parameters.

```

yd=m1*gu(i-nn)-n1*ku(j-nn)
xd=m1*gu(j-nn)+n1*ku(i-nn)

```

Compute the integer coordinates of array b associated with the above transformation.

```

kx=mm+xd
ky=mm+yd

```

Assign value of a at (j,i) to the nearest cell of array b in which the transformed coordinates lie.

```

b(ky+1,kx-1)=a(i,j)
b(ky,kx-1)=a(i,j)
b(ky-1,kx-1)=a(i,j)
b(ky+1,kx)=a(i,j)
b(ky,kx)=a(i,j)
b(ky-1,kx)=a(i,j)
b(ky+1,kx+1)=a(i,j)
b(ky,kx+1)=a(i,j)
b(ky-1,kx+1)=a(i,j)

```

```

enddo
enddo
return
end

```

```

      subroutine irtr_lrp_neig(x,m,y,n)
      real x(256,256),y(256,256),xx(256,256)
      *****
FUNCTION: This subroutine computes the inverse radon
      transform by:
          1. Back_projecting all the projections stored
             in array x giving the back_projection
             function y.
          2. Deconvolving the array y using the inverse_
             filter  $\text{SQRT}(k_x*k_x + k_y*k_y)$ .
External subroutines: bpr_nearest_neig
                     back_projection_deconvolution
Internal subroutines: none
      *****
Send message to user 'Filtering Radon Transform...'

      write(6,1)
      format(1x,'Filtering Radon Transform...')
      call projection_filtering(x,m)
Send message to user 'Computing Inverse Radon Transform...'

      write(6,2)
      format(1x,'Computing Inverse Radon Transform...')
Initialize array xx
      do i=1,n
      do j=1,n
      xx(j,i)=0.
      enddo
      enddo

Back project projections stored in x

      call bpr_lrp_neig(x,m,y,n)

      return
      end

```

```
subroutine irtr_2rp_neig(x,m,y,n)
real x(256,256),y(256,256),xx(256,256)
```

```
*****
```

```
FUNCTION: This subroutine computes the inverse radon
transform by:
```

1. Back_projecting all the projections stored in array x giving the back_projection function y.
2. Deconvolving the array y using the inverse filter $\text{SQRT}(k_x^2 + k_y^2)$.

```
External subroutines: bpr_nearest_neig
back_projection_deconvolution
```

```
Internal subroutines: none
```

```
*****
```

```
Send message to user 'Filtering Radon Transform...'
```

```
write(6,1)
format(1x,'Filtering Radon Transform...'$)
call projection_filtering(x,m)
```

```
Send message to user 'Computing Inverse Radon Transform...'
```

```
write(6,2)
format(1x,'Computing Inverse Radon Transform...'$)
```

```
Initialize array xx
```

```
do i=1,n
do j=1,n
xx(j,i)=0.
enddo
enddo
```

```
Back project projections stored in x
```

```
call bpr_2rp_neig(x,m,y,n)
```

```
return
end
```

```

      subroutine irtr_nearest(x,m,y,n)
      real x(256,256),y(256,256),xx(256,256)
      *****
FUNCTION: This subroutine computes the inverse radon
      transform by:
      1. Back_projecting all the projections stored
         in array x giving the back_projection
         function y.
      2. Deconvolving the array y using the inverse_
         filter SQRT(kx*kx + ky*ky*).
External subroutines: bpr_nearest_neig
                     back_projection_deconvolution
Internal subroutines: none
      *****
Filtering Radon Transform.
Send message to user ' Filtering Radon Transform...'

      write(6,1)
      format(1x,'Filtering Radon Transform...'$)

      call projection_filtering(x,m)

Send message to user 'Computing Inverse Radon Transform...'

      write(6,2)
      format(1x,'Computing Inverse Radon Transform...'$)
Initialize array xx
      do i=1,n
      do j=1,n
      xx(j,i)=0.
      enddo
      enddo

Back project projections stored in x

      call bpr_nearest(x,m,y,n)

      return
      end

```

```
subroutine irtr_nearest_neig(x,m,y,n)
real x(256,256),y(256,256),xx(256,256)
```

```
*****
```

```
FUNCTION: This subroutine computes the inverse radon
transform by:
```

1. Back_projecting all the projections stored in array x giving the back_projection function y.
2. Deconvolving the array y using the inverse_filter $\text{SQRT}(k_x*k_x + k_y*k_y)$.

```
External subroutines: bpr_nearest_neig
back_projection_deconvolution
```

```
Internal subroutines: none
```

```
*****
```

```
Filtering Radon Transform.
```

```
Send message to user ' Filtering Radon Transform...'
```

```
write(6,1)
format(1x,'Filtering Radon Transform...'$)
```

```
call projection_filtering(x,m)
```

```
Send message to user 'Computing Inverse Radon Transform...'
```

```
write(6,2)
format(1x,'Computing Inverse Radon Transform...'$)
```

```
Initialize array xx
```

```
do i=1,n
do j=1,n
xx(j,i)=0.
enddo
enddo
```

```
Back project projections stored in x
```

```
call bpr_nearest_neig(x,m,y,n)
```

```
return
end
```

```

      subroutine irtr_four(x,m,y,n)
      real x(256,256),y(256,256),xx(256,256)
      *****
FUNCTION: This subroutine computes the inverse radon
      transform by:
      1. Back_projecting all the projections stored
         in array x giving the back_projection
         function y.
      2. Deconvolving the array y using the inverse_
         filter SQRT(kx*kx + ky*ky*).
External subroutines: bpr_nearest_neig
                     back_projection_deconvolution
Internal subroutines: none
      *****
Send message to user 'Filtering Radon Transform...'

      write(6,1)
      format(1x,'Filtering Radon Transform...'$)
      call projection_filtering(x,m)
Send message to user 'Computing Inverse Radon Transform...'

      write(6,2)
      format(1x,'Computing Inverse Radon Transform...'$)
Initialize array xx
      do i=1,n
      do j=1,n
      xx(j,i)=0.
      enddo
      enddo

Back project projections stored in x

      call bpr_four(x,m,y,n)

      return
      end

```

```
subroutine irtr_four_neig(x,m,y,n)
real x(256,256),y(256,256),xx(256,256)
```

```
*****
```

```
FUNCTION: This subroutine computes the inverse radon
transform by:
```

1. Back_projecting all the projections stored in array x giving the back_projection function y.
2. Deconvolving the array y using the inverse filter $\text{SQRT}(k_x^2 + k_y^2)$.

```
External subroutines: bpr_four_neig
back_projection_deconvolution
```

```
Internal subroutines: none
```

```
*****
```

```
Filtering Radon Transform.
```

```
Send message to user ' Filtering Radon Transform...'
```

```
write(6,1)
format(1x,'Filtering Radon Transform...'$)
call projection_filtering(x,m)
```

```
Send message to user 'Computing Inverse Radon Transform...'
```

```
write(6,2)
format(1x,'Computing Inverse Radon Transform...'$)
```

```
Initialize array xx
```

```
do i=1,n
do j=1,n
xx(j,i)=0.
enddo
enddo
```

```
Back project projections stored in x
```

```
call bpr_four_neig(x,m,y,n)
```

```
open(unit=1,file='datair.dat',status='new')
```

```
format(4(1x,f14.4))
```

```
write(1,119)((y(j,i),j=1,128),i=1,128)
```

```
close(1)
```

```
return
```

```
end
```

119

```
subroutine irtr_eight(x,m,y,n)
  real x(256,256),y(256,256),xx(256,256)
```

```
*****
```

```
FUNCTION: This subroutine computes the inverse radon
transform by:
```

1. Back_projecting all the projections stored
in array x giving the back_projection
function y.
2. Deconvolving the array y using the inverse_
filter $\text{SQRT}(k_x^2 + k_y^2)$.

```
External subroutines: bpr_nearest_neig
                     back_projection_deconvolution
```

```
Internal subroutines: none
```

```
*****
```

```
Filtering Radon Transform.
```

```
Send message to user ' Filtering Radon Transform...'
```

```
  write(6,1)
  format(1x,'Filtering Radon Transform...'$)
```

```
  call projection_filtering(x,m)
```

```
Send message to user 'Computing Inverse Radon Transform...'
```

```
  write(6,2)
  format(1x,'Computing Inverse Radon Transform...'$)
```

```
Initialize array xx
```

```
  do i=1,n
  do j=1,n
  xx(j,i)=0.
  enddo
  enddo
```

```
Back project projections stored in x
```

```
  call bpr_eight(x,m,y,n)
```

```
  return
end
```



```

      subroutine irtr_eight_neig(x,m,y,n)
      real x(256,256),y(256,256),xx(256,256)
c *****
c FUNCTION: This subroutine computes the inverse radon
c transform by:
c     1. Back_projecting all the projections stored
c        in array x giving the back_projection
c        function y.
c     2. Deconvolving the array y using the inverse_
c        filter SQRT(kx*kx + ky*ky*).
c External subroutines: bpr_eight_neig
c                      back_projection_deconvolution
c Internal subroutines: none
c *****
c Filtering Radon Transform.
c Send message to user ' Filtering Radon Transform...'
c
c     write(6,1)
c     format(1x,'Filtering Radon Transform...'$)
c
c     call projection_filtering(x,m)
c
c Send message to user 'Computing Inverse Radon Transform...'
c
c     write(6,2)
c     format(1x,'Computing Inverse Radon Transform...'$)
c Initialize array xx
c     do i=1,n
c     do j=1,n
c     xx(j,i)=0.
c     enddo
c     enddo
c
c Back project projections stored in x
c
c     call bpr_eight_neig(x,m,y,n)
c
c     return
c     end

```

```

subroutine bpr_lrp_neig(b,m,a,n)
  real a(256,256),b(256,256),xx(256,256)
c *****
c Function: this routine back-projects the
c           data stored in b to reconstruct
c           the image a.
c External subroutines: Rotate_nearest_neig
c Internal subroutines: None
c Parameters: b _ input, size m X m
c             a _ output, size n X n
c *****
c x, y and sum are working space
  real x(256,256),y(256,256)
  real sum(256,256)
c Send message to user 'Computing back_projection ...'
c
  write(6,1)
1  format(1x,'Computing back_projection...'$)
c Compute size of image in which b will be rotated to prevent
c corner clipping.
  l=nint(sqrt(2.)*float(m))
c Compute centers of arrays to be used later
  nn=n/2+1
  ll=l/2+1
  ln=ll-nn
c Initialize working space
  do i=1,l
  do j=1,l
    x(j,i)=0.
    y(j,i)=0.
    sum(j,i)=0.
    a(j,i)=0.
    xx(j,i)=0.
  enddo
  enddo
c Compute the angle of rotation in the reverse direction
  do k=1,m
    theta=float(1-k)
c Back_project projection
  do i=1,m
  do j=1,m
    x(j,i)=b(j,k)
  enddo
  enddo
c Rotate x by angle at which projection was taken
  call rotate_lrp_neig(x,m,y,l,theta)
c Add result to previous back_projection function
  do i=1,l
  do j=1,l
    sum(j,i)=sum(j,i)+y(j,i)
  enddo
  enddo
c
c
  enddo
c Window the back_projection function to extract reconstruction
c (size n X n) and write it to output
  do i=1,n
  do j=1,n
    a(j,i)=sum(j+ln,i+ln)
  enddo
  enddo
c
  return
end

```

```

subroutine bpr_2rp_neig(b,m,a,n)
  real a(256,256),b(256,256),xx(256,256)
c *****
c Function: this routine back-projects the
c           data stored in b to reconstruct
c           the image a.
c External subroutines: Rotate_nearest_neig
c Internal subroutines: None
c Parameters: b _ input, size m X m
c             a _ output, size n X n
c *****
c x, y and sum are working space
  real x(256,256),y(256,256)
  real sum(256,256)
c Send message to user 'Computing back_projection ...'
c
  write(6,1)
1  format(1x,'Computing back_projection...'$)
c Compute size of image in which b will be rotated to prevent
c corner clipping.
  l=nint(sqrt(2.)*float(m))
c Compute centers of arrays to be used later
  nn=n/2+1
  ll=l/2+1
  ln=ll-nn
c Initialize working space
  do i=1,l
  do j=1,l
    x(j,i)=0.
    y(j,i)=0.
    sum(j,i)=0.
    a(j,i)=0.
    xx(j,i)=0.
  enddo
  enddo
c Compute the angle of rotation in the reverse direction
  do k=1,m
    theta=float(1-k)
c Back_project projection
  do i=1,m
  do j=1,m
    x(j,i)=b(j,k)
  enddo
  enddo
c Rotate x by angle at which projection was taken
  call rotate_2rp_neig(x,m,y,l,theta)
c Add result to previous back_projection function
  do i=1,l
  do j=1,l
    sum(j,i)=sum(j,i)+y(j,i)
  enddo
  enddo
c
c
  enddo
c Window the back_projection function to extract reconstruction
c (size n X n) and write it to output
  do i=1,n
  do j=1,n
    a(j,i)=sum(j+ln,i+ln)
  enddo
  enddo
c
  return
end

```

```

      subroutine bpr_nearest(b,m,a,n)
      real a(256,256),b(256,256),xx(256,256)
c *****
c Function: this routine back-projects the
c           data stored in b to reconstruct
c           the image a.
c External subroutines: Rotate_nearest_neig
c Internal subroutines: None
c Parameters: b _ input, size m X m
c             a _ output, size n X n
c *****
c x, y and sum are working space
      real x(256,256),y(256,256)
      real sum(256,256)
c Send message to user 'Computing back_projection ...'
c
      write(6,1)
1      format(1x,'Computing back_projection...'$)
c Compute size of image in which b will be rotated to prevent
c corner clipping.
      l=nint(sqrt(2.)*float(m))
c Compute centers of arrays to be used later
      nn=n/2+1
      ll=l/2+1
      ln=ll-nn
c Initialize working space
      do i=1,l
      do j=1,l
      x(j,i)=0.
      y(j,i)=0.
      sum(j,i)=0.
      a(j,i)=0.
      xx(j,i)=0.
      enddo
      enddo
c Compute the angle of rotation in the reverse direction
      do k=1,m
      theta=float(1-k)
c Back_project projection
      do i=1,m
      do j=1,m
      x(j,i)=b(j,k)
      enddo
      enddo
c Rotate x by angle at which projection was taken
      call rotate_nearest(x,m,y,l,theta)
c Add result to previous back_projection function
      do i=1,l
      do j=1,l
      sum(j,i)=sum(j,i)+y(j,i)
      enddo
      enddo
c
c
      enddo
c Window the back_projection function to extract reconstruction
c (size n X n) and write it to output
      do i=1,n
      do j=1,n
      a(j,i)=sum(j+ln,i+ln)
      enddo
      enddo
c
      return
      end

```

```

      subroutine bpr_nearest_neig(b,m,a,n)
      real a(256,256),b(256,256),xx(256,256)
c *****
c Function: this routine back-projects the
c           data stored in b to reconstruct
c           the image a.
c External subroutines: Rotate_nearest_neig
c Internal subroutines: None
c Parameters: b _ input, size m X m
c              a _ output, size n X n
c *****
c x, y and sum are working space
      real x(256,256),y(256,256)
      real sum(256,256)
c Send message to user 'Computing back_projection ...'
c
      write(6,1)
1      format(1x,'Computing back_projection...'$)
c Compute size of image in which b will be rotated to prevent
c corner clipping.
      l=nint(sqrt(2.)*float(m))
c Compute centers of arrays to be used later
      nn=n/2+1
      ll=l/2+1
      ln=ll-nn
c Initialize working space
      do i=1,l
      do j=1,l
      x(j,i)=0.
      y(j,i)=0.
      sum(j,i)=0.
      a(j,i)=0.
      xx(j,i)=0.
      enddo
      enddo
c Compute the angle of rotation in the reverse direction
      do k=1,m
      theta=float(1-k)
c Back_project projection
      do i=1,m
      do j=1,m
      x(j,i)=b(j,k)
      enddo
      enddo
c Rotate x by angle at which projection was taken
      call rotate_nearest_neig(x,m,y,l,theta)
c Add result to previous back_projection function
      do i=1,l
      do j=1,l
      sum(j,i)=sum(j,i)+y(j,i)
      enddo
      enddo
c
c
      enddo
c Window the back_projection function to extract reconstruction
c (size n X n) and write it to output
      do i=1,n
      do j=1,n
      a(j,i)=sum(j+ln,i+ln)
      enddo
      enddo
c
      return
      end

```

```

subroutine bpr_four(b,m,a,n)
  real a(256,256),b(256,256),xx(256,256)
c *****
c Function: this routine back-projects the
c           data stored in b to reconstruct
c           the image a.
c External subroutines: Rotate_nearest_neig
c Internal subroutines: None
c Parameters: b _ input, size m X m
c             a _ output, size n X n
c *****
c x, y and sum are working space
  real x(256,256),y(256,256)
  real sum(256,256)
c Send message to user 'Computing back_projection ...'
c
  write(6,1)
1  format(1x,'Computing back_projection...'$)
c Compute size of image in which b will be rotated to prevent
c corner clipping.
  l=nint(sqrt(2.)*float(m))
c Compute centers of arrays to be used later
  nn=n/2+1
  ll=l/2+1
  ln=ll-nn
c Initialize working space
  do i=1,l
  do j=1,l
    x(j,i)=0.
    y(j,i)=0.
    sum(j,i)=0.
    a(j,i)=0.
    xx(j,i)=0.
  enddo
  enddo
c Compute the angle of rotation in the reverse direction
  do k=1,m
    theta=float(1-k)
c Back_project projection
  do i=1,m
  do j=1,m
    x(j,i)=b(j,k)
  enddo
  enddo
c Rotate x by angle at which projection was taken
  call rotate_four(x,m,y,l,theta)
c Add result to previous back_projection function
  do i=1,l
  do j=1,l
    sum(j,i)=sum(j,i)+y(j,i)
  enddo
  enddo
c
c
  enddo
c Window the back_projection function to extract reconstruction
c (size n X n) and write it to output
  do i=1,n
  do j=1,n
    a(j,i)=sum(j+ln,i+ln)
  enddo
  enddo
c
  return
end

```

```

      subroutine bpr_four_neig(b,m,a,n)
      real a(256,256),b(256,256),xx(256,256)
c *****
c Function: this routine back-projects the
c           data stored in b to reconstruct
c           the image a.
c External subroutines: Rotate_four_neig
c Internal subroutines: None
c Parameters: b _ input, size m X m
c             a _ output, size n X n
c *****
c x, y and sum are working space
      real x(256,256),y(256,256)
      real sum(256,256)
c Send message to user 'Computing back_projection ...'
c
      write(6,1)
1      format(1x,'Computing back_projection...'$)
c Compute size of image in which b will be rotated to prevent
c corner clipping.
      l=nint(sqrt(2.)*float(m))
c Compute centers of arrays to be used later
      nn=n/2+1
      ll=l/2+1
      ln=ll-nn
c Initialize working space
      do i=1,l
      do j=1,l
      x(j,i)=0.
      y(j,i)=0.
      sum(j,i)=0.
      a(j,i)=0.
      xx(j,i)=0.
      enddo
      enddo
c Compute the angle of rotation in the reverse direction
      do k=1,m
      theta=float(1-k)
c Back_project projection
      do i=1,m
      do j=1,m
      x(j,i)=b(j,k)
      enddo
      enddo
c Rotate x by angle at which projection was taken
      call rotate_four_neig(x,m,y,l,theta)
c Add result to previous back_projection function
      do i=1,l
      do j=1,l
      sum(j,i)=sum(j,i)+y(j,i)
      enddo
      enddo
c
c
      enddo
c Window the back_projection function to extract reconstruction
c (size n X n) and write it to output
      do i=1,n
      do j=1,n
      a(j,i)=sum(j+ln,i+ln)
      enddo
      enddo
c
      return
      end

```

```

      subroutine bpr_eight(b,m,a,n)
      real a(256,256),b(256,256),xx(256,256)
c *****
c Function: this routine back-projects the
c           data stored in b to reconstruct
c           the image a.
c External subroutines: Rotate_nearest_neig
c Internal subroutines: None
c Parameters: b _ input, size m X m
c             a _ output, size n X n
c *****
c x, y and sum are working space
      real x(256,256),y(256,256)
      real sum(256,256)
c Send message to user 'Computing back_projection ...'
c
      write(6,1)
1      format(1x,'Computing back_projection...'$)
c Compute size of image in which b will be rotated to prevent
c corner clipping.
      l=nint(sqrt(2.)*float(m))
c Compute centers of arrays to be used later
      nn=n/2+1
      ll=l/2+1
      ln=ll-nn
c Initialize working space
      do i=1,l
      do j=1,l
      x(j,i)=0.
      y(j,i)=0.
      sum(j,i)=0.
      a(j,i)=0.
      xx(j,i)=0.
      enddo
      enddo
c Compute the angle of rotation in the reverse direction
      do k=1,m
      theta=float(1-k)
c Back_project projection
      do i=1,m
      do j=1,m
      x(j,i)=b(j,k)
      enddo
      enddo
c Rotate x by angle at which projection was taken
      call rotate_eight(x,m,y,l,theta)
c Add result to previous back_projection function
      do i=1,l
      do j=1,l
      sum(j,i)=sum(j,i)+y(j,i)
      enddo
      enddo
c
c
      enddo
c Window the back_projection function to extract reconstruction
c (size n X n) and write it to output
      do i=1,n
      do j=1,n
      a(j,i)=sum(j+ln,i+ln)
      enddo
      enddo
c
      return
      end

```



```

      subroutine bpr_eight_neig(b,m,a,n)
      real a(256,256),b(256,256),xx(256,256)
c *****
c Function: this routine back-projects the
c           data stored in b to reconstruct
c           the image a.
c External subroutines: Rotate_eight_neig
c Internal subroutines: None
c Parameters: b _ input, size m X m
c             a _ output, size n X n
c *****
c x, y and sum are working space
      real x(256,256),y(256,256)
      real sum(256,256)
c Send message to user 'Computing back_projection ...'
c
      write(6,1)
1      format(1x,'Computing back_projection...'$)
c Compute size of image in which b will be rotated to prevent
c corner clipping.
      l=nint(sqrt(2.)*float(m))
c Compute centers of arrays to be used later
      nn=n/2+1
      ll=l/2+1
      ln=ll-nn
c Initialize working space
      do i=1,l
      do j=1,l
      x(j,i)=0.
      y(j,i)=0.
      sum(j,i)=0.
      a(j,i)=0.
      xx(j,i)=0.
      enddo
      enddo
c Compute the angle of rotation in the reverse direction
      do k=1,m
      theta=float(1-k)
c Back_project projection
      do i=1,m
      do j=1,m
      x(j,i)=b(j,k)
      enddo
      enddo
c Rotate x by angle at which projection was taken
      call rotate_eight_neig(x,m,y,l,theta)
c Add result to previous back_projection function
      do i=1,l
      do j=1,l
      sum(j,i)=sum(j,i)+y(j,i)
      enddo
      enddo
c
c
      enddo
c Window the back_projection function to extract reconstruction
c (size n X n) and write it to output
      do i=1,n
      do j=1,n
      a(j,i)=sum(j+ln,i+ln)
      enddo
      enddo
c
      return
      end

```

```
subroutine projection_filtering(a,n)
```

```
c
c Function: This subroutine filters a Radon Transform
c           obtained by straight projection (filtered)
c           using the appropriate inverse filter, i.e.
c           ABS(kx) where kx is the spatial frequency.
c
```

```
real a(256,256),y(256,256),c(256,256),za(256)
real zb(256),zc(256),zd(256)
```

```
c
c initialize array c
c
```

```
do i=1,n
do j=1,n
c(j,i)=0.
enddo
enddo
do j=1,256
za(j)=0.
zb(j)=0.
zc(j)=0.
zd(j)=0.
enddo
```

```
c
c take Fourier transform of a
c
```

```
nn=256/2+1
jframe=nint(float(256-n)/2.)
do i=1,n
do j=1,n
za(j+jframe)=a(j,i)
enddo
call fftld(za,zb,256,-1)
```

```
c
c multiply spectrum by inverse filter
c
```

```
do j=1,256
xk=abs(float(j-nn)/float(256))
zc(j)=za(j)*xk
zd(j)=zb(j)*xk
enddo
```

```
c
c call fftld(zc,zd,256,1)
c
```

```
do j=1,n
c(j,i)=zc(j+jframe)
enddo
enddo
```

```
c write output
c
```

```
do i=1,n
do j=1,n
a(j,i)=c(j,i)
enddo
enddo
```

```
c
c return
c end
```

```

      subroutine project(a,b,n)
c *****
c Function: This subroutine sums
c           over all rows of an n X n array a giving 1D
c projection of size n _ array b
c *****
      real a(256,256),b(256),s(256,256),z(256)
c initialize arrays s & b
      do j=1,n
        b(j)=0.
        z(j)=0.
        do i=1,n
          s(j,i)=0.
        enddo
      enddo
c set initial value for summing process and sum all pixels along
c row i together
      do j=1,n
        s(j,1)=a(j,1)
        do i=2,n
          s(j,i)=s(j,i-1)+a(j,i)
        enddo
c Assign value of sum at column j to output _b(j)
        b(j)=s(j,n)
      enddo
c
      return
      end

```

```
subroutine back_projection_deconvolution(a,n)
```

```
c  
c Function: This subroutine deconvolves a reconstruction  
c obtained by straight backprojection (un-filtered)  
c using the appropriate inverse filter, i.e.  
c  $\sqrt{k_x^2 + k_y^2}$  where  $k_x$  and  $k_y$  are the  
c spatial frequencies.  
c
```

```
real a(256,256),aa(256,256),xx(256,256)  
real b(256,256),bb(256,256)
```

```
c  
c initialize array b  
c
```

```
do i=1,n  
do j=1,n  
aa(j,i)=0.  
bb(j,i)=0.  
enddo  
enddo
```

```
c  
c take Fourier transform of a  
c
```

```
call fft2d(a,aa,n,-1)
```

```
c  
c multiply spectrum by inverse filter  
c
```

```
nn=n/2+1  
do i=1,n  
do j=1,n  
xk=float(i-nn)/float(n)  
yk=float(j-nn)/float(n)  
fil=sqrt(xk*xk+yk*yk)!value of inverse filter at (xk,yk)  
b(j,i)=a(j,i)*fil  
bb(j,i)=aa(j,i)*fil  
enddo  
enddo
```

```
c  
c call fft2d(b,bb,n,1)  
c
```

```
c  
c write output  
c
```

```
do i=1,n  
do j=1,n  
a(j,i)=b(j,i)  
enddo  
enddo
```

```
c  
c return  
c end
```

```

subroutine gnoise(s,n,iseed)
real s(256),w1(256)

```

```

C
C *****
C
C FUNCTION: Generates an n size array s of Gaussian distributed noise
C           with zero mean and unit variance.
C
C AUTHOR: J M Blackledge
C
C PARAMETERS
C
C Input: iseed -set seed to a positive integer.
C        n - array size
C
C Output: s - fGaussian distributed random signal.
C
C EXTERNAL SUBROUTINES: None
C
C INTERNAL SUBROUTINES: gasdev, ran1
C
C NB : output consists of positive numbers only.
C *****

```

```

C use function gasdev to generate array of random numbers.
C

```

```

        idum=-iseed
        do i=1,n
C NB : output consists of positive and negative numbers.

```

```

        s(i)=gasdev(idum)
C
C NB : output consists of positive numbers only.

```

```

        s(i)=abs(gasdev(idum))
C
        enddo
        return
        end

```

```

C *****
C
C function gasdev(idum)

```

```

C FUNCTION: Returns a Gaussian distributed deviate with zero mean and unit
C           variance, using RAN1(IDUM) as the source of uniform deviates.
C           (from 'Numerical Recipes' by Press et al. 1987).
C

```

```

        data iset/0/
        if (iset.eq.0) then
1      v1=2.*rang(idum) -1.
        v2=2.*rang(idum) -1.
        r=v1**2.+v2**2.
        if (r.ge.1.) goto 1
        fac=sqrt(-2.*log(r)/r)
        gset=v1*fac
        gasdev=v2*fac
        iset=1
        else
        gasdev=gset
        iset=0
        endif

```

```

C
        return
        end

```

```

C *****
C
C function rang(idum)

```

```

C

```

c FUNCTION: This function returns a uniform random number between 0. and 1.
c set IDUM to any negative value to initiate or reinitiate the
c sequence. This source code is taken from 'Numerical Recipes'
c by Press et al.

dimension r(97)
parameter(m1=259200,ia1=7141,ic1=54773,rm1=1./m1)
parameter(m2=134456,ia2=8121,ic2=28411,rm2=1./m2)
parameter(m3=243000,ia3=4561,ic3=51349)
data iff /0/

c
if (idum.lt.0.or.iff.eq.0) then
iff=1

ix1=mod(ic1-idum,m1)
ix1=mod(ia1*ix1+ic1,m1)
ix2=mod(ix1,m2)
ix1=mod(ia1*ix1+ic1,m1)
ix3=mod(ix1,m3)

c
do i=1,97
ix1=mod(ia1*ix1+ic1,m1)
ix2=mod(ia2*ix2+ic2,m2)
r(i)=(float(ix1)+float(ix2)*rm2)*rm1
enddo
idum=1

c
endif

c
ix1=mod(ia1*ix1+ic1,m1)
ix2=mod(ia2*ix2+ic2,m2)
ix3=mod(ia3*ix3+ic3,m3)

c
i=1+(97*ix3)/m3
if (i.gt.97.or.i.lt.1) pause
rang=r(i)
r(i)=(float(ix1)+float(ix2)*rm2)*rm1
return
end

subroutine fft2d(a,b,n,isgn)

```
C
C *****
C
C FUNCTION: This subroutine computes the DFT of a 2D array with
C           real and imaginary parts a and b respectively.
C
C           Note: This routine computes the 'optical' transform - the
C                 zero frequency occurs at (n/2+1,n/2+1).
C
C PARAMETERS
C
C Input: a - real part of image
C        b - imaginary part of image
C        n - image size (=n x n where n=2**k; k-integer)
C           Note: n must be less than or equal to 512.
C        isgn = -1 gives forward Fourier transform
C              = 1 gives inverse Fourier transform
C
C Output:
C
C EXTERNAL SUBROUTINES: None
C
C INTERNAL SUBROUTINES: fft1d
C
C *****
C
C Use subroutine fft1d to design a 2D FFT by computing
C the 1D FFT of the columns and rows separately.
C
C       real a(256,256),b(256,256),c(256,256),d(256,256)
C       real aa(256),bb(256),cc(256),dd(256)
C
C Put columns into single-dimension array and call fft1d.
C initialize arrays
C   do i=1,n
C   aa(i)=0.
C   bb(i)=0.
C   cc(i)=0.
C   dd(i)=0.
C   enddo
C   do i=1,n
C   do j=1,n
C   c(j,i)=0.
C   d(j,i)=0.
C   enddo
C   enddo
C   do i=1,n
C   do j=1,n
C   aa(j)=a(j,i)
C   bb(j)=b(j,i)
C   enddo
C
C   call fft1d(aa,bb,n,isgn)
C
C   do j=1,n
C   c(j,i)=aa(j)
C   d(j,i)=bb(j)
C   enddo
C   enddo
C
C Put rows into single-dimension array and call fft1d.
C
C   do j=1,n
C   do i=1,n
C   cc(i)=c(j,i)
C   dd(i)=d(j,i)
```

```

enddo

c
call fftld(cc,dd,n,isgn)

do i=1,n
a(j,i)=cc(i)
b(j,i)=dd(i)
enddo
enddo

c
return
end

c
*****

c
subroutine fftld(a,b,n,isgn)
c
c This subroutine computes the DFT of a complex array whose
c real and imaginary parts are a and b respectively
c using the successive doubling method ('Digital Image Processing'
c by Gonzalez and Wintz, Addison-Wesley publishing company, 1987).
c
c The subroutine returns the real (a) and imaginary (b) parts
c of the DFT.
c
c The size of the arrays n must be an integer power of 2
c less than or equal to 512. If arrays larger than 512
c are required, then the dimensions of a,b and c must
c be adjusted accordingly.
c
c Note: Zero frequency occurs at n/2+1.
c
c By convention, the forward Fourier transform is obtained
c when isgn=-1 and the inverse Fourier transform is obtained
c when isgn=1.
c
real a(256),b(256),aa(256),bb(256),cc(256),dd(256)
complex c(512),u,v,w,cmplx
if(isgn.lt.0)den=float(n)
if(isgn.gt.0)den=1.
pi=4.*atan(1.)
p=float(n)
q=log(p)/log(2.)
k=int(q)
nh=n/2
nm=n-1
c initialize complex array c
do j=1,n
c(j)=(0.,0.)
enddo
j=1
do 1 i=nh+1,n
c(j)=cmplx(a(i),b(i))
j=j+1
do 2 i=1,nh
c(j)=cmplx(a(i),b(i))
j=j+1
j=1
do 3 i=1,nm
if(i.ge.j)goto 10
v=c(j)
c(j)=c(i)
c(i)=v
jj=nh
if(jj.ge.j)goto 3
j=j-jj
jj=jj/2
goto 4
10
4

```



```

3      j=j+jj
      do 6 l=1,k
      l1=2**l
      l2=l1/2
      u=(1.,0.)
      x=pi*float(isgn)/l2
      w=cmlpx(cos(x),sin(x))
      do 6 j=1,l2
      do 5 i=j,n,l1
      ii=i+l2
      v=c(ii)*u
      c(ii)=c(i)-v
5      c(i)=c(i)+v
6      u=u*w
      j=1
      do 7 i=1,n
      c(i)=c(i)/den
c7      c(i)=c(i)/float(n)
      do 8 i=nh+1,n
      a(j)=real(c(i))
      b(j)=aimag(c(i))
8      j=j+1
      do 9 i=1,nh
      a(j)=real(c(i))
      b(j)=aimag(c(i))
9      j=j+1
      return
      end

```

subroutine fftld(a,b,n,isgn)

```
C
C *****
C
C This subroutine computes the DFT of a complex array whose
C real and imaginary parts are a and b respectively
C using the successive doubling method ('Digital Image Processing'
C by Gonzalez and Wintz, Addison-Wesley publishing company, 1987).
C
C The subroutine returns the real (a) and imaginary (b) parts
C of the DFT.
C
C The size of the arrays n must be an integer power of 2
C less than or equal to 512. If arrays larger than 512
C are required, then the dimensions of a,b and c must
C be adjusted accordingly.
C
C Note: Zero frequency occurs at n/2+1.
C
C By convention, the forward Fourier transform is obtained
C when isgn=-1 and the inverse Fourier transform is obtained
C when isgn=1.
C
      real a(256),b(256),za(256),zb(256),zc(256),zd(256)
      complex c(512),u,v,w,cmplx
      if(isgn.lt.0)den=float(n)
      if(isgn.gt.0)den=1.
      pi=4.*atan(1.)
      p=float(n)
      q=log(p)/log(2.)
      k=int(q)
      nh=n/2
      nm=n-1
C initialize complex array c
      do j=1,n
        c(j)=(0.,0.)
      enddo
      j=1
      do 1 i=nh+1,n
        c(i)=cmplx(a(i),b(i))
        j=j+1
      do 2 i=1,nh
        c(j)=cmplx(a(i),b(i))
        j=j+1
      do 3 i=1,nm
        if(i.ge.j)goto 10
        v=c(j)
        c(j)=c(i)
        c(i)=v
        jj=nh
        if(jj.ge.j)goto 3
        j=j-jj
        jj=jj/2
        goto 4
      do 5 i=j,n,11
        j=j+jj
        do 6 l=1,k
          ll=2**l
          l2=ll/2
          u=(1.,0.)
          x=pi*float(isgn)/l2
          w=cmplx(cos(x),sin(x))
          do 6 j=1,l2
            do 5 i=j,n,11
              ii=i+l2
              v=c(ii)*u
              c(ii)=c(i)-v
```

```
5      c(i)=c(i)+v
6      u=u*w
      j=1
      do 7 i=1,n
7      c(i)=c(i)/den
c7      c(i)=c(i)/float(n)
      do 8 i=nh+1,n
      a(j)=real(c(i))
      b(j)=aimag(c(i))
8      j=j+1
      do 9 i=1,nh
      a(j)=real(c(i))
      b(j)=aimag(c(i))
9      j=j+1
      return
      end
```

```

      subroutine signal_noise_ratio(y,n)
      real y(256,256),xȳ(256,256)
c *****
c This subroutine add the required Signal to Noise ratio (SNR) to
c the appropriate signal.
c *****
c Create working space area
      real se(256,256),sn(256),snr
      do i=1,n
      do j=1,n
      se(j,i)=0.
      enddo
      enddo

c
      do i=1,n
      do j=1,n
      se(j,i)=y(j,i)
      enddo
      enddo

c
      write(6,*)'enter Signal to Noise Ratio='
      read(5,*)snr
      do i=1,n
      do j=1,n
      sn(j)=se(j,i)*(1./snr)
      se(j,i)=se(j,i)+sn(j)
      enddo
      enddo

c
      do i=1,n
      do j=1,n
      y(j,i)=se(j,i)
      enddo
      enddo
      return
      end

```

subroutine norm_signal(x1,n)

FUNCTION: returns an image with absolute maximum value of 1.

PARAMETERS

Input: x1 - signal
n - size of signal

Output: x1 - normalized signal

INTERNAL SUBROUTINES: None

EXTERNAL SUBROUTINES: None

real x1(256),s1(256),w1(256)

real max1

Generate working space

do i=1,n
s1(i)=x1(i)
enddo

Set the negative value to zero

do i=1,n
if(s1(i).lt.0.) s1(i)=0.
enddo

Find the absolute maximum value of the image.

max1=0.
do i=1,n
xm=s1(i)
if(xm.gt.max1) max1=xm
enddo

write(6,*)'max1=',xm

Normalize the image.

do i=1,n
x1(i)=s1(i)/max1
enddo

return
end

```
subroutine radon_gauss_noise(xy,m)
real xy(256,256),yt(256,256),ye(256,256)
```

```
c
c *****
c
c Function: This subroutine creates Gaussian noise with zero
c           with a selected Signal to Noise Ratio mean and
c           variance 1 and add it to the Radon Transform.
c *****
c
c           real s(256,256),xl(256),wl(256)
c           real max(256)
c Generate working space
c           do i=1,m
c             do j=1,m
c               s(j,i)=xy(j,i)
c             enddo
c           enddo
c Set the negative value to zero
c           do i=1,m
c             do j=1,m
c               if(s(j,i).lt.0.) s(j,i)=0.
c             enddo
c           enddo
c Find the absolute maximum value of the image.
c
c           do i=1,m
c             max(i)=0.
c           enddo
c           do i=1,m
c             do j=1,m
c               xl(i)=s(j,i)
c               if(xl(i).gt.max(i)) max(i)=xl(i)
c             enddo
c           enddo
c
c create normalized Gaussian noise image.
c
c           do i=1,m
c             iseed=i*10
c             call gnoise (wl,m,iseed)
c             call norm_signal (wl,m)
c             do j=1,m
c               ye(j,i)=wl(j)
c             enddo
c           enddo
c
c Add Radon transform image to noise image.
c
c           write (6,*) 'enter Signal to Noise Ratio:'
c           read(5,*) snr
c           do i=1,m
c             do j=1,m
c               yt(j,i)=s(j,i) * snr + ye(j,i)*max(i)
c             enddo
c           enddo
c           do i=1,m
c             do j=1,m
c               xy(j,i)=yt(j,i)
c             enddo
c           enddo
c
c           return
c           end
```

```

subroutine rtr_rectangle_sum(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
*****
Function: This subroutine computes the Radon Transform of
n X n image.
External subroutines: Rotate_rectangle_sum
Project
Internal subroutines: None
Parameters: x - n X n input array
y - m X m output array
*****
Send message to user 'Computing Radon Transform...'

write(6,1)
format(1x,'Computing Radon Transform...'$)
initialize array y
do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo
Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.
do i=1,m
theta=float(i-1)
call rotate_rectangle_sum(x,n,w,m,theta)
call project(w,z,m)
Stack projections together to form image of "Radon space"
data.
do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```

```

subroutine rtr triangle_sum(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
c *****
C Function: This subroutine computes the Radon Transform of
c      n X n image.
c External subroutines: Rotate_triangle_sum
c                      Project
c Internal subroutines: None
c Parameters: x - n X n input array
c            y - m X m output array
c *****
c Send message to user 'Computing Radon Transform...'
c
      write(6,1)
1      format(1x,'Computing Radon Transform...'$)
c initialize array y
      do j=1,m
      do i=1,m
      y(j,i)=0.
      enddo
      enddo
c Compute angle of rotation in steps of 1 degree and rotate
c the object, take projection of rotated object.
      do i=1,m
      theta=float(i-1)
      call rotate_triangle_sum(x,n,w,m,theta)
      call project(w,z,m)
c Stack projections together to form image of "Radon space"
c data.
      do j=1,m
      y(j,i)=z(j)
      enddo
      enddo
      return
      end

```



```

subroutine rtr divide_rectangle_sum(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
c *****
C Function: This subroutine computes the Radon Transform of
c          n X n image.
c External subroutines: Rotate_nearest_neig
c                      Project
c Internal subroutines: None
c Parameters: x - n X n input array
c             y - m X m output array
c *****
c Send message to user 'Computing Radon Transform...'
c
      write(6,1)
1      format(1x,'Computing Radon Transform...'$)
c initialize array y
      do j=1,m
      do i=1,m
      y(j,i)=0.
      enddo
      enddo
c Compute angle of rotation in steps of 1 degree and rotate
c the object, take projection of rotated object.
      do i=1,m
      theta=float(i-1)
      call rotate_divide_rectangle_sum(x,n,w,m,theta)
      call project(w,z,m)
c Stack projections together to form image of "Radon space"
c data.
      do j=1,m
      y(j,i)=z(j)
      enddo
      enddo
      return
      end

```

```

subroutine rtr divide triangle sum(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
c *****
C Function: This subroutine computes the Radon Transform of
c      n X n image.
c External subroutines: Rotate_nearest_neig
c                      Project
c Internal subroutines: None
c Parameters: x - n X n input array
c             y - m X m output array
c *****
c Send message to user 'Computing Radon Transform...'
c
      write(6,1)
1      format(1x,'Computing Radon Transform...'$)
c initialize array y
      do j=1,m
      do i=1,m
      y(j,i)=0.
      enddo
      enddo
c Compute angle of rotation in steps of 1 degree and rotate
c the object, take projection of rotated object.
      do i=1,m
      theta=float(i-1)
      call rotate_divide_triangle_sum(x,n,w,m,theta)
      call project(w,z,m)
c Stack projections together to form image of "Radon space"
c data.
      do j=1,m
      y(j,i)=z(j)
      enddo
      enddo
      return
      end

```

```

subroutine rtr 2dlinear sum(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
c *****
C Function: This subroutine computes the Radon Transform of
c      n X n image.
c External subroutines: Rotate_2dlinear_sum
c                      Project
c Internal subroutines: None
c Parameters: x - n X n input array
c              y - m X m output array
c *****
c Send message to user 'Computing Radon Transform...'
c
      write(6,1)
1      format(1x,'Computing Radon Transform...'$)
c initialize array y
      do j=1,m
      do i=1,m
      y(j,i)=0.
      enddo
      enddo
c Compute angle of rotation in steps of 1 degree and rotate
c the object, take projection of rotated object.
      do i=1,m
      theta=float(i-1)
      call rotate_2dlinear_sum(x,n,w,m,theta)
      call project(w,z,m)
c Stack projections together to form image of "Radon space"
c data.
      do j=1,m
      y(j,i)=z(j)
      enddo
      enddo
      return
      end

```

```

subroutine rtr divide 2dlinear_sum(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
c *****
C Function: This subroutine computes the Radon Transform of
c      n X n image.
c External subroutines: Rotate_nearest_neig
c                      Project
c Internal subroutines: None
c Parameters: x - n X n input array
c             y - m X m output array
c *****
c Send message to user 'Computing Radon Transform...'
c
      write(6,1)
1      format(1x,'Computing Radon Transform...'$)
c initialize array y
      do j=1,m
      do i=1,m
      y(j,i)=0.
      enddo
      enddo
c Compute angle of rotation in steps of 1 degree and rotate
c the object, take projection of rotated object.
      do i=1,m
      theta=float(i-1)
      call rotate_divide_2dlinear_sum(x,n,w,m,theta)
      call project(w,z,m)
c Stack projections together to form image of "Radon space"
c data.
      do j=1,m
      y(j,i)=z(j)
      enddo
      enddo
      return
      end

```

```

      subroutine rotate_rectangle_sum(a,n,b,m,theta)
      real a(256,256),b(256,256),ss(256,256),w(256,256)
C FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
C            THROUGH AN ANGLE THETA (given in degree)
C            anticlockwise placing the result at the
C            center of an array mxm b
C
C      convert theta into radians
C
C            pi = 4. * atan(1.)
C            theta = theta * (pi/180.)
C            costh = cos(theta)
C            sinth = sin(theta)
C Zero pad image a with aint((m-n+1)/2) storing the result in the array ss.
C The function aint is used in the case of (m-n+1) is even
C            nw=m-n+1
C            ml=m+nw-1
C            iframe=aint(float(m-n)/2.)
C      initial ss
C            do i=1,ml
C            do j=1,ml
C            ss(j,i)=0.
C            enddo
C            enddo
C            do i=1,n
C            do j=1,n
C            ss(j+iframe,i+iframe)=a(j,i)
C            enddo
C            enddo
C      initial b
C            do i = 1,m
C            do j = 1,m
C            b(j,i) = 0.
C            w(j,i) = 0.
C            enddo
C            enddo
C
C      compute new coordinates (xd,yd) of rotate array
C
C            nn = m/2 + 1
C            mm = ml/2 + 1
C Compensation of the center position for rotation.
C            xcomp=nn+nn*(sinth - costh)
C            ycomp=nn-nn*(sinth + costh)
C
C      start process
C
C            do j = 1,m
C            do i = 1,m
C
C      generate 4-quadrant cartesian plan
C            y = float(j-nn)
C            x = float(i-nn)
C            xd = x * costh + y * sinth
C            yd = -x * sinth + y * costh
C
C      compute both real and integer coordinates of array b
C      associated with the above transformation for later use
C            ax =nn+ xd
C            ay = nn+ yd
C      assign value of a at (j,i) to 9-point cell
C      of array b in wich the transformed coordinates
C      lie.
C            fx=(ax-aint(ax))
C            fy=(ay-aint(ay))
C            kx=aint(ax)
C            ky=aint(ay)
C            dl=(1.-fx)*(1.-fy)*ss(ky,kx)

```

```
d2=fx*fy* ss(ky+1,kx+1)
d3=fy*(1.-fx)*ss(ky+1,kx)
d4=fx*(1.-fy)*ss(ky,kx+1)
```

```
b(j,i)=d1+d2+d3+d4
```

```
enddo
enddo
return
end
```

```

subroutine rotate_triangle_sum(a,n,b,m,theta)
real a(256,256),b(256,256),ss(256,256),w(256,256)
c FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
c           THROUGH AN ANGLE THETA (given in degree)
c           anticlockwise placing the result at the
c           center of an array mxm b
c
c           convert theta into radians
c
c           pi = 4. * atan(1.)
c           theta = theta * (pi/180.)
c           costh = cos(theta)
c           sinth = sin(theta)
c Zero pad image a with aint((m-n+1)/2) storing the result in the array ss.
c The function aint is used in the case of (m-n+1) is even
c           nw=m-n+1
c           ml=m+nw-1
c           iframe=aint(float(m-n)/2.)
c initial ss
c           do i=1,ml
c           do j=1,ml
c           ss(j,i)=0.
c           enddo
c           enddo
c           do i=1,n
c           do j=1,n
c           ss(j+iframe,i+iframe)=a(j,i)
c           enddo
c           enddo
c initial b
c           do i = 1,m
c           do j = 1,m
c           b(j,i) = 0.
c           w(j,i) = 0.
c           enddo
c           enddo
c
c compute new coordinates (xd,yd) of rotate array
c
c           nn = m/2 + 1
c           nm = ml/2 + 1
c Compensation of the center position for rotation.
c           xcomp=nn+nn*(sinth - costh)
c           ycomp=nn-nn*(sinth + costh)
c
c start process
c
c           do j = 1,m
c           do i = 1,m
c
c generate 4-quadrant cartesian plan
c           y = float(j-nn)
c           x = float(i-nn)
c           xd = x * costh + y * sinth
c           yd = -x * sinth + y * costh
c
c compute both real and integer coordinates of array b
c associated with the above transformation for later use
c           ax =nn+ xd
c           ay = nn+ yd
c assign value of a at (j,i) to 9-point cell
c of array b in wich the transformed coordinates
c lie.
c           fx=ax-aint(ax)
c           fy=ay-aint(ay)
c           kx=aint(ax)
c           ky=aint(ay)
c

```

```
d1=(1.-fy)*ss(ky,kx)
d2=fy* ss(ky+1,kx+1)
d3=(1.-fx)*ss(ky+1,kx)
d4=fx*ss(ky,kx+1)
```

```
b(j,i)=(d1+d2+d3+d4)*.5
```

```
enddo
enddo
return
end
```



```

subroutine rotate divide rectangle sum(a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
C FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
C            THROUGH AN ANGLE THETA (given in degree)
C            anticlockwise placing the result at the
C            center of an array mxm b
C
C            convert theta into radians
C
C            pi = 4. * atan(1.)
C            theta = theta * (pi/180.)
C            costh = cos(theta)
C            sinth = sin(theta)
C
C            initial b
C
C            do i = 1,m
C            do j = 1,m
C            b(j,i) = 0.
C            w(j,i) = 0.
C            enddo
C            enddo
C
C            compute new coordinates (xd,yd) of rotate array
C
C            nn = n/2 + 1
C            mm = m/2 + 1
C            Compensation of the center position for rotation.
C            xcomp=nn+nn*(sinth - costh)
C            ycomp=nn-nn*(sinth + costh)
C
C            start process
C
C            do j = 1,m
C            do i = 1,m
C
C            generate 4-quadrant cartesian plan
C            y = float(j-nn)
C            x = float(i-nn)
C            xd = x * costh - y * sinth
C            yd = x * sinth + y * costh
C
C            compute both real and integer coordinates of array b
C            associated with the above transformation for later use
C            ax = mm+ xd
C            ay = mm+ yd
C            assign value of a at (j,i) to 9-point cell
C            of array b in wich the transformed coordinates
C            lie.
C
C            fx=(ax-aint(ax))
C            fy=(ay-aint(ay))
C            kx=aint(ax)
C            ky=aint(ay)
C            d1=(1.-fx)*(1.-fy)*a(j,i)
C            d2=fy*(1.-fx)*a(j,i)
C            d3=fx*(1.-fy)*a(j,i)
C            d4=fx*fy*a(j,i)
C
C            b(ky,kx)=b(ky,kx)+d1
C            b(ky+1,kx)=b(ky+1,kx)+d2
C            b(ky,kx+1)=b(ky,kx+1)+d3
C            b(ky+1,kx+1)=b(ky+1,kx+1)+d4
C
C            enddo
C            enddo
C            return
C            end

```

```

subroutine divide triangle sum(a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
C FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
C           THROUGH AN ANGLE THETA (given in degree)
C           anticlockwise placing the result at the
C           center of an array mxm b
C
C           convert theta into radians
C
C           pi = 4. * atan(1.)
C           theta = theta * (pi/180.)
C           costh = cos(theta)
C           sinth = sin(theta)
C
C           initial b
C
C           do i = 1,m
C           do j = 1,m
C           b(j,i) = 0.
C           w(j,i) = 0.
C           enddo
C           enddo
C
C           compute new coordinates (xd,yd) of rotate array
C
C           nn = n/2 + 1
C           mm = m/2 + 1
C           Compensation of the center position for rotation.
C           xcomp=nn+nn*(sinth - costh)
C           ycomp=nn-nn*(sinth + costh)
C
C           start process
C
C           do j = 1,m
C           do i = 1,m
C
C           generate 4-quadrant cartesian plan
C           y = float(j-nn)
C           x = float(i-nn)
C           xd = x * costh - y * sinth
C           yd = x * sinth + y * costh
C
C           compute both real and integer coordinates of array b
C           associated with the above transformation for later use
C           ax =mm+ xd
C           ay = mm+ yd
C           assign value of a at (j,i) to 9-point cell
C           of array b in wich the transformed coordinates
C           lie.
C           fx=.5*(ax-aint(ax))
C           fy=.5*(ay-aint(ay))
C           kx=aint(ax)
C           ky=aint(ay)
C           d1=(.5-fy)*a(j,i)
C           d2=(.5-fx)*a(j,i)
C           d3=fx*a(j,i)
C           d4=fy*a(j,i)
C
C           b(ky,kx)=b(ky,kx)+d1
C           b(ky+1,kx)=b(ky+1,kx)+d2
C           b(ky,kx+1)=b(ky,kx+1)+d3
C           b(ky+1,kx+1)=b(ky+1,kx+1)+d4
C
C           enddo
C           enddo
C           return
C           end

```

```

subroutine rotate 2dlinear sum(a,n,b,m,theta)
real a(256,256),b(256,256),ss(256,256),w(256,256)
c FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
c THROUGH AN ANGLE THETA (given in degree)
c anticlockwise placing the result at the
c center of an array mxm b
c
c convert theta into radians
c
c pi = 4. * atan(1.)
c theta = theta * (pi/180.)
c costh = cos(theta)
c sinth = sin(theta)
c Zero pad image a with aint((m-n+1)/2) storing the result in the array ss.
c The function aint is used in the case of (m-n+1) is even
c nw=m-n+1
c ml=m+nw-1
c iframe=aint(float(m-n)/2.)
c initial ss
c do i=1,ml
c do j=1,ml
c ss(j,i)=0.
c enddo
c enddo
c do i=1,n
c do j=1,n
c ss(j+iframe,i+iframe)=a(j,i)
c enddo
c enddo
c initial b
c do i = 1,m
c do j = 1,m
c b(j,i) = 0.
c w(j,i) =0.
c enddo
c enddo
c compute new coordinates (xc,yd) of rotate array
c
c nn = m/2 + 1
c mm = ml/2 + 1
c Compensation of the center position for rotation.
c xcomp=nn+nn*(sinth - costh)
c ycomp=nn-nn*(sinth + costh)
c start process
c
c do j = 1,m
c do i = 1,m
c
c generate 4-quadrant cartesian plan
c y = float(j-nn)
c x = float(i-nn)
c xd = x * costh + y * sinth
c yd = -x * sinth + y * costh
c
c compute both real and integer coordinates of array b
c associated with the above transformation for later use
c ax =nn+ xd
c ay = nn+ yd
c assign value of a at (j,i) to 9-point cell
c of array b in wich the transformed coordinates
c lie.
c
c kx=aint(ax)
c ky=aint(ay)
c xm1=sqrt((ky-ay)**2.+(kx-ax)**2.)
c xm4=sqrt((ky+1.-ay)**2.+(kx+1.-ax)**2.)
c xm2=sqrt((ky+1.-ay)**2.+(kx-ax)**2.)

```

```
xm3=sqrt((ky-ay)**2.+(kx+1.-ax)**2.)
d1=xm4*ss(ky,kx)
d2=xm1* ss(ky+1,kx+1)
d3=xm3*ss(ky+1,kx)
d4=xm2*ss(ky,kx+1)

b(j,i)=d1+d2+d3+d4

enddo
enddo
return
end
```

```

      subroutine divide 2dlinear sum(a,n,b,m,theta)
      real a(256,256),b(256,256),w(256,256)
C FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
C            THROUGH AN ANGLE THETA (given in degree)
C            anticlockwise placing the result at the
C            center of an array mxm b
C
      convert theta  into radians
C
      pi = 4. * atan(1.)
      theta = theta * (pi/180.)
      costh = cos(theta)
      sinth = sin(theta)
C
C  initial b
      do i = 1,m
      do j = 1,m
      b(j,i) = 0.
      w(j,i) = 0.
      enddo
      enddo
      sqrt2=sqrt(2.)
C compute new coordinates (xd,yd) of rotate array
C
      nn = n/2 + 1
      mm = m/2 + 1
C Compensation of the center position for rotation.
      xcomp=nn+nn*(sinth - costh)
      ycomp=nn-nn*(sinth + costh)
C start process
C
      do j = 1,m
      do i = 1,m
C generate 4-quadrant cartesian plan
      y = float(j-nn)
      x = float(i-nn)
      xd = x * costh - y * sinth
      yd = x * sinth + y * costh
C compute both real and integer coordinates of array b
C associated with the above transformation for later use
      ax =mm+ xd
      ay = mm+ yd
C assign value of a at (j,i) to 9-point cell
C of array b in wich the transformed coordinates
C lie.
      kx=aint(ax)
      ky=aint(ay)
      xm1=sqrt((ky-ay)**2.+(kx-ax)**2.)
      xm4=sqrt((ky+1.-ay)**2.+(kx+1.-ax)**2.)
      xm2=sqrt((ky+1.-ay)**2.+(kx-ax)**2.)
      xm3=sqrt((ky-ay)**2.+(kx+1.-ax)**2.)
      d1=xm4*a(j,i)
      d2=xm3*a(j,i)
      d3=xm2*a(j,i)
      d4=xm1*a(j,i)
C
      b(ky,kx)=b(ky,kx)+d1
      b(ky+1,kx)=b(ky+1,kx)+d2
      b(ky,kx+1)=b(ky,kx+1)+d3
      b(ky+1,kx+1)=b(ky+1,kx+1)+d4
C
      enddo
      enddo
      return
      end

```

```

      subroutine intr_rectangle_sum(x,m,y,n)
      real x(256,256),y(256,256),xx(256,256)
c *****
c FUNCTION: This subroutine computes the inverse radon
c           transform by:
c           1. Back_projecting all the projections stored
c              in array x giving the back_projection
c              function y.
c           2. Deconvolving the array y using the inverse_
c              filter SQRT(kx*kx + ky*ky*).
c External subroutines: bpr_rectangle_sum
c                      back_projection_deconvolution
c Internal subroutines: none
c *****
c Filtering Radon Transform.
c Send message to user ' Filtering Radon Transform...'
c
c       write(6,1)
c       format(1x,'Filtering Radon Transform...'$)
c       call projection_filtering(x,m)
c Send message to user 'Computing Inverse Radon Transform...'
c
c       write(6,2)
c       format(1x,'Computing Inverse Radon Transform...'$)
c Initialize array xx
c       do i=1,n
c       do j=1,n
c       xx(j,i)=0.
c       enddo
c       enddo
c
c Back project projections stored in x
c
c       call bpr_rectangle_sum(x,m,y,n)
c
c       return
c       end

```

```

subroutine irtr_triangle_sum(x,m,y,n)
real x(256,256),y(256,256),xx(256,256)
c *****
c FUNCTION: This subroutine computes the inverse radon
c transform by:
c     1. Back_projecting all the projections stored
c        in array x giving the back_projection
c        function y.
c     2. Deconvolving the array y using the inverse_
c        filter SQRT(kx*kx + ky*ky*).
c External subroutines: bpr_rectangle_sum
c                      back_projection_deconvolution
c Internal subroutines: none
c *****
c Filtering Radon Transform.
c Send message to user ' Filtering Radon Transform...'
c
c     write(6,1)
c     format(1x,'Filtering Radon Transform...'$)
c     call projection_filtering(x,m)
c Send message to user 'Computing Inverse Radon Transform...'
c
c     write(6,2)
c     format(1x,'Computing Inverse Radon Transform...'$)
c Initialize array xx
c     do i=1,n
c     do j=1,n
c     xx(j,i)=0.
c     enddo
c     enddo
c
c Back project projections stored in x
c
c     call bpr_triangle_sum(x,m,y,n)
c
c     return
c     end

```

```

subroutine irtr_divide_rectangle_sum(x,m,y,n)
real x(256,256),y(256,256),xx(256,256)
c *****
c FUNCTION: This subroutine computes the inverse radon
c transform by:
c     1. Back_projecting all the projections stored
c        in array x giving the back_projection
c        function y.
c     2. Deconvolving the array y using the inverse_
c        filter SQRT(kx*kx + ky*ky*).
c External subroutines: bpr_nearest_neig
c                      back_projection_deconvolution
c Internal subroutines: none
c *****
c Filtering Radon Transform.
c Send message to user ' Filtering Radon Transform...'
c
c     write(6,1)
c     format(1x,'Filtering Radon Transform...'$)
c     call projection_filtering(x,m)
c Send message to user 'Computing Inverse Radon Transform...'
c
c     write(6,2)
c     format(1x,'Computing Inverse Radon Transform...'$)
c Initialize array xx
c     do i=1,n
c     do j=1,n
c     xx(j,i)=0.
c     enddo
c     enddo
c
c Back project projections stored in x
c
c     call bpr_divide_rectangle_sum(x,m,y,n)
c
c     return
c     end

```



```

      subroutine irtr_divide_triangle_sum(x,m,y,n)
      real x(256,256),y(256,256),xx(256,256)
c *****
c FUNCTION: This subroutine computes the inverse radon
c           transform by:
c           1. Back_projecting all the projections stored
c              in array x giving the back_projection
c              function y.
c           2. Deconvolving the array y using the inverse_
c              filter SQRT(kx*kx + ky*ky*).
c External subroutines: bpr_nearest_neig
c                       back_projection_deconvolution
c Internal subroutines: none
c *****
c Filtering Radon Transform.
c Send message to user ' Filtering Radon Transform...'
c
      write(6,1)
1      format(1x,'Filtering Radon Transform...'$)
      call projection_filtering(x,m)
c Send message to user 'Computing Inverse Radon Transform...'
c
      write(6,2)
2      format(1x,'Computing Inverse Radon Transform...'$)
c Initialize array xx
      do i=1,n
      do j=1,n
      xx(j,i)=0.
      enddo
      enddo
c
c Back project projections stored in x
c
      call bpr_divide_triangle_sum(x,m,y,n)
c
      return
      end

```

```

subroutine irtr_2dlinear_sum(x,m,y,n)
real x(256,256),y(256,256),xx(256,256)
c *****
c FUNCTION: This subroutine computes the inverse radon
c transform by:
c     1. Back_projecting all the projections stored
c        in array x giving the back_projection
c        function y.
c     2. Deconvolving the array y using the inverse_
c        filter SQRT(kx*kx + ky*ky*).
c External subroutines: bpr_2dlinear_sum
c                      back_projection_deconvolution
c Internal subroutines: none
c *****
c Filtering Radon Transform.
c Send message to user ' Filtering Radon Transform...'
c
c     write(6,1)
c     format(1x,'Filtering Radon Transform...'$)
c     call projection_filtering(x,m)
c Send message to user 'Computing Inverse Radon Transform...'
c
c     write(6,2)
c     format(1x,'Computing Inverse Radon Transform...'$)
c Initialize array xx
c     do i=1,n
c     do j=1,n
c     xx(j,i)=0.
c     enddo
c     enddo
c
c Back project projections stored in x
c
c     call bpr_2dlinear_sum(x,m,y,n)
c
c     return
c     end

```

```

subroutine irtr_divide_2dlinear_sum(x,m,y,n)
real x(256,256),y(256,256),xx(256,256)
c *****
c FUNCTION: This subroutine computes the inverse radon
c transform by:
c     1. Back_projecting all the projections stored
c        in array x giving the back_projection
c        function y.
c     2. Deconvolving the array y using the inverse_
c        filter SQRT(kx*kx + ky*ky*).
c External subroutines: bpr_nearest_neig
c                      back_projection_deconvolution
c Internal subroutines: none
c *****
c Filtering Radon Transform.
c Send message to user ' Filtering Radon Transform...'
c
c     write(6,1)
c     format(1x,'Filtering Radon Transform...')
c     call projection_filtering(x,m)
c Send message to user 'Computing Inverse Radon Transform...'
c
c     write(6,2)
c     format(1x,'Computing Inverse Radon Transform...')
c Initialize array xx
c     do i=1,n
c     do j=1,n
c     xx(j,i)=0.
c     enddo
c     enddo
c
c Back project projections stored in x
c
c     call bpr_divide_2dlinear_sum(x,m,y,n)
c
c     return
c     end

```

subroutine projection_filtering(a,n)

c
c Function: This subroutine filters a Radon Transform
c obtained by straight projection (filtered)
c using the appropriate inverse filter, i.e.
c ABS(kx) where kx is the spatial frequency.
c

real a(256,256),y(256,256),c(256,256),za(256)
real zb(256),zc(256),zd(256)

c
c initialize array c
c

do i=1,n
do j=1,n
c(j,i)=0.
enddo
enddo
do j=1,256
za(j)=0.
zb(j)=0.
zc(j)=0.
zd(j)=0.
enddo

c
c take Fourier transform of a
c

nn=256/2+1
jframe=nint(float(256-n)/2.)
do i=1,n
do j=1,n
za(j+jframe)=a(j,i)
enddo
call fftld(za,zb,256,-1)

c
c multiply spectrum by inverse filter
c

do j=1,256
xk=abs(float(j-nn)/float(256))
zc(j)=za(j)*xk
zd(j)=zb(j)*xk
enddo

c
c call fftld(zc,zd,256,1)
c

do j=1,n
c(j,i)=zc(j+jframe)
enddo
enddo

c write output
c

do i=1,n
do j=1,n
a(j,i)=c(j,i)
enddo
enddo

c
c return
c end

```

      subroutine project(a,b,n)
c *****
c Function: This subroutine sums
c           over all rows of an n X n array a giving 1D
c projection of size n _ array b
c *****
      real a(256,256),b(256),s(256,256),z(256)
c initialize arrays s & b
      do j=1,n
        b(j)=0.
        z(j)=0.
        do i=1,n
          s(j,i)=0.
        enddo
      enddo
c set initial value for summing process and sum all pixels along
c row i together
      do j=1,n
        s(j,1)=a(j,1)
        do i=2,n
          s(j,i)=s(j,i-1)+a(j,i)
        enddo
c Assign value of sum at column j to output _b(j)
        b(j)=s(j,n)
      enddo
c
      return
      end

```

```

      subroutine bpr_rectangle_sum(b,m,a,n)
      real a(256,256),b(256,256),xx(256,256)
c *****
c Function: this routine back-projects the
c           data stored in b to reconstruct
c           the image a.
c External subroutines: Rotate_rectangle_sum
c Internal subroutines: None
c Parameters: b _ input, size m X m
c             a _ output, size n X n
c *****
c x, y and sum are working space
      real x(256,256),y(256,256)
      real sum(256,256)
c Send message to user 'Computing back_projection ...'
c
      write(6,1)
1      format(1x,'Computing back_projection...'$)
c Compute size of image in which b will be rotated to prevent
c corner clipping.
      l=nint(sqrt(2.)*float(m))
c Compute centers of arrays to be used later
      nn=n/2+1
      ll=l/2+1
      ln=ll-nn
c Initialize working space
      do i=1,l
      do j=1,l
      x(j,i)=0.
      y(j,i)=0.
      sum(j,i)=0.
      a(j,i)=0.
      xx(j,i)=0.
      enddo
      enddo
c Compute the angle of rotation in the reverse direction
      do k=1,m
      theta=float(1-k)
c Back_project projection
      do i=1,m
      do j=1,m
      x(j,i)=b(j,k)
      enddo
      enddo
c Rotate x by angle at which projection was taken
      call rotate_rectangle_sum(x,m,y,l,theta)
c Add result to previous back_projection function
      do i=1,l
      do j=1,l
      sum(j,i)=sum(j,i)+y(j,i)
      enddo
      enddo
c
c
      enddo
c Window the back_projection function to extract reconstruction
c (size n X n) and write it to output
      do i=1,n
      do j=1,n
      a(j,i)=sum(j+ln,i+ln)
      enddo
      enddo
c
      return
      end

```

```

subroutine bpr_triangle sum(b,m,a,n)
  real a(256,256),b(256,256),xx(256,256)
c *****
c Function: this routine back-projects the
c           data stored in b to reconstruct
c           the image a.
c External subroutines: 30:Rotation_techniques
c Internal subroutines: None
c Parameters: b _ input, size m X m
c             a _ output, size n X n
c *****
c x, y and sum are working space
  real x(256,256),y(256,256)
  real sum(256,256)
c Send message to user 'Computing back_projection ...'
c
  write(6,1)
1  format(1x,'Computing back_projection...'$)
c Compute size of image in which b will be rotated to prevent
c corner clipping.
  l=nint(sqrt(2.)*float(m))
c Compute centers of arrays to be used later
  nn=n/2+1
  ll=l/2+1
  ln=ll-nn
c Initialize working space
  do i=1,l
  do j=1,l
    x(j,i)=0.
    y(j,i)=0.
    sum(j,i)=0.
    a(j,i)=0.
    xx(j,i)=0.
  enddo
  enddo
c Compute the angle of rotation in the reverse direction
  do k=1,m
    theta=float(1-k)
c Back_project projection
  do i=1,m
  do j=1,m
    x(j,i)=b(j,k)
  enddo
  enddo
c Rotate x by angle at which projection was taken
  call rotate_triangle_sum(x,m,y,l,theta)
c Add result to previous back_projection function
  do i=1,l
  do j=1,l
    sum(j,i)=sum(j,i)+y(j,i)
  enddo
  enddo
c
c
  enddo
c Window the back_projection function to extract reconstruction
c (size n X n) and write it to output
  do i=1,n
  do j=1,n
    a(j,i)=sum(j+ln,i-ln)
  enddo
  enddo
c
  return
end

```

```

subroutine bpr_divide_rectangle_sum(b,m,a,n)
  real a(256,256),b(256,256),xx(256,256)
c *****
c Function: this routine back-projects the
c           data stored in b to reconstruct
c           the image a.
c External subroutines: Rotate_nearest_neig
c Internal subroutines: None
c Parameters: b _ input, size m X m
c             a _ output, size n X n
c *****
c x, y and sum are working space
  real x(256,256),y(256,256)
  real sum(256,256)
c Send message to user 'Computing back_projection ...'
c
  write(6,1)
1  format(1x,'Computing back_projection...'$)
c Compute size of image in which b will be rotated to prevent
c corner clipping.
  l=nint(sqrt(2.)*float(m))
c Compute centers of arrays to be used later
  nn=n/2+1
  ll=l/2+1
  ln=ll-nn
c Initialize working space
  do i=1,l
  do j=1,l
    x(j,i)=0.
    y(j,i)=0.
    sum(j,i)=0.
    a(j,i)=0.
    xx(j,i)=0.
  enddo
  enddo
c Compute the angle of rotation in the reverse direction
  do k=1,m
    theta=float(1-k)
c Back_project projection
  do i=1,m
  do j=1,m
    x(j,i)=b(j,k)
  enddo
  enddo
c Rotate x by angle at which projection was taken
  call rotate_divide_rectangle_sum(x,m,y,l,theta)
c Add result to previous back_projection function
  do i=1,l
  do j=1,l
    sum(j,i)=sum(j,i)+y(j,i)
  enddo
  enddo
c
c
  enddo
c Window the back_projection function to extract reconstruction
c (size n X n) and write it to output
  do i=1,n
  do j=1,n
    a(j,i)=sum(j+ln,i+ln)
  enddo
  enddo
c
  return
end

```



```

subroutine bpr_divide_triangle sum(b,m,a,n)
  real a(256,256),b(256,256),xx(256,256)
  *****
Function: this routine back-projects the
         data stored in b to reconstruct
         the image a.
External subroutines: Rotate_nearest_neig
Internal subroutines: None
Parameters: b _ input, size m X m
           a _ output, size n X n
  *****
x, y and sum are working space
  real x(256,256),y(256,256)
  real sum(256,256)
Send message to user 'Computing back_projection ...'

  write(6,1)
  format(1x,'Computing back_projection...'$)

Compute size of image in which b will be rotated to prevent
corner clipping.
  l=nint(sqrt(2.)*float(m))
Compute centers of arrays to be used later
  nn=n/2+1
  ll=l/2+1
  ln=ll-nn
Initialize working space
  do i=1,l
  do j=1,l
    x(j,i)=0.
    y(j,i)=0.
    sum(j,i)=0.
    a(j,i)=0.
    xx(j,i)=0.
  enddo
  enddo
Compute the angle of rotation in the reverse direction
  do k=1,m
    theta=float(1-k)
Back_project projection
  do i=1,m
  do j=1,m
    x(j,i)=b(j,k)
  enddo
  enddo
Rotate x by angle at which projection was taken
  call rotate_divide_triangle_sum(x,m,y,l,theta)
Add result to previous back_projection function
  do i=1,l
  do j=1,l
    sum(j,i)=sum(j,i)+y(j,i)
  enddo
  enddo

  enddo

Window the back_projection function to extract reconstruction
(size n X n) and write it to output
  do i=1,n
  do j=1,n
    a(j,i)=sum(j+ln,i+ln)
  enddo
  enddo

  return
end

```

```

subroutine bpr 2dlinear sum(b,m,a,n)
  real a(256,256),b(256,256),xx(256,256)
c *****
c Function: this routine back-projects the
c           data stored in b to reconstruct
c           the image a.
c External subroutines: Rotate_2dlinear_sum
c Internal subroutines: None
c Parameters: b _ input, size m X m
c              a _ output, size n X n
c *****
c x, y and sum are working space
  real x(256,256),y(256,256)
  real sum(256,256)
c Send message to user 'Computing back_projection ...'
c
  write(6,1)
1  format(1x,'Computing back_projection...'$)
c Compute size of image in which b will be rotated to prevent
c corner clipping.
  l=nint(sqrt(2.)*float(m))
c Compute centers of arrays to be used later
  nn=n/2+1
  ll=l/2+1
  ln=ll-nn
c Initialize working space
  do i=1,l
  do j=1,l
    x(j,i)=0.
    y(j,i)=0.
    sum(j,i)=0.
    a(j,i)=0.
    xx(j,i)=0.
  enddo
  enddo
c Compute the angle of rotation in the reverse direction
  do k=1,m
    theta=float(1-k)
c Back_project projection
  do i=1,m
  do j=1,m
    x(j,i)=b(j,k)
  enddo
  enddo
c Rotate x by angle at which projection was taken
  call rotate_2dlinear_sum(x,m,y,l,theta)
c Add result to previous back_projection function
  do i=1,l
  do j=1,l
    sum(j,i)=sum(j,i)+y(j,i)
  enddo
  enddo
c
c
  enddo
c Window the back_projection function to extract reconstruction
c (size n X n) and write it to output
  do i=1,n
  do j=1,n
    a(j,i)=sum(j+ln,i+ln)
  enddo
  enddo
c
  return
end

```

```

subroutine bpr divide_2dlinear sum(b,m,a,n)
  real a(256,256),b(256,256),xx(256,256)
c *****
c Function: this routine back-projects the
c           data stored in b to reconstruct
c           the image a.
c External subroutines: Rotate_nearest_neig
c Internal subroutines: None
c Parameters: b _ input, size m X m
c             a _ output, size n X n
c *****
c x, y and sum are working space
  real x(256,256),y(256,256)
  real sum(256,256)
c Send message to user 'Computing back_projection ...'
c
  write(6,1)
1  format(1x,'Computing back_projection...'$)
c Compute size of image in which b will be rotated to prevent
c corner clipping.
  l=nint(sqrt(2.)*float(m))
c Compute centers of arrays to be used later
  nn=n/2+1
  ll=l/2+1
  ln=ll-nn
c Initialize working space
  do i=1,l
  do j=1,l
  x(j,i)=0.
  y(j,i)=0.
  sum(j,i)=0.
  a(j,i)=0.
  xx(j,i)=0.
  enddo
  enddo
c Compute the angle of rotation in the reverse direction
  do k=1,m
  theta=float(1-k)
c Back_project projection
  do i=1,m
  do j=1,m
  x(j,i)=b(j,k)
  enddo
  enddo
c Rotate x by angle at which projection was taken
  call rotate_divide_2dlinear_sum(x,m,y,l,theta)
c Add result to previous back_projection function
  do i=1,l
  do j=1,l
  sum(j,i)=sum(j,i)+y(j,i)
  enddo
  enddo
c
c
  enddo
c Window the back_projection function to extract reconstruction
c (size n X n) and write it to output
  do i=1,n
  do j=1,n
  a(j,i)=sum(j+ln,i+ln)
  enddo
  enddo
c
  return
end

```

```

subroutine rtr_rectangle_12neigh_sum(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
c *****
C Function: This subroutine computes the Radon Transform of
c      n X n image.
c External subroutines: Rotate_nearest_neig
c                      Project
c Internal subroutines: None
c Parameters: x - n X n input array
c             y - m X m output array
c *****
c Send message to user 'Computing Radon Transform...'
c
      write(6,1)
1      format(1x,'Computing Radon Transform...'$)
c initialize array y
      do j=1,m
      do i=1,m
      y(j,i)=0.
      enddo
      enddo
c Compute angle of rotation in steps of 1 degree and rotate
c the object, take projection of rotated object.
      do i=1,m
      theta=float(i-1)
      call rotate_rectangle_12neigh_sum(x,n,w,m,theta)
      call project(w,z,m)
c Stack projections together to form image of "Radon space"
c data.
      do j=1,m
      y(j,i)=z(j)
      enddo
      enddo
      return
      end

```

```

subroutine rtr divide_rectangle12n_sum(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
c *****
C Function: This subroutine computes the Radon Transform of
c      n X n image.
c External subroutines: Rotate_nearest_neig
c                      Project
c Internal subroutines: None
c Parameters: x - n X n input array
c             y - m X m output array
c *****
c Send message to user 'Computing Radon Transform...'
c
      write(6,1)
1      format(1x,'Computing Radon Transform...'$)
c initialize array y
      do j=1,m
      do i=1,m
      y(j,i)=0.
      enddo
      enddo
c Compute angle of rotation in steps of 1 degree and rotate
c the object, take projection of rotated object.
      do i=1,m
      theta=float(i-1)
      call rotate_divide_rectangle12n_sum(x,n,w,m,theta)
      call project(w,z,m)
c Stack projections together to form image of "Radon space"
c data.
      do j=1,m
      y(j,i)=z(j)
      enddo
      enddo
      return
      end

```

```

subroutine rtr_rectangle_16neigh_sum(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
c *****
C Function: This subroutine computes the Radon Transform of
c      n X n image.
c External subroutines: Rotate_nearest_neig
c                      Project
c Internal subroutines: None
c Parameters: x - n X n input array
c             y - m X m output array
c *****
c Send message to user 'Computing Radon Transform...'
c
      write(6,1)
1      format(1x,'Computing Radon Transform...'$)
c initialize array y
      do j=1,m
      do i=1,m
      y(j,i)=0.
      enddo
      enddo
c Compute angle of rotation in steps of 1 degree and rotate
c the object, take projection of rotated object.
      do i=1,m
      theta=float(i-1)
      call rotate_rectangle_16neigh_sum(x,n,w,m,theta)
      call projec(w,z,m)
c Stack projections together to form image of "Radon space"
c data.
      do j=1,m
      y(j,i)=z(j)
      enddo
      enddo
      return
      end

```

```

subroutine rtr_triangle_12neigh_sum(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
c *****
C Function: This subroutine computes the Radon Transform of
c      n X n image.
c External subroutines: Rotate_triangle_16neigh_sum
c                      Project
c Internal subroutines: None
c Parameters: x - n X n input array
c             y - m X m output array
c *****
c Send message to user 'Computing Radon Transform...'
c
      write(6,1)
      format(1x,'Computing Radon Transform...'$)
c initialize array y
      do j=1,m
      do i=1,m
      y(j,i)=0.
      enddo
      enddo
c Compute angle of rotation in steps of 1 degree and rotate
c the object, take projection of rotated object.
      do i=1,m
      theta=float(i-1)
      call rotate_triangle_12neigh_sum(x,n,w,m,theta)
      call project(w,z,m)
c Stack projections together to form image of "Radon space"
c data.
      do j=1,m
      y(j,i)=z(j)
      enddo
      enddo
      return
      end

```

```

subroutine rtr_triangle_16neigh_sum(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
c *****
c Function: This subroutine computes the Radon Transform of
c           n X n image.
c External subroutines: Rotate_triangle_16neigh_sum
c                       Project
c Internal subroutines: None
c Parameters: x - n X n input array
c              y - m X m output array
c *****
c Send message to user 'Computing Radon Transform...'
c
      write(6,1)
1      format(1x,'Computing Radon Transform...'$)
c initialize array y
      do j=1,m
      do i=1,m
      y(j,i)=0.
      enddo
      enddo
c Compute angle of rotation in steps of 1 degree and rotate
c the object, take projection of rotated object.
      do i=1,m
      theta=float(i-1)
      call rotate_triangle_16neigh_sum(x,n,w,m,theta)
      call project(w,z,m)
c Stack projections together to form image of "Radon space"
c data.
      do j=1,m
      y(j,i)=z(j)
      enddo
      enddo
      return
      end

```



```

subroutine rtr_triangle_16neigh_sum1(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
c *****
C Function: This subroutine computes the Radon Transform of
c      n X n image.
c External subroutines: Rotate_triangle_16neigh_sum
c                      Project
c Internal subroutines: None
c Parameters: x - n X n input array
c             y - m X m output array
c *****
c Send message to user 'Computing Radon Transform...'
c
      write(6,1)
1      format(1x,'Computing Radon Transform...'$)
c initialize array y
      do j=1,m
      do i=1,m
      y(j,i)=0.
      enddo
      enddo
c Compute angle of rotation in steps of 1 degree and rotate
c the object, take projection of rotated object.
      do i=1,m
      theta=float(i-1)
      call rotate_triangle_16neigh_sum1(x,n,w,m,theta)
      call project(w,z,m)
c Stack projections together to form image of "Radon space"
c data.
      do j=1,m
      y(j,i)=z(j)
      enddo
      enddo
      return
      end

```

```

subroutine rotate_rectangle_12neigh sum(a,n,b,m,theta)
real a(256,256),b(256,256),ss(256,256),w(256,256)
FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
           THROUGH AN ANGLE THETA (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b

convert theta into radians

pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)

Zero pad image a with aint((m-n+1)/2) storing the result in the array ss.
The function aint is used in the case of (m-n+1) is even
nw=m-n+1
ml=m+nw-1
iframe=aint(float(m-n)/2.)

initial ss
do i=1,ml
doj=1,ml
ss(j,i)=0.
enddo
enddo
do i=1,n
do j=1,n
ss(j+iframe,i+iframe)=a(j,i)
enddo
enddo

initial b
do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) = 0.
enddo
enddo

compute new coordinates (xd,yd) of rotate array

nn = m/2 + 1
mm = ml/2 + 1
Compensation of the center position for rotation.
xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)

start process

do j = 1,m
do i = 1,m

generate 4-quadrant cartesian plan
y = float(j-nn)
x = float(i-nn)
xd = x * costh + y * sinth
yd = -x * sinth + y * costh

compute both real and integer coordinates of array b
associated with the above transformation for later use
ax = nn+ xd
ay = nn+ yd

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.
fx=(ax-aint(ax))
fy=(ay-aint(ay))
kx=aint(ax)
ky=aint(ay)
dl=(1.+fy)*(1.-fx)/7.

```

```
d2=(2.-fx)*(1.-fy)/7.  
d3=(2.-fy)*fx/7.  
d4=fy*(1.+fx)/7.  
d5=(1.-fy)*(1.-fx)/7.  
d6=(1.-fy)*fx/7.  
d7=fy*fx/7.  
d8=fy*(1.-fx)/7.  
d9=fy*(2.-fx)/7.  
d10=(2.-fy)*(1.-fx)/7.  
d11=(1.+fx)*(1.-fy)/7.  
d12=(1.+fy)*fx/7.
```

```
e1=d1*ss(ky+1,kx)  
e2=d2* ss(ky,kx)  
e3=d3*ss(ky,kx+1)  
e4=d4*ss(ky+1,kx+1)  
e5=d5*ss(ky-1,kx)  
e6=d6* ss(ky,kx+2)  
e7=d7*ss(ky+2,kx+1)  
e8=d8*ss(ky+1,kx-1)  
e9=d9*ss(ky+2,kx)  
e10=d10* ss(ky,kx-1)  
e11=d11*ss(ky-1,kx+1)  
e12=d12*ss(ky+1,kx+2)
```

```
b(j,i)=e1+e2+e3+e4+e5+e6+e7+e8+e9+e10+e11+e12
```

```
enddo  
enddo  
return  
end
```

```

subroutine divide rectangle12n_sum(a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)

```

```

C FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
C            THROUGH AN ANGLE THETA (given in degree)
C            anticlockwise placing the result at the
C            center of an array mxm b
C
C            convert theta into radians
C
C            pi = 4. * atan(1.)
C            theta = theta * (pi/180.)
C            costh = cos(theta)
C            sinth = sin(theta)
C
C initial ss
C
C            do i=1,m
C            do j=1,m
C            b(j,i)=0.
C            w(j,i)=0.
C            enddo
C            enddo
C
C compute new coordinates (xd,yd) of rotate array
C
C            nn = n/2 + 1
C            mm = m/2 + 1
C Compensation of the center position for rotation.
C            xcomp=nn+nn*(sinth - costh)
C            ycomp=nn-nn*(sinth + costh)
C
C start process
C
C            do j = 1,m
C            do i = 1,m
C
C generate 4-quadrant cartesian plan
C            y = float(j-nn)
C            x = float(i-nn)
C            xd = x * costh - y * sinth
C            yd = x * sinth + y * costh
C
C compute both real and integer coordinates of array b
C associated with the above transformation for later use
C            ax =mm+ xd
C            ay = mm+ yd
C assign value of a at (j,i) to 9-point cell
C of array b in wich the transformed coordinates
C lie.
C            fx=(ax-aint(ax))
C            fy=(ay-aint(ay))
C            kx=aint(ax)
C            ky=aint(ay)
C            d1=(1.+fy)*(1.-fx)/7.
C            d2=(2.-fx)*(1.-fy)/7.
C            d3=(2.-fy)*fx/7.
C            d4=fy*(1.+fx)/7.
C            d5=(1.-fy)*(1.-fx)/7.
C            d6=(1.-fy)*fx/7.
C            d7=fy*fx/7.
C            d8=fy*(1.-fx)/7.
C            d9=fy*(2.-fx)/7.
C            d10=(2.-fy)*(1.-fx)/7.
C            d11=(1.+fx)*(1.-fy)/7.
C            d12=(1.+fy)*fx/7.
C
C            e1=d1*a(j,i)
C            e2=d2*a(j,i)
C            e3=d3*a(j,i)

```

e4=d4*a(j,i)
e5=d1*a(j,i)
e6=d2* a(j,i)
e7=d3*a(j,i)
e8=d4*a(j,i)
e9=d1*a(j,i)
e10=d2* a(j,i)
e11=d3*a(j,i)
e12=d4*a(j,i)
e13=d13*a(j,i)
e14=d14*a(j,i)
e15=d15* a(j,i)
e16=d16*a(j,i)

b(ky,kx)=b(ky,kx)+e2
b(ky,kx+1)=b(ky,kx+1)+e3
b(ky+1,kx+1)=b(ky+1,kx+1)+e4
b(ky+1,kx)=b(ky+1,kx)+e1
b(ky-1,kx)=b(ky-1,kx)+e5
b(ky,kx+2)=b(ky,kx+2)+e6
b(ky+2,kx+1)=b(ky+2,kx+1)+e7
b(ky+1,kx-1)=b(ky+1,kx-1)+e8
b(ky-1,kx+1)=b(ky-1,kx+1)+e11
b(ky+1,kx+2)=b(ky+1,kx+2)+e12
b(ky+2,kx)=b(ky+2,kx)+e9
b(ky,kx-1)=b(ky,kx-1)+e10

enddo
enddo
return
end

```

subroutine rotate_rectangle_16neigh sum(a,n,b,m,theta)
real a(256,256),b(256,256),ss(256,256),w(256,256)
c FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
c — THROUGH AN ANGLE THETA (given in degree)
c anticlockwise placing the result at the
c center of an array mxm b
c
c convert theta into radians
c
c pi = 4. * atan(1.)
c theta = theta * (pi/180.)
c costh = cos(theta)
c sinth = sin(theta)
c Zero pad image a with aint((m-n+1)/2) storing the result in the array ss.
c The function aint is used in the case of (m-n+1) is even
c nw=m-n+1
c ml=m+nw-1
c iframe=aint(float(m-n)/2.)
c initial ss
c do i=1,ml
c do j=1,ml
c ss(j,i)=0.
c enddo
c enddo
c do i=1,n
c do j=1,n
c ss(j+iframe,i+iframe)=a(j,i)
c enddo
c enddo
c initial b
c do i = 1,m
c do j = 1,m
c b(j,i) = 0.
c w(j,i) = 0.
c enddo
c enddo
c compute new coordinates (xd,yd) of rotate array
c
c nn = m/2 + 1
c mm = ml/2 + 1
c Compensation of the center position for rotation.
c xcomp=nn+nn*(sinth - costh)
c ycomp=nn-nn*(sinth + costh)
c start process
c
c do j = 1,m
c do i = 1,m
c
c generate 4-quadrant cartesian plan
c y = float(j-nn)
c x = float(i-nn)
c xd = x * costh + y * sinth
c yd = -x * sinth + y * costh
c compute both real and integer coordinates of array b
c associated with the above transformation for later use
c ax = nn+ xd
c ay = nn+ yd
c assign value of a at (j,i) to 9-point cell
c of array b in wich the transformed coordinates
c lie.
c fx=(ax-aint(ax))
c fy=(ay-aint(ay))
c kx=aint(ax)
c ky=aint(ay)
c dl=(1.+fy)*(2.-fx)/16.

```

```

d2=(2.-fx)*(2.-fy)/16.
d3=(2.-fy)*(1.+fx)/16.
d4=(1.+fy)*(1.+fx)/16.
d5=(1.-fy)*(1.-fx)/16.
d6=fx*(1.-fy)/16.
d7=fy*fx/16.
d8=fy*(1.-fx)/16.
d9=fy*(2.-fx)/16.
d10=(2.-fy)*(1.-fx)/16.
d11=(1.+fx)*(1.-fy)/16.
d12=(1.+fy)*fx/16.
d13=(1.-fx)*(1.+fy)/16.
d14=(2.-fx)*(1.-fy)/16.
d15=fx*(2.-fy)/16.
d16=(1.+fx)*fy/16.

```

```

e1=d1*ss(ky+1,kx)
e2=d2* ss(ky,kx)
e3=d3*ss(ky,kx+1)
e4=d4*ss(ky+1,kx+1)
e5=d5*ss(ky-1,kx-1)
e6=d6* ss(ky-1,kx+2)
e7=d7*ss(ky+2,kx+2)
e8=d8*ss(ky+2,kx-1)
e9=d9*ss(ky+1,kx-1)
e10=d10* ss(ky-1,kx)
e11=d11*ss(ky,kx+2)
e12=d12*ss(ky+2,kx+1)
e13=d13*ss(ky+2,kx)
e14=d14*ss(ky,kx-1)
e15=d15*ss(ky-1,kx+1)
e16=d16*ss(ky+1,kx+2)

```

```

b(j,i)=e1+e2+e3+e4+e5+e6+e7+e8+e9+e10+e11+e12+e13+e14+e15+e16

```

```

enddo
enddo
return
end

```

```

subroutine rotate triangle 12neigh sum(a,n,b,m,theta)
real a(256,256),b(256,256),ss(256,256),w(256,256)
C FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
C            THROUGH AN ANGLE THETA (given in degree)
C            anticlockwise placing the result at the
C            center of an array mxm b
C
C            convert theta into radians
C
C            pi = 4. * atan(1.)
C            theta = theta * (pi/180.)
C            costh = cos(theta)
C            sinth = sin(theta)
C Zero pad image a with aint((m-n+1)/2) storing the result in the array ss.
C The function aint is used in the case of (m-n+1) is even
C
C            nw=m-n+1
C            ml=m+nw-1
C            iframe=aint(float(m-n)/2.)
C
C            initial ss
C
C            do i=1,ml
C            do j=1,ml
C            ss(j,i)=0.
C            enddo
C            enddo
C            do i=1,n
C            do j=1,n
C            ss(j+iframe,i+iframe)=a(j,i)
C            enddo
C            enddo
C
C            initial b
C
C            do i = 1,m
C            do j = 1,m
C            b(j,i) = 0.
C            w(j,i) =0.
C            enddo
C            enddo
C
C compute new coordinates (xd,yd) of rotate array
C
C            nn = m/2 + 1
C            mm = ml/2 + 1
C Compensation of the center position for rotation.
C            xcomp=nn+nn*(sinth - costh)
C            ycomp=nn-nn*(sinth + costh)
C
C start process
C
C            do j = 1,m
C            do i = 1,m
C
C generate 4-quadrant cartesian plan
C            y = float(j-nn)
C            x = float(i-nn)
C            xd = x * costh + y * sinth
C            yd = -x * sinth + y * costh
C
C compute both real and integer coordinates of array b
C associated with the above transformation for later use
C            ax =nn+ xd
C            ay = nn+ yd
C assign value of a at (j,i) to 9-point cell
C of array b in wich the transformed coordinates
C lie.
C            fx=(ax-aint(ax))
C            fy=(ay-aint(ay))
C            kx=aint(ax)
C            ky=aint(ay)
C            dl=.1*(2.-fy)

```



```
d2=.1*(1.+fx)
d3=.1*(1.+fy)
d4=.1*(2.-fx)
d5=.1*(1.-fx)
d6=.1*(1.-fy)
d7=.1*fx
d8=.1*fy
```

```
e1=d1*ss(ky,kx)
e2=d2* ss(ky,kx+1)
e3=d3*ss(ky+1,kx+1)
e4=d4*ss(ky+1,kx)
e5=d5*ss(ky+2,kx)
e6=d6* ss(ky,kx-1)
e7=d7*ss(ky-1,kx+1)
e8=d8*ss(ky+1,kx+2)
e9=d6*ss(ky,kx+2)
e10=d7* ss(ky+2,kx+1)
e11=d8*ss(ky+1,kx-1)
e12=d5*ss(ky-1,kx)
```

```
b(j,i)=e1+e2+e3+e4+e5+e6+e7+e8+e9+e10+e11+e12
```

```
enddo
enddo
return
end
```

```

subroutine rotate triangle 16neigh sum(a,n,b,m,theta)
real a(256,256),b(256,256),ss(256,256),w(256,256)
FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
           THROUGH AN ANGLE THETA (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b

convert theta into radians

pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)

Zero pad image a with aint((m-n+1)/2) storing the result in the array ss.
The function aint is used in the case of (m-n+1) is even
nw=m-n+1
ml=m+nw-1
iframe=aint(float(m-n)/2.)

initial ss
do i=1,ml
doj=1,ml
ss(j,i)=0.
enddo
enddo
do i=1,n
do j=1,n
ss(j+iframe,i+iframe)=a(j,i)
enddo
enddo

initial b
do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) = 0.
enddo
enddo

compute new coordinates (xd,yd) of rotate array

nn = m/2 + 1
mm = ml/2 + 1
Compensation of the center position for rotation.
xcomp=nn+nn*(sinth - costh)
ycomp=nn-nn*(sinth + costh)

start process

do j = 1,m
do i = 1,m

generate 4-quadrant cartesian plan
y = float(j-nn)
x = float(i-nn)
xd = x * costh + y * sinth
yd = -x * sinth + y * costh

compute both real and integer coordinates of array b
associated with the above transformation for later use
ax =nn+ xd
ay = nn+ yd

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.
fx=(ax-aint(ax))
fy=(ay-aint(ay))
kx=aint(ax)
ky=aint(ay)
dl=.05*(2.-fy)

```

```
d2=.05*(1.+fx)
d3=.05*(1.+fy)
d4=.05*(2.-fx)
d13=.05*fy
d14=.05*(1.-fx)
d15=.05*(1.-fy)
d16=.05*fx
```

```
e1=d1*ss(ky,kx)
e2=d2*ss(ky,kx+1)
e3=d3*ss(ky+1,kx+1)
e4=d4*ss(ky+1,kx)
e5=d1*ss(ky-1,kx)
e6=d2* ss(ky,kx+2)
e7=d3*ss(ky+2,kx+1)
e8=d4*ss(ky+1,kx-1)
e9=d1*ss(ky-1,kx+1)
e10=d2* ss(ky+1,kx+2)
e11=d3*ss(ky+2,kx)
e12=d4*ss(ky,kx-1)
e13=d13*ss(ky+2,kx+2)
e14=d14*ss(ky+2,kx-1)
e15=d15* ss(ky-1,kx-1)
e16=d16*ss(ky-1,kx+2)
```

```
b(j,i)=e1+e2+e3+e4+e5+e6+e7+e8+e9+e10+e11+e12+e13+e14+e15+e16
```

```
enddo
enddo
return
end
```

```

subroutine rotate triangle 16neigh sum1(a,n,b,m,theta)
real a(256,256),b(256,256),ss(256,256),w(256,256)
c FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
c THROUGH AN ANGLE THETA (given in degree)
c anticlockwise placing the result at the
c center of an array mxm b
c
c convert theta into radians
c
c pi = 4. * atan(1.)
c theta = theta * (pi/180.)
c costh = cos(theta)
c sinth = sin(theta)
c Zero pad image a with anint((m-n+1)/2) storing the result in the array ss.
c The function anint is used in the case of (m-n+1) is even
c nw=m-n+1
c ml=m+nw-1
c iframe=anint(float(m-n)/2.)
c initial ss
c do i=1,ml
c do j=1,ml
c ss(j,i)=0.
c enddo
c enddo
c do i=1,n
c do j=1,n
c ss(j+iframe,i+iframe)=a(j,i)
c enddo
c enddo
c initial b
c do i = 1,m
c do j = 1,m
c b(j,i) = 0.
c w(j,i) = 0.
c enddo
c enddo
c compute new coordinates (xd,yd) of rotate array
c
c nn = m/2 + 1
c mm = ml/2 + 1
c Compensation of the center position for rotation.
c xcomp=nn+nn*(sinth - costh)
c ycomp=nn-nn*(sinth + costh)
c start process
c
c do j = 1,m
c do i = 1,m
c
c generate 4-quadrant cartesian plan
c y = float(j-nn)
c x = float(i-nn)
c xd = x * costh + y * sinth
c yd = -x * sinth + y * costh
c compute both real and integer coordinates of array b
c associated with the above transformation for later use
c ax = nn+ xd
c ay = nn+ yd
c assign value of a at (j,i) to 9-point cell
c of array b in wich the transformed coordinates
c lie.
c fx=(ax-aint(ax))
c fy=(ay-aint(ay))
c kx=aint(ax)
c ky=aint(ay)
c dl=(1.-fx)/32.

```

```
d2=(1.-fy)/32.  
d3=fx/32.  
d4=fy/32.  
d5=(2.-fy)/32.  
d6=(1.+fy)/32.  
d7=(2.-fx)/32.  
d8=(1.+fx)/32.
```

```
e1=d1*ss(ky+2,kx-1)  
e2=d2*ss(ky-1,kx-1)  
e3=d3*ss(ky-1,kx+2)  
e4=d4*ss(ky+2,kx+2)  
e5=d5*ss(ky-1,kx)  
e6=d6* ss(ky+2,kx+1)  
e7=d7*ss(ky+1,kx-1)  
e8=d8*ss(ky,kx+2)  
e9=3.*d3*ss(ky-1,kx+1)  
e10=3.*d2* ss(ky,kx-1)  
e11=3.*d4*ss(ky+1,kx+2)  
e12=3.*d1*ss(ky+2,kx)  
e13=3.*d8*ss(ky,kx+1)  
e14=3.*d6*ss(ky+1,kx+1)  
e15=3.*d7* ss(ky+1,kx)  
e16=3.*d5*ss(ky,kx)
```

```
b(j,i)=e1+e2+e3+e4+e5+e6+e7+e8+e9+e10+e11+e12+e13+e14+e15+e16
```

```
enddo  
enddo  
return  
end
```

```

subroutine irtr_rectangle_12neigh_sum(x,m,y,n)
real x(256,256),y(256,256),xx(256,256)
*****
FUNCTION: This subroutine computes the inverse radon
transform by:
    1. Back_projecting all the projections stored
       in array x giving the back_projection
       function y.
    2. Deconvolving the array y using the inverse_
       filter SQRT(kx*kx + ky*ky*).
External subroutines: bpr_nearest_neig
                     back_projection_deconvolution
Internal subroutines: none
*****
Send message to user 'Filtering Radon Transform...'

    write(6,1)
    format(1x,'Filtering Radon Transform...')
    call projection_filtering(x,m)
Send message to user 'Computing Inverse Radon Transform...'

    write(6,2)
    format(1x,'Computing Inverse Radon Transform...')
Initialize array xx
do i=1,n
do j=1,n
xx(j,i)=0.
enddo
enddo

Back project projections stored in x

    call bpr_rectangle_12neigh_sum(x,m,y,n)

return
end

```

```

subroutine irtr_divide_rectangle12n_sum(x,m,y,n)
real x(256,256),y(256,256),xx(256,256)
c *****
c FUNCTION: This subroutine computes the inverse radon
c transform by:
c     1. Back_projecting all the projections stored
c        in array x giving the back_projection
c        function y.
c     2. Deconvolving the array y using the inverse_
c        filter SQRT(kx*kx + ky*ky*).
c External subroutines: bpr_nearest_neig
c                      back_projection_deconvolution
c Internal subroutines: none
c *****
c Send message to user 'Filtering Radon Transform...'
c
c     write(6,1)
c     format(1x,'Filtering Radon Transform...'$)
c     call projection_filtering(x,m)
c Send message to user 'Computing Inverse Radon Transform...'
c
c     write(6,2)
c     format(1x,'Computing Inverse Radon Transform...'$)
c Initialize array xx
c     do i=1,n
c     do j=1,n
c     xx(j,i)=0.
c     enddo
c     enddo
c
c Back project projections stored in x
c
c     call bpr_divide_rectangle12n_sum(x,m,y,n)
c
c     return
c     end

```

```

subroutine irtr_rectangle_16neigh_sum(x,m,y,n)
  real x(256,256),y(256,256),xx(256,256)
  *****
FUNCTION: This subroutine computes the inverse radon
  transform by:
    1. Back_projecting all the projections stored
      in array x giving the back_projection
      function y.
    2. Deconvolving the array y using the inverse_
      filter SQRT(kx*kx + ky*ky*).
External subroutines: bpr_nearest_neig
                     back_projection_deconvolution
Internal subroutines: none
  *****
Send message to user 'Filtering Radon Transform...'

  write(6,1)
  format(1x,'Filtering Radon Transform...'$)
  call projection_filtering(x,m)
Send message to user 'Computing Inverse Radon Transform...'

  write(6,2)
  format(1x,'Computing Inverse Radon Transform...'$)
Initialize array xx
  do i=1,n
  do j=1,n
  xx(j,i)=0.
  enddo
  enddo

Back project projections stored in x

  call bpr_rectangle_16neigh_sum(x,m,y,n)

  return
end

```



```

      subroutine irtr_triangle_12neigh_sum(x,m,y,n)
      real x(256,256),y(256,256),xx(256,256)
c *****
c FUNCTION: This subroutine computes the inverse radon
c           transform by:
c           1. Back_projecting all the projections stored
c              in array x giving the back_projection
c              function y.
c           2. Deconvolving the array y using the inverse_
c              filter SQRT(kx*kx + ky*ky*).
c External subroutines: bpr_triangle_16neigh_sum
c                      back_projection_deconvolution
c Internal subroutines: none
c *****
c Send message to user 'Filtering Radon Transform...'
c
      write(6,1)
1      format(1x,'Filtering Radon Transform...')
      call projection_filtering(x,m)
c Send message to user 'Computing Inverse Radon Transform...'
c
      write(6,2)
2      format(1x,'Computing Inverse Radon Transform...')
c Initialize array xx
      do i=1,n
      do j=1,n
      xx(j,i)=0.
      enddo
      enddo
c
c Back project projections stored in x
c
      call bpr_triangle_12neigh_sum(x,m,y,n)
c
      return
      end

```

```

subroutine irtr_triangle_16neigh_sum(x,m,y,n)
real x(256,256),y(256,256),xx(256,256)
*****
c FUNCTION: This subroutine computes the inverse radon
c transform by:
c     1. Back_projecting all the projections stored
c        in array x giving the back_projection
c        function y.
c     2. Deconvolving the array y using the inverse_
c        filter SQRT(kx*kx + ky*ky*).
c External subroutines: bpr_triangle_16neigh_sum
c                      back_projection_deconvolution
c Internal subroutines: none
c *****
c Send message to user 'Filtering Radon Transform...'
c
c     write(6,1)
c     format(1x,'Filtering Radon Transform...'$)
c     call projection_filtering(x,m)
c Send message to user 'Computing Inverse Radon Transform...'
c
c     write(6,2)
c     format(1x,'Computing Inverse Radon Transform...'$)
c Initialize array xx
c     do i=1,n
c     do j=1,n
c     xx(j,i)=0.
c     enddo
c     enddo
c
c Back project projections stored in x
c
c     call bpr_triangle_16neigh_sum(x,m,y,n)
c
c     return
c     end

```

```

      subroutine irtr_triangle_16neigh_sum1(x,m,y,n)
      real x(256,256),y(256,256),xx(256,256)
      *****
FUNCTION: This subroutine computes the inverse radon
      transform by:
      1. Back_projecting all the projections stored
         in array x giving the back_projection
         function y.
      2. Deconvolving the array y using the inverse_
         filter SQRT(kx*kx + ky*ky*).
External subroutines: bpr_triangle_16neigh_sum
                     back_projection_deconvolution
Internal subroutines: none
      *****
Send message to user 'Filtering Radon Transform...'

      write(6,1)
      format(1x,'Filtering Radon Transform...'$)
      call projection_filtering(x,m)
Send message to user 'Computing Inverse Radon Transform...'

      write(6,2)
      format(1x,'Computing Inverse Radon Transform...'$)
Initialize array xx
      do i=1,n
      do j=1,n
      xx(j,i)=0.
      enddo
      enddo

Back project projections stored in x

      call bpr_triangle_16neigh_sum1(x,m,y,n)

      return
      end

```

subroutine projection_filtering(a,n)

Function: This subroutine filters a Radon Transform
obtained by straight projection (filtered)
using the appropriate inverse filter, i.e.
ABS(kx) where kx is the spatial frequency.

```
real a(256,256),y(256,256),c(256,256),za(256)
real zb(256),zc(256),zd(256)
```

initialize array c

```
do i=1,n
do j=1,n
c(j,i)=0.
enddo
enddo
do j=1,256
za(j)=0.
zb(j)=0.
zc(j)=0.
zd(j)=0.
enddo
```

take Fourier transform of a

```
nn=256/2+1
jframe=nint(float(256-n)/2.)
do i=1,n
do j=1,n
za(j+jframe)=a(j,i)
enddo
call fftld(za,zb,256,-1)
```

multiply spectrum by inverse filter

```
do j=1,256
xk=abs(float(j-nn)/float(256))
zc(j)=za(j)*xk
zd(j)=zb(j)*xk
enddo
```

```
call fftld(zc,zd,256,1)
```

```
do j=1,n
c(j,i)=zc(j+jframe)
enddo
enddo
```

write output

```
do i=1,n
do j=1,n
a(j,i)=c(j,i)
enddo
enddo
```

```
return
end
```

```

      subroutine project(a,b,n)
      *****
Function: This subroutine sums
      over all rows of an n X n array a giving 1D
projection of size n _ array b
      *****
      real a(256,256),b(256),s(256,256),z(256)
      initialize arrays s & b
        do j=1,n
          b(j)=0.
          z(j)=0.
          do i=1,n
            s(j,i)=0.
          enddo
        enddo
      set initial value for summing process and sum all pixels along
      row i together
        do j=1,n
          s(j,1)=a(j,1)
          do i=2,n
            s(j,i)=s(j,i-1)+a(j,i)
          enddo
      Assign value of sum at column j to output _b(j)
        b(j)=s(j,n)
      enddo

      return
      end

```

```

subroutine bpr_rectangle_12neigh sum(b,m,a,n)
  real a(256,256),b(256,256),xx(256,256)
  *****
Function: this routine back-projects the
         data stored in b to reconstruct
         the image a.
External subroutines: Rotate_nearest_neig
Internal subroutines: None
Parameters: b _ input, size m X m
            a _ output, size n X n
  *****
x, y and sum are working space
  real x(256,256),y(256,256)
  real sum(256,256)
Send message to user 'Computing back_projection ...'

  write(6,1)
  format(1x,'Computing back_projection...'$)

Compute size of image in which b will be rotated to prevent
corner clipping.
  l=nint(sqrt(2.)*float(m))
Compute centers of arrays to be used later
  nn=n/2+1
  ll=l/2+1
  ln=ll-nn
Initialize working space
  do i=1,l
  do j=1,l
    x(j,i)=0.
    y(j,i)=0.
    sum(j,i)=0.
    a(j,i)=0.
    xx(j,i)=0.
  enddo
  enddo
Compute the angle of rotation in the reverse direction
  do k=1,m
    theta=float(1-k)
Back_project projection
  do i=1,m
  do j=1,m
    x(j,i)=b(j,k)
  enddo
  enddo
Rotate x by angle at which projection was taken
  call rotate_rectangle_12neigh_sum(x,m,y,l,theta)
Add result to previous back_projection function
  do i=1,l
  do j=1,l
    sum(j,i)=sum(j,i)+y(j,i)
  enddo
  enddo

  enddo
Window the back_projection function to extract reconstruction
(size n X n) and write it to output
  do i=1,n
  do j=1,n
    a(j,i)=sum(j+ln,i+ln)
  enddo
  enddo

  return
end

```



```

subroutine bpr_triangle_l2neigh_sum(b,m,a,n)
  real a(256,256),b(256,256),xx(256,256)
  *****
Function: this routine back-projects the
         data stored in b to reconstruct
         the image a.
External subroutines: Rotate_triangle_l6neigh_sum
Internal subroutines: None
Parameters: b _ input, size m X m
            a _ output, size n X n
  *****
x, y and sum are working space
  real x(256,256),y(256,256)
  real sum(256,256)
Send message to user 'Computing back_projection ...'

  write(6,1)
  format(1x,'Computing back_projection...'$)

Compute size of image in which b will be rotated to prevent
corner clipping.
  l=nint(sqrt(2.)*float(m))
Compute centers of arrays to be used later
  nn=n/2+1
  ll=l/2+1
  ln=ll-nn
Initialize working space
  do i=1,l
  do j=1,l
    x(j,i)=0.
    y(j,i)=0.
    sum(j,i)=0.
    a(j,i)=0.
    xx(j,i)=0.
  enddo
  enddo
Compute the angle of rotation in the reverse direction
  do k=1,m
    theta=float(1-k)
Back_project projection
  do i=1,m
  do j=1,m
    x(j,i)=b(j,k)
  enddo
  enddo
Rotate x by angle at which projection was taken
  call rotate_triangle_l2neigh_sum(x,m,y,l,theta)
Add result to previous back_projection function
  do i=1,l
  do j=1,l
    sum(j,i)=sum(j,i)+y(j,i)
  enddo
  enddo

  enddo
Window the back_projection function to extract reconstruction
(size n X n) and write it to output
  do i=1,n
  do j=1,n
    a(j,i)=sum(j+ln,i+ln)
  enddo
  enddo

  return
end

```


subroutine bpr_triangle_16neigh_sum(b,m,a,n)

real a(256,256),b(256,256),xx(256,256)

Function: this routine back-projects the
data stored in b to reconstruct
the image a.

External subroutines: Rotate_triangle_16neigh_sum

Internal subroutines: None

Parameters: b _ input, size m X m

a _ output, size n X n

x, y and sum are working space

real x(256,256),y(256,256)

real sum(256,256)

Send message to user 'Computing back_projection ...'

write(6,1)

format(1x,'Computing back_projection...'\$)

Compute size of image in which b will be rotated to prevent
corner clipping.

l=nint(sqrt(2.)*float(m))

Compute centers of arrays to be used later

nn=n/2+1

ll=l/2+1

ln=ll-nn

Initialize working space

do i=1,l

do j=1,l

x(j,i)=0.

y(j,i)=0.

sum(j,i)=0.

a(j,i)=0.

xx(j,i)=0.

enddo

enddo

Compute the angle of rotation in the reverse direction

do k=1,m

theta=float(1-k)

Back_project projection

do i=1,m

do j=1,m

x(j,i)=b(j,k)

enddo

enddo

Rotate x by angle at which projection was taken

call rotate_triangle_16neigh_sum(x,m,y,l,theta)

Add result to previous back_projection function

do i=1,l

do j=1,l

sum(j,i)=sum(j,i)+y(j,i)

enddo

enddo

enddo

Window the back_projection function to extract reconstruction
(size n X n) and write it to output

do i=1,n

do j=1,n

a(j,i)=sum(j+ln,i+ln)

enddo

enddo

return

end

```

subroutine bpr_triangle_16neigh_sum1(b,m,a,n)
  real a(256,256),b(256,256),xx(256,256)
*****
Function: this routine back-projects the
         data stored in b to reconstruct
         the image a.
External subroutines: Rotate_triangle_16neigh_sum
Internal subroutines: None
Parameters: b _ input, size m X m
           a _ output, size n X n
*****
x, y and sum are working space
  real x(256,256),y(256,256)
  real sum(256,256)
Send message to user 'Computing back_projection ...'

  write(6,1)
  format(1x,'Computing back_projection...'$)

Compute size of image in which b will be rotated to prevent
corner clipping.
  l=nint(sqrt(2.)*float(m))
Compute centers of arrays to be used later
  nn=n/2+1
  ll=l/2+1
  ln=ll-nn
Initialize working space
  do i=1,l
  do j=1,l
  x(j,i)=0.
  y(j,i)=0.
  sum(j,i)=0.
  a(j,i)=0.
  xx(j,i)=0.
  enddo
  enddo
Compute the angle of rotation in the reverse direction
  do k=1,m
  theta=float(1-k)
Back_project projection
  do i=1,m
  do j=1,m
  x(j,i)=b(j,k)
  enddo
  enddo
Rotate x by angle at which projection was taken
  call rotate_triangle_16neigh_sum1(x,m,y,l,theta)
Add result to previous back_projection function
  do i=1,l
  do j=1,l
  sum(j,i)=sum(j,i)+y(j,i)
  enddo
  enddo

  enddo
Window the back_projection function to extract reconstruction
(size n X n) and write it to output
  do i=1,n
  do j=1,n
  a(j,i)=sum(j+ln,i+ln)
  enddo
  enddo

  return
end

```

```
subroutine bpr divide_rectangle12n sum(b,m,a,n)
```

```
real a(256,256),b(256,256),xx(256,256)
```

```
*****
```

Function: this routine back-projects the
data stored in b to reconstruct
the image a.

External subroutines: Rotate_nearest_neig

Internal subroutines: None

Parameters: b _ input, size m X m.

a _ output, size n X n

```
*****
```

x, y and sum are working space

```
real x(256,256),y(256,256)
```

```
real sum(256,256)
```

Send message to user 'Computing back_projection ...'

```
write(6,1)
```

```
format(1x,'Computing back_projection...'$)
```

Compute size of image in which b will be rotated to prevent
corner clipping.

```
l=nint(sqrt(2.)*float(m))
```

Compute centers of arrays to be used later

```
nn=n/2+1
```

```
ll=l/2+1
```

```
ln=ll-nn
```

Initialize working space

```
do i=1,l
```

```
do j=1,l
```

```
x(j,i)=0.
```

```
y(j,i)=0.
```

```
sum(j,i)=0.
```

```
a(j,i)=0.
```

```
xx(j,i)=0.
```

```
enddo
```

```
enddo
```

Compute the angle of rotation in the reverse direction

```
do k=1,m
```

```
theta=float(1-k)
```

Back_project projection

```
do i=1,m
```

```
do j=1,m
```

```
x(j,i)=b(j,k)
```

```
enddo
```

```
enddo
```

Rotate x by angle at which projection was taken

```
call rotate_divide_rectangle12n_sum(x,m,y,l,theta)
```

Add result to previous back_projection function

```
do i=1,l
```

```
do j=1,l
```

```
sum(j,i)=sum(j,i)+y(j,i)
```

```
enddo
```

```
enddo
```

```
enddo
```

Window the back_projection function to extract reconstruction
(size n X n) and write it to output

```
do i=1,n
```

```
do j=1,n
```

```
a(j,i)=sum(j+ln,i+ln)
```

```
enddo
```

```
enddo
```

```
return
```

```
end
```

```

subroutine rtr_nearest(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)

```

Function: This subroutine computes the Radon Transform of
n X n image.

External subroutines: Rotate_nearest_neig
Project

Internal subroutines: None

Parameters: x - n X n input array
y - m X m output array

Send message to user 'Computing Radon Transform...'

```

write(6,1)
format(1x,'Computing Radon Transform...'$)

```

initialize array y

```

do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo

```

Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.

```

do i=1,m
theta=float(i-1)
call rotate_nearest(x,n,w,m,theta)
call project(w,z,m)

```

Stack projections together to form image of "Radon space"
data.

```

do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```

```

subroutine rtr_nearest_neig(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)

```

Function: This subroutine computes the Radon Transform of
n X n image.

External subroutines: Rotate_nearest_neig
Project

Internal subroutines: None

Parameters: x - n X n input array
y - m X m output array

Send message to user 'Computing Radon Transform...'

```

write(6,1)
format(1x,'Computing Radon Transform...'$)

```

initialize array y

```

do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo

```

Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.

```

do i=1,m
theta=float(i-1)
call rotate_nearest_neig(x,n,w,m,theta)
call project(w,z,m)

```

Stack projections together to form image of "Radon space"
data.

```

do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```

```

subroutine rtr_four(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
*****
Function: This subroutine computes the Radon Transform of
         n X n image.
External subroutines: Rotate_nearest_neig
                     Project
Internal subroutines: None
Parameters: x - n X n input array
            y - m X m output array
*****
Send message to user 'Computing Radon Transform...'

write(6,1)
format(1x,'Computing Radon Transform...'$)
initialize array y
do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo
Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.
do i=1,m
theta=float(i-1)
call rotate_four(x,n,w,m,theta)
call project(w,z,m)
Stack projections together to form image of "Radon space"
data.
do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```

```

subroutine rtr four_neig(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)

```

Function: This subroutine computes the Radon Transform of
n X n image.

External subroutines: Rotate_four_neig
Project

Internal subroutines: None

Parameters: x - n X n input array
y - m X m output array

Send message to user 'Computing Radon Transform...'

```

write(6,1)
format(1x,'Computing Radon Transform...'$)

```

initialize array y

```

do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo

```

Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.

```

do i=1,m
theta=float(i-1)
call rotate_four_neig(x,n,w,m,theta)
call project(w,z,m)

```

Stack projections together to form image of "Radon space"
data.

```

do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```

```

subroutine rtr_eight(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)

```

Function: This subroutine computes the Radon Transform of
n X n image.

External subroutines: Rotate_nearest_neig
Project

Internal subroutines: None

Parameters: x - n X n input array
y - m X m output array

Send message to user 'Computing Radon Transform...'

```

write(6,1)
format(1x,'Computing Radon Transform...'$)

```

initialize array y

```

do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo

```

Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.

```

do i=1,m
theta=float(i-1)
call rotate_eight(x,n,w,m,theta)
call project(w,z,m)

```

Stack projections together to form image of "Radon space"
data.

```

do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```



```

subroutine rtr_eight_neig(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)

```

Function: This subroutine computes the Radon Transform of
n X n image.

External subroutines: Rotate_eight_neig
Project

Internal subroutines: None

Parameters: x - n X n input array
y - m X m output array

Send message to user 'Computing Radon Transform...'

```

write(6,1)
format(1x,'Computing Radon Transform...'$)

```

initialize array y

```

do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo

```

Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.

```

do i=1,m
theta=float(i-1)
call rotate_eight_neig(x,n,w,m,theta)
call project(w,z,m)

```

Stack projections together to form image of "Radon space"
data.

```

do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```

```

subroutine rotate_nearest(a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
*****
FUNCTION : This subroutine rotates an n X n array a
           through an angle theta (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b
PARAMETERS: a _ Input image of n X n size
           b _ Rotated image of m X m size
EXTERNAL SUBROUTINE :None
INTERNAL SUBROUTINE :None
*****

```

```

convert theta into radians

```

```

pi = 4. * atan(1.)
theta = theta * (pi/180.)
costh = cos(theta)
sinth = sin(theta)

```

```

initial b

```

```

do i = 1,m
do j = 1,m
b(j,i) = 0.
w(j,i) = 0.
enddo
enddo

```

```

compute new coordinates (xd,yd) of rotate array

```

```

nn = n/2 + 1
mm = m/2 + 1

```

```

start process

```

```

do j = 1,n
do i = 1,n

```

```

generate 4-quadrant cartesian plan

```

```

y = float(j-nn)
x = float(i-nn)
xd = x * costh - y * sinth
yd = x * sinth + y * costh

```

```

compute both real and integer coordinates of array b
associated with the above transformation for later use

```

```

kx = mm+ nint(xd)
ky = mm+ nint(yd)

```

```

assign value of a at (j,i) to the nearest point cell
of array b in wich the transformed coordinates
lie.

```

```

b(ky,kx) = a(j,i)

```

```

enddo
enddo
return
end

```

```

subroutine rotate_nearest_neig (a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
integer xu,yu,xd,yd,dul,eul,du2,eu2,vu(256),hu(256),ku(256)
integer gu(256)

```

Function: This subroutine rotates an nXn array a through an angle theta (given in degrees) placing the result at the center of an mXm array b.

Convert theta into radians.

```

pi=4.*atan(1.)
theta=theta*(pi/180.)

```

initialize b

```

do i=1,m
do j=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo

```

Compute centers of array a and b

```

nn=n/2+1
mm=m/2+1

```

Compute the integer of horizontal and vertical displacement along the x and y-axis which corresponds to n-pixel movement along the u-axis.

```

xu=nint(nn*cos(theta))
yu=nint(nn*sin(theta))
if (xu.lt.0) then
ml=-1
else
ml=1
endif
if (yu.lt.0) then
nl=1
else
nl=-1
endif

```

Compute the rotation parameters

```

dul=2*abs(yu)
du2=2*(abs(yu)-nn)
eul=2*abs(xu)
eu2=2*(abs(xu)-nn)
vu(1)=2*abs(yu)-nn
hu(1)=2*abs(xu)-nn
ku(0)=0
gu(0)=0
do i=1,nn
if (hu(i).ge.0) then
hu(i+1)=hu(i)+eu2
gu(i)=gu(i-1)+1
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
else
hu(i+1)=hu(i)+eul
gu(i)=gu(i-1)
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
endif
enddo

```

Compute new coordinates (xd,yd) of rotated array and assign the value of a at (i,j) to the appropriate cell of b which encloses these coordinates.

Generate 4-quadrant cartesian plane.

```
do i=1,nn
ku(-i)=-ku(i)
gu(-i)=-gu(i)
enddo
```

Start process

```
do j=1,n
do i=1,n
```

Employ coordinate transformation for rotation using the integer parameters.

```
yd=m1*gu(i-nn)-n1*ku(j-nn)
xd=m1*gu(j-nn)+n1*ku(i-nn)
```

Compute the integer coordinates of array b associated with the above transformation.

```
kx=mm+xd
ky=mm+yd
```

Assign value of a at (i,j) to the nearest cell of array b in which the transformed coordinates lie.

```
b(ky+1,kx-1)=a(i,j)
b(ky,kx-1)=a(i,j)
b(ky-1,kx-1)=a(i,j)
b(ky+1,kx)=a(i,j)
b(ky,kx)=a(i,j)
b(ky-1,kx)=a(i,j)
b(ky+1,kx+1)=a(i,j)
b(ky,kx+1)=a(i,j)
b(ky-1,kx+1)=a(i,j)
```

```
enddo
enddo
return
end
```

```

      subroutine rotate four(a,n,b,m,theta)
      real a(256,256),b(256,256),w(256,256)
*****
FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
           THROUGH AN ANGLE THETA (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b
PARAMETERS: a _ Input image of n X n size
           b _ Rotated image of m X m size
EXTERNAL SUBROUTINE :None
INTERNAL SUBROUTINE :None
*****

```

convert theta into radians

```

      pi = 4. * atan(1.)
      theta = theta * (pi/180.)
      costh = cos(theta)
      sinth = sin(theta)

```

initial b

```

      do i = 1,m
      do j = 1,m
      b(j,i) = 0.
      w(j,i) = 0.
      enddo
      enddo

```

compute new coordinates (xd,yd) of rotate array

```

      nn = n/2 + 1
      mm = m/2 + 1

```

Compensation of the center position for rotation.

```

      xcomp=nn+nn*(sinth - costh)
      ycomp=nn-nn*(sinth + costh)

```

start process

```

      do j = 1,n
      do i = 1,n

```

generate 4-quadrant cartesian plan

```

      y = float(j-nn)
      x = float(i-nn)
      xd = x * costh - y * sinth
      yd = x * sinth + y * costh

```

compute both real and integer coordinates of array b
associated with the above transformation for later use

```

      kx = mm+ nint(xd)
      ky = mm+ nint(yd)

```

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.

```

      b(ky,kx-1) = a(j,i)
      b(ky+1,kx) = a(j,i)
      b(ky,kx) = a(j,i)
      b(ky-1,kx) = a(j,i)
      b(ky,kx+1) = a(j,i)
      enddo
      enddo
      return
      end

```

```

subroutine rotate four neig (a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
integer xu,yu,xd,yd,dul,eul,du2,eu2,vu(256),hu(256),ku(256)
integer gu(256)

```

Function: This subroutine rotates an nXn array a through an angle theta (given in degrees) placing the result at the center of an mXm array b.

Convert theta into radiants.

```

pi=4.*atan(1.)
theta=theta*(pi/180.)

```

initialize b

```

do i=1,m
do j=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo

```

Compute centers of array a and b

```

nn=n/2+1
mm=m/2+1

```

Compute the integer of horizontal and vertical displacement along the x and y-axis which corresponds to n-pixel movement along the u-axis.

```

xu=nint(nn*cos(theta))
yu=nint(nn*sin(theta))
if (xu.lt.0) then
ml=-1
else
ml=1
endif
if (yu.lt.0) then
nl=1
else
nl=-1
endif

```

Compute the rotation parameters

```

dul=2*abs(yu)
du2=2*(abs(yu)-nn)
eul=2*abs(xu)
eu2=2*(abs(xu)-nn)
vu(1)=2*abs(yu)-nn
hu(1)=2*abs(xu)-nn
ku(0)=0
gu(0)=0
do i=1,nn
if (hu(i).ge.0) then
hu(i+1)=hu(i)+eu2
gu(i)=gu(i-1)+1
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
else
hu(i+1)=hu(i)+eul
gu(i)=gu(i-1)
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
endif
enddo

```

Compute new coordinates (xd,yd) of rotated array and assign the value of a at (i,j) to the appropriate cell of b which encloses these coordinates.

Generate 4-quadrant cartesian plane.

```
do i=1,nn
ku(-i)=-ku(i)
gu(-i)=-gu(i)
enddo
```

Start process

```
do j=1,n
do i=1,n
```

Employ coordinate transformation for rotation using the integer parameters.

```
yd=m1*gu(i-nn)-n1*ku(j-nn)
xd=m1*gu(j-nn)+n1*ku(i-nn)
```

Compute the integer coordinates of array b associated with the above transformation.

```
kx=mn+xd
ky=mn+yd
```

Assign value of a at (i,j) to the nearest cell of array b in which the transformed coordinates lie.

```
b(ky+1,kx-1)=a(i,j)
b(ky,kx-1)=a(i,j)
b(ky-1,kx-1)=a(i,j)
b(ky+1,kx)=a(i,j)
b(ky,kx)=a(i,j)
b(ky-1,kx)=a(i,j)
b(ky+1,kx+1)=a(i,j)
b(ky,kx+1)=a(i,j)
b(ky-1,kx+1)=a(i,j)
```

```
enddo
enddo
return
end
```

```

      subroutine rotate_eight(a,n,b,m,theta)
      real a(256,256),b(256,256),w(256,256)
      *****
FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
          THROUGH AN ANGLE THETA (given in degree)
          anticlockwise placing the result at the
          center of an array mxm b
PARAMETERS: a _ Input image of n X n size
          b _ Rotated image of m X m size
EXTERNAL SUBROUTINE :None
INTERNAL SUBROUTINE :None
      *****

```

convert theta into radians

```

      pi = 4. * atan(1.)
      theta = theta * (pi/180.)
      costh = cos(theta)
      sinth = sin(theta)

```

initial b

```

      do i = 1,m
      do j = 1,m
      b(j,i) = 0.
      w(j,i) = 0.
      enddo
      enddo

```

compute new coordinates (xd,yd) of rotate array

```

      nn = n/2 + 1
      mm = m/2 + 1

```

start process

```

      do j = 1,n
      do i = 1,n

```

generate 4-quadrant cartesian plan

```

      y = float(j-nn)
      x = float(i-nn)
      xd = x * costh - y * sinth
      yd = x * sinth + y * costh

```

compute both real and integer coordinates of array b
associated with the above transformation for later use

```

      kx = mm+ nint(xd)
      ky = mm+ nint(yd)

```

assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.

```

      b(ky+1,kx-1) = a(j,i)
      b(ky,kx-1) = a(j,i)
      b(ky-1,kx-1) = a(j,i)
      b(ky+1,kx) = a(j,i)
      b(ky,kx) = a(j,i)
      b(ky-1,kx) = a(j,i)
      b(ky+1,kx+1) = a(j,i)
      b(ky,kx+1) = a(j,i)
      b(ky-1,kx+1) = a(j,i)
      enddo
      enddo
      return
      end

```



```

subroutine rotate eight neig (a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
integer xu,yu,xd,yd,dul,eul,du2,eu2,vu(256),hu(256),ku(256)
integer gu(256)

```

Function: This subroutine rotates an nXn array a through an angle theta (given in degrees) placing the result at the center of an mXm array b.

Convert theta into radients.

```

pi=4.*atan(1.)
theta=theta*(pi/180.)

```

initialize b

```

do i=1,m
do j=1,m
b(j,i)=0
enddo
enddo

```

Compute centers of array a and b

```

nn=n/2+1
mm=m/2+1

```

Compute the integer of horizontal and vertical displacement along the x and y-axis which corresponds to n-pixel movement along the u-axis.

```

xu=nint(nn*cos(theta))
yu=nint(nn*sin(theta))
if (xu.lt.0) then
m1=-1
else
m1=1
endif
if (yu.lt.0) then
n1=1
else
n1=-1
endif

```

Compute the rotation parameters

```

dul=2*abs(yu)
du2=2*(abs(yu)-nn)
eul=2*abs(xu)
eu2=2*(abs(xu)-nn)
vu(1)=2*abs(yu)-nn
hu(1)=2*abs(xu)-nn
ku(0)=0
gu(0)=0
do i=1,nn
if (hu(i).ge.0) then
hu(i+1)=hu(i)+eu2
gu(i)=gu(i-1)+1
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
else
hu(i+1)=hu(i)+eul
gu(i)=gu(i-1)
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
endif
enddo

```

Compute new coordinates (xd,yd) of rotated array and assign the value of a at (i,j) to the appropriate cell of b which encloses these coordinates.
Generate 4-quadrant cartesian plane.

```
do i=1,nn
ku(-i)=-ku(i)
gu(-i)=-gu(i)
enddo
```

Start process

```
do j=1,n
do i=1,n
```

Employ coordinate transformation for rotation using the integer parameters.

```
yd=m1*gu(i-nn)-n1*ku(j-nn)
xd=m1*gu(j-nn)+n1*ku(i-nn)
```

Compute the integer coordinates of array b associated with the above transformation.

```
kx=mm+xd
ky=mm+yd
```

Assign value of a at (j,i) to the nearest cell of array b in which the transformed coordinates lie.

```
b(ky+1,kx-1)=a(i,j)
b(ky,kx-1)=a(i,j)
b(ky-1,kx-1)=a(i,j)
b(ky+1,kx)=a(i,j)
b(ky,kx)=a(i,j)
b(ky-1,kx)=a(i,j)
b(ky+1,kx+1)=a(i,j)
b(ky,kx+1)=a(i,j)
b(ky-1,kx+1)=a(i,j)
```

```
enddo
enddo
return
end
```

```

      subroutine irtr_nearest(x,m,y,n)
      real x(256,256),y(256,256),xx(256,256)
      *****
FUNCTION: This subroutine computes the inverse radon
      transform by:
      1. Back_projecting all the projections stored
         in array x giving the back_projection
         function y.
      2. Deconvolving the array y using the inverse_
         filter  $\text{SQRT}(k_x*k_x + k_y*k_y)$ .
External subroutines: bpr_nearest_neig
                     back_projection_deconvolution
Internal subroutines: none
      *****
Differentiating and taking Hilbert Transform.
Send message to user
' Differentiating and taking Hilbert Transform...'

      write(6,1)
      format(1x,'Differentiating and taking Hilbert Transform...'$)

      call filtered_back_projection(x,m)

Send message to user 'Computing Inverse Radon Transform...'

      write(6,2)
      format(1x,'Computing Inverse Radon Transform...'$)
Initialize array xx
      do i=1,n
      do j=1,n
      xx(j,i)=0.
      enddo
      enddo

Back project projections stored in x

      call bpr_nearest(x,m,y,n)

      return
      end

```

```

subroutine irtr_nearest_neig(x,m,y,n)
real x(256,256),y(256,256),xx(256,256)
*****
FUNCTION: This subroutine computes the inverse radon
transform by:
1. Back_projecting all the projections stored
in array x giving the back_projection
function y.
2. Deconvolving the array y using the inverse_
filter SQRT(kx*kx + ky*ky*).
External subroutines: bpr_nearest_neig
back_projection_deconvolution
Internal subroutines: none
*****
Differentiating and taking Hilbert Transform.
Send message to user
' Differentiating and taking Hilbert Transform...'

write(6,1)
format(1x,'Differentiating and taking Hilbert Transform...'$)

call filtered_back_projection(x,m)

Send message to user 'Computing Inverse Radon Transform...'

write(6,2)
format(1x,'Computing Inverse Radon Transform...'$)
Initialize array xx
do i=1,n
do j=1,n
xx(j,i)=0.
enddo
enddo

Back project projections stored in x

call bpr_nearest_neig(x,m,y,n)

return
end

```

```

      subroutine irtr_four(x,m,y,n)
      real x(256,256),y(256,256),xx(256,256)
      *****
FUNCTION: This subroutine computes the inverse radon
      transform by:
      1. Back_projecting all the projections stored
         in array x giving the back_projection
         function y.
      2. Deconvolving the array y using the inverse_
         filter SQRT(kx*kx + ky*ky*).
External subroutines: bpr_nearest_neig
                     back_projection_deconvolution
Internal subroutines: none
      *****
Differentiating and taking Hilbert Transform.
Send message to user
' Differentiating and taking Hilbert Transform...'

      write(6,1)
      format(1x,'Differentiating and taking Hilbert Transform...'$)

      call filtered_back_projection(x,m)

Send message to user 'Computing Inverse Radon Transform...'

      write(6,2)
      format(1x,'Computing Inverse Radon Transform...'$)
Initialize array xx
      do i=1,n
      do j=1,n
      xx(j,i)=0.
      enddo
      enddo

Back project projections stored in x

      call bpr_four(x,m,y,n)

      return
      end

```

```

      subroutine irtr_four_neig(x,m,y,n)
      real x(256,256),y(256,256),xx(256,256)
      *****
FUNCTION: This subroutine computes the inverse radon
      transform by:
      1. Back_projecting all the projections stored
         in array x giving the back_projection
         function y.
      2. Deconvolving the array y using the inverse_
         filter SQRT(kx*kx + ky*ky*).
External subroutines: bpr_four_neig
                     back_projection_deconvolution
Internal subroutines: none
      *****
Differentiating and taking Hilbert Transform.
Send message to user
' Differentiating and taking Hilbert Transform...'

      write(6,1)
      format(1x,'Differentiating and taking Hilbert Transform...'$)

      call filtered_back_projection(x,m)

Send message to user 'Computing Inverse Radon Transform...'

      write(6,2)
      format(1x,'Computing Inverse Radon Transform...'$)
Initialize array xx
      do i=1,n
      do j=1,n
      xx(j,i)=0.
      enddo
      enddo

Back project projections stored in x

      call bpr_four_neig(x,m,y,n)

      open(unit=1,file='datair.dat',status='new')
      format(4(1x,f14.4))
119 write(1,119)((y(j,i),j=1,128),i=1,128)
      close(1)
      return
      end

```

```

      subroutine irtr_eight(x,m,y,n)
      real x(256,256),y(256,256),xx(256,256)
      *****
FUNCTION: This subroutine computes the inverse radon
      transform by:
      1. Back_projecting all the projections stored
         in array x giving the back_projection
         function y.
      2. Deconvolving the array y using the inverse_
         filter SQRT(kx*kx + ky*ky*).
External subroutines: bpr_nearest_neig
                     back_projection_deconvolution
Internal subroutines: none
      *****
Differentiating and taking Hilbert Transform.
Send message to user
' Differentiating and taking Hilbert Transform...'

      write(6,1)
      format(1x,'Differentiating and taking Hilbert Transform...')
      call filtered_back_projection(x,m)

Send message to user 'Computing Inverse Radon Transform...'

      write(6,2)
      format(1x,'Computing Inverse Radon Transform...')
Initialize array xx
      do i=1,n
      do j=1,n
      xx(j,i)=0.
      enddo
      enddo

Back project projections stored in x

      call bpr_eight(x,m,y,n)

      return
      end

```

```

      subroutine irtr_eight_neig(x,m,y,n)
      real x(256,256),y(256,256),xx(256,256)
      *****
FUNCTION: This subroutine computes the inverse radon
      transform by:
      1. Back_projecting all the projections stored
         in array x giving the back_projection
         function y.
      2. Deconvolving the array y using the inverse_
         filter SQRT(kx*kx + ky*ky*).
External subroutines: bpr_eight_neig
                     back_projection_deconvolution
Internal subroutines: none
      *****
Differentiating and taking Hilbert Transform.
Send message to user
' Differentiating and taking Hilbert Transform...'

      write(6,1)
      format(1x,'Differentiating and taking Hilbert Transform...'$)

      call filtered_back_projection(x,m)

Send message to user 'Computing Inverse Radon Transform...'

      write(6,2)
      format(1x,'Computing Inverse Radon Transform...'$)
Initialize array xx
      do i=1,n
      do j=1,n
      xx(j,i)=0.
      enddo
      enddo

Back project projections stored in x

      call bpr_eight_neig(x,m,y,n)

      return
      end

```



```

      subroutine bpr_nearest(b,m,a,n)
      real a(256,256),b(256,256),xx(256,256)
      *****
Function: this routine back-projects the
      data stored in b to reconstruct
      the image a.
External subroutines: Rotate_nearest_neig
Internal subroutines: None
Parameters: b _ input, size m X m
      a _ output, size n X n
      *****
x, y and sum are working space
      real x(256,256),y(256,256)
      real sum(256,256)
Send message to user 'Computing back_projection ...'

      write(6,1)
      format(1x,'Computing back_projection...'$)

Compute size of image in which b will be rotated to prevent
corner clipping.
      l=nint(sqrt(2.)*float(m))
Compute centers of arrays to be used later
      nn=n/2+1
      ll=l/2+1
      ln=ll-nn
Initialize working space
      do i=1,l
      do j=1,l
      x(j,i)=0.
      y(j,i)=0.
      sum(j,i)=0.
      a(j,i)=0.
      xx(j,i)=0.
      enddo
      enddo
Compute the angle of rotation in the reverse direction
      do k=1,m
      theta=float(1-k)
Back_project projection
      do i=1,m
      do j=1,m
      x(j,i)=b(j,k)
      enddo
      enddo
Rotate x by angle at which projection was taken
      call rotate_nearest(x,m,y,l,theta)
Add result to previous back_projection function
      do i=1,l
      do j=1,l
      sum(j,i)=sum(j,i)+y(j,i)
      enddo
      enddo

      enddo
Window the back_projection function to extract reconstruction
(size n X n) and write it to output
      do i=1,n
      do j=1,n
      a(j,i)=sum(j+ln,i+ln)
      enddo
      enddo

      return
      end

```

```

      subroutine bpr_nearest_neig(b,m,a,n)
      real a(256,256),b(256,256),xx(256,256)
      *****
Function: this routine back-projects the
      data stored in b to reconstruct
      the image a.
External subroutines: Rotate_nearest_neig
Internal subroutines: None
Parameters: b _ input, size m X m
      a _ output, size n X n
      *****
x, y and sum are working space
      real x(256,256),y(256,256)
      real sum(256,256)
Send message to user 'Computing back_projection ...'

      write(6,1)
      format(1x,'Computing back_projection...'$)

Compute size of image in which b will be rotated to prevent
corner clipping.
      l=nint(sqrt(2.)*float(m))
Compute centers of arrays to be used later
      nn=n/2+1
      ll=l/2+1
      ln=ll-nn
Initialize working space
      do i=1,l
      do j=1,l
      x(j,i)=0.
      y(j,i)=0.
      sum(j,i)=0.
      a(j,i)=0.
      xx(j,i)=0.
      enddo
      enddo
Compute the angle of rotation in the reverse direction
      do k=1,m
      theta=float(1-k)
Back_project projection
      do i=1,m
      do j=1,m
      x(j,i)=b(j,k)
      enddo
      enddo
Rotate x by angle at which projection was taken
      call rotate_nearest_neig(x,m,y,l,theta)
Add result to previous back_projection function
      do i=1,l
      do j=1,l
      sum(j,i)=sum(j,i)+y(j,i)
      enddo
      enddo

      enddo
Window the back_projection function to extract reconstruction
(size n X n) and write it to output
      do i=1,n
      do j=1,n
      a(j,i)=sum(j+ln,i+ln)
      enddo
      enddo

      return
      end

```

```

subroutine bpr_four(b,m,a,n)
real a(256,256),b(256,256),xx(256,256)

```

```

*****

```

Function: this routine back-projects the data stored in b to reconstruct the image a.

External subroutines: Rotate_nearest_neig

Internal subroutines: None

Parameters: b _ input, size m X m

a _ output, size n X n

```

*****

```

x, y and sum are working space

```

real x(256,256),y(256,256)

```

```

real sum(256,256)

```

Send message to user 'Computing back_projection ...'

```

write(6,1)

```

```

format(1x,'Computing back_projection...'$)

```

Compute size of image in which b will be rotated to prevent corner clipping.

```

l=nint(sqrt(2.)*float(m))

```

Compute centers of arrays to be used later

```

nn=n/2+1

```

```

ll=l/2+1

```

```

ln=ll-nn

```

Initialize working space

```

do i=1,l

```

```

do j=1,l

```

```

x(j,i)=0.

```

```

y(j,i)=0.

```

```

sum(j,i)=0.

```

```

a(j,i)=0.

```

```

xx(j,i)=0.

```

```

enddo

```

```

enddo

```

Compute the angle of rotation in the reverse direction

```

do k=1,m

```

```

theta=float(1-k)

```

Back_project projection

```

do i=1,m

```

```

do j=1,m

```

```

x(j,i)=b(j,k)

```

```

enddo

```

```

enddo

```

Rotate x by angle at which projection was taken

```

call rotate_four(x,m,y,l,theta)

```

Add result to previous back_projection function

```

do i=1,l

```

```

do j=1,l

```

```

sum(j,i)=sum(j,i)+y(j,i)

```

```

enddo

```

```

enddo

```

```

enddo

```

Window the back_projection function to extract reconstruction (size n X n) and write it to output

```

do i=1,n

```

```

do j=1,n

```

```

a(j,i)=sum(j+ln,i+ln)

```

```

enddo

```

```

enddo

```

```

return

```

```

end

```

```

subroutine bpr_four_neig(b,m,a,n)
real a(256,256),b(256,256),xx(256,256)

```

```

*****

```

Function: this routine back-projects the data stored in b to reconstruct the image a.

External subroutines: Rotate_four_neig

Internal subroutines: None

Parameters: b _ input, size m X m

a _ output, size n X n

```

*****

```

x, y and sum are working space

```

real x(256,256),y(256,256)

```

```

real sum(256,256)

```

Send message to user 'Computing back_projection ...'

```

write(6,1)

```

```

format(1x,'Computing back_projection...'$)

```

Compute size of image in which b will be rotated to prevent corner clipping.

```

l=nint(sqrt(2.)*float(m))

```

Compute centers of arrays to be used later

```

nn=n/2+1

```

```

ll=l/2+1

```

```

ln=ll-nn

```

Initialize working space

```

do i=1,l

```

```

do j=1,l

```

```

x(j,i)=0.

```

```

y(j,i)=0.

```

```

sum(j,i)=0.

```

```

a(j,i)=0.

```

```

xx(j,i)=0.

```

```

enddo

```

```

enddo

```

Compute the angle of rotation in the reverse direction

```

do k=1,m

```

```

theta=float(1-k)

```

Back_project projection

```

do i=1,m

```

```

do j=1,m

```

```

x(j,i)=b(j,k)

```

```

enddo

```

```

enddo

```

Rotate x by angle at which projection was taken

```

call rotate_four_neig(x,m,y,l,theta)

```

Add result to previous back_projection function

```

do i=1,l

```

```

do j=1,l

```

```

sum(j,i)=sum(j,i)+y(j,i)

```

```

enddo

```

```

enddo

```

```

enddo

```

Window the back_projection function to extract reconstruction (size n X n) and write it to output

```

do i=1,n

```

```

do j=1,n

```

```

a(j,i)=sum(j+ln,i+ln)

```

```

enddo

```

```

enddo

```

```

return

```

```

end

```

```

subroutine bpr_eight(b,m,a,n)
real a(256,256),b(256,256),xx(256,256)

```

```

*****

```

Function: this routine back-projects the data stored in b to reconstruct the image a.

External subroutines: Rotate_nearest_neig

Internal subroutines: None

Parameters: b _ input, size m X m

a _ output, size n X n

```

*****

```

x, y and sum are working space

```

real x(256,256),y(256,256)

```

```

real sum(256,256)

```

Send message to user 'Computing back_projection ...'

```

write(6,1)

```

```

format(1x,'Computing back_projection...'$)

```

Compute size of image in which b will be rotated to prevent corner clipping.

```

l=nint(sqrt(2.)*float(m))

```

Compute centers of arrays to be used later

```

nn=n/2+1

```

```

ll=l/2+1

```

```

ln=ll-nn

```

Initialize working space

```

do i=1,l

```

```

do j=1,l

```

```

x(j,i)=0.

```

```

y(j,i)=0.

```

```

sum(j,i)=0.

```

```

a(j,i)=0.

```

```

xx(j,i)=0.

```

```

enddo

```

```

enddo

```

Compute the angle of rotation in the reverse direction

```

do k=1,m

```

```

theta=float(1-k)

```

Back_project projection

```

do i=1,m

```

```

do j=1,m

```

```

x(j,i)=b(j,k)

```

```

enddo

```

```

enddo

```

Rotate x by angle at which projection was taken

```

call rotate_eight(x,m,y,l,theta)

```

Add result to previous back_projection function

```

do i=1,l

```

```

do j=1,l

```

```

sum(j,i)=sum(j,i)+y(j,i)

```

```

enddo

```

```

enddo

```

```

enddo

```

Window the back_projection function to extract reconstruction (size n X n) and write it to output

```

do i=1,n

```

```

do j=1,n

```

```

a(j,i)=sum(j+ln,i+ln)

```

```

enddo

```

```

enddo

```

```

return

```

```

end

```

```

      subroutine bpr_eight_neig(b,m,a,n)
      real a(256,256),b(256,256),xx(256,256)
      *****
Function: this routine back-projects the
      data stored in b to reconstruct
      the image a.
External subroutines: Rotate_eight_neig
Internal subroutines: None
Parameters: b _ input, size m X m
      a _ output, size n X n
      *****
x, y and sum are working space
      real x(256,256),y(256,256)
      real sum(256,256)
Send message to user 'Computing back_projection ...'

      write(6,1)
      format(1x,'Computing back_projection...'$)

Compute size of image in which b will be rotated to prevent
corner clipping.
      l=nint(sqrt(2.)*float(m))
Compute centers of arrays to be used later
      nn=n/2+1
      ll=l/2+1
      ln=ll-nn
Initialize working space
      do i=1,l
      do j=1,l
      x(j,i)=0.
      y(j,i)=0.
      sum(j,i)=0.
      a(j,i)=0.
      xx(j,i)=0.
      enddo
      enddo
Compute the angle of rotation in the reverse direction
      do k=1,m
      theta=float(1-k)
Back_project projection
      do i=1,m
      do j=1,m
      x(j,i)=b(j,k)
      enddo
      enddo
Rotate x by angle at which projection was taken
      call rotate_eight_neig(x,m,y,l,theta)
Add result to previous back_projection function
      do i=1,l
      do j=1,l
      sum(j,i)=sum(j,i)+y(j,i)
      enddo
      enddo

      enddo
Window the back_projection function to extract reconstruction
(size n X n) and write it to output
      do i=1,n
      do j=1,n
      a(j,i)=sum(j+ln,i+ln)
      enddo
      enddo

      return
      end

```

```
subroutine hilbert(s,n)
```

```
*****
```

```
Function: This subroutine filters a Radon Transform  
obtained by straight projection (filtered)  
using the appropriate inverse filter, i.e.  
ABS(kx) where kx is the spatial frequency.
```

```
*****
```

```
real s(512),za(512)  
real zb(512),zc(512),zd(512)
```

```
initialize arrays.
```

```
do j=1,n  
za(j)=0.  
zb(j)=0.  
zc(j)=0.  
zd(j)=0.  
enddo
```

```
Generate working space.
```

```
do j=1,n  
za(j)=s(j)  
enddo
```

```
take Fourier transform of za  
call fftld(za,zb,n,-1)
```

```
set za and zb to ZERO for all negative values of j.  
(since ZERO frequency occurs at  $j=n/2+1$  ).
```

```
nn=n/2+1  
do j=1,nn-1  
za(j)=0.  
zb(j)=0.  
enddo  
do j=nn,n  
zc(j)=2.*za(j)  
zd(j)=2.*zb(j)  
enddo
```

```
call fftld(zc,zd,n,1)
```

```
write output
```

```
do j=1,n  
s(j)=zd(j)  
enddo  
return  
end
```

```
subroutine filtered_back_projection(a,n)
```

```
*****
```

```
Function: This subroutine filters a Radon Transform  
obtained by straight projection  
using the appropriate differentiation and  
applying Hilbert transform.
```

```
*****
```

```
real a(256,256),y(256,256),c(256,256),ze(256)
```

```
real zf(256),x(256,256)
```

```
Send message to user
```

```
' Computing backward Difference approximation ...'
```

```
write(6,1)
```

```
format(1x,'Computing Backward Difference approximation...'$)
```

```
initialize array c
```

```
do i=1,n
```

```
do j=1,n
```

```
c(j,i)=0.
```

```
enddo
```

```
enddo
```

```
do j=1,256
```

```
ze(j)=0.
```

```
zf(j)=0.
```

```
enddo
```

```
jframe=nint(float(256-n)/2.)
```

```
ze(n+jframe+1)=0.
```

```
do i=1,n
```

```
do j=1,n
```

```
ze(j+jframe)=a(j,i)
```

```
Differentiating ze using :
```

```
1_ Forward difference approximation.
```

```
zf(j+jframe)=ze(j+jframe+1)-ze(j+jframe)
```

```
2_ Backward difference approximation.
```

```
zf(j+jframe)=ze(j+jframe)-ze(j+jframe-1)
```

```
3_ Central difference approximation.
```

```
zf(j+jframe)=(ze(j+jframe+1)-ze(j+jframe-1))/2.
```

```
enddo
```

```
Computing Hilbert transform.
```

```
call hilbert(zf,256)
```

```
do j=1,n
```

```
c(j,i)=zf(j+jframe)
```

```
enddo
```

```
enddo
```

```
write output
```

```
do i=1,n
```

```
do j=1,n
```

```
a(j,i)=c(j,i)
```

```
enddo
```

```
enddo
```

```
return
```

```
end
```


subroutine back_projection_deconvolution(a,n)

Function: This subroutine deconvolves a reconstruction obtained by straight backprojection (un-filtered) using the appropriate inverse filter, i.e. $\sqrt{kx*kx+ky*ky}$ where kx and ky are the spatial frequencies.

```
real a(256,256),aa(256,256),xx(256,256)
real b(256,256),bb(256,256)
```

initialize array b

```
do i=1,n
do j=1,n
aa(j,i)=0.
bb(j,i)=0.
enddo
enddo
```

take Fourier transform of a

```
call fft2d(a,aa,n,-1)
```

multiply spectrum by inverse filter

```
nn=n/2+1
do i=1,n
do j=1,n
xk=float(i-nn)/float(n)
yk=float(j-nn)/float(n)
fil=sqrt(xk*xk+yk*yk)!value of inverse filter at (xk,yk)
b(j,i)=a(j,i)*fil
bb(j,i)=aa(j,i)*fil
enddo
enddo
```

```
call fft2d(b,bb,n,1)
```

write output

```
do i=1,n
do j=1,n
a(j,i)=b(j,i)
enddo
enddo
```

```
return
end
```

subroutine norm_signal(x1,n)

FUNCTION: returns an image with absolute maximum value of 1.

PARAMETERS

Input: x1 - signal
 n - size of signal

Output: x1 - normalized signal

INTERNAL SUBROUTINES: None

EXTERNAL SUBROUTINES: None

 real x1(256),s1(256),w1(256)

 real max1

Generate working space

 do i=1,n

 s1(i)=x1(i)

 enddo

Set the negative value to zero

 do i=1,n

 if(s1(i).lt.0.) s1(i)=0.

 enddo

Find the absolute maximum value of the image.

 max1=0.

 do i=1,n

 xm=s1(i)

 if(xm.gt.max1) max1=xm

 enddo

 write(6,*)'max1=',xm

Normalize the image.

 do i=1,n

 x1(i)=s1(i)/max1

 enddo

 return

 end

```

      subroutine project(a,b,n)
      *****
Function: This subroutine sums
      over all rows of an n X n array a giving 1D
projection of size n _ array b
      *****
      real a(256,256),b(256),s(256,256),z(256)
initialize arrays s & b
      do j=1,n
        b(j)=0.
        z(j)=0.
        do i=1,n
          s(j,i)=0.
        enddo
      enddo
set initial value for summing process and sum all pixels along
row i together
      do j=1,n
        s(j,1)=a(j,1)
        do i=2,n
          s(j,i)=s(j,i-1)+a(j,i)
        enddo
Assign value of sum at column j to output _b(j)
        b(j)=s(j,n)
      enddo

      return
      end

```

```
subroutine radon_gauss_noise(xy,m)
real xy(256,256),yt(256,256),ye(256,256)
```

```
*****
```

Function: This subroutine creates Gaussian noise with zero
with a selected Signal to Noise Ratio mean and
variance 1 and add it to the Radon Transform.

```
*****
```

```
real s(256,256),xl(256),wl(256)
real max(256)
```

Generate working space

```
do i=1,m
do j=1,m
s(j,i)=xy(j,i)
enddo
enddo
```

Set the negative value to zero

```
do i=1,m
do j=1,m
if(s(j,i).lt.0.) s(j,i)=0.
enddo
enddo
```

Find the absolute maximum value of the image.

```
do i=1,m
max(i)=0.
enddo
do i=1,m
do j=1,m
xl(i)=s(j,i)
if(xl(i).gt.max(i)) max(i)=xl(i)
enddo
enddo
```

create normalized Gaussian noise image.

```
do i=1,m
iseed=i*10
call gnoise (wl,m,iseed)
call norm_signal (wl,m)
do j=1,m
ye(j,i)=wl(j)
enddo
enddo
```

Add Radon transform image to noise image.

```
write (6,*) 'enter Signal to Noise Ratio:'
read(5,*) snr
do i=1,m
do j=1,m
yt(j,i)=s(j,i) * snr + ye(j,i)*max(i)
enddo
enddo
do i=1,m
do j=1,m
xy(j,i)=yt(j,i)
enddo
enddo

return
end
```

```

subroutine rtr nearest(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)

```

Function: This subroutine computes the Radon Transform of
n X n image.

External subroutines: Rotate_nearest_neig
Project

Internal subroutines: None

Parameters: x - n X n input array
y - m X m output array

Send message to user 'Computing Radon Transform...'

```

write(6,1)
format(1x,'Computing Radon Transform...'$)

```

initialize array y

```

do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo

```

Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.

```

do i=1,m
theta=float(i-1)
call rotate_nearest(x,n,w,m,theta)
call project(w,z,m)

```

Stack projections together to form image of "Radon space"
data.

```

do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```

```

      subroutine rtr_nearest_neig(x,n,y,m)
      real x(256,256),y(256,256),xy(256,256)
      real w(256,256),z(256)
*****
Function: This subroutine computes the Radon Transform of
      n X n image.
External subroutines: Rotate_nearest_neig
      Project
Internal subroutines: None
Parameters: x - n X n input array
      y - m X m output array
*****
Send message to user 'Computing Radon Transform...'

      write(6,1)
      format(1x,'Computing Radon Transform...'$)
Initialize array y
      do j=1,m
      do i=1,m
      y(j,i)=0.
      enddo
      enddo
Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.
      do i=1,m
      theta=float(i-1)
      call rotate_nearest_neig(x,n,w,m,theta)
      call project(w,z,m)
Stack projections together to form image of "Radon space"
data.
      do j=1,m
      y(j,i)=z(j)
      enddo
      enddo
      return
      end

```

```

subroutine rtr_lrp_neig(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
*****
Function: This subroutine computes the Radon Transform of
n X n image.
External subroutines: Rotate_nearest_neig
Project
Internal subroutines: None
Parameters: x - n X n input array
y - m X m output array
*****
Send message to user 'Computing Radon Transform...'

write(6,1)
format(1x,'Computing Radon Transform...'$)
initialize array y
do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo
Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.
do i=1,m
theta=float(i-1)
call rotate_lrp_neig(x,n,w,m,theta)
call project(w,z,m)
Stack projections together to form image of "Radon space"
data.
do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```

```

subroutine rtr_2rp_neig(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
*****
Function: This subroutine computes the Radon Transform of
         n X n image.
External subroutines: Rotate_nearest_neig
                     Project
Internal subroutines: None
Parameters: x - n X n input array
            y - m X m output array
*****
Send message to user 'Computing Radon Transform...'

      write(6,1)
      format(1x,'Computing Radon Transform...'$)
initialize array y
do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo
Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.
do i=1,m
theta=float(i-1)
call rotate_2rp_neig(x,n,w,m,theta)
call project(w,z,m)
Stack projections together to form image of "Radon space"
data.
do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```



```

subroutine rtr_four(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
*****
Function: This subroutine computes the Radon Transform of
n X n image.
External subroutines: Rotate_nearest_neig
Project
Internal subroutines: None
Parameters: x - n X n input array
y - m X m output array
*****
Send message to user 'Computing Radon Transform...'

write(6,1)
format(1x,'Computing Radon Transform...'$)
initialize array y
do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo
Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.
do i=1,m
theta=float(i-1)
call rotate_four(x,n,w,m,theta)
call project(w,z,m)
Stack projections together to form image of "Radon space"
data.
do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```

```

subroutine rtr_four_neig(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)

```

```

*****

```

```

Function: This subroutine computes the Radon Transform of
n X n image.

```

```

External subroutines: Rotate_four_neig
Project

```

```

Internal subroutines: None

```

```

Parameters: x - n X n input array
y - m X m output array

```

```

*****

```

```

Send message to user 'Computing Radon Transform...'

```

```

write(6,1)
format(1x,'Computing Radon Transform...'$)

```

```

initialize array y

```

```

do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo

```

```

Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.

```

```

do i=1,m
theta=float(i-1)
call rotate_four_neig(x,n,w,m,theta)
call project(w,z,m)

```

```

Stack projections together to form image of "Radon space"
data.

```

```

do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```

```

      subroutine rtr_eight(x,n,y,m)
      real x(256,256),y(256,256),xy(256,256)
      real w(256,256),z(256)
      *****
Function: This subroutine computes the Radon Transform of
      n X n image.
External subroutines: Rotate_nearest_neig
                      Project
Internal subroutines: None
Parameters: x - n X n input array
            y - m X m output array
      *****
Send message to user 'Computing Radon Transform...'

      write(6,1)
      format(1x,'Computing Radon Transform...'$)
Initialize array y
      do j=1,m
      do i=1,m
      y(j,i)=0.
      enddo
      enddo
Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.
      do i=1,m
      theta=float(i-1)
      call rotate_eight(x,n,w,m,theta)
      call project(w,z,m)
Stack projections together to form image of "Radon space"
data.
      do j=1,m
      y(j,i)=z(j)
      enddo
      enddo
      return
      end

```

```

subroutine rtr_eight_neig(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
*****
Function: This subroutine computes the Radon Transform of
n X n image.
External subroutines: Rotate_eight_neig
Project
Internal subroutines: None
Parameters: x - n X n input array
y - m X m output array
*****
Send message to user 'Computing Radon Transform...'

write(6,1)
format(1x,'Computing Radon Transform...'$)
initialize array y
do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo
Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.
do i=1,m
theta=float(i-1)
call rotate_eight_neig(x,n,w,m,theta)
call project(w,z,m)
Stack projections together to form image of "Radon space"
data.
do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```

```
subroutine rotate_nearest(a,n,b,m,theta)
```

```
real a(256,256),b(256,256),w(256,256)
```

```
*****
```

```
FUNCTION : This subroutine rotates an n X n array a  
through an angle theta (given in degree)  
anticlockwise placing the result at the  
center of an array mxm b
```

```
PARAMETERS: a _ Input image of n X n size
```

```
          b _ Rotated image of m X m size
```

```
EXTERNAL SUBROUTINE :None
```

```
INTERNAL SUBROUTINE :None
```

```
*****
```

```
convert theta into radians
```

```
pi = 4. * atan(1.)
```

```
theta = theta * (pi/180.)
```

```
costh = cos(theta)
```

```
sinth = sin(theta)
```

```
initial b
```

```
do i = 1,m
```

```
do j = 1,m
```

```
b(j,i) = 0.
```

```
w(j,i) = 0.
```

```
enddo
```

```
enddo
```

```
compute new coordinates (xd,yd) of rotate array
```

```
nn = n/2 + 1
```

```
mm = m/2 + 1
```

```
start process
```

```
do j = 1,n
```

```
do i = 1,n
```

```
Exclude the rotation of zero value of array a
```

```
if (a(j,i).eq.0.0) goto 100
```

```
generate 4-quadrant cartesian plan
```

```
y = float(j-nn)
```

```
x = float(i-nn)
```

```
xd = x * costh - y * sinth
```

```
yd = x * sinth + y * costh
```

```
compute both real and integer coordinates of array b
```

```
associated with the above transformation for later use
```

```
kx = mm+ nint(xd)
```

```
ky = mm+ nint(yd)
```

```
assign value of a at (j,i) to the nearest_point cell
```

```
of array b in wich the transformed coordinates
```

```
lie.
```

```
b(ky,kx) = a(j,i)
```

```
00      enddo
```

```
      enddo
```

```
      return
```

```
      end
```

```
subroutine rotate eight(a,n,b,m,theta)
```

```
real a(256,256),b(256,256),w(256,256)
```

```
*****
```

```
FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A  
THROUGH AN ANGLE THETA (given in degree)  
anticlockwise placing the result at the  
center of an array mxm b
```

```
PARAMETERS: a - Input image of n X n size
```

```
             b - Rotated image of m X m size
```

```
EXTERNAL SUBROUTINE :None
```

```
INTERNAL SUBROUTINE :None
```

```
*****
```

```
convert theta into radians
```

```
pi = 4. * atan(1.)
```

```
theta = theta * (pi/180.)
```

```
costh = cos(theta)
```

```
sinth = sin(theta)
```

```
initial b
```

```
do i = 1,m
```

```
do j = 1,m
```

```
b(j,i) = 0.
```

```
w(j,i) = 0.
```

```
enddo
```

```
enddo
```

```
compute new coordinates (xd,yd) of rotate array
```

```
nn = n/2 + 1
```

```
mm = m/2 + 1
```

```
start process
```

```
do j = 1,n
```

```
do i = 1,n
```

```
Exclude the rotation of zero value of array a
```

```
if (a(j,i).eq.0.0) goto 100
```

```
generate 4-quadrant cartesian plan
```

```
y = float(j-nn)
```

```
x = float(i-nn)
```

```
xd = x * costh - y * sinth
```

```
yd = x * sinth + y * costh
```

```
compute both real and integer coordinates of array b  
associated with the above transformation for later use
```

```
kx = mm+ nint(xd)
```

```
ky = mm+ nint(yd)
```

```
assign value of a at (j,i) to 9-point cell
```

```
of array b in which the transformed coordinates
```

```
lie.
```

```
b(ky+1,kx-1) = a(j,i)
```

```
b(ky,kx-1) = a(j,i)
```

```
b(ky-1,kx-1) = a(j,i)
```

```
b(ky+1,kx) = a(j,i)
```

```
b(ky,kx) = a(j,i)
```

```
b(ky-1,kx) = a(j,i)
```

```
b(ky+1,kx+1) = a(j,i)
```

```
b(ky,kx+1) = a(j,i)
```

```
b(ky-1,kx+1) = a(j,i)
```

```
enddo
```

```
enddo
```

```
return
```

```
end
```

```

        subroutine rotate eight neig (a,n,b,m,theta)
        real a(256,256),b(256,256),w(256,256)
        integer xu,yu,xd,yd,dul,eul,du2,eu2,vu(256),hu(256),ku(256)
        integer gu(256)
Function: This subroutine rotates an nXn array a through an angle theta
        (given in degrees ) placing the result at the center of an mXm
        array b.
Convert theta into radiants.
        pi=4.*atan(1.)
        theta=theta*(pi/180.)
c initialize b
        do i=1,m
        do j=1,m
        b(j,i)=0
        enddo
        enddo
c Compute centers of array a and b
        nn=n/2+1
        mm=m/2+1
c Compute the integer of horizontal and vertical displacement along the x and
y-axis which corresponds to n-pixel movement along the u-axis.
        xu=nint(nn*cos(theta))
        yu=nint(nn*sin(theta))
        if (xu.lt.0) then
        ml=-1
        else
        ml=1
        endif
        if (yu.lt.0) then
        nl=1
        else
        nl=-1
        endif
c Compute the rotation parameters
        dul=2*abs(yu)
        du2=2*(abs(yu)-nn)
        eul=2*abs(xu)
        eu2=2*(abs(xu)-nn)
        vu(1)=2*abs(yu)-nn
        hu(1)=2*abs(xu)-nn
        ku(0)=0
        gu(0)=0
        do i=1,nn
        if (hu(i).ge.0) then
            hu(i+1)=hu(i)+eu2
            gu(i)=gu(i-1)+1
            if (vu(i).ge.0) then
                vu(i+1)=vu(i)+du2
                ku(i)=ku(i-1)+1
            else
                vu(i+1)=vu(i)+dul
                ku(i)=ku(i-1)
            endif
        else
            hu(i+1)=hu(i)+eul
            gu(i)=gu(i-1)
            if (vu(i).ge.0) then
                vu(i+1)=vu(i)+du2
                ku(i)=ku(i-1)+1
            else
                vu(i+1)=vu(i)+dul
                ku(i)=ku(i-1)
            endif
        endif
        enddo
c Compute new coordinates (xd,yd) of rotated array and assign the value of a at
(i,j) to the appropriate cell of b which encloses these coordinates.
c Generate 4-quadrant cartesian plane.

```

```

do i=1,nn
ku(-i)=-ku(i)
gu(-i)=-gu(i)
enddo
c Start process

do j=1,n
do i=1,n

c Exclude the rotation of zero value of array a

if (a(i,j).eq.0.0) goto 100
c Employ coordinate transformation for rotation using the integer parameters.
yd=m1*gu(i-nn)-n1*ku(j-nn)
xd=m1*gu(j-nn)+n1*ku(i-nn)
c Compute the integer coordinates of array b associated with the above
c transformation.
kx=mm+xd
ky=mm+yd
c Assign value of a at (j,i) to the nearest cell of array b in which the
c transformed coordinates lie.
b(ky+1,kx-1)=a(i,j)
b(ky,kx-1)=a(i,j)
b(ky-1,kx-1)=a(i,j)
b(ky+1,kx)=a(i,j)
b(ky,kx)=a(i,j)
b(ky-1,kx)=a(i,j)
b(ky+1,kx+1)=a(i,j)
b(ky,kx+1)=a(i,j)
b(ky-1,kx+1)=a(i,j)

c
100

enddo
enddo
return
end

```



```

      subroutine rotate four(a,n,b,m,theta)
      real a(256,256),b(256,256),w(256,256)
      *****
FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
      THROUGH AN ANGLE THETA (given in degree)
      anticlockwise placing the result at the
      center of an array mxm b
PARAMETERS: a -- Input image of n X n size
      b -- Rotated image of m X m size
EXTERNAL SUBROUTINE :None
INTERNAL SUBROUTINE :None
      *****

      convert theta into radians

      pi = 4. * atan(1.)
      theta = theta * (pi/180.)
      costh = cos(theta)
      sinth = sin(theta)

initial b
      do i = 1,m
      do j = 1,m
      b(j,i) = 0.
      w(j,i) = 0.
      enddo
      enddo

compute new coordinates (xd,yd) of rotate array

      nn = n/2 + 1
      mm = m/2 + 1
Compensation of the center position for rotation.
      xcomp=nn+nn*(sinth - costh)
      ycomp=nn-nn*(sinth + costh)

start process

      do j = 1,n
      do i = 1,n

Exclude the rotation of zero value of array a

      if (a(j,i).eq.0.0) goto 100
generate 4-quadrant cartesian plan
      y = float(j-nn)
      x = float(i-nn)
      xd = x * costh - y * sinth
      yd = x * sinth + y * costh

compute both real and integer coordinates of array b
associated with the above transformation for later use
      kx = mm+ nint(xd)
      ky = mm+ nint(yd)
assign value of a at (j,i) to 9-point cell
of array b in wich the transformed coordinates
lie.
      b(ky,kx-1) = a(j,i)
      b(ky+1,kx) = a(j,i)
      b(ky,kx) = a(j,i)
      b(ky-1,kx) = a(j,i)
      b(ky,kx+1) = a(j,i)
      enddo
      enddo
      return
      end
100

```

```

subroutine rotate four_neig (a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
integer xu,yu,xd,yd,dul,eul,du2,eu2,vu(256),hu(256),ku(256)
integer gu(256)

```

Function: This subroutine rotates an nXn array a through an angle theta (given in degrees) placing the result at the center of an mXm array b.

Convert theta into radiants.

```

pi=4.*atan(1.)
theta=theta*(pi/180.)

```

initialize b

```

do i=1,m
do j=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo

```

Compute centers of array a and b

```

nn=n/2+1
mm=m/2+1

```

Compute the integer of horizontal and vertical displacement along the x and y-axis which corresponds to n-pixel movement along the u-axis.

```

xu=nint(nn*cos(theta))
yu=nint(nn*sin(theta))
if (xu.lt.0) then
ml=-1
else
ml=1
endif
if (yu.lt.0) then
nl=1
else
nl=-1
endif

```

Compute the rotation parameters

```

dul=2*abs(yu)
du2=2*(abs(yu)-nn)
eul=2*abs(xu)
eu2=2*(abs(xu)-nn)
vu(1)=2*abs(yu)-nn
hu(1)=2*abs(xu)-nn
ku(0)=0
gu(0)=0
do i=1,nn
if (hu(i).ge.0) then
hu(i+1)=hu(i)+eu2
gu(i)=gu(i-1)+1
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
else
hu(i+1)=hu(i)+eul
gu(i)=gu(i-1)
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
endif
enddo

```

Compute new coordinates (xd,yd) of rotated array and assign the value of a at (i,j) to the appropriate cell of b which encloses these coordinates.

```

c Generate 4-quadrant cartesian plane.
      do i=1,nn
        ku(-i)=-ku(i)
        gu(-i)=-gu(i)
      enddo
c Start process
      do j=1,n
        do i=1,n
c Exclude the rotation of zero value of array a
          if (a(i,j).eq.0.0) goto 100
c Employ coordinate transformation for rotation using the integer parameters.
          yd=m1*gu(i-nn)-n1*ku(j-nn)
          xd=m1*gu(j-nn)+n1*ku(i-nn)
c Compute the integer coordinates of array b associated with the above
c transformation.
          kx=mm+xd
          ky=mm+yd
c Assign value of a at (i,j) to the nearest cell of array b in which the
c transformed coordinates lie.
          b(ky+1,kx-1)=a(i,j)
          b(ky,kx-1)=a(i,j)
          b(ky-1,kx-1)=a(i,j)
          b(ky+1,kx)=a(i,j)
          b(ky,kx)=a(i,j)
          b(ky-1,kx)=a(i,j)
          b(ky+1,kx+1)=a(i,j)
          b(ky,kx+1)=a(i,j)
          b(ky-1,kx+1)=a(i,j)
100      enddo
        enddo
      return
    end

```

```

subroutine ift_nearest(a,m,b,n)
real a(256,256),b(256,256),xy(256,256)
real r1(256,256),il(256,256),xx(256)

```

```

*****

```

```

FUNCTION : This subroutine rotates an slice n X 1 array a
           through an angle theta (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b

```

```

PARAMETERS: a _ Input image of n X n size
             b _ Rotated image of m X m size

```

```

EXTERNAL SUBROUTINE :None

```

```

INTERNAL SUBROUTINE :None

```

```

*****

```

```

real za(256),zb(256),zc(256),zd(256),c(256,256)
real re(256,256),im(256,256)

```

```

Send message to user 'computing reconstruction using centre slice...'

```

```

write(6,1)
format(1x,'Computing centre_slice_reconstruction...'$)

```

```

initial arrays

```

```

do i = 1,256
do j = 1,256
r1(j,i) = 0.
il(j,i) = 0.
enddo
enddo

```

```

initialize array c

```

```

do i=1,m
do j=1,m
c(j,i)=0.
enddo
enddo

```

```

do i=1,m
do j=1,m
c(j,i)=a(j,i)
enddo
enddo

```

```

do j=1,256
za(j)=0.
zb(j)=0.
zc(j)=0.
zd(j)=0.
enddo

```

```

take Fourier transform of a

```

```

nn1=256/2+1
jframe=nint(float(256-m)/2.)
do i=1,m
do j=1,m
za(j+jframe)=c(j,i)
enddo
call fft1d(za,zb,256,-1)

```

```

do j=1,m
re(j,i)=za(j+jframe)
im(j,i)=zb(j+jframe)
enddo
enddo

```

```

call centre_slice_nearest(re,m)
call centre_slice_nearest(im,m)

```

```

call deconvolution (re,m)

```

```
call deconvolution (im,m)

open (unit=1,file='ft.181',status='new')
format(4(1x,f14.4))
write(1,101)((b(k,j),k=1,181),j=1,181)
close(1)
```

```
do i=1,m
do j=1,m
r1(j+jframe,i+jframe)=re(j,i)
il(j+jframe,i+jframe)=im(j,i)
enddo
enddo
```

```
ake 2DI Fourier transform of re,im arrays
call fft2d(r1,il,256,1)
```

```
iframe=nint(float(256-n)/2.)
do i=1,n
do j=1,n
b(j,i)=r1(j+iframe,i+iframe)
enddo
enddo
```

```
return
end
```

```

      subroutine ift_nearest_neig(a,m,b,n)
      real a(256,256),b(256,256),xy(256,256)
      real r1(256,256),il(256,256),xx(256)
*****
FUNCTION : This subroutine rotates an slice n X 1 array a
           through an angle theta (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b
PARAMETERS: a _ Input image of n X n size
           b _ Rotated image of m X m size
EXTERNAL SUBROUTINE :None
INTERNAL SUBROUTINE :None
*****
      real za(256),zb(256),zc(256),zd(256),c(256,256)
      real re(256,256),im(256,256)
end message to user 'computing reconstruction using centre slice...'

      write(6,1)
      format(1x,'Computing centre_slice_reconstruction...'$)

initial arrays
      do i = 1,256
      do j = 1,256
      r1(j,i) = 0.
      il(j,i) = 0.
      enddo
      enddo

initialize array c

      do i=1,m
      do j=1,m
      c(j,i)=0.
      enddo
      enddo

      do i=1,m
      do j=1,m
      c(j,i)=a(j,i)
      enddo
      enddo

      do j=1,256
      za(j)=0.
      zb(j)=0.
      zc(j)=0.
      zd(j)=0.
      enddo

ke Fourier transform of a

      nn1=256/2+1
      jframe=nint(float(256-m)/2.)
      do i=1,m
      do j=1,m
      za(j+jframe)=c(j,i)
      enddo
      call fft1d(za,zb,256,-1)

      do j=1,m
      re(j,i)=za(j+jframe)
      im(j,i)=zb(j+jframe)
      enddo
      enddo

      call centre_slice_nearest_neig(re,m)
      call centre_slice_nearest_neig(im,m)

      call deconvolution (re,m)

```

```
call deconvolution (im,m)
```

```
open (unit=1,file='ft.181',status='new')  
format(4(1x,f14.4))  
write(1,101)((b(k,j),k=1,181),j=1,181)  
close(1)
```

```
do i=1,m  
do j=1,m  
r1(j+jframe,i+jframe)=re(j,i)  
il(j+jframe,i+jframe)=im(j,i)  
enddo  
enddo
```

```
ake 2DI Fourier transform of re,im arrays  
call fft2d(r1,il,256,1)
```

```
iframe=nint(float(256-n)/2.)  
do i=1,n  
do j=1,n  
b(j,i)=r1(j+iframe,i+iframe)  
enddo  
enddo
```

```
return  
end
```

```

subroutine rotate_2rp_neig (a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
integer xu,yu,xd,yd,dul,eul,du2,eu2,vu(256),hu(256),ku(256)
integer gu(256)

```

Function: This subroutine rotates an nXn array a through an angle theta (given in degrees) placing the result at the center of an mXm array b.

Convert theta into radiants.

```

pi=4.*atan(1.)
theta=theta*(pi/180.)

```

Initialize b

```

do i=1,m
do j=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo

```

Compute centers of array a and b

```

nn=n/2+1
mm=m/2+1

```

Compute the integer of horizontal and vertical displacement along the x and y-axis which corresponds to n-pixel movement along the u-axis.

```

xu=nint(nn*cos(theta))
yu=nint(nn*sin(theta))
if (xu.lt.0) then
ml=-1
else
ml=1
endif
if (yu.lt.0) then
nl=1
else
nl=-1
endif

```

Compute the rotation parameters

```

dul=2*abs(yu)
du2=2*(abs(yu)-nn)
eul=2*abs(xu)
eu2=2*(abs(xu)-nn)
vu(1)=2*abs(yu)-nn
hu(1)=2*abs(xu)-nn
ku(0)=0
gu(0)=0
do i=1,nn
if (hu(i).ge.0) then
hu(i+1)=hu(i)+eu2
gu(i)=gu(i-1)+1
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
else
hu(i+1)=hu(i)+eul
gu(i)=gu(i-1)
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
endif
enddo

```

Compute new coordinates (xd,yd) of rotated array and assign the value of a at (j) to the appropriate cell of b which encloses these coordinates.

Generate 4-quadrant cartesian plane.

```
do i=1,nn
ku(-i)=-ku(i)
gu(-i)=-gu(i)
enddo
```

Start process

```
do j=1,n
do i=1,n
```

Exclude the rotation of zero value of array a

```
if (a(i,j).eq.0.0) goto 100
```

Employ coordinate transformation for rotation using the integer parameters.

```
yd=m1*gu(i-nn)-n1*ku(j-nn)
xd=m1*gu(j-nn)+n1*ku(i-nn)
```

Compute the integer coordinates of array b associated with the above transformation.

```
kx=mm+xd
ky=mm+yd
```

Assign value of a at (i,j) to the nearest cell of array b in which the transformed coordinates lie.

```
b(ky,kx-1)=a(i,j)
b(ky+1,kx)=a(i,j)
b(ky,kx)=a(i,j)
b(ky-1,kx)=a(i,j)
b(ky,kx+1)=a(i,j)
```

```
enddo
enddo
return
end
```

```

subroutine rotate_lrp_neig (a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
integer xu,yu,xd,yd,dul,eul,du2,eu2,vu(256),hu(256),ku(256)
integer gu(256)

```

Function: This subroutine rotates an nXn array a through an angle theta (given in degrees) placing the result at the center of an mXm array b.

Convert theta into radiants.

```

pi=4.*atan(1.)
theta=theta*(pi/180.)

```

Initialize b

```

do i=1,m
do j=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo

```

Compute centers of array a and b

```

nn=n/2+1
mm=m/2+1

```

Compute the integer of horizontal and vertical displacement along the x and y-axis which corresponds to n-pixel movement along the u-axis.

```

xu=nint(nn*cos(theta))
yu=nint(nn*sin(theta))
if (xu.lt.0) then
ml=-1
else
ml=1
endif
if (yu.lt.0) then
nl=1
else
nl=-1
endif

```

Compute the rotation parameters

```

dul=2*abs(yu)
du2=2*(abs(yu)-nn)
eul=2*abs(xu)
eu2=2*(abs(xu)-nn)
vu(1)=2*abs(yu)-nn
hu(1)=2*abs(xu)-nn
ku(0)=0
gu(0)=0
do i=1,nn
if (hu(i).ge.0) then
hu(i+1)=hu(i)+eu2
gu(i)=gu(i-1)+1
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
else
hu(i+1)=hu(i)+eul
gu(i)=gu(i-1)
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
endif
enddo

```

Compute new coordinates (xd,yd) of rotated array and assign the value of a at (i,j) to the appropriate cell of b which encloses these coordinates.

Generate 4-quadrant cartesian plane.

```
do i=1,nn
ku(-i)=-ku(i)
gu(-i)=-gu(i)
enddo
```

Start process

```
do j=1,n
do i=1,n
```

Exclude the rotation of zero value of array a

```
if (a(i,j).eq.0.0) goto 100
```

Employ coordinate transformation for rotation using the integer parameters.

```
yd=m1*gu(i-nn)-n1*ku(j-nn)
xd=m1*gu(j-nn)+n1*ku(i-nn)
```

Compute the integer coordinates of array b associated with the above transformation.

```
kx=mm+xd
ky=mm+yd
```

Assign value of a at (i,j) to the nearest cell of array b in which the transformed coordinates lie.

```
b(ky,kx-1)=a(i,j)
b(ky+1,kx)=a(i,j)
b(ky,kx)=a(i,j)
b(ky-1,kx)=a(i,j)
b(ky,kx+1)=a(i,j)
```

```
enddo
enddo
return
end
```

```

subroutine rotate_lrp_neig (a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
integer xu,yu,xd,yd,du1,eul,du2,eu2,vu(256),hu(256),ku(256)
integer gu(256)

```

Function: This subroutine rotates an nXn array a through an angle theta (given in degrees) placing the result at the center of an mXm array b.

Convert theta into radiants.

```

pi=4.*atan(1.)
theta=theta*(pi/180.)

```

Initialize b

```

do i=1,m
do j=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo

```

compute centers of array a and b

```

nn=n/2+1
mm=m/2+1

```

compute the integer of horizontal and vertical displacement along the x and y-axis which corresponds to n-pixel movement along the u-axis.

```

xu=nint(nn*cos(theta))
yu=nint(nn*sin(theta))
if (xu.lt.0) then
ml=-1
else
ml=1
endif
if (yu.lt.0) then
nl=1
else
nl=-1
endif

```

compute the rotation parameters

```

du1=2*abs(yu)
du2=2*(abs(yu)-nn)
eul=2*abs(xu)
eu2=2*(abs(xu)-nn)
vu(1)=2*abs(yu)-nn
hu(1)=2*abs(xu)-nn
ku(0)=0
gu(0)=0
do i=1,nn
if (hu(i).ge.0) then
hu(i+1)=hu(i)+eu2
gu(i)=gu(i-1)+1
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+du1
ku(i)=ku(i-1)
endif
else
hu(i+1)=hu(i)+eul
gu(i)=gu(i-1)
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+du1
ku(i)=ku(i-1)
endif
endif
enddo

```

compute new coordinates (xd,yd) of rotated array and assign the value of a at (i,j) to the appropriate cell of b which encloses these coordinates.

Generate 4-quadrant cartesian plane.

```
do i=1,nn
ku(-i)=-ku(i)
gu(-i)=-gu(i)
enddo
```

Start process

```
do j=1,n
do i=1,n
```

Exclude the rotation of zero value of array a

```
if (a(i,j).eq.0.0) goto 100
```

Employ coordinate transformation for rotation using the integer parameters.

```
yd=m1*gu(i-nn)-n1*ku(j-nn)
xd=m1*gu(j-nn)+n1*ku(i-nn)
```

Compute the integer coordinates of array b associated with the above transformation.

```
kx=mm+xd
ky=mm+yd
```

Assign value of a at (i,j) to the nearest cell of array b in which the transformed coordinates lie.

```
b(ky+1,kx-1)=a(i,j)
b(ky,kx-1)=a(i,j)
b(ky-1,kx-1)=a(i,j)
b(ky+1,kx)=a(i,j)
b(ky,kx)=a(i,j)
b(ky-1,kx)=a(i,j)
b(ky+1,kx+1)=a(i,j)
b(ky,kx+1)=a(i,j)
b(ky-1,kx+1)=a(i,j)
```

```
enddo
enddo
return
end
```

```

      subroutine ift_lrp_neig(a,m,b,n)
      real a(256,256),b(256,256),xy(256,256)
      real r1(256,256),il(256,256),xx(256)
*****
FUNCTION : This subroutine rotates an slice n X 1 array a
           through an angle theta (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b
PARAMETERS: a _ Input image of n X n size
           b _ Rotated image of m X m size
EXTERNAL SUBROUTINE :None
INTERNAL SUBROUTINE :None
*****
      real za(256),zb(256),zc(256),zd(256),c(256,256)
      real re(256,256),im(256,256)
Send message to user 'computing reconstruction using centre slice...'

      write(6,1)
      format(1x,'Computing centre_slice_reconstruction...'$)

initial arrays
      do i = 1,256
      do j = 1,256
      r1(j,i) = 0.
      il(j,i) = 0.
      enddo
      enddo

initialize array c

      do i=1,m
      do j=1,m
      c(j,i)=0.
      enddo
      enddo

      do i=1,m
      do j=1,m
      c(j,i)=a(j,i)
      enddo
      enddo

      do j=1,256
      za(j)=0.
      zb(j)=0.
      zc(j)=0.
      zd(j)=0.
      enddo

ake Fourier transform of a

      nn1=256/2+1
      jframe=nint(float(256-m)/2.)
      do i=1,m
      do j=1,m
      za(j+jframe)=c(j,i)
      enddo
      call fft1d(za,zb,256,-1)

      do j=1,m
      re(j,i)=za(j+jframe)
      im(j,i)=zb(j+jframe)
      enddo
      enddo

      call centre_slice_lrp_neig(re,m)
      call centre_slice_lrp_neig(im,m)

      call deconvolution (re,m)

```

```
call deconvolution (im,m)
```

```
open (unit=1,file='ft.181',status='new')  
format(4(1x,f14.4))  
write(1,101)((b(k,j),k=1,181),j=1,181)  
close(1)
```

```
do i=1,m  
do j=1,m  
r1(j+jframe,i+jframe)=re(j,i)  
il(j+jframe,i+jframe)=im(j,i)  
enddo  
enddo
```

```
ake 2DI Fourier transform of re,im arrays  
call fft2d(r1,il,256,1)
```

```
iframe=nint(float(256-n)/2.)  
do i=1,n  
do j=1,n  
b(j,i)=r1(j+iframe,i+iframe)  
enddo  
enddo
```

```
return  
end
```

```

      subroutine ift_2rp_neig(a,m,b,n)
      real a(256,256),b(256,256),xy(256,256)
      real r1(256,256),il(256,256),xx(256)
*****
FUNCTION : This subroutine rotates an slice n X 1 array a
           through an angle theta (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b
PARAMETERS: a _ Input image of n X n size
           b _ Rotated image of m X m size
EXTERNAL SUBROUTINE :None
INTERNAL SUBROUTINE :None
*****
      real za(256),zb(256),zc(256),zd(256),c(256,256)
      real re(256,256),im(256,256)
end message to user 'computing reconstruction using centre slice...'

      write(6,1)
      format(1x,'Computing centre_slice_reconstruction...'$)

initial arrays
      do i = 1,256
      do j = 1,256
      r1(j,i) = 0.
      il(j,i) = 0.
      enddo
      enddo

initialize array c

      do i=1,m
      do j=1,m
      c(j,i)=0.
      enddo
      enddo

      do i=1,m
      do j=1,m
      c(j,i)=a(j,i)
      enddo
      enddo

      do j=1,256
      za(j)=0.
      zb(j)=0.
      zc(j)=0.
      zd(j)=0.
      enddo

like Fourier transform of a

      nn1=256/2+1
      jframe=nint(float(256-m)/2.)
      do i=1,m
      do j=1,m
      za(j+jframe)=c(j,i)
      enddo
      call fft1d(za,zb,256,-1)

      do j=1,m
      re(j,i)=za(j+jframe)
      im(j,i)=zb(j+jframe)
      enddo
      enddo

      call centre_slice_2rp_neig(re,m)
      call centre_slice_2rp_neig(im,m)

      call deconvolution (re,m)

```



```
call deconvolution (im,m)
```

```
open (unit=1,file='ft.181',status='new')  
format(4(1x,f14.4))  
write(1,101)((b(k,j),k=1,181),j=1,181)  
close(1)
```

```
do i=1,m  
do j=1,m  
r1(j+jframe,i+jframe)=re(j,i)  
il(j+jframe,i+jframe)=im(j,i)  
enddo  
enddo
```

```
ake 2DI Fourier transform of re,im arrays  
call fft2d(r1,il,256,1)
```

```
iframe=nint(float(256-n)/2.)  
do i=1,n  
do j=1,n  
b(j,i)=r1(j+iframe,i+iframe)  
enddo  
enddo
```

```
return  
end
```

```

      subroutine ift_four_neig(a,m,b,n)
      real a(256,256),b(256,256),xy(256,256)
      real r1(256,256),il(256,256),xx(256)
*****
FUNCTION : This subroutine rotates an slice n X 1 array a
           through an angle theta (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b
PARAMETERS: a _ Input image of n X n size
           b _ Rotated image of m X m size
EXTERNAL SUBROUTINE :None
INTERNAL SUBROUTINE :None
*****
      real za(256),zb(256),zc(256),zd(256),c(256,256)
      real re(256,256),im(256,256)
end message to user 'computing reconstruction using centre slice...'

      write(6,1)
      format(1x,'Computing centre_slice_reconstruction...'$)

initial arrays
      do i = 1,256
      do j = 1,256
      r1(j,i) = 0.
      il(j,i) = 0.
      enddo
      enddo

initialize array c

      do i=1,m
      do j=1,m
      c(j,i)=0.
      enddo
      enddo

      do i=1,m
      do j=1,m
      c(j,i)=a(j,i)
      enddo
      enddo

      do j=1,256
      za(j)=0.
      zb(j)=0.
      zc(j)=0.
      zd(j)=0.
      enddo

like Fourier transform of a

      nn1=256/2+1
      jframe=nint(float(256-m)/2.)
      do i=1,m
      do j=1,m
      za(j+jframe)=c(j,i)
      enddo
      call fft1d(za,zb,256,-1)

      do j=1,m
      re(j,i)=za(j+jframe)
      im(j,i)=zb(j+jframe)
      enddo
      enddo

      call centre_slice_four_neig(re,m)
      call centre_slice_four_neig(im,m)

      call deconvolution (re,m)

```

```
call deconvolution (im,m)

open (unit=1,file='ft.181',status='new')
format(4(1x,f14.4))
write(1,101)((b(k,j),k=1,181),j=1,181)
close(1)
```

```
do i=1,m
do j=1,m
r1(j+jframe,i+jframe)=re(j,i)
il(j+jframe,i+jframe)=im(j,i)
enddo
enddo
```

```
ake 2DI Fourier transform of re,im arrays
call fft2d(r1,il,256,1)
```

```
iframe=nint(float(256-n)/2.)
do i=1,n
do j=1,n
b(j,i)=r1(j+iframe,i+iframe)
enddo
enddo
```

```
return
end
```

```

      subroutine ift_four(a,m,b,n)
      real a(256,256),b(256,256),xy(256,256)
      real rl(256,256),il(256,256),xx(256)
*****
FUNCTION : This subroutine rotates an slice n X 1 array a
           through an angle theta (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b
PARAMETERS: a _ Input image of n X n size
           b _ Rotated image of m X m size
EXTERNAL SUBROUTINE :None
INTERNAL SUBROUTINE :None
*****
      real za(256),zb(256),zc(256),zd(256),c(256,256)
      real re(256,256),im(256,256)
end message to user 'computing reconstruction using centre slice...'

      write(6,1)
      format(1x,'Computing centre_slice_reconstruction...'$)

initial arrays
      do i = 1,256
      do j = 1,256
      rl(j,i) = 0.
      il(j,i) = 0.
      enddo
      enddo

initialize array c

      do i=1,m
      do j=1,m
      c(j,i)=0.
      enddo
      enddo

      do i=1,m
      do j=1,m
      c(j,i)=a(j,i)
      enddo
      enddo

      do j=1,256
      za(j)=0.
      zb(j)=0.
      zc(j)=0.
      zd(j)=0.
      enddo

like Fourier transform of a

      nn1=256/2+1
      jframe=nint(float(256-m)/2.)
      do i=1,m
      do j=1,m
      za(j+jframe)=c(j,i)
      enddo
      call fft1d(za,zb,256,-1)

      do j=1,m
      re(j,i)=za(j+jframe)
      im(j,i)=zb(j+jframe)
      enddo
      enddo

      call centre_slice_four(re,m)
      call centre_slice_four(im,m)

      call deconvolution (re,m)

```

```
call deconvolution (im,m)

open (unit=1,file='ft.181',status='new')
format(4(1x,f14.4))
write(1,101)((b(k,j),k=1,181),j=1,181)
close(1)
```

```
do i=1,m
do j=1,m
r1(j+jframe,i+jframe)=re(j,i)
il(j+jframe,i+jframe)=im(j,i)
enddo
enddo
```

```
ake 2DI Fourier transform of re,im arrays
call fft2d(r1,il,256,1)
```

```
iframe=nint(float(256-n)/2.)
do i=1,n
do j=1,n
b(j,i)=r1(j+iframe,i+iframe)
enddo
enddo
```

```
return
end
```

```

      subroutine ift_eight(a,m,b,n)
      real a(256,256),b(256,256),xy(256,256)
      real r1(256,256),il(256,256),xx(256)
*****
FUNCTION : This subroutine rotates an slice n X 1 array a
           through an angle theta (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b
PARAMETERS: a _ Input image of n X n size
           b _ Rotated image of m X m size
EXTERNAL SUBROUTINE :None
INTERNAL SUBROUTINE :None
*****
      real za(256),zb(256),zc(256),zd(256),c(256,256)
      real re(256,256),im(256,256)
Send message to user 'computing reconstruction using centre slice...'

      write(6,1)
      format(1x,'Computing centre_slice_reconstruction...'$)

initial arrays
      do i = 1,256
      do j = 1,256
      r1(j,i) = 0.
      il(j,i) = 0.
      enddo
      enddo

nititalize array c

      do i=1,m
      do j=1,m
      c(j,i)=0.
      enddo
      enddo

      do i=1,m
      do j=1,m
      c(j,i)=a(j,i)
      enddo
      enddo

      do j=1,256
      za(j)=0.
      zb(j)=0.
      zc(j)=0.
      zd(j)=0.
      enddo

ake Fourier transform of a

      nn1=256/2+1
      jframe=nint(float(256-m)/2.)
      do i=1,m
      do j=1,m
      za(j+jframe)=c(j,i)
      enddo
      call fft1d(za,zb,256,-1)

      do j=1,m
      re(j,i)=za(j+jframe)
      im(j,i)=zb(j+jframe)
      enddo
      enddo

      call centre_slice_eight(re,m)
      call centre_slice_eight(im,m)

      call deconvolution (re,m)

```

```
call deconvolution (im,m)

open (unit=1,file='ft.181',status='new')
format(4(1x,f14.4))
write(1,101)((b(k,j),k=1,181),j=1,181)
close(1)
```

```
do i=1,m
do j=1,m
r1(j+jframe,i+jframe)=re(j,i)
il(j+jframe,i+jframe)=im(j,i)
enddo
enddo
```

```
ake 2DI Fourier transform of re,im arrays
call fft2d(r1,il,256,1)
```

```
iframe=nint(float(256-n)/2.)
do i=1,n
do j=1,n
b(j,i)=r1(j+iframe,i+iframe)
enddo
enddo
```

```
return
end
```

```

      subroutine ift_eight_neig(a,m,b,n)
      real a(256,256),b(256,256),xy(256,256)
      real rl(256,256),il(256,256),xx(256)
*****
FUNCTION : This subroutine rotates an slice n X 1 array a
           through an angle theta (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b
PARAMETERS: a _ Input image of n X n size
           b _ Rotated image of m X m size
EXTERNAL SUBROUTINE :None
INTERNAL SUBROUTINE :None
*****
      real za(256),zb(256),zc(256),zd(256),c(256,256)
      real re(256,256),im(256,256)
end message to user 'computing reconstruction using centre slice...'

      write(6,1)
      format(1x,'Computing centre_slice_reconstruction...'$)

initial arrays
      do i = 1,256
      do j = 1,256
      rl(j,i) = 0.
      il(j,i) = 0.
      enddo
      enddo

initialize array c

      do i=1,m
      do j=1,m
      c(j,i)=0.
      enddo
      enddo

      do i=1,m
      do j=1,m
      c(j,i)=a(j,i)
      enddo
      enddo

      do j=1,256
      za(j)=0.
      zb(j)=0.
      zc(j)=0.
      zd(j)=0.
      enddo

ake Fourier transform of a

      nn1=256/2+1
      jframe=nint(float(256-m)/2.)
      do i=1,m
      do j=1,m
      za(j+jframe)=c(j,i)
      enddo
      call fft1d(za,zb,256,-1)

      do j=1,m
      re(j,i)=za(j+jframe)
      im(j,i)=zb(j+jframe)
      enddo
      enddo

      call centre_slice_eight_neig(re,m)
      call centre_slice_eight_neig(im,m)

      call deconvolution (re,m)

```



```
call deconvolution (im,m)

open (unit=1,file='ft.181',status='new')
format(4(1x,f14.4))
write(1,101)((b(k,j),k=1,181),j=1,181)
close(1)
```

```
do i=1,m
do j=1,m
r1(j+jframe,i+jframe)=re(j,i)
il(j+jframe,i+jframe)=im(j,i)
enddo
enddo
```

```
ake 2DI Fourier transform of re,im arrays
call fft2d(r1,il,256,1)
```

```
iframe=nint(float(256-n)/2.)
do i=1,n
do j=1,n
b(j,i)=r1(j+iframe,i+iframe)
enddo
enddo
```

```
return
end
```

```

subroutine centre slice nearest(a,m)
real a(256,256),b(256,256),w(256,256)
real cm(256,256),am(256,256),bc(256,256)
*****
FUNCTION : This subroutine rotates an slice n X 1 array a
           through an angle theta (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b
PARAMETERS: a _ Input image of n X n size
           b _ Rotated image of m X m size
EXTERNAL SUBROUTINE :None
INTERNAL SUBROUTINE :None
*****
real za(256),zb(256),zc(256),zd(256),c(256,256)

initial arrays
do i = 1,m
do j = 1,m
am(j,i) = 0.
enddo
enddo

do i = 1,m
do j = 1,m
cm(j,i) = 0.
b(j,i) =0.
bc(j,i) =0.
enddo
enddo

compute array centre
mm=m/2+1

start process
copy each profile to the horizontal array through the image centre.
do i = 1,m
do k = 1,m
if (k.eq.mm) then
do j=1,m
am(j,k)=a(j,i)
enddo
else
do j=1,m
am(j,k)=0.0
enddo
endif
enddo

theta = float(i-1)
call rotate_nearest(am,m,bc,m,theta)
do kk=1,m
do jj=1,m
cm(kk,jj)=cm(kk,jj)+bc(kk,jj)
enddo
enddo

enddo

do i = 1,m
do j = 1,m
a(j,i) =cm(j,i)
enddo
enddo

open (unit=1,file='ft.181',status='new')
format(4(1x,f14.4))
write(1,101)((b(k,j),k=1,181),j=1,181)
close(1)

```

return
end

```

subroutine centre slice nearest neig(a,m)
real a(256,256),b(256,256),w(256,256)
real cm(256,256),am(256,256),bc(256,256)
*****
FUNCTION : This subroutine rotates an slice n X 1 array a
           through an angle theta (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b
PARAMETERS: a _ Input image of n X n size
           b _ Rotated image of m X m size
EXTERNAL SUBROUTINE :None
INTERNAL SUBROUTINE :None
*****
real za(256),zb(256),zc(256),zd(256),c(256,256)

initial arrays
do i = 1,m
do j = 1,m
am(j,i) = 0.
enddo
enddo

do i = 1,m
do j = 1,m
cm(j,i) = 0.
b(j,i) =0.
bc(j,i) =0.
enddo
enddo

comput array centre
mm=m/2+1

start process
copy each profile to the horizontal array through the image centre.
do i = 1,m
do k = 1,m
if (k.eq.mm) then
do j=1,m
am(k,j)=a(j,i)
enddo
else
do j=1,m
am(k,j)=0.0
enddo
endif
enddo

theta = float(1-i)
call rotate_nearest_neig(am,m,bc,m,theta)
do kk=1,m
do jj=1,m
cm(kk,jj)=cm(kk,jj)+bc(kk,jj)
enddo
enddo

enddo

do i = 1,m
do j = 1,m
a(j,i) =cm(j,i)
enddo
enddo

open (unit=1,file='ft.181',status='new')
format(4(1x,f14.4))
write(1,101)((b(k,j),k=1,181),j=1,181)
close(1)

```

return
end

```

      subroutine centre_slice_lrp_neig(a,m)
      real a(256,256),b(256,256),w(256,256)
      real cm(256,256),am(256,256),bc(256,256)
*****
FUNCTION : This subroutine rotates an slice n X 1 array a
           through an angle theta (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b
PARAMETERS: a _ Input image of n X n size
           b _ Rotated image of m X m size
EXTERNAL SUBROUTINE :None
INTERNAL SUBROUTINE :None
*****
      real za(256),zb(256),zc(256),zd(256),c(256,256)

initial arrays
      do i = 1,m
      do j = 1,m
      am(j,i) = 0.
      enddo
      enddo

      do i = 1,m
      do j = 1,m
      cm(j,i) = 0.
      b(j,i) =0.
      bc(j,i) =0.
      enddo
      enddo

comput array centre
      mm=m/2+1

start process
copy each profile to the horizontal array through the image centre.
      do i = 1,m
      do k = 1,m
      if (k.eq.mm) then
      do j=1,m
      am(k,j)=a(j,i)
      enddo
      else
      do j=1,m
      am(k,j)=0.0
      enddo
      endif
      enddo

      theta = float(i-1)
      call rotate_lrp_neig(am,m,bc,m,theta)
      do kk=1,m
      do jj=1,m
      cm(kk,jj)=cm(kk,jj)+bc(kk,jj)
      enddo
      enddo

      enddo

      do i = 1,m
      do j = 1,m
      a(j,i) =cm(j,i)
      enddo
      enddo

      open (unit=1,file='ft.181',status='new')
      format(4(1x,f14.4))
      write(1,101)((b(k,j),k=1,181),j=1,181)
      close(1)

```

return
end

```

      subroutine centre_slice 2rp_neig(a,m)
      real a(256,256),b(256,256),w(256,256)
      real cm(256,256),am(256,256),bc(256,256)
*****
FUNCTION : This subroutine rotates an slice n X 1 array a
           through an angle theta (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b
PARAMETERS: a _ Input image of n X n size
           b _ Rotated image of m X m size
EXTERNAL SUBROUTINE :None
INTERNAL SUBROUTINE :None
*****
      real za(256),zb(256),zc(256),zd(256),c(256,256)

initial arrays
      do i = 1,m
      do j = 1,m
      am(j,i) = 0.
      enddo
      enddo

      do i = 1,m
      do j = 1,m
      cm(j,i) = 0.
      b(j,i) =0.
      bc(j,i) =0.
      enddo
      enddo

omput array centre
      mm=m/2+1

tart process
opy each profile to the horizontal array through the image centre.
      do i = 1,m
      do k = 1,m
      if (k.eq.mm) then
      do j=1,m
      am(k,j)=a(j,i)
      enddo
      else
      do j=1,m
      am(k,j)=0.0
      enddo
      endif
      enddo

      theta = float(i-1)
      call rotate_2rp_neig(am,m,bc,m,theta)
      do kk=1,m
      do jj=1,m
      cm(kk,jj)=cm(kk,jj)+bc(kk,jj)
      enddo
      enddo

      enddo

      do i = 1,m
      do j = 1,m
      a(j,i) =cm(j,i)
      enddo
      enddo

      open (unit=1,file='ft.181',status='new')
      format(4(1x,f14.4))
      write(1,101)((b(k,j),k=1,181),j=1,181)
      close(1)

```


return
end

```

      subroutine centre slice four(a,m)
      real a(256,256),b(256,256),w(256,256)
      real cm(256,256),am(256,256),bc(256,256)
*****
FUNCTION : This subroutine rotates an slice n X 1 array a
           through an angle theta (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b
PARAMETERS: a _ Input image of n X n size
           b _ Rotated image of m X m size
EXTERNAL SUBROUTINE :None
INTERNAL SUBROUTINE :None
*****
      real za(256),zb(256),zc(256),zd(256),c(256,256)

initial arrays
      do i = 1,m
      do j = 1,m
      am(j,i) = 0.
      enddo
      enddo

      do i = 1,m
      do j = 1,m
      cm(j,i) = 0.
      b(j,i) =0.
      bc(j,i) =0.
      enddo
      enddo

compute array centre
      mm=m/2+1

start process
copy each profile to the horizontal array through the image centre.
      do i = 1,m
      do k = 1,m
      if (k.eq.mm) then
      do j=1,m
      am(k,j)=a(j,i)
      enddo
      else
      do j=1,m
      am(k,j)=0.0
      enddo
      endif
      enddo

      theta = float(i-1)
      call rotate_four(am,m,bc,m,theta)
      do kk=1,m
      do jj=1,m
      cm(kk,jj)=cm(kk,jj)+bc(kk,jj)
      enddo
      enddo

      enddo

      do i = 1,m
      do j = 1,m
      a(j,i) =cm(j,i)
      enddo
      enddo

      open (unit=1,file='ft.181',status='new')
      format(4(1x,f14.4))
      write(1,101)((b(k,j),k=1,181),j=1,181)
      close(1)

```

return
end

```

subroutine centre slice four neig(a,m)
real a(256,256),b(256,256),w(256,256)
real cm(256,256),am(256,256),bc(256,256)
*****
FUNCTION : This subroutine rotates an slice n X 1 array a
through an angle theta (given in degree)
anticlockwise placing the result at the
center of an array mxm b
PARAMETERS: a _ Input image of n X n size
b _ Rotated image of m X m size
EXTERNAL SUBROUTINE :None
INTERNAL SUBROUTINE :None
*****
real za(256),zb(256),zc(256),zd(256),c(256,256)

initial arrays
do i = 1,m
do j = 1,m
am(j,i) = 0.
enddo
enddo

do i = 1,m
do j = 1,m
cm(j,i) = 0.
b(j,i) =0.
bc(j,i) =0.
enddo
enddo

omput array centre
mm=m/2+1

tart process
opy each profile to the horizontal array through the image centre.
do i = 1,m
do k = 1,m
if (k.eq.mm) then
do j=1,m
am(k,j)=a(j,i)
enddo
else
do j=1,m
am(k,j)=0.0
enddo
endif
enddo

theta = float(i-1)
call rotate_four_neig(am,m,bc,m,theta)
do kk=1,m
do jj=1,m
cm(kk,jj)=cm(kk,jj)+bc(kk,jj)
enddo
enddo

enddo

do i = 1,m
do j = 1,m
a(j,i) =cm(j,i)
enddo
enddo

open (unit=1,file='ft.181',status='new')
format(4(1x,f14.4))
write(1,101)((b(k,j),k=1,181),j=1,181)
close(1)

```

return
end

```

subroutine centre slice eight(a,m)
real a(256,256),b(256,256),w(256,256)
real cm(256,256),am(256,256),bc(256,256)
*****
FUNCTION : This subroutine rotates an slice n X 1 array a
through an angle theta (given in degree)
anticlockwise placing the result at the
center of an array mxm b
PARAMETERS: a _ Input image of n X n size
b _ Rotated image of m X m size
EXTERNAL SUBROUTINE :None
INTERNAL SUBROUTINE :None
*****
real za(256),zb(256),zc(256),zd(256),c(256,256)

initial arrays
do i = 1,m
do j = 1,m
am(j,i) = 0.
enddo
enddo

do i = 1,m
do j = 1,m
cm(j,i) = 0.
b(j,i) = 0.
bc(j,i) = 0.
enddo
enddo

compute array centre
mm=m/2+1

start process
copy each profile to the horizontal array through the image centre.
do i = 1,m
do k = 1,m
if (k.eq.mm) then
do j=1,m
am(k,j)=a(j,i)
enddo
else
do j=1,m
am(k,j)=0.0
enddo
endif
enddo

theta = float(i-1)
call rotate_eight(am,m,bc,m,theta)
do kk=1,m
do jj=1,m
cm(kk,jj)=cm(kk,jj)+bc(kk,jj)
enddo
enddo

enddo

do i = 1,m
do j = 1,m
a(j,i) =cm(j,i)
enddo
enddo

open (unit=1,file='ft.181',status='new')
format(4(1x,f14.4))
write(1,101)((b(k,j),k=1,181),j=1,181)
close(1)

```

return
end

```

subroutine centre_slice eight_neig(a,m)
real a(256,256),b(256,256),w(256,256)
real cm(256,256),am(256,256),bc(256,256)
*****
FUNCTION : This subroutine rotates an slice n X 1 array a
through an angle theta (given in degree)
anticlockwise placing the result at the
center of an array mxm b
PARAMETERS: a _ Input image of n X n size
b _ Rotated image of m X m size
EXTERNAL SUBROUTINE :None
INTERNAL SUBROUTINE :None
*****
real za(256),zb(256),zc(256),zd(256),c(256,256)

initial arrays
do i = 1,m
do j = 1,m
am(j,i) = 0.
enddo
enddo

do i = 1,m
do j = 1,m
cm(j,i) = 0.
b(j,i) =0.
bc(j,i) =0.
enddo
enddo

omput array centre
mm=m/2+1

tart process
opy each profile to the horizontal array through the image centre.
do i = 1,m
do k = 1,m
if (k.eq.mm) then
do j=1,m
am(k,j)=a(j,i)
enddo
else
do j=1,m
am(k,j)=0.0
enddo
endif
enddo

theta = float(i-1)
call rotate_eight_neig(am,m,bc,m,theta)
do kk=1,m
do jj=1,m
cm(kk,jj)=cm(kk,jj)+bc(kk,jj)
enddo
enddo

enddo

do i = 1,m
do j = 1,m
a(j,i) =cm(j,i)
enddo
enddo

open (unit=1,file='ft.181',status='new')
format(4(1x,f14.4))
write(1,101)((b(k,j),k=1,181),j=1,181)
close(1)

```


return
end

subroutine deconvolution(a,m)

Function: This subroutine deconvolves a reconstruction obtained by straight backprojection (un-filtered) using the appropriate inverse filter, i.e. $\sqrt{kx*kx+ky*ky}$ where kx and ky are the spatial frequencies.

```
real a(256,256),aa(256,256),xx(256,256)
real b(256,256),bb(256,256)
```

Initialize array b

```
do i=1,m
do j=1,m
aa(j,i)=0.
bb(j,i)=0.
enddo
enddo
```

Take Fourier transform of a

```
call fft2d(a,aa,n,-1)
```

Multiply spectrum by inverse filter

```
mm=m/2+1
do i=1,m
do j=1,m
xk=float(i-mm)/float(m)
yk=float(j-mm)/float(m)
fil=sqrt(xk*xk+yk*yk)!value of inverse filter at (xk,yk)
b(j,i)=a(j,i)*fil
cc bb(j,i)=aa(j,i)*fil
enddo
enddo
```

```
call fft2d(b,bb,n,1)
```

Write output

```
do i=1,m
do j=1,m
a(j,i)=b(j,i)
enddo
enddo
```

```
return
end
```

```

subroutine project(a,b,n)
*****
Function: This subroutine sums
          over all rows of an n X n array a giving 1D
projection of size n _ array b
*****
      real a(256,256),b(256),s(256,256),z(256)
initialize arrays s & b
      do j=1,n
        b(j)=0.
        z(j)=0.
        do i=1,n
          s(j,i)=0.
        enddo
      enddo
set initial value for summing process and sum all pixels along
row i together
      do j=1,n
        s(j,1)=a(j,1)
        do i=2,n
          s(j,i)=s(j,i-1)+a(j,i)
        enddo
assign value of sum at column j to output _b(j)
        b(j)=s(j,n)
      enddo

      return
end

```

```

subroutine rtr_nearest(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
*****
Function: This subroutine computes the Radon Transform of
n X n image.
External subroutines: Rotate_nearest_neig
Project
Internal subroutines: None
Parameters: x - n X n input array
y - m X m output array
*****
Send message to user 'Computing Radon Transform...'

write(6,1)
format(1x,'Computing Radon Transform...'$)
Initialize array y
do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo
Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.
do i=1,m
theta=float(i-1)
call rotate_nearest(x,n,w,m,theta)
call project(w,z,m)
Stack projections together to form image of "Radon space"
data.
do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```

```

subroutine rtr_nearest_neig(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
*****
Function: This subroutine computes the Radon Transform of
n X n image.
External subroutines: Rotate_nearest_neig
Project
Internal subroutines: None
Parameters: x - n X n input array
y - m X m output array
*****
Send message to user 'Computing Radon Transform...'

write(6,1)
format(1x,'Computing Radon Transform...'$)
Initialize array y
do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo
Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.
do i=1,m
theta=float(i-1)
call rotate_nearest_neig(x,n,w,m,theta)
call project(w,z,m)
Stack projections together to form image of "Radon space"
data.
do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```

```

subroutine rtr_lrp_neig(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
*****
Function: This subroutine computes the Radon Transform of
n X n image.
External subroutines: Rotate_nearest_neig
Project
Internal subroutines: None
Parameters: x - n X n input array
y - m X m output array
*****
Send message to user 'Computing Radon Transform...'

write(6,1)
format(1x,'Computing Radon Transform...'$)
Initialize array y
do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo
Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.
do i=1,m
theta=float(i-1)
call rotate_lrp_neig(x,n,w,m,theta)
call project(w,z,m)
Stack projections together to form image of "Radon space"
data.
do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```

```

subroutine rtr_2rp_neig(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
*****
Function: This subroutine computes the Radon Transform of
n X n image.
External subroutines: Rotate_nearest_neig
Project
Internal subroutines: None
Parameters: x - n X n input array
y - m X m output array
*****
Send message to user 'Computing Radon Transform...'

write(6,1)
format(1x,'Computing Radon Transform...')
Initialize array y
do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo
Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.
do i=1,m
theta=float(i-1)
call rotate_2rp_neig(x,n,w,m,theta)
call project(w,z,m)
Stack projections together to form image of "Radon space"
data.
do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```

```

subroutine rtr_four(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
*****
Function: This subroutine computes the Radon Transform of
n X n image.
External subroutines: Rotate_nearest_neig
Project
Internal subroutines: None
Parameters: x - n X n input array
y - m X m output array
*****
Send message to user 'Computing Radon Transform...'

write(6,1)
format(1x,'Computing Radon Transform...'$)
Initialize array y
do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo
Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.
do i=1,m
theta=float(i-1)
call rotate_four(x,n,w,m,theta)
call project(w,z,m)
Stack projections together to form image of "Radon space"
data.
do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```



```

subroutine rtr_four_neig(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
*****
Function: This subroutine computes the Radon Transform of
n X n image.
External subroutines: Rotate_four_neig
Project
Internal subroutines: None
Parameters: x - n X n input array
y - m X m output array
*****
Send message to user 'Computing Radon Transform...'

write(6,1)
format(1x,'Computing Radon Transform...'$)
Initialize array y
do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo
Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.
do i=1,m
theta=float(i-1)
call rotate_four_neig(x,n,w,m,theta)
call project(w,z,m)
Stack projections together to form image of "Radon space"
data.
do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```

```

subroutine rtr_eight(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
*****
Function: This subroutine computes the Radon Transform of
n X n image.
External subroutines: Rotate_nearest_neig
Project
Internal subroutines: None
Parameters: x - n X n input array
y - m X m output array
*****
Send message to user 'Computing Radon Transform...'

write(6,1)
format(1x,'Computing Radon Transform...'$)
Initialize array y
do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo
Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.
do i=1,m
theta=float(i-1)
call rotate_eight(x,n,w,m,theta)
call project(w,z,m)
Stack projections together to form image of "Radon space"
data.
do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```

```

subroutine rtr_eight_neig(x,n,y,m)
real x(256,256),y(256,256),xy(256,256)
real w(256,256),z(256)
*****
Function: This subroutine computes the Radon Transform of
n X n image.
External subroutines: Rotate_eight_neig
Project
Internal subroutines: None
Parameters: x - n X n input array
y - m X m output array
*****
Send message to user 'Computing Radon Transform...'

write(6,1)
format(1x,'Computing Radon Transform...'$)
Initialize array y
do j=1,m
do i=1,m
y(j,i)=0.
enddo
enddo
Compute angle of rotation in steps of 1 degree and rotate
the object, take projection of rotated object.
do i=1,m
theta=float(i-1)
call rotate_eight_neig(x,n,w,m,theta)
call project(w,z,m)
Stack projections together to form image of "Radon space"
data.
do j=1,m
y(j,i)=z(j)
enddo
enddo
return
end

```

```

      subroutine rotate_nearest(a,n,b,m,theta)
      real a(256,256),b(256,256),w(256,256)
*****
FUNCTION : This subroutine rotates an n X n array a
           through an angle theta (given in degree)
           anticlockwise placing the result at the
           center of an array mxm b
PARAMETERS: a _ Input image of n X n size
           b _ Rotated image of m X m size
EXTERNAL SUBROUTINE :None
INTERNAL SUBROUTINE :None
*****

      convert theta into radians

           pi = 4. * atan(1.)
           theta = theta * (pi/180.)
           costh = cos(theta)
           sinth = sin(theta)

initial b
           do i = 1,m
           do j = 1,m
           b(j,i) = 0.
           w(j,i) = 0.
           enddo
           enddo

compute new coordinates (xd,yd) of rotate array

           nn = n/2 + 1
           mm = m/2 + 1

start process
           do j = 1,n
           do i = 1,n

generate 4-quadrant cartesian plan
           y = float(j-nn)
           x = float(i-nn)
           xd = x * costh - y * sinth
           yd = x * sinth + y * costh

compute both real and integer coordinates of array b
associated with the above transformation for later use
           kx = mm+ nint(xd)
           ky = mm+ nint(yd)

assign value of a at (j,i) to the nearest_point cell
of array b in which the transformed coordinates
lie.
           b(ky,kx) = a(j,i)

           enddo
           enddo
           return
           end

```

```

subroutine rotate_nearest_neig (a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
integer xu,yu,xd,yd,dul,eul,du2,eu2,vu(256),hu(256),ku(256)
integer gu(256)

```

Function: This subroutine rotates an nXn array a through an angle theta (given in degrees) placing the result at the center of an mXm array b.

Convert theta into radians.

```

pi=4.*atan(1.)
theta=theta*(pi/180.)

```

initialize b

```

do i=1,m
do j=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo

```

Compute centers of array a and b

```

nn=n/2+1
mm=m/2+1

```

Compute the integer of horizontal and vertical displacement along the x and y-axis which corresponds to n-pixel movement along the u-axis.

```

xu=nint(nn*cos(theta))
yu=nint(nn*sin(theta))
if (xu.lt.0) then
ml=-1
else
ml=1
endif
if (yu.lt.0) then
nl=1
else
nl=-1
endif

```

compute the rotation parameters

```

dul=2*abs(yu)
du2=2*(abs(yu)-nn)
eul=2*abs(xu)
eu2=2*(abs(xu)-nn)
vu(1)=2*abs(yu)-nn
hu(1)=2*abs(xu)-nn
ku(0)=0
gu(0)=0
do i=1,nn
if (hu(i).ge.0) then
hu(i+1)=hu(i)+eu2
gu(i)=gu(i-1)+1
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
else
hu(i+1)=hu(i)+eul
gu(i)=gu(i-1)
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
endif
enddo

```

compute new coordinates (xd,yd) of rotated array and assign the value of a at (i,j) to the appropriate cell of b which encloses these coordinates.

Generate 4-quadrant cartesian plane.

```
do i=1,nn
  ku(-i)=-ku(i)
  gu(-i)=-gu(i)
enddo
```

Start process

```
do j=1,n
  do i=1,n
```

Employ coordinate transformation for rotation using the integer parameters.

```
  yd=m1*gu(i-nn)-n1*ku(j-nn)
  xd=m1*gu(j-nn)+n1*ku(i-nn)
```

Compute the integer coordinates of array b associated with the above transformation.

```
  kx=mm+xd
  ky=mm+yd
```

Assign value of a at (i,j) to the nearest cell of array b in which the transformed coordinates lie.

```
  b(ky+1,kx-1)=a(i,j)
  b(ky,kx-1)=a(i,j)
  b(ky-1,kx-1)=a(i,j)
  b(ky+1,kx)=a(i,j)
  b(ky,kx)=a(i,j)
  b(ky-1,kx)=a(i,j)
  b(ky+1,kx+1)=a(i,j)
  b(ky,kx+1)=a(i,j)
  b(ky-1,kx+1)=a(i,j)
```

```
enddo
enddo
return
end
```

```

subroutine rotate_lrp neig (a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
integer xu,yu,xd,yd,dul,eul,du2,eu2,vu(256),hu(256),ku(256)
integer gu(256)

```

Function: This subroutine rotates an nXn array a through an angle theta (given in degrees) placing the result at the center of an mXm array b.

Convert theta into radiants.

```

pi=4.*atan(1.)
theta=theta*(pi/180.)

```

initialize b

```

do i=1,m
do j=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo

```

Compute centers of array a and b

```

nn=n/2+1
mm=m/2+1

```

Compute the integer of horizontal and vertical displacement along the x and y-axis which corresponds to n-pixel movement along the u-axis.

```

xu=nint(nn*cos(theta))
yu=nint(nn*sin(theta))
if (xu.lt.0) then
m1=-1
else
m1=1
endif
if (yu.lt.0) then
n1=1
else
n1=-1
endif

```

compute the rotation parameters

```

dul=2*abs(yu)
du2=2*(abs(yu)-nn)
eul=2*abs(xu)
eu2=2*(abs(xu)-nn)
vu(1)=2*abs(yu)-nn
hu(1)=2*abs(xu)-nn
ku(0)=0
gu(0)=0
do i=1,nn
if (hu(i).ge.0) then
hu(i+1)=hu(i)+eu2
gu(i)=gu(i-1)+1
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
else
hu(i+1)=hu(i)+eul
gu(i)=gu(i-1)
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
endif
enddo

```

compute new coordinates (xd,yd) of rotated array and assign the value of a at (i,j) to the appropriate cell of b which encloses these coordinates.

Generate 4-quadrant cartesian plane.

```
do i=1,nn
ku(-i)=-ku(i)
gu(-i)=-gu(i)
enddo
```

Start process

```
do j=1,n
do i=1,n
```

Employ coordinate transformation for rotation using the integer parameters.

```
yd=m1*gu(i-nn)-n1*ku(j-nn)
xd=m1*gu(j-nn)+n1*ku(i-nn)
```

Compute the integer coordinates of array b associated with the above transformation.

```
kx=mm+xd
ky=mm+yd
```

Assign value of a at (i,j) to the nearest cell of array b in which the transformed coordinates lie.

```
b(ky,kx-1)=a(i,j)
b(ky+1,kx)=a(i,j)
b(ky,kx)=a(i,j)
b(ky-1,kx)=a(i,j)
b(ky,kx+1)=a(i,j)
```

```
enddo
enddo
return
end
```



```

subroutine rotate 2rp neig (a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
integer xu,yu,xd,yd,dul,eul,du2,eu2,vu(256),hu(256),ku(256)
integer gu(256)

```

Function: This subroutine rotates an nXn array a through an angle theta (given in degrees) placing the result at the center of an mXm array b.

Convert theta into radians.

```

pi=4.*atan(1.)
theta=theta*(pi/180.)

```

initialize b

```

do i=1,m
do j=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo

```

Compute centers of array a and b

```

nn=n/2+1
mm=m/2+1

```

Compute the integer of horizontal and vertical displacement along the x and y-axis which corresponds to n-pixel movement along the u-axis.

```

xu=nint(nn*cos(theta))
yu=nint(nn*sin(theta))
if (xu.lt.0) then
m1=-1
else
m1=1
endif
if (yu.lt.0) then
n1=1
else
n1=-1
endif

```

compute the rotation parameters

```

dul=2*abs(yu)
du2=2*(abs(yu)-nn)
eul=2*abs(xu)
eu2=2*(abs(xu)-nn)
vu(1)=2*abs(yu)-nn
hu(1)=2*abs(xu)-nn
ku(0)=0
gu(0)=0
do i=1,nn
if (hu(i).ge.0) then
hu(i+1)=hu(i)+eu2
gu(i)=gu(i-1)+1
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
else
hu(i+1)=hu(i)+eul
gu(i)=gu(i-1)
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
endif
enddo

```

compute new coordinates (xd,yd) of rotated array and assign the value of a at (i,j) to the appropriate cell of b which encloses these coordinates.

Generate 4-quadrant cartesian plane.

```
do i=1,nn
ku(-i)=-ku(i)
gu(-i)=-gu(i)
enddo
```

Start process

```
do j=1,n
do i=1,n
```

Employ coordinate transformation for rotation using the integer parameters.

```
yd=m1*gu(i-nn)-n1*ku(j-nn)
xd=m1*gu(j-nn)+n1*ku(i-nn)
```

Compute the integer coordinates of array b associated with the above transformation.

```
kx=mm+xd
ky=mm+yd
```

Assign value of a at (i,j) to the nearest cell of array b in which the transformed coordinates lie.

```
b(ky,kx-1)=a(i,j)
b(ky+1,kx)=a(i,j)
b(ky,kx)=a(i,j)
b(ky-1,kx)=a(i,j)
b(ky,kx+1)=a(i,j)
```

```
enddo
enddo
return
end
```

```

subroutine rotate four(a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
*****
FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
          THROUGH AN ANGLE THETA (given in degree)
          anticlockwise placing the result at the
          center of an array mxm b
PARAMETERS: a _ Input image of n X n size
            b _ Rotated image of m X m size
EXTERNAL SUBROUTINE :None
INTERNAL SUBROUTINE :None
*****

convert theta into radians

      pi = 4. * atan(1.)
      theta = theta * (pi/180.)
      costh = cos(theta)
      sinth = sin(theta)

initial b
      do i = 1,m
      do j = 1,m
      b(j,i) = 0.
      w(j,i) = 0.
      enddo
      enddo

compute new coordinates (xd,yd) of rotate array

      nn = n/2 + 1
      mm = m/2 + 1
compensation of the center position for rotation.
      xcomp=nn+nn*(sinth - costh)
      ycomp=nn-nn*(sinth + costh)

start process

      do j = 1,n
      do i = 1,n

generate 4-quadrant cartesian plan
      y = float(j-nn)
      x = float(i-nn)
      xd = x * costh - y * sinth
      yd = x * sinth + y * costh

compute both real and integer coordinates of array b
associated with the above transformation for later use
      kx = mm+ nint(xd)
      ky = mm+ nint(yd)

assign value of a at (j,i) to 9-point cell
of array b in which the transformed coordinates
lie.
      b(ky,kx-1) = a(j,i)
      b(ky+1,kx) = a(j,i)
      b(ky,kx) = a(j,i)
      b(ky-1,kx) = a(j,i)
      b(ky,kx+1) = a(j,i)
      enddo
      enddo
      return
      end

```

```

subroutine rotate four neig (a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
integer xu,yu,xd,yd,dul,eul,du2,eu2,vu(256),hu(256),ku(256)
integer gu(256)

```

Function: This subroutine rotates an nXn array a through an angle theta (given in degrees) placing the result at the center of an mXm array b.

Convert theta into radiants.

```

pi=4.*atan(1.)
theta=theta*(pi/180.)

```

Initialize b

```

do i=1,m
do j=1,m
b(j,i)=0.
w(j,i)=0.
enddo
enddo

```

Compute centers of array a and b

```

nn=n/2+1
mm=m/2+1

```

Compute the integer of horizontal and vertical displacement along the x and y-axis which corresponds to n-pixel movement along the u-axis.

```

xu=nint(nn*cos(theta))
yu=nint(nn*sin(theta))
if (xu.lt.0) then
ml=-1
else
ml=1
endif
if (yu.lt.0) then
nl=1
else
nl=-1
endif

```

Compute the rotation parameters

```

dul=2*abs(yu)
du2=2*(abs(yu)-nn)
eul=2*abs(xu)
eu2=2*(abs(xu)-nn)
vu(1)=2*abs(yu)-nn
hu(1)=2*abs(xu)-nn
ku(0)=0
gu(0)=0
do i=1,nn
if (hu(i).ge.0) then
hu(i+1)=hu(i)+eu2
gu(i)=gu(i-1)+1
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
else
hu(i+1)=hu(i)+eul
gu(i)=gu(i-1)
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
endif
enddo

```

Compute new coordinates (xd,yd) of rotated array and assign the value of a at (i,j) to the appropriate cell of b which encloses these coordinates.

Generate 4-quadrant cartesian plane.

```
do i=1,nn
ku(-i)=-ku(i)
gu(-i)=-gu(i)
enddo
```

Start process

```
do j=1,n
do i=1,n
```

Employ coordinate transformation for rotation using the integer parameters.

```
yd=m1*gu(i-nn)-n1*ku(j-nn)
xd=m1*gu(j-nn)+n1*ku(i-nn)
```

Compute the integer coordinates of array b associated with the above transformation.

```
kx=mm+xd
ky=mm+yd
```

Assign value of a at (i,j) to the nearest cell of array b in which the transformed coordinates lie.

```
b(ky+1,kx-1)=a(i,j)
b(ky,kx-1)=a(i,j)
b(ky-1,kx-1)=a(i,j)
b(ky+1,kx)=a(i,j)
b(ky,kx)=a(i,j)
b(ky-1,kx)=a(i,j)
b(ky+1,kx+1)=a(i,j)
b(ky,kx+1)=a(i,j)
b(ky-1,kx+1)=a(i,j)
```

```
enddo
enddo
return
end
```

```

      subroutine rotate_eight(a,n,b,m,theta)
      real a(256,256),b(256,256),w(256,256)
*****
FUNCTION : THIS SUBROUTINE ROTATES AN NXN ARRAY A
          THROUGH AN ANGLE THETA (given in degree)
          anticlockwise placing the result at the
          center of an array mxm b
PARAMETERS: a _ Input image of n X n size
          b _ Rotated image of m X m size
EXTERNAL SUBROUTINE :None
INTERNAL SUBROUTINE :None
*****

      convert theta into radians

          pi = 4. * atan(1.)
          theta = theta * (pi/180.)
          costh = cos(theta)
          sinth = sin(theta)

initial b

          do i = 1,m
          do j = 1,m
          b(j,i) = 0.
          w(j,i) = 0.
          enddo
          enddo

compute new coordinates (xd,yd) of rotate array

          nn = n/2 + 1
          mm = m/2 + 1

start process

          do j = 1,n
          do i = 1,n

generate 4-quadrant cartesian plan
          y = float(j-nn)
          x = float(i-nn)
          xd = x * costh - y * sinth
          yd = x * sinth + y * costh

compute both real and integer coordinates of array b
associated with the above transformation for later use
          kx = mm+ nint(xd)
          ky = mm+ nint(yd)

assign value of a at (j,i) to 9-point cell
of array b in which the transformed coordinates
lie.
          b(ky+1,kx-1) = a(j,i)
          b(ky,kx-1) = a(j,i)
          b(ky-1,kx-1) = a(j,i)
          b(ky+1,kx) = a(j,i)
          b(ky,kx) = a(j,i)
          b(ky-1,kx) = a(j,i)
          b(ky+1,kx+1) = a(j,i)
          b(ky,kx+1) = a(j,i)
          b(ky-1,kx+1) = a(j,i)
          enddo
          enddo
          return
          end

```

```

subroutine rotate_eight_neig (a,n,b,m,theta)
real a(256,256),b(256,256),w(256,256)
integer xu,yu,xd,yd,dul,eul,du2,eu2,vu(256),hu(256),ku(256)
integer gu(256)

```

Function: This subroutine rotates an nXn array a through an angle theta (given in degrees) placing the result at the center of an mXm array b.

Convert theta into radians.

```

pi=4.*atan(1.)
theta=theta*(pi/180.)

```

initialize b

```

do i=1,m
do j=1,m
b(j,i)=0
enddo
enddo

```

Compute centers of array a and b

```

nn=n/2+1
mm=m/2+1

```

Compute the integer of horizontal and vertical displacement along the x and y-axis which corresponds to n-pixel movement along the u-axis.

```

xu=nint(nn*cos(theta))
yu=nint(nn*sin(theta))
if (xu.lt.0) then
ml=-1
else
ml=1
endif
if (yu.lt.0) then
nl=1
else
nl=-1
endif

```

compute the rotation parameters

```

dul=2*abs(yu)
du2=2*(abs(yu)-nn)
eul=2*abs(xu)
eu2=2*(abs(xu)-nn)
vu(1)=2*abs(yu)-nn
hu(1)=2*abs(xu)-nn
ku(0)=0
gu(0)=0
do i=1,nn
if (hu(i).ge.0) then
hu(i+1)=hu(i)+eu2
gu(i)=gu(i-1)+1
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
else
hu(i+1)=hu(i)+eul
gu(i)=gu(i-1)
if (vu(i).ge.0) then
vu(i+1)=vu(i)+du2
ku(i)=ku(i-1)+1
else
vu(i+1)=vu(i)+dul
ku(i)=ku(i-1)
endif
endif
enddo

```

compute new coordinates (xd,yd) of rotated array and assign the value of a at (i,j) to the appropriate cell of b which encloses these coordinates.
generate 4-quadrant cartesian plane.

```
do i=1,nn
ku(-i)=-ku(i)
gu(-i)=-gu(i)
enddo
```

Start process

```
do j=1,n
do i=1,n
```

Employ coordinate transformation for rotation using the integer parameters.

```
yd=m1*gu(i-nn)-n1*ku(j-nn)
xd=m1*gu(j-nn)+n1*ku(i-nn)
```

Compute the integer coordinates of array b associated with the above transformation.

```
kx=mm+xd
ky=mm+yd
```

Assign value of a at (j,i) to the nearest cell of array b in which the transformed coordinates lie.

```
b(ky+1,kx-1)=a(i,j)
b(ky,kx-1)=a(i,j)
b(ky-1,kx-1)=a(i,j)
b(ky+1,kx)=a(i,j)
b(ky,kx)=a(i,j)
b(ky-1,kx)=a(i,j)
b(ky+1,kx+1)=a(i,j)
b(ky,kx+1)=a(i,j)
b(ky-1,kx+1)=a(i,j)
```

```
enddo
enddo
return
end
```



```

subroutine irtr_nearest(x,m,y,n)
real x(256,256),y(256,256),xx(256,256)
*****
FUNCTION: This subroutine computes the inverse radon
transform by:
    1. Back_projecting all the projections stored
       in array x giving the back_projection
       function y.
    2. Deconvolving the array y using the inverse_
       filter SQRT(kx*kx + ky*ky*).
External subroutines: bpr_nearest_neig
                     back_projection_deconvolution
Internal subroutines: none
*****

Send message to user 'Computing Inverse Radon Transform...'

    write(6,1)
    format(1x,'Computing Inverse Radon Transform...')
Initialize array xx
do i=1,n
do j=1,n
xx(j,i)=0.
enddo
enddo

Back project projections stored in x

    call bpr_nearest(x,m,y,n)

Send message to user' Deconvolve the reconstruction...'
    write(6,2)
    format(1x,'Deconvolving the reconstruction process...')

    call back_projection_deconvolution(y,n)

return
end

```

```

subroutine irtr_nearest_neig(x,m,y,n)
real x(256,256),y(256,256),xx(256,256)
*****
FUNCTION: This subroutine computes the inverse radon
transform by:
    1. Back_projecting all the projections stored
        in array x giving the back_projection
        function y.
    2. Deconvolving the array y using the inverse_
        filter SQRT(kx*kx + ky*ky*).
External subroutines: bpr_nearest_neig
                    back_projection_deconvolution
Internal subroutines: none
*****

Send message to user 'Computing Inverse Radon Transform...'

    write(6,1)
    format(1x,'Computing Inverse Radon Transform...')
Initialize array xx
do i=1,n
do j=1,n
xx(j,i)=0.
enddo
enddo

Back project projections stored in x

    call bpr_nearest_neig(x,m,y,n)

Send message to user' Deconvolve the reconstruction...'
    write(6,2)
    format(1x,'Deconvolving the reconstruction process...')

    call back_projection_deconvolution(y,n)

return
end

```

```

subroutine irtr_lrp_neig(x,m,y,n)
real x(256,256),y(256,256),xx(256,256)
*****
FUNCTION: This subroutine computes the inverse radon
transform by:
    1. Back_projecting all the projections stored
       in array x giving the back_projection
       function y.
    2. Deconvolving the array y using the inverse_
       filter SQRT(kx*kx + ky*ky*).
External subroutines: bpr_nearest_neig
                     back_projection_deconvolution
Internal subroutines: none
*****
Send message to user 'Computing Inverse Radon Transform...'

    write(6,1)
    format(1x,'Computing Inverse Radon Transform...')
Initialize array xx
do i=1,n
do j=1,n
xx(j,i)=0.
enddo
enddo

Back project projections stored in x

    call bpr_lrp_neig(x,m,y,n)

Send message to user' Deconvolve the reconstruction...'
    write(6,2)
    format(1x,'Deconvolving the reconstruction process...')

    call back_projection_deconvolution(y,n)
    return
end

```

```

subroutine irtr_2rp_neig(x,m,y,n)
real x(256,256),y(256,256),xx(256,256)
*****
FUNCTION: This subroutine computes the inverse radon
transform by:
    1. Back_projecting all the projections stored
       in array x giving the back_projection
       function y.
    2. Deconvolving the array y using the inverse_
       filter SQRT(kx*kx + ky*ky*).
External subroutines: bpr_nearest_neig
                     back_projection_deconvolution
Internal subroutines: none
*****
Send message to user 'Computing Inverse Radon Transform...'

    write(6,1)
    format(1x,'Computing Inverse Radon Transform...')
Initialize array xx
do i=1,n
do j=1,n
xx(j,i)=0.
enddo
enddo

Back project projections stored in x

    call bpr_2rp_neig(x,m,y,n)

Send message to user' Deconvolve the reconstruction...'
    write(6,2)
    format(1x,'Deconvolving the reconstruction process...')

    call back_projection_deconvolution(y,n)
return
end

```

```

subroutine irtr_four(x,m,y,n)
real x(256,256),y(256,256),xx(256,256)
*****
FUNCTION: This subroutine computes the inverse radon
transform by:
    1. Back_projecting all the projections stored
       in array x giving the back_projection
       function y.
    2. Deconvolving the array y using the inverse_
       filter SQRT(kx*kx + ky*ky*).
External subroutines: bpr_nearest_neig
                     back_projection_deconvolution
Internal subroutines: none
*****
Send message to user 'Computing Inverse Radon Transform...'

    write(6,1)
    format(1x,'Computing Inverse Radon Transform...')
Initialize array xx
do i=1,n
do j=1,n
xx(j,i)=0.
enddo
enddo

Back project projections stored in x

    call bpr_four(x,m,y,n)

Send message to user' Deconvolve the reconstruction...'
    write(6,2)
    format(1x,'Deconvolving the reconstruction process...')

    call back_projection_deconvolution(y,n)

return
end

```

```

subroutine irtr_four_neig(x,m,y,n)
real x(256,256),y(256,256),xx(256,256)
*****
FUNCTION: This subroutine computes the inverse radon
transform by:
    1. Back_projecting all the projections stored
       in array x giving the back_projection
       function y.
    2. Deconvolving the array y using the inverse_
       filter SQRT(kx*kx + ky*ky*).
External subroutines: bpr_four_neig
                     back_projection_deconvolution
Internal subroutines: none
*****
Send message to user 'Computing Inverse Radon Transform...'

    write(6,1)
    format(1x,'Computing Inverse Radon Transform...')
Initialize array xx
do i=1,n
do j=1,n
xx(j,i)=0.
enddo
enddo

Back project projections stored in x

    call bpr_four_neig(x,m,y,n)

    open(unit=1,file='datair.dat',status='new')
    format(4(1x,f14.4))
    write(1,119)((y(j,i),j=1,128),i=1,128)
    close(1)
end message to user' Deconvolve the reconstruction...'
    write(6,2)
    format(1x,'Deconvolving the reconstruction process...')

    call back_projection_deconvolution(y,n)

return
end

```

```

subroutine irtr_eight(x,m,y,n)
real x(256,256),y(256,256),xx(256,256)
*****
FUNCTION: This subroutine computes the inverse radon
transform by:
    1. Back_projecting all the projections stored
       in array x giving the back_projection
       function y.
    2. Deconvolving the array y using the inverse_
       filter SQRT(kx*kx + ky*ky*).
External subroutines: bpr_nearest_neig
                     back_projection_deconvolution
Internal subroutines: none
*****
Send message to user 'Computing Inverse Radon Transform...'

    write(6,1)
    format(1x,'Computing Inverse Radon Transform...')
Initialize array xx
do i=1,n
do j=1,n
xx(j,i)=0.
enddo
enddo

Back project projections stored in x

    call bpr_eight(x,m,y,n)

Send message to user' Deconvolve the reconstruction...'
    write(6,2)
    format(1x,'Deconvolving the reconstruction process...')

    call back_projection_deconvolution(y,n)
    return
end

```

```

subroutine irtr_eight_neig(x,m,y,n)
real x(256,256),y(256,256),xx(256,256)
*****
FUNCTION: This subroutine computes the inverse radon
transform by:
    1. Back_projecting all the projections stored
       in array x giving the back_projection
       function y.
    2. Deconvolving the array y using the inverse_
       filter SQRT(kx*kx + ky*ky*).
External subroutines: bpr_eight_neig
                     back_projection_deconvolution
Internal subroutines: none
*****
Send message to user 'Computing Inverse Radon Transform...'

    write(6,1)
    format(1x,'Computing Inverse Radon Transform...')
Initialize array xx
do i=1,n
do j=1,n
xx(j,i)=0.
enddo
enddo

Back project projections stored in x

    call bpr_eight_neig(x,m,y,n)

Send message to user' Deconvolve the reconstruction...'
    write(6,2)
    format(1x,'Deconvolving the reconstruction process...')

    call back_projection_deconvolution(y,n)

return
end

```



```

      subroutine bpr_nearest(b,m,a,n)
      real a(256,256),b(256,256),xx(256,256)
      *****
Function: this routine back-projects the
      data stored in b to reconstruct
      the image a.
External subroutines: Rotate_nearest_neig
Internal subroutines: None
Parameters: b _ input, size m X m
      a _ output, size n X n
      *****
x, y and sum are working space
      real x(256,256),y(256,256)
      real sum(256,256)
Send message to user 'Computing back_projection ...'

      write(6,1)
      format(1x,'Computing back_projection...'$)

Compute size of image in which b will be rotated to prevent
corner clipping.
      l=nint(sqrt(2.)*float(m))
Compute centers of arrays to be used later
      nn=n/2+1
      ll=l/2+1
      ln=ll-nn
Initialize working space
      do i=1,l
      do j=1,l
      x(j,i)=0.
      y(j,i)=0.
      sum(j,i)=0.
      a(j,i)=0.
      xx(j,i)=0.
      enddo
      enddo
Compute the angle of rotation in the reverse direction
      do k=1,m
      theta=float(1-k)
back_project projection
      do i=1,m
      do j=1,m
      x(j,i)=b(j,k)
      enddo
      enddo
otate x by angle at which projection was taken
      call rotate_nearest(x,m,y,l,theta)
dd result to previous back_projection function
      do i=1,l
      do j=1,l
      sum(j,i)=sum(j,i)+y(j,i)
      enddo
      enddo

      enddo
indow the back_projection function to extract reconstruction
size n X n) and write it to output
      do i=1,n
      do j=1,n
      a(j,i)=sum(j+ln,i+ln)
      enddo
      enddo

      return
      end

```

```

      subroutine bpr_nearest_neig(b,m,a,n)
      real a(256,256),b(256,256),xx(256,256)
      *****
Function: this routine back-projects the
      data stored in b to reconstruct
      the image a.
External subroutines: Rotate_nearest_neig
Internal subroutines: None
Parameters: b _ input, size m X m
      a _ output, size n X n
      *****
x, y and sum are working space
      real x(256,256),y(256,256)
      real sum(256,256)
Send message to user 'Computing back_projection ...'

      write(6,1)
      format(1x,'Computing back_projection...'$)

Compute size of image in which b will be rotated to prevent
corner clipping.
      l=nint(sqrt(2.)*float(m))
Compute centers of arrays to be used later
      nn=n/2+1
      ll=l/2+1
      ln=ll-nn
Initialize working space
      do i=1,l
      do j=1,l
      x(j,i)=0.
      y(j,i)=0.
      sum(j,i)=0.
      a(j,i)=0.
      xx(j,i)=0.
      enddo
      enddo
Compute the angle of rotation in the reverse direction
      do k=1,m
      theta=float(1-k)
back_project projection
      do i=1,m
      do j=1,m
      x(j,i)=b(j,k)
      enddo
      enddo
otate x by angle at which projection was taken
      call rotate_nearest_neig(x,m,y,l,theta)
dd result to previous back_projection function
      do i=1,l
      do j=1,l
      sum(j,i)=sum(j,i)+y(j,i)
      enddo
      enddo

      enddo

indow the back_projection function to extract reconstruction
size n X n) and write it to output
      do i=1,n
      do j=1,n
      a(j,i)=sum(j+ln,i+ln)
      enddo
      enddo

      return
      end

```

```

      subroutine bpr_lrp_neig(b,m,a,n)
      real a(256,256),b(256,256),xx(256,256)
      *****
Function: this routine back-projects the
      data stored in b to reconstruct
      the image a.
External subroutines: Rotate_nearest_neig
Internal subroutines: None
Parameters: b _ input, size m X m
      a _ output, size n X n
      *****
x, y and sum are working space
      real x(256,256),y(256,256)
      real sum(256,256)
Send message to user 'Computing back_projection ...'

      write(6,1)
      format(1x,'Computing back_projection...'$)

Compute size of image in which b will be rotated to prevent
corner clipping.
      l=nint(sqrt(2.)*float(m))
Compute centers of arrays to be used later
      nn=n/2+1
      ll=l/2+1
      ln=ll-nn
Initialize working space
      do i=1,l
      do j=1,l
      x(j,i)=0.
      y(j,i)=0.
      sum(j,i)=0.
      a(j,i)=0.
      xx(j,i)=0.
      enddo
      enddo
Compute the angle of rotation in the reverse direction
      do k=1,m
      theta=float(1-k)
Back_project projection
      do i=1,m
      do j=1,m
      x(j,i)=b(j,k)
      enddo
      enddo
Rotate x by angle at which projection was taken
      call rotate_lrp_neig(x,m,y,l,theta)
Add result to previous back_projection function
      do i=1,l
      do j=1,l
      sum(j,i)=sum(j,i)+y(j,i)
      enddo
      enddo

      enddo

Show the back_projection function to extract reconstruction
size n X n) and write it to output
      do i=1,n
      do j=1,n
      a(j,i)=sum(j+ln,i+ln)
      enddo
      enddo

      return
      end

```

```

      subroutine bpr_2rp_neig(b,m,a,n)
      real a(256,256),b(256,256),xx(256,256)
      *****
Function: this routine back-projects the
      data stored in b to reconstruct
      the image a.
External subroutines: Rotate_nearest_neig
Internal subroutines: None
Parameters: b _ input, size m X m
      a _ output, size n X n
      *****
x, y and sum are working space
      real x(256,256),y(256,256)
      real sum(256,256)
Send message to user 'Computing back_projection ...'

      write(6,1)
      format(1x,'Computing back_projection...'$)

Compute size of image in which b will be rotated to prevent
corner clipping.
      l=nint(sqrt(2.)*float(m))
Compute centers of arrays to be used later
      nn=n/2+1
      ll=l/2+1
      ln=ll-nn
Initialize working space
      do i=1,l
      do j=1,l
      x(j,i)=0.
      y(j,i)=0.
      sum(j,i)=0.
      a(j,i)=0.
      xx(j,i)=0.
      enddo
      enddo
Compute the angle of rotation in the reverse direction
      do k=1,m
      theta=float(1-k)
back_project projection
      do i=1,m
      do j=1,m
      x(j,i)=b(j,k)
      enddo
      enddo
otate x by angle at which projection was taken
      call rotate_2rp_neig(x,m,y,l,theta)
dd result to previous back_projection function
      do i=1,l
      do j=1,l
      sum(j,i)=sum(j,i)+y(j,i)
      enddo
      enddo

      enddo
indow the back_projection function to extract reconstruction
size n X n) and write it to output
      do i=1,n
      do j=1,n
      a(j,i)=sum(j+ln,i+ln)
      enddo
      enddo

      return
      end

```

```

      subroutine bpr_four(b,m,a,n)
      real a(256,256),b(256,256),xx(256,256)
*****
Function: this routine back-projects the
          data stored in b to reconstruct
          the image a.
External subroutines: Rotate_nearest_neig
Internal subroutines: None
Parameters: b _ input, size m X m
            a _ output, size n X n
*****
x, y and sum are working space
      real x(256,256),y(256,256)
      real sum(256,256)
Send message to user 'Computing back_projection ...'

      write(6,1)
      format(1x,'Computing back_projection...'$)

Compute size of image in which b will be rotated to prevent
corner clipping.
      l=nint(sqrt(2.)*float(m))
Compute centers of arrays to be used later
      nn=n/2+1
      ll=l/2+1
      ln=ll-nn
Initialize working space
      do i=1,l
      do j=1,l
      x(j,i)=0.
      y(j,i)=0.
      sum(j,i)=0.
      a(j,i)=0.
      xx(j,i)=0.
      enddo
      enddo
Compute the angle of rotation in the reverse direction
      do k=1,m
      theta=float(1-k)
back_project projection
      do i=1,m
      do j=1,m
      x(j,i)=b(j,k)
      enddo
      enddo
otate x by angle at which projection was taken
      call rotate_four(x,m,y,l,theta)
dd result to previous back_projection function
      do i=1,l
      do j=1,l
      sum(j,i)=sum(j,i)+y(j,i)
      enddo
      enddo

      enddo
indow the back_projection function to extract reconstruction
size n X n) and write it to output
      do i=1,n
      do j=1,n
      a(j,i)=sum(j+ln,i+ln)
      enddo
      enddo

      return
      end

```

```

      subroutine bpr_four_neig(b,m,a,n)
      real a(256,256),b(256,256),xx(256,256)
      *****
Function: this routine back-projects the
      data stored in b to reconstruct
      the image a.
External subroutines: Rotate_four_neig
Internal subroutines: None
Parameters: b _ input, size m X m
      a _ output, size n X n
      *****
x, y and sum are working space
      real x(256,256),y(256,256)
      real sum(256,256)
Send message to user 'Computing back_projection ...'

      write(6,1)
      format(1x,'Computing back_projection...'$)

Compute size of image in which b will be rotated to prevent
corner clipping.
      l=nint(sqrt(2.)*float(m))
Compute centers of arrays to be used later
      nn=n/2+1
      ll=l/2+1
      ln=ll-nn
Initialize working space
      do i=1,l
      do j=1,l
      x(j,i)=0.
      y(j,i)=0.
      sum(j,i)=0.
      a(j,i)=0.
      xx(j,i)=0.
      enddo
      enddo
Compute the angle of rotation in the reverse direction
      do k=1,m
      theta=float(1-k)
Back_project projection
      do i=1,m
      do j=1,m
      x(j,i)=b(j,k)
      enddo
      enddo
Rotate x by angle at which projection was taken
      call rotate_four_neig(x,m,y,l,theta)
Add result to previous back_projection function
      do i=1,l
      do j=1,l
      sum(j,i)=sum(j,i)+y(j,i)
      enddo
      enddo

      enddo
Window the back_projection function to extract reconstruction
size n X n) and write it to output
      do i=1,n
      do j=1,n
      a(j,i)=sum(j+ln,i+ln)
      enddo
      enddo

      return
      end

```

```

      subroutine bpr_eight(b,m,a,n)
      real a(256,256),b(256,256),xx(256,256)
*****
Function: this routine back-projects the
          data stored in b to reconstruct
          the image a.
External subroutines: Rotate_nearest_neig
Internal subroutines: None
Parameters: b _ input, size m X m
            a _ output, size n X n
*****
x, y and sum are working space
      real x(256,256),y(256,256)
      real sum(256,256)
Send message to user 'Computing back_projection ...'

      write(6,1)
      format(1x,'Computing back_projection...'$)

Compute size of image in which b will be rotated to prevent
corner clipping.
      l=nint(sqrt(2.)*float(m))
Compute centers of arrays to be used later
      nn=n/2+1
      ll=l/2+1
      ln=ll-nn
Initialize working space
      do i=1,l
      do j=1,l
      x(j,i)=0.
      y(j,i)=0.
      sum(j,i)=0.
      a(j,i)=0.
      xx(j,i)=0.
      enddo
      enddo
Compute the angle of rotation in the reverse direction
      do k=1,m
      theta=float(1-k)
Back_project projection
      do i=1,m
      do j=1,m
      x(j,i)=b(j,k)
      enddo
      enddo
Rotate x by angle at which projection was taken
      call rotate_eight(x,m,y,l,theta)
Add result to previous back_projection function
      do i=1,l
      do j=1,l
      sum(j,i)=sum(j,i)+y(j,i)
      enddo
      enddo

      enddo

Window the back_projection function to extract reconstruction
size n X n) and write it to output
      do i=1,n
      do j=1,n
      a(j,i)=sum(j+ln,i+ln)
      enddo
      enddo

      return
      end

```

```

      subroutine bpr_eight_neig(b,m,a,n)
      real a(256,256),b(256,256),xx(256,256)
      *****
Function: this routine back-projects the
      data stored in b to reconstruct
      the image a.
External subroutines: Rotate_eight_neig
Internal subroutines: None
Parameters: b _ input, size m X m
      a _ output, size n X n
      *****
x, y and sum are working space
      real x(256,256),y(256,256)
      real sum(256,256)
Send message to user 'Computing back_projection ...'

      write(6,1)
      format(1x,'Computing back_projection...'$)

Compute size of image in which b will be rotated to prevent
corner clipping.
      l=nint(sqrt(2.)*float(m))
Compute centers of arrays to be used later
      nn=n/2+1
      ll=l/2+1
      ln=ll-nn
Initialize working space
      do i=1,l
      do j=1,l
      x(j,i)=0.
      y(j,i)=0.
      sum(j,i)=0.
      a(j,i)=0.
      xx(j,i)=0.
      enddo
      enddo
Compute the angle of rotation in the reverse direction
      do k=1,m
      theta=float(1-k)
Back_project projection
      do i=1,m
      do j=1,m
      x(j,i)=b(j,k)
      enddo
      enddo
Rotate x by angle at which projection was taken
      call rotate_eight_neig(x,m,y,l,theta)
Add result to previous back_projection function
      do i=1,l
      do j=1,l
      sum(j,i)=sum(j,i)+y(j,i)
      enddo
      enddo

      enddo
Window the back_projection function to extract reconstruction
size n X n) and write it to output
      do i=1,n
      do j=1,n
      a(j,i)=sum(j+ln,i+ln)
      enddo
      enddo

      return
      end

```


subroutine back_projection_deconvolution(a,n)

Function: This subroutine deconvolves a reconstruction obtained by straight backprojection (un-filtered) using the appropriate inverse filter, i.e. $\sqrt{k_x^2 + k_y^2}$ where k_x and k_y are the spatial frequencies.

```
real a(256,256),aa(256,256),xx(256,256)
real b(256,256),bb(256,256)
```

initialize array b

```
do i=1,n
do j=1,n
aa(j,i)=0.
bb(j,i)=0.
enddo
enddo
```

take Fourier transform of a

```
call fft2d(a,aa,n,-1)
```

multiply spectrum by inverse filter

```
nn=n/2+1
do i=1,n
do j=1,n
xk=float(i-nn)/float(n)
yk=float(j-nn)/float(n)
fil=sqrt(xk*xk+yk*yk)!value of inverse filter at (xk,yk)
b(j,i)=a(j,i)*fil
bb(j,i)=aa(j,i)*fil
enddo
enddo
```

```
call fft2d(b,bb,n,1)
```

write output

```
do i=1,n
do j=1,n
a(j,i)=b(j,i)
enddo
enddo
```

```
return
end
```

```
subroutine project(a,b,n)
```

```
*****
```

```
Function: This subroutine sums
```

```
over all rows of an n X n array a giving 1D
```

```
projection of size n array b
```

```
*****
```

```
real a(256,256),b(256),s(256,256),z(256)
```

```
initialize arrays s & b
```

```
do j=1,n
```

```
b(j)=0.
```

```
z(j)=0.
```

```
do i=1,n
```

```
s(j,i)=0.
```

```
enddo
```

```
enddo
```

```
set initial value for summing process and sum all pixels along
```

```
row i together
```

```
do j=1,n
```

```
s(j,1)=a(j,1)
```

```
do i=2,n
```

```
s(j,i)=s(j,i-1)+a(j,i)
```

```
enddo
```

```
Assign value of sum at column j to output _b(j)
```

```
b(j)=s(j,n)
```

```
enddo
```

```
return
```

```
end
```

APPENDIX D

COMMON ROUTINES

```

subroutine read_image(x,n)
real x(256,256)
character*6 answer

```

FUNCTION: Reads (using free format) an image from a given data file.

PARAMETERS

Input: x - image
n - size of image (=n x n)

Output:

EXTERNAL SUBROUTINES: None

INTERNAL SUBROUTINES: open_read_image

Call subroutine open_read_image to assigns unit number 1 to data file
and output appropriate response to user (i.e. size and name of file).

```

continue
call open_read_image(n)

```

Send message to user 'reading data...'

```

write(6,1)
format(1x,'Reading data...' $)

```

Read data from file (unit number = 1) to array x
(using implied do-loop).

```

read(1,*,end=99,err=88)((x(j,i),j=1,n),i=1,n)
goto 3

```

ERROR CONDITIONS:

If read goes beyond end of file, output appropriate message
with options allowing user to either input another file or
stop process.

```

write(6,*)'Attempt to read beyond end of file!'
write(6,*)!skip a line
write(6,7)
format(1x,'Continue? <n>' $)
read(5,4)answer
format(a6)
if(answer.eq.'y')goto 2
if(answer.ne.'y')goto 8
stop

```

If error occurs while reading file, output appropriate message
with option of allowing user to input another file.

```

write(6,*)'Error when reading data from file!'
write(6,*)!skip a line
write(6,9)
format(1x,'Continue? <n>' $)
read(5,10)answer
format(a6)
if(answer.eq.'y')goto 2
if(answer.ne.'y')goto 8

```

continue

Close file.

close(1)

Skip a line to indicate that process has finished.

write(6,*)

return
end

subroutine open_read_image(n)
character*40 filer

FUNCTION: Asks the user to 'enter size of image...'
and 'enter name of image data file...'.

Prompt user to input size of image

ask whether image is square or otherwise

write(6,1)
format(1x,'Enter size of image (<=512):'\$)
read(5,*,err=99)n

check to see if input is correct

call check_size(n,iopt)
if(iopt.eq.1)goto 2
if(iopt.eq.2)goto 3

If error occurs in input type message to '...try again!'

write(6,*)'Wrong type, try again!'
goto 2
continue

Prompt user to input name of data file

00 write(6,101)
format(1x,'Enter name of data file (input):'\$)
read(5,102)filer

format(a40)
open(unit=1,file=filer,status='old',err=66)
open(unit=1,file=filer,status='old',err=66,form='unformatted')
rewind 1
goto 1001

write(6,103)
format(1x,'File does not exist!')
goto 1000
1 continue

return
end

subroutine check_size(n,iopt)

UNCTION: Checks to see if the size of the array n
is consistent with bounds, i.e. $0 \leq n \leq 512$.

If iopt=1 then error has been made.
If iopt=2 then value of n is within bounds.

```
if((n.gt.512).or.(n.le.0))then  
write(6,1)  
format(1x,'Wrong size, try again!')  
iopt=1  
else  
iopt=2  
endif
```

```
return  
end
```

```
subroutine write_image(x,n)
real x(256,256)
```

FUNCTION: Writes (using free format) array x to a named data file.

PARAMETERS

Input: x - image
n - size of image (=n x n)

Output:

EXTERNAL SUBROUTINES: None

INTERNAL SUBROUTINES: open_write_image

call open_write_image to assign unit number
and prompt user to enter name of file.

```
call open_write_image
```

Send message to user 'writing data...'

```
write(6,1)
format(1x,'Writing data...'$)
```

Write out array x to data file with unit number 1.

```
write(1,*)((x(j,i),j=1,n),i=1,n)
```

Close file.

```
close(1)
```

Skip a line to indicate that process has finished.

```
write(6,*)
```

```
return
end
```

```
subroutine open_write_image
character*40 filew
character*6 answer
```

FUNCTION: Asks the user to and 'enter name of data file...'
and open file.

```
0 write(6,101)
format(1x,'Enter name of data file (output):'$)
read(5,102)filew
format(a40)
```

```
print *,'File name is:',filew
write(6,103)
format(1x,'Is this name OK? <y>'$)
read(5,104)answer
format(a6)
if(answer.eq.'n')goto 1000
```

```
open(unit=1,file=filew,status='new')
```

return
end


```
subroutine write_signal(s,n)
real s(512)
```

FUNCTION: Writes (free format) a signal to a named data file.

AUTHOR: J M Blackledge

PARAMETERS: s, n

Input: s - Data (digital signal).

n - Size of signal.

Note: maximum size of n is 512.

Output: None

GLOBAL VARIABLES: None

LOCAL VARIABLES: None

EXTERNAL FUNCTIONS: None

INTERNAL FUNCTIONS: None

EXTERNAL SUBROUTINES: None

INTERNAL SUBROUTINES: open_write_signal

call open_write_signal to assign unit number for file
and to prompt user to enter size and name of file
to be created.

call open_write_signal

Send message to user 'writing data...'

write(6,1)

format(1x,'Writing data...'\$)

write out array s to data file with unit number 1

do i=1,n

write(1,*)s(i)

enddo

close file

close(1)

skip a line to indicate that process has finished.

write(6,*)

return

end

subroutine open_write_signal

character*40 filew

character*6 answer

FUNCTION: Asks user to 'enter name of data file...'
and opens file to write data.

prompt user to input name of data file

```
000  write(6,101)
001  format(1x,'Enter name of data file (output):'$)
002  read(5,102)filew
002  format(a40)
002  print *,'File name is:',filew
002  write(6,103)
003  format(1x,'Is this name OK? <y>','$)
003  read(5,104)answer
004  format(a6)
004  if(answer.eq.'n')goto 1000
```

open file - unit file number=1

```
open(unit=1,file=filew,status='new')
```

```
return
end
```

```
subroutine display_image(z,n,icol,ipol)
real z(256,256),a(256,256),xx(256,256),xy(256,256)
```

FUNCTION: Uses UNIRAS-GIMAGE to display a 4-bit (16-level) image.

PARAMETERS

Input: z - image

n - size of image (=n x n).

icol - colour specification:

icol = 1 gives UNIRAS default colour scale 1.
icol = 2 gives UNIRAS default colour scale 2
(spectral colour code - blue to red).
icol = 3 gives grey scale
icol = 4 gives topographic colour scale.
icol = 5 gives seismic colour scale.

ipol - polarity of scale:

ipol = 1 gives positive image.
ipol = -1 gives negative image.

Output: None

EXTERNAL SUBROUTINES: UNIRAS library - rcsini
rshade
rclass
gimage
gempty

DIPLIB - write_image

INTERNAL SUBROUTINES: None

Declare internal work space.

```
real zz(512,512),zcl(15)
real maxp,maxn
character*6 answer
character*150 title ,title1
```

Specify data for a 4-bit display.

```
data zcl/1.,2.,3.,4.,5.,6.,7.,8.,9.,10.,11.,12.,13.,14.,15./
```

Find maximum positive value of data maxp
and maximum negative value of data maxn.

If negative numbers exist in the image,
then the largest negative value is set
to zero and the largest positive value
is set to 16. In this way, any image
can be observed regardless of whether
it has single or dual polarity.

```
maxp=0.
maxn=0.
do i=1,n
do j=1,n
if(z(j,i).lt.maxn)maxn=z(j,i)
if(z(j,i).gt.maxp)maxp=z(j,i)
enddo
```

```
enddo
```

Note: the default UNIRAS gray scale gives
a 0 to 16 - white to black image
respectively (i.e. a negative).
Hence, if icol=3 and ipol = 1
the order of the numbers is
reversed, thus:

```
if((icol.eq.3).and.(ipol.gt.0))then
do i=1,n
do j=1,n
zz(j,i)=16.-(16.*(z(j,i)-maxn)/(maxp-maxn))
enddo
enddo
```

Draw frame so that extent of the image can
be observed inside the UNIRAS window.

```
do i=1,n
zz(1,i)=0.
zz(n,i)=0.
enddo
do j=1,n
zz(j,1)=0.
zz(j,n)=0.
enddo
```

```
endif
```

Compute negative image (in the case when
icol = 3 and ipol = -1).

```
if((icol.eq.3).and.(ipol.lt.0))then
do j=1,n
do i=1,n
zz(j,i)=16.*(z(j,i)-maxn)/(maxp-maxn)
enddo
enddo
endif
```

Compute positive colour coded image
in any case where ipol = 1 and
icol not= 3.

```
if((icol.ne.3).and.(ipol.gt.0))then
do i=1,n
do j=1,n
zz(j,i)=16.*(z(j,i)-maxn)/(maxp-maxn)
enddo
enddo
endif
```

Compute negative colour coded image
in any case where ipol = -1 and
icol not= 3.

```
if((icol.ne.3).and.(ipol.lt.0))then
do i=1,n
do j=1,n
zz(j,i)=16.-(16.*(z(j,i)-maxn)/(maxp-maxn))
enddo
enddo
endif
```

all relevant subroutines from UNIRAS 6 Library.

```
call rcsini !set default color scale definitions.
```

```

call rshade(icol,0) !Colour coding.
call rclass(zcl,15,0) !Class limits.
call gimage(zz,n,0) !Initialization of gimage.

```

Build matrix column by column.

```

do i=1,n
call gimage(zz(1,i),n,1) !Plot the scan-line
enddo
call gimage(zz,n,9999) !Terminate gimage

```

Display title

```

0      write(6,*) 'enter main title '
      read (5,10) title
0      format (a)
      write(6,*) 'Enter Horizontal_start'
      read(5,*) xh
      write(6,*) 'enter Vertical_start'
      read(5,*) yv
      call rtx (len(title), title, xh, yv)
      call rtxhei(.5)
call rtx (len(title), title, 45.0, 176.0)
      write(6,*) 'enter subtitle'
      read(5,40) title1
0      format(a)
      write(6,*) 'enter another Vertical_start'
      read(5,*) yva
call rtx (len(title1), title1, xh, yva)
      call rtxhei(.5)
call rtx (len(title1), title1, 80.0, 169.0)
call gempty !Empty graphics buffer
      write(6,*) 'would you like to change the position'
      read (5,20) c
0      format(a1)
      if ((c.eq.'Y') .or. (c.eq.'y')) then
goto 30
      else
      endif

```

Hold display until user 'hits' Return,
giving time to inspect the image.

```

      write(6,*) 'Hit Return to continue'
      read '(a)'

```

ask user if he wants to store the image in
a named data file

```

      write(6,11)
      format(1x,'store file image? <n>'$)
      read(5,12) answer
      format(a6)
      if(answer.eq.'y') then
      call write_image(z,n)
      else
      endif

```

```

return
end

```

```
subroutine display_image_grey(z,n,icol,ipol)
real z(256,256)
```

FUNCTION: Uses UNIRAS-GIMAGE to display a 4-bit (16-level) image.

PARAMETERS

Input: z - image

n - size of image (=n x n).

icol - colour specification:

icol = 1 gives UNIRAS default colour scale 1.
icol = 2 gives UNIRAS default colour scale 2
(spectral colour code - blue to red).
icol = 3 gives grey scale
icol = 4 gives topographic colour scale.
icol = 5 gives seismic colour scale.

ipol - polarity of scale:

ipol = 1 gives positive image.
ipol = -1 gives negative image.

Output: None

EXTERNAL SUBROUTINES: UNIRAS library - rcsini
rshade
rclass
gimage
gempty

DIPLIB - write_image

INTERNAL SUBROUTINES: None

Declare internal work space.

real zz(256,256),zcl(256)
real maxp,maxn
character*6 answer
character*150 title ,title1

Specify data for a 4-bit display.

data zcl/1.,2.,3.,4.,5.,6.,7.,8.,9.,10.,11.,12.,13.,14.,15./

Specify data for a n-bit display.

write(6,*)'enter no. of gray level'
read(5,*) nm
do i=1,nm
zcl(i)=i
enddo

Find maximum positive value of data maxp
and maximum negative value of data maxn.

If negative numbers exist in the image,
then the largest negative value is set
to zero and the largest positive value
is set to 16. In this way, any image
can be observed regardless of whether
it has single or dual polarity.

maxp=0.
maxn=0.

```

do i=1,n
do j=1,n
if(z(j,i).lt.maxn)maxn=z(j,i)
if(z(j,i).gt.maxp)maxp=z(j,i)
enddo
enddo

```

Note: the default UNIRAS gray scale gives
a 0 to 16 - white to black image
respectively (i.e. a negative).
Hence, if icol=3 and ipol = 1
the order of the numbers is
reversed, thus:

```

if((icol.eq.3).and.(ipol.gt.0))then
do i=1,n
do j=1,n
zz(j,i)=zcl(nm) - (zcl(nm) * (z(j,i)-maxn) / (maxp-maxn))
enddo
enddo

```

Draw frame so that extent of the image can
be observed inside the UNIRAS window.

```

do i=1,n
zz(1,i)=0.
zz(n,i)=0.
enddo
do j=1,n
zz(j,1)=0.
zz(j,n)=0.
enddo

```

```

endif

```

Compute negative image (in the case when
icol = 3 and ipol = -1).

```

if((icol.eq.3).and.(ipol.lt.0))then
do j=1,n
do i=1,n
zz(j,i)=zcl(nm) * (z(j,i)-maxn) / (maxp-maxn)
enddo
enddo
endif

```

Compute positive colour coded image
in any case where ipol = 1 and
icol not= 3.

```

if((icol.ne.3).and.(ipol.gt.0))then
do i=1,n
do j=1,n
zz(j,i)=zcl(nm) * (z(j,i)-maxn) / (maxp-maxn)
enddo
enddo
endif

```

compute negative colour coded image
in any case where ipol = -1 and
icol not= 3.

```

if((icol.ne.3).and.(ipol.lt.0))then
do i=1,n
do j=1,n
zz(j,i)=zcl(nm) - (zcl(nm) * (z(j,i)-maxn) / (maxp-maxn))
enddo
enddo

```

```
endif
```

Call relavent subroutines from UNIRAS 6 Library.

```
call rcsini !set default color scale definitions.
call rshade(icol,0) !Colour coding.
call rclass(zcl,nm-1,0) !Class limits.
call gimage(zz,n,0) !Initialization of gimage.
```

Build matrix column by column.

```
do i=1,n
call gimage(zz(1,i),n,1) !Plot the scan-line
enddo
call gimage(zz,n,9999) !Terminate gimage
```

Display title

```
write(6,*)'enter main title '
read (5,10) title
format (a)
write(6,*) 'Enter Horizontal_start'
read(5,*) xh
write(6,*) 'enter Vertical_start'
read(5,*) yv
call rtx (len(title), title, xh, yv)
call rtxhei(.5)
call rtx (len(title), title, 45.0, 176.0)
write(6,*) 'enter subtitle'
read(5,40) title1
format(a)
write(6,*) 'enter another Vertical_start'
read(5,*) yva
call rtx (len(title1), title1, xh, yva)
call rtxhei(.5)
call rtx (len(title1), title1, 80.0, 169.0)
call gempty !Empty graphics buffer
write(6,*) 'would you like to change the position'
read (5,20) c
format(a1)
if ((c.eq.'Y') .or. (c.eq.'y')) then
goto 30
else
endif
```

Hold display until user 'hits' Return,
giving time to inspect the image.

```
write(6,*)'Hit Return to continue'
read '(a)'
```

ask user if he wants to store the image in
a named data file

```
write(6,11)
format(1x,'store file image? <n>'$)
read(5,12)answer
format(a6)
if(answer.eq.'y')then
call write_image(z,n)
else
endif

return
end
```



```

subroutine display_signal(s,n)
dimension x(512),y(512),s(512)
character*6 answer
character*150 title ,title1

```

FUNCTION: Displays a digital signal held in array s
using appropriate UNIRAS 6.1 modules.

NOTE: The program automatically scales the axis so
that the display lies within the bounds of
the x and y axis.

groute and gopen must be called before this module
or a series of calls to it are made.

gclose must be called after using this module
or a sequence of calls to it.

AUTHOR: J M Blackledge

PARAMETERS:

Input: s - digital signal
n - no. of samples (<=512)

Output: None

GLOBAL VARIABLES: None

LOCAL VARIABLES: x - x-axis
y - y-axis

EXTERNAL FUNCTIONS: None

INTERNAL FUNCTIONS: None

EXTERNAL SUBROUTINES: DSPLIB - write_signal
UNIRAS - gempty, gclear

INTERNAL SUBROUTINES: axis, plot

```

real ymax,ymin,xmin,xmax

```

Generate x and y values.

```

do i=1,n
x(i)=float(i-1)!line elements or lixels
y(i)=s(i)!data - z unchanged on output
enddo

```

Compute largest value of y, ymax.

```

ymax=0.
do i=1,n
if(y(i).gt.ymax)ymax=y(i)
enddo

```

Comput the smallest value of y, ymin.

```

ymin=ymax
do i=1,n
if(y(i).lt.ymin)ymin=y(i)
enddo
ymin=0.

```

Define the minumum and maximum values of the x-axis.

```
xmin=0.  
xmax=float(n-1)
```

Plot axis.

```
call axis(xmin,xmax,ymin,ymax,0.7,0.03)
```

Plot signal.

```
call plot(x,y,n)  
write(6,*) 'enter histogram title'  
read(5,40) title1  
format(a)  
write(6,*) 'enter horizontal& Vertical_start'  
read(5,*) xv,yv  
call rtx(len(title1), title1, xh, yva)  
call rtxhei(.5)  
xm=.05*(xmax-xmin)+xmin  
ym=1.05*(ymax-ymin)+ymin  
call rtx(len(title1),title1,xm,ym)  
call gempty !empty buffer
```

Keep plot until user 'hits' the Return key.

```
write(6,*) 'hit Return to continue'  
read '(a)'
```

Ask user if data file is required for generating a hardcopy.

```
write(6,10)  
format(1x,'file graph <n>?' '$)  
read(5,11) answer  
format(a6)  
if(answer.eq.'y') then  
call write_signal(z,n)  
else  
endif
```

Clear plot in UNIRAS window.

```
call greset  
call gclear  
  
return  
end
```

```
subroutine axis(xmin,xmax,ymin,ymax,sizfac,heifac)
```

FUNCTION: Plots the axes of the user 2D coordinate system.

NOTE: This program is a modification of the UNIRAS example program AXI001 (see UNIRAS AGL/AXES Version 6).

```
REAL XMIN,YMIN,XMAX,YMAX  
REAL XLEN,YLEN,ZLEN  
REAL SIZFAC, HEIFAC, HEIGHB  
REAL BOX2MM  
EXTERNAL BOX2MM
```

Step 1: Define the 2D limits of the axes

```
CALL GLIMIT(XMIN,XMAX,YMIN,YMAX,0.0,0.0)
```

Step 2: Calculate for a suitable square viewport size

```
CALL GRPSIZ(XSIZMM,YSIZMM)
SIZE = SIZFAC * MIN(XSIZMM,YSIZMM)
XSIZE = SIZE
YSIZE = SIZE
```

Set up the viewport

```
XORG = (XSIZMM-XSIZE)/2.
YORG = (YSIZMM-YSIZE)/2.
CALL GVPOR(XORG,YORG,XSIZE,YSIZE)
```

Step 3: Define the 2D workbox aspect ratio
to be equal to the viewport's.

1st method: Make XLEN and YLEN to be equal
to the viewport sides.

2nd method: Assign the shorter side of the
workbox to be 1. and the other
side proportional to it.

Set ZLEN = 0. for both methods.

The second method is used here.

```
XLEN = XSIZE/MIN(XSIZE,YSIZE)
YLEN = YSIZE/MIN(XSIZE,YSIZE)
ZLEN = 0.
CALL GWBOX(XLEN,YLEN,ZLEN)
```

Step 4: Activate 2D scaling - scaling must be active,
otherwise, no axes will be plotted.

```
CALL GSCALE
```

Step 5: Calculate a suitable axis-label height
based on the shorter side of the workbox.
Note: The function "BOX2MM" converts box
units to mm.

```
HEIGHB = HEIFAC*BOX2MM(1.0)
```

Step 6: Plot two numeric and user-labeled axes
at the limits of the 2D coordinate system

```
heighb2=2.*heighb
```

```
call raxdis(6,1,0)
call raxtex(6,-1,'Lixels',999.999,999.999,heighb2)
CALL RAXIS(1,YMIN,HEIGHB,1)
call raxtex(6,-1,'Amplitude',999.999,999.999,heighb2)
CALL RAXIS(2,XMIN,HEIGHB,1)
```

```
call raxdis(2,0,0)
call raxdis(3,0,0)
call raxdis(4,0,0)
call raxis(1,ymax,heighb,2)
call raxis(2,xmax,heighb,2)
call raxdis(2,1,0)
call raxdis(3,1,0)
call raxdis(4,1,0)
```

```
RETURN
END
```

```
subroutine plot(x,y,n)
real x(512),y(512)
```

FUNCTION: Plots the signal

EXTERNAL SUBROUTINES: UNIRAS 6 - bdimx, blivec

```
idir=1
call bdimx(idir,x,n)
iextr=0
itreat=0
call blivec(y,n,iextr,itreat)
```

```
return
end
```

```

subroutine inverse_filter_2d(array,n)
*****
This subroutine computes the Foward, Backward and Central
difference approximation of the first order differential
equation.
*****
real array(512,512),a(512,512)
character*12 answer,yes,no
Initialize array a
do i=1,n
do j=1,n
a(j,i)=0.
array(j,i)=0.
enddo
enddo
nn=n/2+1
do i=1,n
do j=1,n
xx=( float(i-nn)/float(n))
yy=( float(j-nn)/float(n))
array(j,i)=sqrt(xx**2.+yy**2.)
enddo
enddo
write(6,7)
format(1x,'another difference (answer "y" or "n"):'$)
read(5,8)answer
format(a12)
if((answer.eq.'y').or.(answer.eq.'Y')) then
go to 2
else
endif

return
end

```

```

subroutine histogram_signal(x,n,z,lev)
real x(256),z(256),y(256)
real h(256),xx(256),hl(256)
real max,yk(256)

```

FUNCTION: Computes the histogram h of a digital image x.

PARAMETERS

Input: x - signal.
n - size of signal
lev - number of grey levels (<=256)

Output: z - histogram (<=256 - an 8-bit signal).

INTERNAL SUBROUTINES: None

EXTERNAL SUBROUTINES: DIPLIB - Norm_signal

Initialize arrays

```

do i=1,lev
z(i)=0.
h(i)=0.
hl(i)=0.
enddo
do i=1,n
y(i)=0.
enddo

```

Generate work space.

```

do i=1,n
y(i)=x(i)
enddo

```

Normalize signal.

```

call norm_signal(y,n)

```

Quantize signal (truncate decimal parts of array
using FORTRAN function int) between 0 and lev-1.

```

do i=1,n
yk(i)=anint(y(i)*float(lev-1))
enddo

```

Count the number of pixels which occupy each level
between 0 and lev-1.

```

do k=1,lev
z(k)=0.
enddo
do k=1,lev
do i=1,n
if(yk(i).eq.float(k-1)) z(k)=z(k)+1.
enddo
enddo
return
end

```

```

subroutine histogram_image(x,n,z,lev,gsum)
real x(256,256),z(256),y(256,256)
real h(256),h1(256),h2(256),xx(256,256)
real h3(256),h4(256),h5(256)
real max,yk(256,256)

```

FUNCTION: Computes the histogram h of a digital image x.

PARAMETERS

Input: x - image.

n - size of image = n x n

lev - number of grey levels (<=256)

Output: z - histogram (<=256 - an 8-bit image).

gsum Summing up grey_level from 9_16 level

INTERNAL SUBROUTINES: None

EXTERNAL SUBROUTINES: DIPLIB - Norm_image

Initialize arrays

```

do i=1,lev

```

```

z(i)=0.

```

```

h(i)=0.

```

```

h1(i)=0.

```

```

h2(i)=0.

```

```

h3(i)=0.

```

```

h4(i)=0.

```

```

h5(i)=0.

```

```

enddo

```

```

do i=1,n

```

```

do j=1,n

```

```

y(j,i)=0.

```

```

enddo

```

```

enddo

```

Generate work space.

```

do i=1,n

```

```

do j=1,n

```

```

y(j,i)=x(j,i)

```

```

enddo

```

```

enddo

```

Normalize image.

```

call norm_image(y,n)

```

Quantize image (truncate decimal parts of array
using FORTRAN function int) between 0 and lev-1.

```

do i=1,n

```

```

do j=1,n

```

```

yk(j,i)=aint(y(j,i)*float(lev-1))

```

```

enddo

```

```

enddo

```

Count the number of pixels which occupy each level
between 0 and lev-1.

```

do k=1,lev

```

```

z(k)=0.

```

```

enddo

```

```

do k=1,lev

```

```

do i=1,n

```

```
do j=1,n
  if(yk(j,i).eq.float(k-1)) z(k)=z(k)+1.
enddo
enddo
enddo
```

Summing up the grey_level from 9 to 16 level.

```
gsum=0.0
do k=9,lev
  gsum=gsum+z(k)
enddo
return
end
```



```

subroutine head_phantom(array,n)
*****
This subroutine generates a circle with a given radius
and grey-tone in an array "array" at a given point.
*****

```

```

real array(256,256),a(256,256)
character*12 answer,yes,no

```

Initialize array a

```

do i=1,n
do j=1,n
a(j,i)=0.
enddo
enddo
write(6,1)
format(1x,'input radius of ellipse in pixels: '$)
read(5,*)rad
write(6,3)
format(1x,'coordinates of centre of ellipse: '$)
read(5,*)icenx,iceny
write(6,5)
format(1x,'Major and minor axis of ellipse: '$)
read(5,*)ax,ay
write(6,4)
format(1x,'grey-tone: '$)
read(5,*)grey
do j=1,n
do i=1,n
ss=sqrt((float(icenx-i)/ax)**2.+(float(iceny-j)/ay)**2.)
if(ss.le.1) array(i,j)=grey
enddo
enddo
write(6,7)
format(1x,'another ellipse (answer "y" or "n"): '$)
read(5,8)answer
format(a12)
if((answer.eq.'y').or.(answer.eq.'Y')) then
go to 10
else
endif

return
end

```

```
subroutine gnoise(s,n,iseed)
real s(256),w1(256)
```

FUNCTION: Generates an n size array s of Gaussian distributed noise
with zero mean and unit variance.

AUTHOR: J M Blackledge

PARAMETERS

Input: iseed -set seed to a positive integer.
n - array size

Output: s - fGaussian distributed random signal.

EXTERNAL SUBROUTINES: None

INTERNAL SUBROUTINES: gasdev, ran1

NB : output consists of positive numbers only.

use function gasdev to generate array of random numbers.

```
idum=-iseed
do i=1,n
```

NB : output consists of positive and negative numbers.

```
s(i)=gasdev(idum)
```

NB : output consists of positive numbers only.

```
s(i)=abs(gasdev(idum))
```

```
enddo
return
end
```

```
function gasdev(idum)
```

FUNCTION: Returns a Gaussian distributed deviate with zero mean and unit
variance, using RAN1(IDUM) as the source of uniform deviates.
(from 'Numerical Recipes' by Press et al. 1987).

```
data iset/0/
if (iset.eq.0) then
v1=2.*rang(idum) -1.
v2=2.*rang(idum) -1.
r=v1**2.+v2**2.
if (r.ge.1.) goto 1
fac=sqrt(-2.*log(r)/r)
gset=v1*fac
gasdev=v2*fac
iset=1
else
gasdev=gset
iset=0
endif

return
end
```

```
function rang(idum)
```

FUNCTION: This function returns a uniform random number between 0. and 1.
set IDUM to any negative value to initiate or reinitiate the
sequence. This source code is taken from 'Numerical Recipes'
by Press et al.

```
dimension r(97)
parameter(m1=259200,ia1=7141,ic1=54773,rm1=1./m1)
parameter(m2=134456,ia2=8121,ic2=28411,rm2=1./m2)
parameter(m3=243000,ia3=4561,ic3=51349)
data iff /0/
```

```
if (idum.lt.0.or.iff.eq.0) then
```

```
  iff=1
```

```
  ix1=mod(ic1-idum,m1)
```

```
  ix1=mod(ia1*ix1+ic1,m1)
```

```
  ix2=mod(ix1,m2)
```

```
  ix1=mod(ia1*ix1+ic1,m1)
```

```
  ix3=mod(ix1,m3)
```

```
  do i=1,97
```

```
    ix1=mod(ia1*ix1+ic1,m1)
```

```
    ix2=mod(ia2*ix2+ic2,m2)
```

```
    r(i)=(float(ix1)+float(ix2)*rm2)*rm1
```

```
  enddo
```

```
  idum=1
```

```
endif
```

```
ix1=mod(ia1*ix1+ic1,m1)
```

```
ix2=mod(ia2*ix2+ic2,m2)
```

```
ix3=mod(ia3*ix3+ic3,m3)
```

```
i=1+(97*ix3)/m3
```

```
if (i.gt.97.or.i.lt.1) pause
```

```
rang=r(i)
```

```
r(i)=(float(ix1)+float(ix2)*rm2)*rm1
```

```
return
```

```
end
```

```

subroutine generate_vertical_lines(array,n)
*****
This subroutine generates a rectangle with a given width
,height and grey-tone in an array "array" at a given point.
real array(256,256),a(256,256)
character*12 answer,yes,no
*****

```

Initialize arrays

```

do i=1,n
do j=1,n
a(j,i)=0.
array(j,i)=0.
enddo
enddo
write(6,4)
format(1x,'grey-tone: '$)
read(5,*)grey
do i=1,n,4
do j=1,n
array(j,i)=grey
enddo
enddo
write(6,7)

```

```

format(1x,'another rectangle (answer "yes" or "no"):$)
read(5,8)answer
format(a12)
if(answer.eq.'yes')then
go to 1
else
endif

```

```

return
end

```

```
integer function action_routine(status,timer_data)
*****
function action_routine called by lib$ show_timer run time libcommand.
writes the results to the creen.
parameter stats      :cpu data string
               timer_data :libparameter
*****
```

```
character *(*) status
integer*4 timer_data
```

```
write(6,*)status
write(6,100)
format(/,11x,'press return to continue'$)
read(5,*)
action_routine=1
return
end
```

```

subroutine exp_2d(array,n)
*****
This subroutine generates a Exponential Parabola with
a given Alpha and Beta in an array "array" at a given point.
*****
real array(256,256),a(256,256)
character*12 answer,yes,no

```

Initialize array a

```

do i=1,n
do j=1,n
a(j,i)=0.
array(j,i)=0.
enddo
enddo
write(6,1)
format(1x,'input Alpha of Exp. in pixels:'$)
read(5,*)alpha
write(6,4)
format(1x,'grey-tone:'$)
read(5,*)grey
nn=n/2+1
do i=1,n
do j=1,n
xx=float(i-nn)
yy=float(j-nn)
array(j,i)=exp(-alpha*(xx**2.+yy**2.))
enddo
enddo
write(6,7)
format(1x,'another Exp_2dim. (answer "y" or "n"):'$)
read(5,8)answer
format(a12)
if((answer.eq.'y').or.(answer.eq.'Y')) then
go to 2
else
endif

return
end

```

```

subroutine display_histogram_norm(z,n)
real y(256),z(256)
real h(256),h1(256),h2(256)
real h3(256),h4(256),h5(256)
character*6 answer
character*150 title,title1

```

FUNCTION: Displays a barline histogram of data stored
in array z using UNIRAS 6.

PARAMETERS

Input: z - histogram.

n - input array size (<=256 - an 8-bit image).

icolbar - defines colour of barline.

icolbar = 1 - white

= 2 - red

= 3 - green

= 4 - blue

= other integer values define different colours

(see UNIRAS 6 AGL/CHARTS for details).

Output: None

NOTES:

The program automatically scales each axis so
so that the display lies within the bounds
of the x and y axis - xmin to xmax and
ymin to ymax respectively.

EXTERNAL SUBROUTINES: Called from UNIRAS 6 Library

INTERNAL SUBROUTINES: axis_histogram
plot_histogram
write_histogram

Define as real xmin, xmax, ymin, ymax.

```

real xmin,xmax,ymin,ymax

```

```

write(6,*)'      UNIRAS 6 colour scales'

```

```

write(6,*)

```

```

write(6,*)'icolbar _ defines colour of barline.'

```

```

write(6,*)'icolbar = 1. white '

```

```

write(6,*)'      = 2. red '

```

```

write(6,*)'      = 3. green'

```

```

write(6,*)'      = 4. blue'

```

```

write(6,*)

```

```

write(6,10)

```

```

format(1x,'Scale (1-4)?'$)

```

```

read(5,*)icolbar

```

Initialize arrays

```

do i=1,n

```

```

y(i)=0.

```

```

h(i)=0.

```

```

h1(i)=0.

```

```

h2(i)=0.

```

```

h3(i)=0.

```

```

h4(i)=0.

```

```

h5(i)=0.

```

```

enddo

```

Generate work space.

```

do i=1,n
y(i)=z(i)!Data - z unchanged on output.
enddo

```

Compute largest value of y, ymax.

```

ymax=0.
do i=1,n
if(y(i).gt.ymax)ymax=y(i)
enddo

```

Compute smallest value of y, ymin.

```

ymin=ymax
do i=1,n
if(y(i).lt.ymin)ymin=y(i)
enddo

```

Define the minumum and maximum values of the x-axis.

```

xmin=0.
xmax=1.
ymax=1.
ymin=0

```

Plot axes.

```

call axis_histogram(xmin,xmax,ymin,ymax)

```

Plot histogram

```

call plot_histogram(y,n,ymin,icolbar)
write(6,*) 'enter normalized hist.title'
read(5,40) title1
format(a)
write(6,*) 'enter another Vertical_start'
read(5,*) yva
call rtx (len(title1), title1, xh, yva)
call rtxhei(.5)
call rtx (len(title1), title1,.05*xmax ,1.05*ymax )
call gempty !empty graphics buffer

```

Keep plot until user 'hits' the Return key.

```

write(6,1)
format(1x,'hit Return to continue'$)
read '(a)'

```

Ask user if data file is required for generating a hardcopy.

```

write(6,2)
format(1x,'file histogram <n>? '$)
read(5,3)answer
format(a6)

```

If answer is 'yes' then call write_signal.

```

if(answer.eq.'y')then
call write_histogram(z,n)
else
endif

```

clear plot in UNIRAS window

```

call gclear

return

```


end

subroutine axis_histogram(xmin,xmax,ymin,ymax)

FUNCTION: Plots the axes of the user 2D coordinate system.

PARAMETERS IN:

xmin - lower bound of x-axis.
xmax - upper bound of x-axis.
ymin - lower bound of y-axis.
ymax - upper bound of y-axis.

Program Histogram_axis - based on the UNIRAS example program
AXI001 given in AGL/AXES.

REAL XMIN,YMIN,XMAX,YMAX
REAL XLEN,YLEN,ZLEN
REAL SIZFAC, HEIFAC, HEIGHB
REAL BOX2MM
EXTERNAL BOX2MM

Step 1: Define the 2D limits of the axes

CALL GLIMIT(XMIN,XMAX,YMIN,YMAX,0.0,0.0)

Step 2: Calculate for a suitable square viewport size

SIZFAC=0.7!Determines size of Plot
HEIFAC=0.03!Determines size of axis-text.

CALL GRPSIZ(XSIZMM,YSIZMM)
SIZE = SIZFAC * MIN(XSIZMM,YSIZMM)
XSIZE = SIZE
YSIZE = SIZE

Set up the viewport

XORG = (XSIZMM-XSIZE)/2.
YORG = (YSIZMM-YSIZE)/2.
CALL GVPORT(XORG,YORG,XSIZE,YSIZE)

Step 3: Define the 2D workbox aspect ratio
to be equal to the viewport's.

1st method: Make XLEN and YLEN to be equal
to the viewport sides.

2nd method: Assign the shorter side of the
workbox to be 1. and the other
side proportional to it.

Set ZLEN = 0. for both methods.

The second method is used here.

XLEN = XSIZE/MIN(XSIZE,YSIZE)
YLEN = YSIZE/MIN(XSIZE,YSIZE)
ZLEN = 0.
CALL GWBOX(XLEN,YLEN,ZLEN)

Step 4: Activate 2D scaling - scaling must be active,
otherwise, no axes will be plotted.

CALL GSCALE

Step 5: Calculate a suitable axis-label height
based on the shorter side of the workbox.
Note: The function "BOX2MM" converts box
units to mm.

HEIGHB = HEIFAC*BOX2MM(1.0)

Step 6: Plot two numeric and user-labeled axes
at the limits of the 2D coordinate system

heighb2=1.5*heighb

CALL RAXDIS(6,1,0)
CALL RAXDIS(3,0,0)
CALL RAXLAS(2)
CALL RAXTEX(6,-1,'Grey Level',999.999,999.999,HEIGHB2)
CALL RAXIS(1,YMIN,HEIGHB,1)
CALL RAXTEX(6,-1,'Frequency',999.999,999.999,HEIGHB2)
CALL RAXIS(2,XMIN,HEIGHB,1)

CALL RAXDIS(2,0,0)
CALL RAXDIS(4,0,0)
CALL RAXIS(1,YMAX,HEIGHB,2)
CALL RAXIS(2,XMAX,HEIGHB,2)
CALL RAXDIS(2,1,0)
CALL RAXDIS(4,1,0)

RETURN
END

subroutine plot_histogram(y,n,ymin,icolbar)
real y(256)

FUNCTION: Call's UNIRAS 6 subroutines for plotting
a barline histogram.

ARRAYS IN:
y - histogram.

PARAMETERS IN:
n - array size (<=256).
ymin - lower bound of y-axis.
icolbar - defines colour of barline.

Specify colour of barlines.

call blcol(icolbar,1)!1 colour only.

Define width of bar lines.

call blwid(1.)!set to 1 mm.

Define regular x - data

1 - horizontal x-direction.
n - number of points.
0.0 - Default grace clearance.

call bdimr(1,n,0.0)

Plot bar lines.

ymin - starting y-value (minumum value of input array).
y - end y-values (input array values).
n - number of data points -
the minus sign indicates that bars start at a constant value.

0.0 - bar lines are centered.

```
call bbarli(ymin,y,-n,0.0)
```

```
return
```

```
end
```

```

subroutine display_histogram(z,n)
real y(256),z(256)
real h(256),h1(256),h2(256)
real h3(256),h4(256),h5(256)
character*6 answer
character*150 title ,title1

```

FUNCTION: Displays a barline histogram of data stored
in array z using UNIRAS 6.

PARAMETERS

Input: z - histogram.
n - input array size (<=256 - an 8-bit image).
icolbar - defines colour of barline.
icolbar = 1 - white
 = 2 - red
 = 3 - green
 = 4 - blue
 = other integer values define different colours
 (see UNIRAS 6 AGL/CHARTS for details).

Output: None

NOTES:

The program automatically scales each axis so
so that the display lies within the bounds
of the x and y axis - xmin to xmax and
ymin to ymax respectively.

EXTERNAL SUBROUTINES: Called from UNIRAS 6 Library

INTERNAL SUBROUTINES: axis_histogram
 plot_histogram
 write_histogram

Define as real xmin, xmax, ymin, ymax.

```

real xmin,xmax,ymin,ymax

write(6,*)'      UNIRAS 6 colour scales'
write(6,*)
write(6,*)'icolbar _ defines colour of barline.'
write(6,*)'icolbar = 1. white '
write(6,*)'      = 2. red '
write(6,*)'      = 3. green'
write(6,*)'      = 4. blue'
write(6,*)
write(6,10)
format(1x,'Scale (1-4)?'$)
read(5,*)icolbar

```

Initialize arrays

```

do i=1,n
y(i)=0.
h(i)=0.
h1(i)=0.
h2(i)=0.
h3(i)=0.
h4(i)=0.
h5(i)=0.
enddo

```

Generate work space.

```

do i=1,n
y(i)=z(i)!Data - z unchanged on output.
enddo

```

Compute largest value of y, ymax.

```

ymax=0.
do i=1,n
if(y(i).gt.ymax)ymax=y(i)
enddo

```

Compute smallest value of y, ymin.

```

ymin=ymax
do i=1,n
if(y(i).lt.ymin)ymin=y(i)
enddo

```

Define the minumum and maximum values of the x-axis.

```

xmin=0.
xmax=float(n-1)

```

Plot axes.

```

call axis_histogram(xmin,xmax,ymin,ymax)

```

Plot histogram

```

call plot_histogram(y,n,ymin,icolbar)
call rtxang(90.)
2  write(6,*) 'enter histogram title'
write(6,*) 'enter histogram title'
read(5,40) title1
format(a)
write(6,*) 'enter horizontal& Vertical_start'
read(5,*) xv,yv
call rtx(len(title1), title1, xh, yva)
call rtxhei(.5)
xm=.05*(xmax-xmin)+xmin
ym=1.05*(ymax-ymin)+ymin
call rtx(len(title1),title1,xm,ym)
call rtxang(0.)
call gempty !empty graphics buffer
write(6,*)' enter adjust the coordinate Y/N'
read(5,41)re
1  format(a)
if((re.eq.'Y').or.(re.eq.'y')) goto 42

```

Keep plot until user 'hits' the Return key.

```

write(6,1)
format(1x,'hit Return to continue'$)
read '(a)'

```

Ask user if data file is required for generating a hardcopy.

```

write(6,2)
format(1x,'file histogram <n>?'$)
read(5,3)answer
format(a6)

```

If answer is 'yes' then call write_signal.

```

if(answer.eq.'y')then
call write_histogram(z,n)
else

```

endif

clear plot in UNIRAS window

call gclear

return

end

subroutine axis_histogram(xmin,xmax,ymin,ymax)

FUNCTION: Plots the axes of the user 2D coordinate system.

PARAMETERS IN:

xmin - lower bound of x-axis.

xmax - upper bound of x-axis.

ymin - lower bound of y-axis.

ymax - upper bound of y-axis.

Program Histogram_axis - based on the UNIRAS example program
AXI001 given in AGL/AXES.

REAL XMIN,YMIN,XMAX,YMAX

REAL XLEN,YLEN,ZLEN

REAL SIZFAC, HEIFAC, HEIGHB

REAL BOX2MM

EXTERNAL BOX2MM

Step 1: Define the 2D limits of the axes

CALL GLIMIT(XMIN,XMAX,YMIN,YMAX,0.0,0.0)

Step 2: Calculate for a suitable square viewport size

SIZFAC=0.7!Determines size of Plot

HEIFAC=0.03!Determines size of axis-text.

CALL GRPSIZ(XSIZMM,YSIZMM)

SIZE = SIZFAC * MIN(XSIZMM,YSIZMM)

XSIZE = SIZE

YSIZE = SIZE

Set up the viewport

XORG = (XSIZMM-XSIZE)/2.

YORG = (YSIZMM-YSIZE)/2.

CALL GVPORT(XORG,YORG,XSIZE,YSIZE)

Step 3: Define the 2D workbox aspect ratio
to be equal to the viewport's.

1st method: Make XLEN and YLEN to be equal
to the viewport sides.

2nd method: Assign the shorter side of the
workbox to be 1. and the other
side proportional to it.

Set ZLEN = 0. for both methods.

The second method is used here.

XLEN = XSIZE/MIN(XSIZE,YSIZE)

YLEN = YSIZE/MIN(XSIZE,YSIZE)

ZLEN = 0.

CALL GWBOX(XLEN,YLEN,ZLEN)

Step 4: Activate 2D scaling - scaling must be active,
otherwise, no axes will be plotted.

CALL GSCALE

Step 5: Calculate a suitable axis-label height
based on the shorter side of the workbox.
Note: The function "BOX2MM" converts box
units to mm.

HEIGHB = HEIFAC*BOX2MM(1.0)

Step 6: Plot two numeric and user-labeled axes
at the limits of the 2D coordinate system

heighb2=1.5*heighb

CALL RAXDIS(6,1,0)

CALL RAXDIS(3,0,0)

CALL RAXLAS(2)

CALL RAXTEX(6,-1,'Grey Level',999.999,999.999,HEIGHB2)

CALL RAXIS(1,YMIN,HEIGHB,1)

CALL RAXTEX(6,-1,'Frequency',999.999,999.999,HEIGHB2)

CALL RAXIS(2,XMIN,HEIGHB,1)

CALL RAXDIS(2,0,0)

CALL RAXDIS(4,0,0)

CALL RAXIS(1,YMAX,HEIGHB,2)

CALL RAXIS(2,XMAX,HEIGHB,2)

CALL RAXDIS(2,1,0)

CALL RAXDIS(4,1,0)

RETURN

END

subroutine plot_histogram(y,n,ymin,icolbar)

real y(256),z(256),h(256),h1(256),h2(256)

real h3(256),h4(256),h5(256)

FUNCTION: Call's UNIRAS 6 subroutines for plotting
a barline histogram.

ARRAYS IN:

y - histogram.

PARAMETERS IN:

n - array size (<=256).

ymin - lower bound of y-axis.

icolbar - defines colour of barline.

Specify colour of barlines.

call blcol(icolbar,1)!1 colour only.

Define width of bar lines.

call blwid(1.)!set to 1 mm.

Define regular x - data

l - horizontal x-direction.

n - number of points.

0.0 - Default grace clearance.

```
call bdimr(1,n,0.0)
```

Plot bar lines.

ymin - starting y-value (minumum value of input array).

y - end y-values (input array values).

n - number of data points -

the minus sign indicates that bars start at a constant value.

0.0 - bar lines are centered.

```
call bbarli(ymin,y,-n,0.0)
```

```
return
```

```
end
```



```

subroutine circle(array,n)
*****
This subroutine generates a circle with a given radius
and grey-tone in an array "array" at a given point.
*****
real array(256,256),a(256,256)
character*12 answer,yes,no

```

```

Initialize array a
do i=1,n
do j=1,n
a(j,i)=0.
enddo
enddo
write(6,1)
format(1x,'input radius of circle in pixels:')
read(5,*)rad
write(6,3)
format(1x,'coordinates of centre of circle:')
read(5,*)icenx,iceny
write(6,4)
format(1x,'grey-tone:')
read(5,*)grey
do j=1,n
do i=1,n
ss=sqrt(float((icenx-i)**2+(iceny-j)**2))
if(ss.gt.rad)go to 100
array(i,j)=grey
enddo
enddo
write(6,7)
format(1x,'another circle (answer "y" or "n"):')
read(5,8)answer
format(a12)
if((answer.eq.'y').or.(answer.eq.'Y')) then
go to 2
else
endif

return
end

```

```

subroutine blpf_image(x,n,icut,iord)
real x(256,256),xx(256,256)
integer icut,iord

```

FUNCTION: This program uses the Butterworth lowpass filter to
reduce high frequency noise in an image x.

PARAMETERS

Input: x - digital image
n - size of image = n x n (n<=512)
icut - cutoff frequency in pixels (positive integer)
iord - order of the filter (positive integer)

Output: x - filtered image

EXTERNAL SUBROUTINES: DIPLIB - fft2d

INTERNAL SUBROUTINES: None

Declare internal workspace.

```

real xr(256,256),xi(256,256)
real yr(256,256),yi(256,256)
real b(256,256)

```

Initialize arrays

```

do i=1,n
do j=1,n
xx(j,i)=0.
xr(j,i)=0.
xi(j,i)=0.
yi(j,i)=0.
yr(j,i)=0.
enddo
enddo

```

Compute the Butterworth lowpass filter and
the real and imaginary arrays that will be
used later in the call to fft2d.

```

nn=n/2+1
do i=1,n
do j=1,n
a=sqrt(float(nn-j)**2.+float(nn-i)**2.)
b(j,i)=1./(1.+(a/icut)**(2.*iord))
xr(j,i)=x(j,i)
xi(j,i)=0.
enddo
enddo

```

Take the Fourier transform of the image.

```

call fft2d(xr,xi,n,-1)

```

Multiply the complex spectrum of the image with
the Butterworth lowpass filter (array b).

```

do i=1,n
do j=1,n
yr(j,i)=xr(j,i)*b(j,i)
yi(j,i)=xi(j,i)*b(j,i)
enddo
enddo

```

Take the inverse Fourier transform

and write out the result.

```
call fft2d(yr,yi,n,1)
```

```
do i=1,n
```

```
do j=1,n
```

```
x(j,i)=yr(j,i)
```

```
enddo
```

```
enddo
```

```
return
```

```
end
```

subroutine axis_3d(z,n)

Plot the axes of the 3D Cartesian coordinate system

real a(256,256),f(256,256),f1(256,256),af(256,256)

real pf(256,256)

REAL XORG,YORG,XSIZE,YSIZE,XSIZMM,YSIZMM,SIZEMM

REAL XMIN,YMIN,XMAX,YMAX,ZMIN,ZMAX

REAL SIZFAC,HEIFAC,HEIGHB

REAL XLEN,YLEN,ZLEN,SIZE

REAL BOX2MM,HUNIT,z(256,256)

real x(256),y(256),xarr(256),yarr(256),zarr(256)

INTEGER NTEX

PARAMETER (NTEX = 6, NPLANE=3,IPROJ=2)

CHARACTER*15 TEXARR(NTEX)

CHARACTER*150 title,title1

INTEGER LENARR(NTEX),IPLANE(NPLANE)

EXTERNAL BOX2MM

Select "best plane" for the orientation of the axis scales

DATA IPLANE/1,1,1/

DATA XMIN,XMAX,YMIN,YMAX,ZMIN,ZMAX

+ / 0.,100., 0., 10., 0., 50./

DATA USRLAB /'1st','2nd','3rd','4th','...','Nth'/

DATA LENARR /NLAB*-2/

DATA SIZFAC,HEIFAC/1.0,0.035/

Send message to user 'Display 3-dimension object...'

write(6,1)

format(1x,'Displaying 3-Dimension Object...'\$)

DATA TEXARR/

+' ',

+'X-direction',

+' ',

+'Y-direction',

+' ',

+'Z-direction'/

Remove all trailing blanks in the axis-texts

DATA LENARR /0,-2,0,-2,0,-2/

DATA IUNDEF/9999/

Set the best fit font.

CALL RDEFON('C*4',0)

do i=1,n

x(i)=float(i-1)

y(i)=float(i-1)

enddo

xmin=0.

xmax=float(n-1)

ymin=0.

ymax=float(n-1)

Compute largest value of z, zmax.

zmax=0.

do i=1,n

do j=1,n

if(z(j,i).gt.zmax) zmax=z(j,i)

enddo

enddo

Compute the smallest value of z , $zmin$.

```
zmin=zmax
do i=1,n
do j=1,n
if(z(j,i).lt.zmin) zmin=z(j,i)
enddo
enddo
```

Set world coordinate system up vector

```
IPROJ = 1 the default "Y-up, X-right"
IPROJ = 2 "Z-up, Y-right"
CALL GVPROJ(IPROJ)
```

Step 1: Define the 3D limits of the axes

```
CALL GLIMIT(XMIN,XMAX,YMIN,YMAX,ZMIN,ZMAX)
```

Step 2: Calculate a suitable square viewport size.

Note: The viewport aspect ratio does not have to be unity.

```
CALL GRPSIZ(XSIZMM,YSIZMM)
SIZEMM = SIZEFAC * MIN(XSIZMM,YSIZMM)
XSIZE = SIZEMM
YSIZE = SIZEMM
```

Set up the viewport

```
XORG = (XSIZMM-XSIZE)/2.
YORG = (YSIZMM-YSIZE)/2.
CALL GVPORT(XORG,YORG,XSIZE,YSIZE)
```

Step 3: Define the 3D workbox aspect ratio

such that $XLEN/YLEN$ is the same as the viewport and $ZLEN = MAX(XLEN,YLEN)$

Note: The shortest side of the box is unity

```
SIZE= MIN(XSIZE,YSIZE)
XLEN = XSIZE/SIZE
YLEN = YSIZE/SIZE
ZLEN = MAX(XLEN,YLEN)
CALL GWBOX(XLEN,YLEN,ZLEN)
```

Set axel color to red

```
call raxaxl (iundef,2,iundef,iundef)
```

Set axel lable, unit and text to blue

```
call raxtea(4,4,iundef,iundef,iundef,iundef,iundef)
call raxtea(5,4,iundef,iundef,iundef,iundef,iundef)
call raxtea(6,4,iundef,iundef,iundef,iundef,iundef)
```

Step 4: Activate the XY-plane at $Z = ZMIN$

```
CALL RWALL(1,2,3,ZMIN)
```

Step 5: Calculate a suitable box unit

for the axis-label height based on the shortest side of the workbox which has previously been defined to be unity.

Note: Convert the box unit to

MM for the axis-label height

after the wall has been activated.

```
HUNIT = HEIFAC
```

```
HEIGHB = BOX2MM(HUNIT)
```

Plot one numeric X-axis and one user-labeled Y-axis
at YMIN and XMIN.

```
CALL RAXIS(1,YMIN,HEIGHB,1)
```

```
CALL RAXIS(2,XMIN,HEIGHB,1)
```

```
CALL RUXIS(2,XMIN,HEIGHB,1,LENARR,USRLAB,NLAB)
```

Step 6: Activate the YZ-plane at X = XMIN

```
CALL RWALL(-3,2,1,XMIN)
```

Plot the world X-axis at Y = YMIN to represent
the Cartesian Z-axis.

```
HEIGHB = BOX2MM(HUNIT)
```

```
CALL RAXIS(1,YMIN,HEIGHB,1)
```

```
do i=1,n
```

```
do j=1,n
```

```
xarr(j)=x(i)
```

```
yarr(j)=y(j)
```

```
zarr(j)=z(j,i)
```

```
enddo
```

Transform and plot the 3D points

```
call gtrans(xarr,yarr,zarr,n)
```

```
call gscamm
```

Set 3d surface to Green

```
call gwicol(iundef,3)
```

```
call gvect(xarr,yarr,n)
```

```
enddo
```

```
do j=1,n
```

```
do i=1,n
```

```
xarr(i)=x(i)
```

```
yarr(i)=y(j)
```

```
zarr(i)=z(j,i)
```

```
enddo
```

Transform and plot the 3D points

```
call gtrans(xarr,yarr,zarr,n)
```

```
call gscamm
```

Set 3d surface to Black

```
call gwicol(iundef,1)
```

```
call gvect(xarr,yarr,n)
```

```
enddo
```

Display title

```
write(6,*)'enter title '
```

```
read(5,10) title
```

```
format(a)
```

```
call rtx(len(title), title, 15.0, 175.0)
```

```
CALL RAXDIS(1,1,IUNDEF)
```

```
CALL RAXDIS(4,1,1)
```

```
CALL RAXDIS(6,1,IUNDEF)
```

```
CALL RAXDIS(7,0,IUNDEF)
```

```
CALL RAXIS3(0,HEIGHB,IPLANE,LENARR,TEXARR)
```

```
return
```

```
END
```

```

      subroutine axis_3d_phase(z,n)
      *****
      Plot the axes of the 3D Cartesian coordinate system
      *****
      real a(256,256),pf(256,256)

```

```

      REAL XORG,YORG,XSIZE,YSIZE,XSIZMM,YSIZMM,SIZEMM
      REAL XMIN,YMIN,XMAX,YMAX,ZMIN,ZMAX
      REAL SIZFAC, HEIFAC, HEIGHB
      REAL XLEN,YLEN,ZLEN,SIZE
      REAL BOX2MM,HUNIT,z(256,256)
      real x(256),y(256),xarr(256),yarr(256),zarr(256)
      INTEGER NTEX
      PARAMETER (NTEX = 6, NPLANE=3,IPROJ=2)
      CHARACTER*15 TEXARR(NTEX)
      CHARACTER*150 title ,title1
      INTEGER LENARR(NTEX),IPLANE(NPLANE)
      EXTERNAL BOX2MM

```

Select "best plane" for the orientation of the axis scales

```

      DATA IPLANE/1,1,1/

```

```

      DATA XMIN,XMAX,YMIN,YMAX,ZMIN,ZMAX
      + / 0.,100., 0., 10., 0., 50./

```

```

      DATA USRLAB /'1st','2nd','3rd','4th','...','Nth'/
      DATA LENARR /NLAB*-2/

```

```

      DATA SIZFAC,HEIFAC/1.0,0.035/

```

Send message to user 'Display 3-dimension object...'

```

      write(6,1)
      format(1x,'Displaying 3-Dimension Object...'$)

```

```

      DATA TEXARR/
      +' ',
      +'X-direction',
      +' ',
      +'Y-direction',
      +' ',
      +'Z-direction'/

```

Remove all trailing blanks in the axis-texts

```

      DATA LENARR /0,-2,0,-2,0,-2/
      DATA IUNDEF/9999/

```

Set the best fit font.
CALL RDEFON('C*4',0)

```

      do i=1,n
      x(i)=float(i-1)
      y(i)=float(i-1)
      enddo

```

```

      xmin=0.
      xmax=float(n-1)
      ymin=0.
      ymax=float(n-1)

```

Compute largest value of z, zmax.

```

      zmin=0.0
      zmax=90.0
      do i=1,n
      do j=1,n

```

```

        if(z(j,i).gt.zmax) zmax=z(j,i)
        enddo
    enddo

```

Comput the smallest value of z, zmin.

```

    zmin=zmax
    do i=1,n
    do j=1,n
        if(z(j,i).lt.zmin) zmin=z(j,i)
    enddo
    enddo

```

Set world coordinate system up vector

```

    IPROJ = 1 the default "Y-up, X-right"
    IPROJ = 2             "Z-up, Y-right"
    CALL GVPROJ(IPROJ)

```

Step 1: Define the 3D limits of the axes

```

    CALL GLIMIT(XMIN,XMAX,YMIN,YMAX,ZMIN,ZMAX)

```

Step 2: Calculate a suitable square viewport size.

Note: The viewport aspect ratio does not have to be unity.

```

    CALL GRPSIZ(XSIZMM,YSIZMM)
    SIZEMM = SIZEFAC * MIN(XSIZMM,YSIZMM)
    XSIZE = SIZEMM
    YSIZE = SIZEMM

```

Set up the viewport

```

    XORG = (XSIZMM-XSIZE)/2.
    YORG = (YSIZMM-YSIZE)/2.
    CALL GVPORT(XORG,YORG,XSIZE,YSIZE)

```

Step 3: Define the 3D workbox aspect ratio

such that XLEN/YLEN is the same as the viewport and ZLEN = MAX(XLEN,YLEN)

Note: The shortest side of the box is unity

```

    SIZE= MIN(XSIZE,YSIZE)
    XLEN = XSIZE/SIZE
    YLEN = YSIZE/SIZE
    ZLEN = MAX(XLEN,YLEN)
    CALL GWBOX(XLEN,YLEN,ZLEN)

```

Set axel color to red

```

    call raxaxl (iundef,2,iundef,iundef)

```

Set axel lable, unit and text to blue

```

    call raxtea(4,4,iundef,iundef,iundef,iundef)
    call raxtea(5,4,iundef,iundef,iundef,iundef)
    call raxtea(6,4,iundef,iundef,iundef,iundef)

```

Step 4: Activate the XY-plane at Z = ZMIN

```

    CALL RWALL(1,2,3,ZMIN)

```

Step 5: Calculate a suitable box unit

for the axis-label height based on the shortest side of the workbox which has previously been defined to be unity.

Note: Convert the box unit to

MM for the axis-label height

after the wall has been activated.


```
HUNIT = HEIFAC
HEIGHB = BOX2MM(HUNIT)
```

Plot one numeric X-axis and one user-labeled Y-axis
at YMIN and XMIN.

```
CALL RAXIS(1,YMIN,HEIGHB,1)
CALL RAXIS(2,XMIN,HEIGHB,1)
CALL RUXIS(2,XMIN,HEIGHB,1,LENARR,USRLAB,NLAB)
```

Step 6: Activate the YZ-plane at X = XMIN

```
CALL RWALL(-3,2,1,XMIN)
```

Plot the world X-axis at Y = YMIN to represent
the Cartesian Z-axis.

```
HEIGHB = BOX2MM(HUNIT)
CALL RAXIS(1,YMIN,HEIGHB,1)
```

```
do i=1,n
do j=1,n
xarr(j)=x(i)
yarr(j)=y(j)
zarr(j)=z(j,i)
enddo
```

Transform and plot the 3D points
call gtrans(xarr,yarr,zarr,n)
call gscamm

Set 3d surface to Green
call gwicol(iundef,3)
call gvect(xarr,yarr,n)
enddo

```
do j=1,n
do i=1,n
xarr(i)=x(i)
yarr(i)=y(j)
zarr(i)=z(j,i)
enddo
```

Transform and plot the 3D points
call gtrans(xarr,yarr,zarr,n)
call gscamm

Set 3d surface to Black
call gwicol(iundef,1)
call gvect(xarr,yarr,n)
enddo

Display title

```
write(6,*)'enter title '
read (5,10) title
format (a)
call rtx (len(title), title, 15.0, 175.0)
CALL RAXDIS(1,1,IUNDEF)
CALL RAXDIS(4,1,1)
CALL RAXDIS(6,1,IUNDEF)
CALL RAXDIS(7,0,IUNDEF)
CALL RAXIS3(0,HEIGHB,IPLANE,LENARR,TEXARR)
return
END
```

```
subroutine write_histogram(h,n)
real h(256)
```

FUNCTION: Writes (free format) array h to
a named data file.

PARAMETERS

Input: h - histogram.
n - size of array h.

Output: None

INTERNAL SUBROUTINES: open_write_histogram

EXTERNAL SUBROUTINES: None

Call open_write_histogram to assign unit number for file
and to prompt user to enter size and name of file
to be created.

```
call open_write_histogram
```

Send message to user 'writing data...'

```
write(6,1)
format(1x,'Writing data...')$)
```

Write out array s to data file.

```
do i=1,n
write(1,*)h(i)
enddo
```

Close file.

```
close(1)
```

Skip a line to indicate that process has finished.

```
write(6,*)
```

```
return
end
```

```
subroutine open_write_histogram
character*40 filew
character*6 answer
```

FUNCTION: Asks user to 'enter name of data file...'
and opens file to write data.

Prompt user to input name of data file.

```
00 write(6,101)
1 format(1x,'Enter name of data file (output):')$)
read(5,102)filew
2 format(a40)
print *,'File name is:',filew
write(6,103)
3 format(1x,'Is this name OK? <y>')$)
read(5,104)answer
```

```
04      format(a6)  
      if(answer.eq.'n')goto 1000
```

```
Open file - unit file number=1.
```

```
      open(unit=1,file=filew,status='new')
```

```
      return
```

```
      end
```

```
subroutine var_rectangle(array,n)
```

```
*****
```

```
This subroutine generates a rectangle with a given width  
,height and grey-tone in an array "array" at a given point.  
real array(256,256),a(256,256)
```

```
character*12 answer,yes,no
```

```
*****
```

```
Initialize arrays
```

```
do i=1,n
```

```
do j=1,n
```

```
a(j,i)=0.
```

```
array(j,i)=0.
```

```
enddo
```

```
enddo
```

```
write(6,1)
```

```
format(1x,'input width of square in pixels:','$)
```

```
read(5,*)wid
```

```
write(6,3)
```

```
format(1x,'coordinates of the Center pixel of rectangle:','$)
```

```
read(5,*)icenx,iceny
```

```
write(6,4)
```

```
format(1x,'grey-tone:','$)
```

```
read(5,*)grey
```

```
l=ix+iwid-1
```

```
m=jsy+jhei-1
```

```
do j=1,n
```

```
do i=1,n
```

```
ss=sqrt(abs(float((icenx-i)*(iceny-j))))
```

```
if(ss.gt.wid)go to 100
```

```
array(i,j)=grey
```

```
enddo
```

```
enddo
```

```
do j=1,n
```

```
do i=1,n
```

```
s1=float(icenx-i)
```

```
s2=float(iceny-j)
```

```
if((i.ge.ix).and.(i.le.l)) then
```

```
if((j.ge.jsy).and.(j.le.m)) then
```

```
array(i,j)=grey
```

```
endif
```

```
endif
```

```
enddo
```

```
enddo
```

```
write(6,7)
```

```
format(1x,'another rectangle (answer "yes" or "no"):$)
```

```
read(5,8)answer
```

```
format(a12)
```

```
if(answer.eq.'yes')then
```

```
go to 2
```

```
else
```

```
endif
```

```
return
```

```
end
```

```
subroutine timer_start(context)
```

```
routine to initialize the internal clock to allow recording processing  
time
```

```
integer *4 status,context,lib$ init_timer
```

```
status=lib$ init_timer(context)
```

```
if( .not.status)call lib$ signal(%val(status))
```

```
return
```

```
end
```

```
subroutine time_elapsed(context)
```

```
function.... utilises system run time lib. routines to  
record program cpu times used.
```

```
external action_routine
```

```
integer status,context,lib$ show_timer,libfree_timer,data
```

```
status=lib$ show_timer(context,2,action_routine,data)
```

```
if( .not. status)call lib$ signal (%val(status))
```

```
status=lib$ free_timer(context)
```

```
if ( .not. status)call lib$ signal (%val(status))
```

```
return
```

```
end
```

subroutine spike_rectangle(array,n)

This subroutine generates a rectangle with a given width
,height and grey-tone in an array "array" at a given point.

real array(256,256),a(256,256)

character*12 answer,yes,no

Initialize arrays

do i=1,n

do j=1,n

a(j,i)=0.

array(j,i)=0.

enddo

enddo

write(6,4)

format(1x,'grey-tone: '\$)

read(5,*)grey

do j=1,n

do i=1,n

if (i mod 20) then

array(i,j)=3*grey

else

array(i,j)=grey

endif

enddo

enddo

write(6,7)

format(1x,'another rectangle (answer "yes" or "no"):\$)

read(5,8)answer

format(a12)

if(answer.eq.'yes')then

go to 2

else

endif

return

end

```

subroutine sinc_interpolate(x,n,y,m)
real x(256,256),y(256,256),xr(256,256),xi(256,256)
real yr(256,256),yi(256,256)

```

FUNCTION: Interpolate an image by zero padding its complex spectrum

AUTHOR: J M Blackledge

PARAMETERS

Input: x - digital image
n - array size
m - array size of output

Note: both n and m must be of size 2^k where k
is a positive integer.

Output: y - output

EXTERNAL SUBROUTINES: fft2d

INTERNAL SUBROUTINES: None

Declare internal workspace.

Initialize array

```

do i=1,n
do j=1,n
xr(j,i)=x(j,i)
xi(j,i)=0.
enddo
enddo

```

```

do i=1,n
do j=1,n
yr(j,i)=0.
yi(j,i)=0.
enddo
enddo

```

Take 2D FFT

```

call fft2d(xr,xi,n,-1)

```

Define positions of DCs.

```

nn=n/2+1
mm=m/2+1

```

Zero pad xr and xi

```

kk=1
do i=mm-nn+1,mm-nn+n
k=1
do j=mm-nn+1,mm-nn+n
yr(j,i)=xr(k,kk)
yi(j,i)=xi(k,kk)
k=k+1
enddo
kk=kk+1
enddo

```

Inverse FFT

```

call fft2d(yr,yi,m,1)

```

Write output


```
do i=1,m
do j=1,m
y(j,i)=yr(j,i)
enddo
enddo
return
end
```

```
subroutine rectangle(array,n)
```

```
*****
```

```
This subroutine generates a rectangle with a given width  
,height and grey-tone in an array "array" at a given point.
```

```
real array(256,256),a(256,256)
```

```
character*12 answer,yes,no
```

```
*****
```

```
Initialize arrays
```

```
do i=1,n
```

```
do j=1,n
```

```
a(j,i)=0.
```

```
array(j,i)=0.
```

```
enddo
```

```
enddo
```

```
write(6,1)
```

```
format(1x,'input width & height of rectangle in pixels:','$)
```

```
read(5,*)iwid,jhei
```

```
write(6,3)
```

```
format(1x,'coordinates of the starting pixel of rectangle:','$)
```

```
read(5,*)isx,jsy
```

```
write(6,4)
```

```
format(1x,'grey-tone:','$)
```

```
read(5,*)grey
```

```
l=isx+iwid-1
```

```
m=jsy+jhei-1
```

```
do j=1,n
```

```
do i=1,n
```

```
s1=float(icenx-i)
```

```
s2=float(iceny-j)
```

```
if((i.ge.isx).and.(i.le.l)) then
```

```
if((j.ge.jsy).and.(j.le.m)) then
```

```
array(i,j)=grey
```

```
endif
```

```
endif
```

```
enddo
```

```
enddo
```

```
write(6,7)
```

```
format(1x,'another rectangle (answer "yes" or "no"):$)
```

```
read(5,8)answer
```

```
format(a12)
```

```
if(answer.eq.'yes')then
```

```
go to 2
```

```
else
```

```
endif
```

```
return
```

```
end
```

```
      subroutine projection_square_error(x,y,n,er)
      real y(256,256),x(256,256),er(256)
```

```
*****
```

```
FUNCTION: Computes the least square error for each angle of
          rotation (projection).
```

```
PARAMETERS:
```

```
INPUT      : y _ input image of n X n size
            : x _ input image of n X n size
```

```
OUTPUT      : er _ Rotation square error array
```

```
EXTERNAL SUBROUTINE: Norm_image
```

```
INTERNAL SUBROUTINE: None
```

```
*****
```

```
  Initialize error array.
```

```
    do i=1,n
      er(i)=0.
    enddo
```

```
Compute rotation square error
```

```
  ern=0.
  do i=1,n
    do j=1,n
      ern=ern+abs((x(j,i))**2.)
    enddo
  enddo
  do i=1,n
    do j=1,n
      er(i)=er(i)+abs((x(j,i)-y(j,i))**2./ern)
    enddo
  enddo
  return
end
```

```

subroutine phase_2d_inv_filter(array,n)
*****
This subroutine computes the Foward, Backward and Central
difference approximation of the first order differential
equation.
*****
real array(256,256),a(256,256),pf(256,256)
character*12 answer,yes,no
Initialize array a
do i=1,n
do j=1,n
a(j,i)=0.
array(j,i)=0.
enddo
enddo
write(6,7)
format(1x,'another difference (answer "y" or "n"):'$)
read(5,8)answer
format(a12)
if((answer.eq.'y').or.(answer.eq.'Y')) then
go to 2
else
endif

return
end

```

subroutine option(iresponse)

FUNCTION: Provides option for repeating a process (i.e. typically to
'display another' image using different input parameters)

AUTHOR: J M Blackledge

PARAMETERS

Input: None

Output: irepponse

= 1 for 'yes to repeat process'

= 0 for 'no to continue'

EXTERNAL SUBROUTINES: None

INTERNAL SUBROUTINES: None

character*6 answer

Write out appropriate response to user.

write(6,1)

format(1x,'Display another <n> ?'\$)

read(5,2)answer

format(a6)

Provide logical output: default no (return) - iresponse = 0
option yes (input 'y') - iresponse = 1

if(answer.eq.'y')iresponse=1

if(answer.ne.'y')iresponse=0

return

end

subroutine norm_signal(x,n)

FUNCTION: returns an image with absolute maximum value of 1.

PARAMETERS

Input: x -signal
n - size of signal

Output: x - normalized signal

INTERNAL SUBROUTINES: None

EXTERNAL SUBROUTINES: None

real x(256),s(256),y(256)

real max

Generate working space

do i=1,n

s(i)=x(i)

enddo

Set the negative value to zero

do i=1,n

if(s(i).lt.0.) s(i)=0.

enddo

Find the absolute maximum value of the signal.

max=0.

do i=1,n

x1=s(i)

if(x1.gt.max) max=x1

enddo

write(6,*)'max=',x1

Normalize the signal.

do i=1,n

x(i)=s(i)/max

enddo

return

end

subroutine norm_image(x,n)

FUNCTION: returns an image with absolute maximum value of 1.

PARAMETERS

Input: x -image
n - size of image = n x n

Output: x - normalized image

INTERNAL SUBROUTINES: None

EXTERNAL SUBROUTINES: None

real x(256,256),s(256,256)

real max

Generate working space

do i=1,n

do j=1,n

s(j,i)=x(j,i)

enddo

enddo

Set the negative value to zero

do i=1,n

do j=1,n

if(s(j,i).lt.0.) s(j,i)=0.

enddo

enddo

Find the absolute maximum value of the image.

max=0.

do i=1,n

do j=1,n

x1=s(j,i)

if(x1.gt.max) max=x1

enddo

enddo

write(6,*)'max=',x1

Normalize the image.

do i=1,n

do j=1,n

x(j,i)=s(j,i)/max

enddo

enddo

return

end

subroutine norm_histogram(hy,levy,n,hx,dev,area)

FUNCTION: returns an image with absolute maximum value of 1.

PARAMETERS

Input: hy - Histogram_image
levy - histogram_level
n - size of the image

Output: h - normalized histogram
dev - deviation

INTERNAL SUBROUTINES: None

EXTERNAL SUBROUTINES: None

real h(256),h1(256),h2(256),h3(256)
real h4(256),h5(256)
real max,hy(256),hx(256)

Find the absolute maximum value of the histogram.

max=0.
do j=1,levy
e=float(j)/float(levy)
xx=abs(hy(j))
if(xx.gt.max) then
max=xx
ls=j
else
endif
enddo

Compute the deviation

dev=float(levy-ls)/float(levy)

Normalize the histogram.

do j=1,levy
e=float(j)/float(levy)
hx(j)=hy(j)/float(n*2)
enddo

Compute the histogram area

area=0.
do j=1,levy-1
area=area+hx(j)
enddo
return
end


```

subroutine least_square_error(x,y,n,er,abser,std)
real y(256,256),x(256,256),r(256,256),abser(256,256)
real abser1(256,256)

```

```

*****

```

```

FUNCTION: Computes the least square error and the normalized
least square error.

```

```

PARAMETERS:

```

```

INPUT      : y _ input image of n X n size
            : x _ input image of n X n size
            : lev_ Grey level

```

```

OUTPUT     : er _ Least square error
            : ernor_ Normalized leaset square error

```

```

EXTERNAL SUBROUTINE: Norm_image

```

```

INTERNAL SUBROUTINE: None

```

```

*****

```

```

Initialize array

```

```

do i=1,n
do j=1,n
abser(j,i)=0.
enddo
enddo

```

```

Compute least square error

```

```

erl=0.
ern=0.
do i=1,n
do j=1,n
ern=ern+abs((x(j,i))**2.)
enddo
enddo
do i=1,n
do j=1,n
erl=erl+abs((x(j,i)-y(j,i))**2./ern)
enddo
enddo
er=erl/float(n**2)

```

```

Absolute square_error array

```

```

do i=1,n
do j=1,n
abser(j,i)=abs((x(j,i)-y(j,i))**2./ern)
enddo
enddo

```

```

Compute standard deviation for image

```

```

stm=0.
do i=1,n
do j=1,n
stm=stm+abs(abser(j,i))
enddo
enddo
ymean=stm/float(n**2)
std2=0.
do i=1,n
do j=1,n
std2=std2+abs((abser(j,i)-ymean)**2)
enddo
enddo
std=sqrt(std2*(1./(float(n**2))))
return
end

```

SUBROUTINE MEAN_standard (y,n,ymean,std)

real Y(256)

INITIALIZE THE SUM VALUE

SUM1=0.0

SUM UP THE VALUES AND DETERMINE THE MEAN

DO I=1,N

SUM1=SUM1+Y(I)

enddo

YMEAN=SUM1/N

SUM2=0.0

DO I=1,N

SUM2=SUM2+(Y(I)-YMEAN)**2

enddo

VAR=SUM2/N

STD=SQRT(VAR)

RETURN

END

subroutine multi_circle(array,n)

This subroutine generates a circle with a given radius
and grey-tone in an array "array" at a given point.

real array(256,256),a(256,256)

character*12 answer,yes,no

1 Initialize array a

do i=1,n

do j=1,n

a(j,i)=0.

enddo

enddo

2 write(6,1)

1 format(1x,'input radius of circle in pixels:')

read(5,*)rad

write(6,3)

3 format(1x,'coordinates of centre of circle:')

read(5,*)icenx,iceny

write(6,4)

4 format(1x,'grey-tone:')

read(5,*)grey

do j=1,n

do i=1,n

ss=sqrt(float((64-i)**2+(64-j)**2))

if(ss.gt.64)go to 100

array(i,j)=10

00 enddo

enddo

do j=1,n

do i=1,n

ss=sqrt(float((64-i)**2+(32-j)**2))

if(ss.gt.20)go to 101

array(i,j)=2

01 enddo

enddo

do j=1,n

do i=1,n

ss=sqrt(float((32-i)**2+(96-j)**2))

if(ss.gt.20)go to 102

array(i,j)=1

02 enddo

enddo

do j=1,n

do i=1,n

ss=sqrt(float((96-i)**2+(96-j)**2))

if(ss.gt.20)go to 103

array(i,j)=14

03 enddo

enddo

write(6,7)

7 format(1x,'another circle (answer "y" or "n"):')

read(5,8)answer

8 format(a12)

if((answer.eq.'y').or.(answer.eq.'Y')) then

go to 2

else

endif

return

end

```
subroutine multi_resolution_image(x,n,y,nz)
real x(256,256),y(256,256)
character*6 answer
```

FUNCTION: Reads (using free format) an image from a given data file.

PARAMETERS

Input: x - image
n - size of image (=n x n)

Output: y - image
nz - multi_size of image

EXTERNAL SUBROUTINES: None

```
write(6,*)'enter resolution 1,2,4,8,16,32 ?'
read(5,*)in
```

```
nz=int(n/in)
```

```
do j=1,nz
do i=1,nz
y(j,i)=x(j*in,i*in)
enddo
enddo
```

Skip a line to indicate that process has finished.

```
write(6,*)
```

```
return
end
```

subroutine line(array,n)

This subroutine generates a line with a given start
and end point in an array "array" .

real array(256,256),a(256,256)

character*12 answer,yes,no

Initialize array a

do i=1,n

do j=1,n

a(j,i)=0.

array(j,i)=0.

enddo

enddo

write(6,1)

format(1x,'input line start in pixels:')\$)

read(5,*)x1,y1

write(6,3)

format(1x,'input line end in pixels:')\$)

read(5,*)x2,y2

write(6,4)

format(1x,'grey-tone:')\$)

read(5,*)grey

do i=1,n

do j=1,n

yy=(float(j)-y1)*(x2-x1)

xx=(float(i)-x1)*(y2-y1)

if(yy.eq.xx) then

array(i,j)=grey

endif

enddo

enddo

write(6,7)

format(1x,'another line (answer "y" or "n"):\$)

read(5,8) answer

format(a12)

if((answer.eq.'y').or.(answer.eq.'Y')) then

go to 2

else

endif

return

end

subroutine fft2d(a,b,n,isgn)

FUNCTION: This subroutine computes the DFT of a 2D array with
real and imaginary parts a and b respectively.

Note: This routine computes the 'optical' transform - the
zero frequency occurs at $(n/2+1, n/2+1)$.

PARAMETERS

Input: a - real part of image
b - imaginary part of image
n - image size (=n x n where $n=2^k$; k-integer)
Note: n must be less than or equal to 512.
isgn = -1 gives forward Fourier transform
 = 1 gives inverse Fourier transform

Output:

EXTERNAL SUBROUTINES: None

INTERNAL SUBROUTINES: fft1d

Use subroutine fft1d to design a 2D FFT by computing
the 1D FFT of the columns and rows separately.

real a(256,256),b(256,256),c(256,256),d(256,256)
real aa(256),bb(256),cc(256),dd(256)

Put columns into single-dimension array and call fft1d.
initialize arrays

```
do i=1,n
  aa(i)=0.
  bb(i)=0.
  cc(i)=0.
  dd(i)=0.
enddo
do i=1,n
  do j=1,n
    c(j,i)=0.
    d(j,i)=0.
  enddo
enddo
do i=1,n
  do j=1,n
    aa(j)=a(j,i)
    bb(j)=b(j,i)
  enddo
```

call fft1d(aa,bb,n,isgn)

```
do j=1,n
  c(j,i)=aa(j)
  d(j,i)=bb(j)
enddo
enddo
```

Put rows into single-dimension array and call fft1d.

```
do j=1,n
  do i=1,n
    cc(i)=c(j,i)
    dd(i)=d(j,i)
```

enddo

call fftld(cc,dd,n,isgn)

do i=1,n
a(j,i)=cc(i)
b(j,i)=dd(i)
enddo
enddo

return
end

subroutine fftld(a,b,n,isgn)

This subroutine computes the DFT of a complex array whose real and imaginary parts are a and b respectively using the successive doubling method ('Digital Image Processing' by Gonzalez and Wintz, Addison-Wesley publishing company, 1987).

The subroutine returns the real (a) and imaginary (b) parts of the DFT.

The size of the arrays n must be an integer power of 2 less than or equal to 512. If arrays larger than 512 are required, then the dimensions of a,b and c must be adjusted accordingly.

Note: Zero frequency occurs at $n/2+1$.

By convention, the forward Fourier transform is obtained when isgn=-1 and the inverse Fourier transform is obtained when isgn=1.

real a(256),b(256),aa(256),bb(256),cc(256),dd(256)
complex c(512),u,v,w,cplx
if(isgn.lt.0)den=float(n)
if(isgn.gt.0)den=1.
pi=4.*atan(1.)
p=float(n)
q=log(p)/log(2.)
k=int(q)
nh=n/2
nm=n-1

initialize complex array c

do j=1,n
c(j)=(0.,0.)
enddo
j=1
do 1 i=nh+1,n
c(j)=cplx(a(i),b(i))
j=j+1
do 2 i=1,nh
c(j)=cplx(a(i),b(i))
j=j+1
j=1
do 3 i=1,nm
if(i.ge.j)goto 10
v=c(j)
c(j)=c(i)
c(i)=v
jj=nh
if(jj.ge.j)goto 3
j=j-jj
jj=jj/2
goto 4

```

j=j+jj
do 6 l=1,k
  l1=2**l
  l2=l1/2
  u=(1.,0.)
  x=pi*float(isgn)/l2
  w=cplx(cos(x),sin(x))
  do 6 j=1,l2
    do 5 i=j,n,l1
      ii=i+l2
      v=c(ii)*u
      c(ii)=c(i)-v
      c(i)=c(i)+v
      u=u*w
    j=1
  do 7 i=1,n
    c(i)=c(i)/den
    c(i)=c(i)/float(n)
  do 8 i=nh+1,n
    a(j)=real(c(i))
    b(j)=aimag(c(i))
    j=j+1
  do 9 i=1,nh
    a(j)=real(c(i))
    b(j)=aimag(c(i))
    j=j+1
  return
end

```


subroutine fftld(a,b,n,isgn)

This subroutine computes the DFT of a complex array whose real and imaginary parts are a and b respectively using the successive doubling method ('Digital Image Processing' by Gonzalez and Wintz, Addison-Wesley publishing company, 1987).

The subroutine returns the real (a) and imaginary (b) parts of the DFT.

The size of the arrays n must be an integer power of 2 less than or equal to 512. If arrays larger than 512 are required, then the dimensions of a,b and c must be adjusted accordingly.

Note: Zero frequency occurs at $n/2+1$.

By convention, the forward Fourier transform is obtained when isgn=-1 and the inverse Fourier transform is obtained when isgn=1.

```
real a(256),b(256),za(256),zb(256),zc(256),zd(256)
complex c(512),u,v,w,cmplx
if(isgn.lt.0)den=float(n)
if(isgn.gt.0)den=1.
pi=4.*atan(1.)
p=float(n)
q=log(p)/log(2.)
k=int(q)
nh=n/2
nm=n-1
```

initialize complex array c

```
do j=1,n
  c(j)=(0.,0.)
enddo
j=1
do 1 i=nh+1,n
  c(j)=cmplx(a(i),b(i))
  j=j+1
do 2 i=1,nh
  c(j)=cmplx(a(i),b(i))
  j=j+1
j=1
do 3 i=1,nm
  if(i.ge.j)goto 10
  v=c(j)
  c(j)=c(i)
  c(i)=v
  jj=nh
  if(jj.ge.j)goto 3
  j=j-jj
  jj=jj/2
  goto 4
  j=j+jj
do 6 l=1,k
  l1=2**l
  l2=l1/2
  u=(1.,0.)
  x=pi*float(isgn)/l2
  w=cmplx(cos(x),sin(x))
do 6 j=1,l2
do 5 i=j,n,l1
  ii=i+l2
  v=c(ii)*u
  c(ii)=c(i)-v
```

```
5      c(i)=c(i)+v
6      u=u*w
      j=1
7      do 7 i=1,n
c7      c(i)=c(i)/den
      c(i)=c(i)/float(n)
      do 8 i=nh+1,n
      a(j)=real(c(i))
      b(j)=aimag(c(i))
8      j=j+1
      do 9 i=1,nh
      a(j)=real(c(i))
      b(j)=aimag(c(i))
9      j=j+1
      return
      end
```