Refraction-Diffraction Model

REF/DIF S
Version 1.3

Documentation and User's Manual

James T. Kirby, Robert A. Dalrymple and Fengyan Shi

Center for Applied Coastal Research
Department of Civil and Environmental Engineering
University of Delaware, Newark, DE 19716

Research Report NO. CACR-04-01

July 2004
Supercedes Version 1.2 (Report No. 94-22, 1994)

Contents

List of Figures

1 INTRODUCTION

REF/DIF S is presently used by hundreds of researchers, practicing engineers and planners worldwide. The program is freely distributed through the web site
http://chinacat.coastal.udel.edu/ kirby/programs/refdif/refdif.html,
and links to various activities using the program are provided. The program is provided without warranty and under the copyright model of the Free Software Foundation, as detailed below.

Work on the present upgrade of **REF/DIF S** is supported by the National Ocean Partnership Program (NOPP) through the project “Development and Verification of a Comprehensive Community Model for Physical Processes in the Nearshore Ocean”, described at **http://chinacat.coastal.udel.edu/ kirby/NOPP/index.html**. The main goal in the upgrade to Version 1.2 has been to provide compatibility between **REF/DIF S** and the Nearshore Community Model system. Similar compatibility is being provided for the monochromatic wave model **REF/DIF 1** (Kirby et al, 2004), and a time dependent refraction/diffraction model developed by Kennedy and Kirby (2002). Each of these models will be documented independently and will be provided as free standing programs and as Nearshore Community Model components.

1.1 Legal Information

This section provides information pertaining to copyright and warranty. This information pertains to the free-standing **REF/DIF S** program and to the Nearshore Community Model framework as a whole.

1.1.1 Limits to Liability

1.1.2 The GNU Public License

REF/DIF S has been developed by James T. Kirby, Robert A. Dalrymple and Fengyan Shi for the purpose of computing the combined refraction/diffraction of spectral surface water waves in the coastal environment. The program is copyright (C) 2002 by James T. Kirby, Robert A. Dalrymple and Fengyan Shi. The particulars of the GNU Public License follow.

1.2 Notes on Using REF/DIF S in the NOPP Nearshore Community Model System.

REF/DIF S has been commonly used as a wave-driver in conjunction with a number of wave-induced nearshore circulation models. At present, a comprehensive community model is under development with the support of the National Ocean Partnership Program (NOPP). As this code is developed, small adjustments will be needed in the **REF/DIF S** program in order to accomodate the needs of the overall modelling system. Changes which are transparent to the users of **REF/DIF S** as a stand-alone program will not trigger a revision of the program documentation. Notes on using **REF/DIF S** in the context of the comprehensive system will appear here.

The NOPP model system is now undergoing preliminary development and documentation, and will be described separately.

1.3 Document and Source Code Generation using NOWEB

The program source and documentation for **REF/DIF S** are maintained using **NOWEB**, which is described at http://www.eecs.harvard.edu/ nr/noweb/.

2 Refraction-Diffraction Model REF/DIF S, Version 1.3.

Program to calculate the forward scattered wave field in regions with slowly varying depth and current, including the effects of refraction and diffraction. The program is based on the parabolic equation method. This program is an extension of REF/DIF 1 for the case where a directional spectral sea is to be simulated.

1. Parabolic approximation:
   (a) Minimax approximation given by Kirby (1986b).
2. Wave nonlinearity: choice of
   (a) Linear.
   (b) Composite nonlinear: approximate model of Kirby and Dalrymple (1986b).
   (c) Stokes nonlinear: model of Kirby and Dalrymple (1983a).
3. Wave breaking:
   (a) Model of Thornton and Guza (198?).

4. Absorbing structures and shorelines:
    (a) Thin film model surrounded by a natural surfzone (Kirby and Dalrymple, 1986a).
5. Energy dissipation: any of
    (a) Turbulent bottom friction damping.
    (b) Porous bottom damping.
    (c) Laminar boundary layer damping.
6. Lateral boundary conditions: either of
    (a) Reflective condition.
    (b) Open boundary condition (Kirby, 1986c).
7. Input wave field: either of
    (a) Model specification of monochromatic or directional wave field.
    (b) Input of initial row of data from disk file.
8. Output wave field:
    (a) Standard output.
    (b) Optional storage of last full calculated row of complex amplitudes.

The documentation of present program is contained in:

Kirby, J. T., Kaihatu, J. M., Özkan-Haller, H. T. and Chawla, A., 2001, "Combined refraction/diffraction model for spectral wave conditions. REF/DIF S, Version 1.3. Documentation and user's manual", Research Report CACR-01-XX, Center for Applied Coastal Research, Department of Civil and Environmental Engineering, University of Delaware.

2.1 Copyright and Warrantee Information

2.2 Point of Contact

James T. Kirby
Center for Applied Coastal Research
University of Delaware
Newark, DE 19716
kirbyudel.edu
(302) 831-2438, FAX (302) 831-1228

This file contains the source code and documentation for version 1.3 of REFDIF S, as of XX 2001. See documentation for REFDIF 1 for further information about basic algorithm changes.

c REFDIFSV12A.F c c This program is a modified version of the released code c REFDIFSV12.F c c It is designed to work specifically with SHORECIRC v1.3.6 c c It contains the following modifications: c c 1) It calculates mass flux for both Stokes drift (outside c surf zone) and rollers (inside surf zone) c c 2) It reads user-defined values of the breaking parameters B, c gamma and sigma ("sg") c c 3) It allows specification of longshore-varying input conditions c c 4) It allows specification of different breaking and decay c mechanisms c c 5) It allows specification of static-area or dynamic-area c rollers c c c initial edit 12/11/00 c c James M. Kaihatu, NRL Code 7322 c Naval Research Laboratory c Stennis Space Center, MS 39529-5004 c kaihatu@nrlssc.navy.mil c c NRL is not responsible for any harm, bodily or otherwise, resulting c from either proper or improper use of this code. (For a full legal c disclaimer, simply take the intro paragraphs from any subroutine of c the SWAN model and replace ”Netherlands” with ”USA.”) c c If that isn’t enough, recall that one thing scientists do not have a c lot of is money. c c cjmk 8/1/01 c c this version of REFDIFS calculates dissipation via the c Stive and deVriend mechanism as an option. c cjmk 8/1/01

⟨*⟩≡
```
      subroutine WaveModule()
      IMPLICIT NONE
      include 'param.h'
      include 'common.h'

c  Common block data passing to Master program.

      include 'pass.h'
      integer i,j

      real*8 urr(ixr,iyr),vrr(ixr,iyr),drr(ixr,iyr)
      integer mrr,nrr

      real*8 dconv,dconv2,dr,ur,vr,dxr,dyr,xr,yr,x,y,d,u,v,dx,dy,q
     $     ,p,sig,dd,an,anl,freqs,fpeak,amp,dir,tide,gam,b,sg
     $     ,psibar,h13,sp,so

      integer mr,nr,ispace,nd,md,iu,iff,icur,ibc,iun,iinput,ioutput,iopt
     $     ,isd,m,n,ntype,iwave,nfreqs,istore,nii,i,nwavs,irol,idecay,
     $     irolsij


c --- master_start=0 or 1 for initialization

      if(Master_Start.eq.1) then
        write(*,*) 'wave module initialization ...'
        else
        write(*,*) 'call wave module ...'
      endif

C  Constants.
      dconv(1)=1.
      dconv(2)=0.30488
      dconv2(1)=1.
      dconv2(2)=14.594


C  Read control parameters and reference grid data.

      call inref

C  Read control parameters and initializing wave data.
```

```
      if(Master_Start.ge.0)then
      call inwave
      close(1)
      endif

C  Pass program control to subroutine |model|.

C  For each frequency component specified in |inwave|, |model| executes the
C  model throughout the entire grid and then reinitializes the model for      t
C  the next frequency.

      if(Master_Start.le.0)then
      call model
      endif

C  All done.

C  Close output data files if |open| and |close| statements are being used.
      do 1 i=1,3
      close(iun(i))
1     continue
      close(iun(5))
      close(9)
      close(10)
      close(12)
      close(13)
      close(14)
      close(15)
      close(16)
      close(17)
      close(18)
      close(19)
      close(20)
      close(21)
      close(22)
      close(23)
      if(iopt.EQ.1)close(36)
c      stop
      return
      end
```

3 INREF.

This subroutine reads in and checks dimensions and values for large scale reference grid values. Wave parameters for the particular run are read in later by subroutine $inwave$.

The following unit (device) numbers are assumed:

- $iun(1)$: input reference grid values of d, u, and v.
- $iun(2)$: input user specified subgrid divisions.
- $iun(3)$: output results at reference grid locations to disk file.
- $iun(8)$: Output image of instantaneous water surface at computational grid resolution. This is interpolated to a regular rectangular grid by the program $surface2hdf.f$ and stored in HDF file format. Usual name for file is $surface.dat$.
- $iun(9)$: Output results for wave angles in file usually named $angle.dat$.
- $iun(10)$: log file for run - store basic run information and log file for error messages.
- $iun(12)$: Output results for significant wave height in file usually named $height.dat$.
- $iun(13)$: Output results for rad. stress Sxx in file usually named $sxx.dat$.
- $iun(14)$: Output results for rad. stress Sxy in file usually named $sxy.dat$.
- $iun(15)$: Output results for rad. stress Syy in file usually named $syy.dat$.
- $iun(16)$: Output results for tide-corrected depth grid in $depth.dat$
- $iun(5)$: Unit for file containing *namelist* input data. Usually named $indat.dat$. This filename is specified in the standard program version. In the LRSS version, an arbitrary filename is entered on the command line.

Variable definitions:

- mr, nr - reference grid dimensions.
- dxr, dyr - grid spacing for reference grid.
- iu - physical unit descriptor (1=mks, 2=english). Default value is 1, mks units.
- dt - depth tolerence value (to check for anomalous depth values).
- $ispace$ - switch to control grid subdivision.
    1. =0, program attempts its own subdivisions.

   2. =1, user specifies subdivisions.

- nd - y direction subdivision ($ispace$=0 or 1). Must be less than $iy/nr - 1$.
- $md(mr - 1)$ - x direction subdivisions (if $ispace$=1). Must be less than $ix - 1$.
- $ntype$ - nonlinearity control parameter.
  1. =0, linear model.
  2. =1, Stokes matched to Hedges in shallow water.
  3. =2, Stokes throughout.
- $icur$ - switch to tell program if current data is to be used and read on input.
  1. =0, no input current data.
  2. =1, input current data to be read.

  Program defaults to $icur$=0.
- ibc - boundary condition switch.
  1. =0, use closed lateral boundaries.
  2. =1, use open lateral conditions.

  Program defaults to ibc=0.
- dr - depths at grid points.
  1. dr .gt. 0, submerged areas.
  2. dr .lt. 0, elevation above surface datum.
- ur - x velocities at grid points (only entered if $icur$=1).
- vr - y velocities at grid points (only entered if $icur$=1).

Data is entered in $namelist$ format from $iun(5)$, which is attached to the file $indat.dat$ in $infile1.f$.

The subroutine is called from $refdifs$ and returns control to calling location, unless a fatal error is found during input data checking.

$\langle$*$\rangle$+$\equiv$

```
      subroutine inref
      IMPLICIT NONE
      include 'param.h'
      include 'pass.h'
      include 'common.h'
      integer i,j

      real*8 urr(ixr,iyr),vrr(ixr,iyr),drr(ixr,iyr)
      integer mrr,nrr

      real*8 dconv,dconv2,dr,ur,vr,dxr,dyr,xr,yr,x,y,d,u,v,dx,dy,q,p,sig
     $     ,dd,an,anl,freqs,fpeak,amp,dir,tide,gam,b,sg,psibar,h13,sp,so


      integer mr,nr,ispace,nd,md,iu,iff,icur,ibc,iun,iinput,ioutput,iopt
     $     ,isd,m,n,ntype,iwave,nfreqs,nwavs,istore,nii,isp,ir,jr,
     $     irol,idecay,irolsij

      real*8 g,test,dcheck,fr

C  Standard file name choices:

C  |fname1| = |refdat.dat|, reference grid data file.

C  |fname2| = |outdat.dat|, standard output data file.

C  |fname3| = |subdat.dat|, user specified subgrids.

C  |fname4| = |wave.dat|, user-specified complex amplitude on row 1 (for |iinput
C|       =2).

C  |fname5| = |owave.dat|, complex amplitude on last row (for |ioutput| = 2).

C  |fname6| = |surface.dat|, instantaneous water surface at computational
Cresolution.

C  |fname7| = |bottom1.dat|, magnitude of bottom velocity at reference grid
C  points.

C  |fname8| = |angle.dat|, wave directions at reference grid points.

C  |fname9| = |bottom2.dat|, normalized bottom velocity skewness at reference
C    grid points.

C  |fname10| = |refdifs.log|, run log for refdifs program.

C  |fname11| = |height.dat|, wave heights at reference grid locations.  For
C  REF/DIF S, the height is given as significant height H1/3.

C  |fname12| = |sxx.dat|, Sxx components at reference grid locations.

C  |fname13| = |sxy.dat|, Sxy components at reference grid locations.

C  |fname14| = |syy.dat|, Syy components at reference grid locations.

C  |fname15| = |depth.dat|, tide-corrected depths at reference grid locations.
C
```

```
C  |fname16| = |fluxus.dat|, Stokes mass flux in x direction

C  |fname17| = |fluxvs.dat|, Stokes mass flux in y direction

C  |fname18| = |diss.dat|, dissipation

C  |fname19| = |e_f_theta.dat|, directional spectra at each point

C  |fname20| = |e_f.dat|, frequency spectra at each point

C  |fname21| = |direct_mom.dat|, directional parameters

C  |fname22| = |fluxur.dat|, roller mass flux in x direction

C  |fname23| = |fluxvr.dat|, roller mass flux in y-direction

C  |fnamein| = |indat.dat|, input namelist file.

      namelist/ingrid/mr,nr,iu,ntype,icur,ibc,dxr,dyr,dt,ispace,nd,iff,i
     &sp,iinput,ioutput/inmd/md/fnames/fname1,fname2,fname3,fname4,fname
     &5,fname6,fname7,fname8,fname9,fname10,fname11,fname12,fname13,fnam
     &e14,fname15,fname16,fname17,fname18,fname19,fname20,fname21,fname2
     &2,fname23

      if(Master_Start.ge.0)then

C  Constants.
      g=9.80621
```

3.1 Specify name of namelist data file.

The LRSS implementation of Ref/Dif 1 imposes the restriction that no file names can be specified within the program itself. This makes it necessary to read in at least one file name as a command line argument. Two options are provided here by means of a subroutine $infile$. The code for the subroutine is provided in either of the files

1. $infile1.f$ - standard version. The program assumes the name $indat.dat$.
2. $infile2.f$ - user specifies the file name using the $igetarg$ command line syntax.

The $igetarg$ structure is supported on Sun Fortran and may be used at all times there. The SGI version tested to date uses a subroutine library $liblrss.a$ provided by SAIC.

⟨*⟩+≡

```
c      call infile(fnamein)
      iun(5)=31
      fnamein='indat.dat'
      open(unit=iun(5),file=fnamein,status='old')
```

3.2 Read remaining file names from namelist.

⟨*⟩+≡

```
      iun(1)=1
      iun(2)=2
      iun(3)=3
      read(iun(5),nml=fnames)

      endif
c ---- skip above after the first call wave module

      open(unit=iun(1),file=fname1,status='old')
      open(unit=iun(3),file=fname2)
      open(9,file=fname8)
      if(fname7.ne.'   ')then
      open(19,file=fname7)
      endif
      open(10,file=fname10)
      open(12,file=fname11)
      if(fname12.NE.'  ')then
      open(13,file=fname12)
      open(14,file=fname13)
      open(15,file=fname14)
      endif
      open(16,file=fname15)
c*jmk 12-13-00
c
c open files for mass flux calculation
c
c*jmk 12-13-00

      if(fname16.NE.'  ')then
      open(17,file=fname16)
      endif
      if(fname17.NE.'  ')then
      open(18,file=fname17)
      endif
      if(fname18.ne.'  ')then
      open(20,file=fname18)
      endif
      if(fname22.ne.'  ')then
      open(22,file=fname22)
      endif
      if(fname23.ne.'  ')then
      open(23,file=fname23)
      endif

c  Print header on log file.
      write(*,120)
      write(*,106)

      if(Master_Start.ge.0)then
```

```
C  Read control data from unit iun(5).
      read(iun(5),nml=ingrid)
      if(ispace.EQ.1)read(iun(5),nml=inmd)
c      mr=mrr  ! for dummy argu.
c      nr=nrr
      write(*,107)mr,nr,dxr,dyr
      if(iu.EQ.1)write(*,114)iu
      if(iu.EQ.2)write(*,115)iu
      if(icur.EQ.0)write(*,200)
      if(icur.EQ.1)write(*,201)
      if(ibc.EQ.0)write(*,202)
      if(ibc.EQ.1)write(*,203)
      if(ispace.EQ.0)write(*,108)
      if(ispace.EQ.1)write(*,109)
      write(*,119)nd
      if(ntype.EQ.0)write(*,110)
      if(ntype.EQ.1)write(*,111)
      if(ntype.EQ.2)write(*,112)

C  Check input from unit |iun(5)|.
      if((mr.GT.ixr).OR.(nr.GT.iyr))then
      write(*,*)'dimensions for reference grid too large, stopping'
c      call exit(1)
       stop
      end if
      if((iu.NE.1).AND.(iu.NE.2))iu=1
      dt=dt*dconv(iu)
      dxr=dxr*dconv(iu)
      dyr=dyr*dconv(iu)
      if(dt.EQ.0.)dt=2.
!wer  IFIX changed to IDINT
      if(nd.GT.(IDINT(dfloat(iy-1)/dfloat(nr-1))))then
      write(*,102)nd
c      call exit(1)
       stop
      endif
      if(ispace.EQ.1)then
      test=0.
      do 1 i=1,mr-1
      if(md(i).GT.(ix-1))then
      write(*,103)md(i),i
      test=1.
      endif
1     continue
      if(test.EQ.1.)stop
      endif

      endif
c --------------- skip above after the first call wave module

C  Pass depth grid and velocities from master program.

      do  i=1,mr
      do  j=1,nr
```

```
          dr(i,j)=Depth_Wave(i,j)
          ur(i,j)=Intp_U_Wave(i,j)
          vr(i,j)=Intp_V_Wave(i,j)
        enddo
        enddo

C  Convert depth and currents to metric units.
        do 5 i=1,mr
        do 5 j=1,nr
        dr(i,j)=dr(i,j)*dconv(iu)
5       continue
        if(icur.EQ.1)then
        do 55 i=1,mr
        do 55 j=1,nr
        ur(i,j)=ur(i,j)*dconv(iu)
        vr(i,j)=vr(i,j)*dconv(iu)
55      continue
        endif

C  Check for large depth changes and large currents in reference grid data.
        do 6 i=2,mr-1
        do 6 j=2,nr-1
        dcheck=(dr(i+1,j)+dr(i-1,j)+dr(i,j-1)+dr(i,j+1))/4.
        if(abs(dcheck-dr(i,j)).GT.dt)write(*,104)dr(i,j),i,j,dt
6       continue
        if(icur.EQ.1)then
        do 7 i=1,mr
        do 7 j=1,nr
        if(dr(i,j).LE.0.0)go to 7
        fr=(ur(i,j)*ur(i,j)+vr(i,j)*vr(i,j))/(g*dr(i,j))
        if(fr.GT.1.)write(*,105)i,j,fr
7       continue
        endif

C  Establish coordinates for reference grid.
        do 8 ir=1,mr
        xr(ir)=dfloat(ir-1)*dxr
8       continue
        do 9 jr=1,nr
        yr(jr)=dfloat(jr-1)*dyr
9       continue

C  Establish |y| coordinates for interpolated grid.
        n=nd*(nr-1)+1
        dy=dyr/dfloat(nd)
        do 10 j=1,n
        y(j)=dfloat(j-1)*dy
10      continue

C  Write grid information on output unit |iun(3)|.
        write(iun(3),*)nr,mr
        write(iun(3),*)(yr(jr)/dconv(iu),jr=1,nr)

C  Check friction values.
```

```
C         |iff(1)|=1, turbulent boundary layer damping everywhere.

C         |iff(2)|=1, porous bottom damping everywhere.

C         |iff(3)|=1, laminar boundary layer damping everywhere.

C  Set friction switches to zero if they are not zero or one.
      do 11 i=1,3
      if((iff(i).NE.0).AND.(iff(i).NE.1))iff(i)=0
11    continue
      write(*,116)(iff(i),i=1,3)

C  Specify whether or not user specified subgrids are to be         read in duri
Cng model operation.

C     |isp|=0, no subgrids specified.

C     |isp|=1, subgrids to be read in later from unit iun(2).
      if(isp.EQ.0)write(*,117)
      if(isp.EQ.1)then
      write(*,118)
      open(unit=iun(2),file=fname3,status='old')
      endif
      if((isp.EQ.1).AND.(ispace.EQ.0))write(*,113)
      if(isp.EQ.0)then
      do 14 ir=1,mr
      do 14 jr=1,nr
      isd(ir,jr)=0
14    continue
      else
      do 15 ir=1,mr-1
      read(iun(2),100)(isd(ir,jr),jr=1,nr-1)
15    continue
      endif

C  Input done, return to main program.
      return
100   format(15i4)
101   format(100f10.4)
102   format(' y-direction subdivision nd=',i4,'too fine.'/' maximum num
     &ber of y grid points will be exceeded.'/' execution terminating.')
103   format(' x-direction subdivision md=',i4,'too fine on grid block',
     &2x,i3/' execution terminating')
104   format(' depth',2x,f7.2,'(m) at reference grid location',2(2x,i3)/
     &' differs from the average of its neighbors by',' more than',2x,f7
     &.2,'(m).'/' execution continuing')
105   format(' ambient current at reference grid location',2(2x,i3),' is
     & supercritical with froude number =',f7.4/' execution continuing')
106   format('0'///20x,'input section, reference grid values'///)
107   format(' reference grid dimensions  mr=',i3/'
     &        nr=',i3///' reference grid spacings   dxr=',f8.4/'
     &                   dyr=',f8.4)
108   format(' '/' ispace =0 chosen, program will attempt its own ','ref
     &erence grid subdivisions')
 109  format(' '/' ispace =1 chosen, subdivision spacings will be',' inp
     &ut as data')
 110  format(' '/' ntype = 0, linear model')
 111  format(' '/' ntype = 1, stokes model matched to hedges model')
 112  format(' '/' ntype = 2, stokes model')
 113  format(' warning: input specifies that user will be supplying',' s
     &pecified subgrids (isp=1),'/' while program has been told to gener
     &ate its own subgrid',' spacings (ispace=0).'/' possible incompatib
     &ility in any or all subgrid blocks')
 114  format(' '/' physical unit switch iu=',i1,',  input in mks units')
 115  format(' '/' physical unit switch iu=',i1,',  input in english uni
     &ts')
 116  format(' '//'   switches for dissipation terms'//' ',i1,'   turbul
     &ent boundary layer'/' ',i1,'   porous bottom'/' ',i1,'   laminar b
     &oundary layer')
 117  format(' '/' isp=0, no user defined subgrids')
 118  format(' '/' isp=1, user defined subgrids to be read')
 119  format(' '/' y-direction subdivision according to nd=',i3)
 120  format(/////20x,'Refraction-Diffraction Model for'/20x,'Spectral
     &Wave Conditions'///20x,'REF/DIF S  Version 1.2'///20x,'Center for
     &Applied Coastal Research'/20x,'Department of Civil Engineering'/20
     &x,'University of Delaware'/20x,'Newark, Delaware 19716'///5x,'Jame
     &s T. Kirby, H.Tuba Ozkan and Arun Chawla, July 1992,'' February 19
     &94, August 1995')
 200  format(' '/' icur=0, no current values read from input files')
 201  format(' '/' icur=1, current values read from data files')
 202  format(' '/' ibc=0, closed (reflective) lateral boundaries')
 203  format(' '/' ibc=1, open lateral boundaries')
      end
```

4 INWAVE.

Read in wave parameters. The variable definitions are:

1. $iinput$ - determine method of specifying the first row of computational values.
   (a) =1, input values from indat.dat according to value of $iwave$. (This option is assumed to be chosen in $REF/DIFS$).
   (b) =2, input complex a values from file $wave.dat$.
2. $ioutput$ - determine whether last row of complex amplitudes are stored in separate file $owave.dat$.
   (a) =1, extra data not stored. (This option is assumed to be chosen in $REF/DIFS$).
   (b) =2, extra data stored in file $owave.dat$.
3. if $iinput$=1:
   (a) $iwave$ - input wave type.
      i. =1, input discrete wave amplitudes and directions.
      ii. =2, read in dominant direction, total average energy density, and spreading factor.
   (b) $nfreqs$ - number of frequency components to be used (separate model run for each component).
   (c) $freqs$ - wave frequency for each of $nfreqs$ runs.
   (d) $tide$ - tidal offset for each of $nfreqs$ runs.
4. If $iwave$ = 1
   (a) $nwavs$ - number of discrete wave components at each of $nfreqs$ runs.
   (b) amp - initial amplitude of each component.
   (c) dir - direction of each discrete component in + or - degrees from the x-direction.
5. If $iinput$=2: (This option not allowed in $REF/DIFS$. If $iinput = 2$ is found in the input data, the value is set equal to 1.)
   (a) $freqs$ - wave frequency for one run.
   (b) $tide$ - tidal offset for one run.

⟨*⟩+≡

```
      subroutine inwave
      IMPLICIT NONE
      include 'param.h'
      include 'common.h'

      real*8 dir2(nnd)
      character*255 complex_amplitude_file,freqfile1,freqfile2

      real*8 dconv,dconv2,dr,ur,vr,dxr,dyr,xr,yr,x,y,d,u,v,dx,dy,q,p,sig
     $     ,dd,an,anl,freqs,fpeak,amp,dir,tide,gam,b,sg,psibar
     $     ,h13,sp,so

      integer mr,nr,ispace,nd,md,iu,iff,icur,ibc,iun,iinput,ioutput,iopt
     $     ,isd,m,n,ntype,iwave,nwavs,istore,nii,ifreq,iwavs
     $     ,nfreqs,i,j,ik,ii,irol,idecay,irolsij

      real*8 pi

      namelist/waves1a/iwave,nfreqs/waves1b/freqs,tide,nwavs,amp,dir/wa
     $ves1c/tide,nwavs/compampfiles/complex_amplitude_file,freqfile1,fr
     $eqfile2

C  Constants.
c       pi=3.1415927
      pi=2.0*dacos(0.D0)

C  Values of |iinput|, |ioutput| already entered in namelist statement in
C|inref|.
      if(iinput.EQ.1)then
      write(*,*)'iinput = 1, program specifies initial row of a'
      else
      if(iinput.eq.2)then
      write(*,*)'iinput=2, user specifies initial row of a'
      else
      write(*,*) 'iinput unspecified, now what?'
      stop
      endif
      endif

C  Read |iwave|, |nfreqs|.
      if(iinput.eq.1)then
      write(*,102)
      read(iun(5),nml=waves1a)
      if(iwave.NE.1)iwave=1
      write(*,103)
      if(nfreqs.GT.ncomp)then
      write(*,104)
      stop
      endif
      write(*,105)nfreqs
```

```
C  Read in frequency, directions, wave height tidal offset.
      read(iun(5),nml=waves1b)
      if(tide.ne.0.0)then
         write(*,*)'tide is in wave model bathy but not hydro bathy'
         stop
      end if
      do 3 ifreq=1,nfreqs
      write(*,107)ifreq,freqs(ifreq),tide
      freqs(ifreq)=2.*pi/freqs(ifreq)
      tide=tide*dconv(iu)
      if(nwavs.GT.nnd)then
      write(*,109)
      stop
      endif
      do 1 iwavs=1,nwavs
      write(10,106)iwavs,amp(ifreq,iwavs),dir(ifreq,iwavs) !wer "*" changed to "10"
      dir(ifreq,iwavs)=dir(ifreq,iwavs)*pi/180.
      amp(ifreq,iwavs)=amp(ifreq,iwavs)*dconv(iu)
1     continue
3     continue
c
c     end of iinput=1
c
      endif
c
c     add bit for iinput=2
c
cjmk 3/26/01

      if(iinput.eq.2)then
      read(iun(5),nml=compampfiles)
      read(iun(5),nml=waves1a)
      read(iun(5),nml=waves1c)
c
c     read complex amplitude file
c
      open(30,file=complex_amplitude_file)
      do ik=1,nr
       do i=1,nfreqs
        do j=1,nwavs
         ii=nwavs*(i-1)+j
         read(30,*) a(1,ik,ii)
        enddo
       enddo
      enddo
      close(30)
      open(34,file=freqfile1)
      write(*,3001)freqfile1
3001  format(a255)
      do i=1,nfreqs
       read(34,*) freqs(i)
       freqs(i)=2.*pi*freqs(i)
      enddo
      close(34)
      open(32,file=freqfile2)
      do j=1,nwavs
       read(32,*) dir2(j)
      enddo
      close(32)
       do i=1,nfreqs
        do j=1,nwavs
         dir(i,j)=dir2(j)
        enddo
       enddo
c
c     end of iinput=2
c
      endif
      return
100   format(15i4)
101   format(500f8.4)
102   format('1'///20x,' input section, wave data values'///)
103   format(' '///' iwave=1, discrete wave amps and directions')
104   format('Too many frequency components; stopping')
105   format(' '///' the model is to be run for',i3,' separate',' freque
     &ncy components')
106   format(' '/' wave component ',i2,', amplitude =',f8.4,', direction
     &=',f8.4)
107   format(' '//' frequency component ',i2//' wave period=',f8.4,'sec.
     &, tidal offset=',f8.4)
108   format(' '/' total variance density =',f8.4,', spreading factor
     &       n=',i2,' seed number =',i5)
109   format('Too many directional components; stopping')
      end
```

5 MODEL.

This subroutine is the control level for the actual wave model. Data read in during $inref$ and $inwave$ is conditioned and passed to the wave model. This routine is executed once for each frequency component specified in $inwave$.

The wave model is split in four parts which are run sequentially for each reference grid row.

1. $grid$ - perform the interpolation of depth and current values.
2. con - calculate the constants needed by the finite difference scheme.
3. $fdcalc$ - perform the finite difference calculations.
4. $rbcon$ - roll the constants back one row in anticipation of next step.

$\langle$*$\rangle$+≡

```
      subroutine model
      IMPLICIT NONE
      include 'param.h'
      include 'common.h'
      include 'pass.h'
      integer i,j

      common/rolbk/phsp(2,iy),thm(2,iy),er(2,iy),disp(2,iy)
      integer npts(ncomp)

      real*8 s(iy),th(iy,nnii),sxy(iy),sxx(iy),syy(iy),sbxy(iy),sbxx(iy)
     $     ,sbyy(iy)
     $     ,sxxbody(iy),sxybody(iy),syybody(iy)
     $     ,hrms(iy),thet(iyr),rolmod(iy),sumk(ncomp)
     $     ,xd(iyr),xu(iyr),xk(iy,ncomp),kp(iyr),cgg(iy),hhh(iy)

      real*8 dconv,dconv2,dr,ur,vr,dxr,dyr,xr,yr,x,y,d,u,v,dx,dy,q,p,sig
     $     ,dd,an,anl,freqs,fpeak,amp,dir,tide,gam,b,sg,psibar,h13,sp,
     $     alpha2,disp,phsp,thm,er,wl0,so,wkh,fff1,fff2,cg0

      integer mr,nr,ispace,nd,md,iu,iff,icur,ibc,iun,iinput
     $     ,ioutput,iopt,isd,m,n,ntype,iwave,nfreqs,nwavs,istore,nii,jr
     $     ,ifreq,ii,iwavs,ir,jh,jj,icomp,mm1,icount,irol,idecay,iro
     $      lsij,ipeak

      real*8 g,rho,pi,eps,akdd,sum1,sum2,rmm,topp,bott,hhh2,wkmean,one
     $     ,two,hbb,ompeak,qp

      namelist/peak/fpeak/breakin/gam,b,sg,irol,idecay,irolsij


C  Constants.
      g=9.80621
      rho=1000.
```

```
c        pi=3.1415927
        pi=2.0*dacos(0.D0)
        eps=1.0e-05
        nii=nwavs*nfreqs
        if(nii.EQ.1)then
        fpeak=freqs(1)/(2.*pi)
        iopt=0
        else
        open(iun(5),file='indat.dat')
        read(iun(5),nml=peak)
        close(iun(5))
        iopt=0
        endif
c
c       read in breaking parameters
c
        open(iun(5),file='indat.dat')
        read(iun(5),nml=breakin)
        close(iun(5))
        write(*,*) 'in model'
         open(71,file='roller_energy.dat')
         open(72,file='roller_area.dat')
         if(irol.eq.2)then
         open(73,file='percent_break.dat')
        endif
C   Specify initial nonlinear parameters once.
        if(ntype.EQ.0)an=0.
        if(ntype.NE.0)an=1.
        if(ntype.NE.2)anl=0.
        if(ntype.EQ.2)anl=1.

C   Compute |kb| on first row, for use in specifying initial condition.
        do 32 jr=1,nr
        xd(jr)=dr(1,jr)
        xu(jr)=ur(1,jr)
32      continue

        call vwnum(xd,xu,freqs,xk,eps,nfreqs,nr)
        do 35 ifreq=1,nfreqs
c        do jr=1,nr
c        write(17,*) xk(jr,ifreq),jr,ifreq,nr,nfreqs
c        enddo
        npts(ifreq)=0.
        sumk(ifreq)=0.
35      continue
        do 36 ifreq=1,nfreqs
        do 37 jr=1,nr
        if(dr(1,jr).GT.0.05)then
        sumk(ifreq)=sumk(ifreq)+xk(jr,ifreq)
        npts(ifreq)=npts(ifreq)+1
        endif
37      continue
        if(npts(ifreq).EQ.0)then
        kb(1,ifreq)=xk(1,ifreq)
```

```
      else
      kb(1,ifreq)=sumk(ifreq)/dfloat(npts(ifreq))
      endif
c      write(*,*) kb(1,ifreq),xk(1,ifreq)
36    continue
c      pause

c
C  Set-up initial condition.
c
c add if-then for iinput=1
c  jmk 3/27/01
c
      if(iinput.eq.1)then
      do 2 ii=1,nii
      do 3 j=1,n
      do 4 i=1,2
      a(i,j,ii)=dcmplx(0.,0.)
4     continue
3     continue
2     continue
      do 5 j=1,n
      do 6 ifreq=1,nfreqs
      do 7 iwavs=1,nwavs
      ii=nwavs*(ifreq-1)+iwavs
      a(1,j,ii)=amp(ifreq,iwavs)*cdexp(dcmplx(0.,kb(1,ifreq)
     &*dsin(dir(ifreq
     &,iwavs))*y(j)))
      istore(ii)=ifreq
      th(j,ii)=dir(ifreq,iwavs)
c      if (j.eq.n/2)then
c      write(45,*) th(j,ii)*180./pi,ii
c      endif
7     continue
6     continue
5     continue
      endif
c
c     if iinput=2 then a(1,j,ii) already defined (in inwave)
c
      write(*,*) a(1,j,ii)
      if(iinput.eq.2)then
      do ifreq=1,nfreqs
       do iwavs=1,nwavs
        ii=nwavs*(ifreq-1)+iwavs
        istore(ii)=ifreq
       enddo
      enddo
      endif
      wl0=g*(1/fpeak)**2/(2*pi)
C  Compute significant waveheight |hs| for initial condition.
      if(nii.EQ.1)then
      do 31 j=1,n
      h13(j)=2.*cdabs(a(1,j,1))
      hrms(j)=h13(j)/sqrt(2.)
      so(j)=hrms(j)/wl0
      write(*,*) so(j)
31    continue
      else
      fff1=freqs(1)/(2.*pi)
      fff2=freqs(2)/(2.*pi)
      ipeak=int((fpeak-fff1)/(fff2-fff1))
      write(*,*) fff1,fff2,freqs(1),freqs(2),fpeak,ipeak
      do 13 j=1,n
      s(j)=0.
      do 14 ii=1,nii
      s(j)=s(j)+((cdabs(a(1,j,ii)))**2.)
14    continue
c
c     find index of peak frequency, calculate group velocity and
c     deep water group velocity, then deep water steepness
c
      wkh=xk(j,ipeak)*xd(j)
      cgg(j)=0.5*(2*pi*fpeak/xk(j,ipeak))*(1+(2*wkh/dsinh(2*wkh)))
      cg0=0.5*(g/(2*pi*fpeak))
      h13(j)=dsqrt(8.*s(j))
      hrms(j)=h13(j)/sqrt(2.)
      hhh(j)=hrms(j)*dsqrt(cgg(j)/cg0)
      so(j)=hhh(j)/wl0
c      write(*,*) so(j),j,hhh(j),wl0,hrms(j),h13(j),ipeak,cgg(j),cg0,
c     1xd(j),xk(ipeak,j)
13    continue
      endif

C  Calculate constants for first row.
       if(ir.EQ.1)then
       call con(ir,1,1)
       endif
cJMK
c   calculate q explicitly here
c
c   this was a problem. the values q were not passing through
c   properly or something so we just explicitly evaluate it
c   here. problem - will NOT work when nd .ne. 1....
cJMK
c
       if(nd.ne.1)then !wer
          write(*,*)'REFDIFS is BROKEN for nd.ne.1' !wer
          stop !wer
       end if !wer

      do j=1,n
      do ii=1,nii
      akdd=xk(1,istore(ii))*xd(j)
      q(1,j,istore(ii))=(1.+akdd/(dsinh(akdd)*dcosh(akdd)))/2.
      enddo
      enddo
c
```

```
cJMK
c

C  Compute radiation stress components for initial grid row.
c

cjmk
c
c     adding roller effects to radiation stress calculation
c     For time being, simply making H of Svendsen’s roller
c     term = H_rms for random waves.
c
c     The real answer would be to integrate Hˆ2 through the
c     Rayleigh probability distribution. Maybe later.
c
cjmk  1-10-01

c
c     first calculate some statistical quantities
c

      do 222 j=1,n
      sum1=0.
      sum2=0.
      do 223 ii=1,nii
       sum1=sum1+kb(1,istore(ii))*cdabs(a(1,j,ii))**2
       sum2=sum2+cdabs(a(1,j,ii))**2
223   continue
      rmm=(hrms(j)**2)/(gam*xd(j))**2
      topp=(gam*xd(j))**4
      bott=((hrms(j)**2)+(gam*xd(j))**2)**2
      hhh2=rmm*(hrms(j)**2)*(1-(topp/bott))
      wkmean=sum1/sum2
      one=1+(hrms(j)/(gam*xd(j)))**2
      two=1-(1/(one**(5/2)))
      hbb=(3.*sqrt(pi)/(4.*(gam*xd(j))**2))*hrms(j)**5*two
222   continue
c
c     now add roller term to radiation stress terms
c
c
c     analytic roller effect not right. will not put in
c     roller effect here. will put it in fdcalc.
c
c     jmk 12/2/01
c

      do 22 j=1,n
      sxx(j)=0.
      sxy(j)=0.
      syy(j)=0.
      sxxbody(j)=0.
      sxybody(j)=0.
      syybody(j)=0.
      do 18 ii=1,nii

!old      sxx(j)=sxx(j)+(cdabs(a(1,j,ii))**2)*((q(1,j,istore(ii))*
!old     &((1+rolmod(j))*(cos(th(j,ii))**2))+1.)-0.5)
        sxx(j)=sxx(j)+(cdabs(a(1,j,ii))**2)*(q(1,j,istore(ii))*
     $       ((1)*dcos(th(j,ii))**2+1.)-0.5) ! changed

        sxxbody(j)=sxxbody(j)+(cdabs(a(1,j,ii))**2)*(q(1,j,istore(ii))*
     $       ((1)*dcos(th(j,ii))**2))/xd(j) !bodyforcing

!old      syy(j)=syy(j)+(cdabs(a(1,j,ii))**2)*((q(1,j,istore(ii))*
!old     &((1+rolmod(j))*(sin(th(j,ii))**2))+1.)-0.5)
        syy(j)=syy(j)+(cdabs(a(1,j,ii))**2)*(q(1,j,istore(ii))*
     $       ((1)*dsin(th(j,ii))**2+1.)-0.5) ! changed

        syybody(j)=syybody(j)+(cdabs(a(1,j,ii))**2)*(q(1,j,istore(ii))*
     $       ((1)*dsin(th(j,ii))**2))/xd(j) ! bodyforciing

      sxy(j)=sxy(j)+q(1,j,istore(ii))*(cdabs(a(1,j,ii))**2)*
     &(1)*dsin(2.*th(j,ii))

      sxybody(j)=sxybody(j)+q(1,j,istore(ii))*(cdabs(a(1,j,ii))**2)*
     &(1)*dsin(2.*th(j,ii))/xd(j) ! bodyforcing

c      write(*,*) q(1,j,istore(ii))

18    continue

      sxx(j)=sxx(j)/2.
      syy(j)=syy(j)/2.
      sxy(j)=sxy(j)/4.
      sxxbody(j)=sxxbody(j)/2.
      syybody(j)=syybody(j)/2.
      sxybody(j)=sxybody(j)/4.
22    continue

C  Spatial smoothing of radiation stresses if |(nd.gt.1)|.
      if(nd.NE.1)then
      jh=dint(dfloat(nd)/2.d0)
      do j=1,n,nd
      sbxx(j)=0.
      sbyy(j)=0.
      sbxy(j)=0.
      if(j.EQ.1)then
      do jj=1,1+jh
      sbxx(1)=sbxx(1)+sxx(jj)
      sbxy(1)=sbxy(1)+sxy(jj)
      sbyy(1)=sbyy(1)+syy(jj)
      end do
      sbxx(1)=sbxx(1)/(jh+1)
      sbxy(1)=sbxy(1)/(jh+1)
      sbyy(1)=sbyy(1)/(jh+1)
      endif
      if(j.EQ.n)then
      do jj=n-jh,n
      sbxx(n)=sbxx(n)+sxx(jj)
      sbxy(n)=sbxy(n)+sxy(jj)
      sbyy(n)=sbyy(n)+syy(jj)
      end do
      sbxx(n)=sbxx(n)/(jh+1)
      sbxy(n)=sbxy(n)/(jh+1)
      sbyy(n)=sbyy(n)/(jh+1)
      endif
      if((j.GT.1).AND.(j.LT.n))then
      do jj=j-jh,j+jh
      sbxx(j)=sbxx(j)+sxx(jj)
      sbxy(j)=sbxy(j)+sxy(jj)
      sbyy(j)=sbyy(j)+syy(jj)
      end do
      sbxx(j)=sbxx(j)/(2*jh+1)
      sbxy(j)=sbxy(j)/(2*jh+1)
      sbyy(j)=sbyy(j)/(2*jh+1)
      endif
      end do
      endif
      if(nd.EQ.1)then
      do j=1,n
      sbxx(j)=sxx(j)
      sbxy(j)=sxy(j)
      sbyy(j)=syy(j)
      end do
      endif

C  Compute an average angle at the reference grid locations.
      ompeak=fpeak*2.*pi
      do jr=1,nr
      j=(jr-1)*nd+1
      call wvnum(xd(jr),xu(jr),ompeak,kp(jr),eps)
      qp=(1./2.)*(1.+2.*kp(jr)*xd(jr)/dsinh(2.*kp(jr)*xd(jr)))
      thet(jr)=(1./2.)*dasin(16.*sbxy(j)/(qp*hrms(j)*hrms(j)))
      thm(1,j)=thet(jr)
      thet(jr)=180.*thet(jr)/pi
      end do
c
c     initialize roller dissipation, if dynamic roller is selected
c
c     jmk 4/30/01
c
      if(irol.eq.2) then
       do jr=1,nr
        j=(jr-1)*nd+1
        er(1,j)=0.
        disp(1,j)=0.
        phsp(1,j)=2*pi*fpeak/wkmean
       enddo
      else
      endif
```

```
C  Initial condition set up, execute model for each grid block.
      do 20 ii=1,nfreqs
      psibar(ii)=0.
20    continue

      do 100 ir=1,(mr-1)

C  Establish interpolated grid block for segment |ir|.
      call grid(ir)

C  Write initial values on iun(3).
      if(ir.EQ.1)then
      write(iun(3),*)x(1)/dconv(iu)
      write(iun(3),*)(d(1,j)/dconv(iu),j=1,n,nd)
      write(12,203)(h13(j)/dconv(iu),j=1,n,nd)
      if(fname8.NE.'  ')then
      write(9,203)(thet(jr),jr=1,nr)
      endif
      if(fname12.NE.'  ')then
      write(13,203)(g*rho*sbxx(j)/dconv2(iu),j=1,n,nd)
      write(14,203)(g*rho*sbxy(j)/dconv2(iu),j=1,n,nd)
      write(15,203)(g*rho*sbyy(j)/dconv2(iu),j=1,n,nd)
      endif
      write(16,203)(d(1,j)/dconv(iu),j=1,n,nd)
      write(17,203)(0.0,j=1,n,nd)
      write(18,203)(0.0,j=1,n,nd)
      write(19,203)(0.0,j=1,n,nd)
      write(20,203)(0.0,j=1,n,nd)
      write(22,203)(0.0,j=1,n,nd)
      write(23,203)(0.0,j=1,n,nd)
      write(71,203)(0.0,j=1,n,nd)
      write(72,203)(0.0,j=1,n,nd)
      if(irol.eq.2)then
      write(73,203)(0.0,j=1,n,nd)
      endif
      if(iopt.EQ.1)then
      write(36,*)x(1)/dconv(iu),psibar(1)
      write(36,*)(d(1,j)/dconv(iu),j=1,n,nd)
!wer      write(36,*)(a(1,j,icomp)/dconv(iu),j=1,n,nd) !wer icomp is undefined, so I
      endif

C  Pass sxx height... at the first row -- Fengyan (01/25/02)

        do j=1,n,nd
          jr=(j-1)/nd +1
          Pass_Sxx(1,jr)=g*rho*sbxx(j)/dconv2(iu)
          Pass_Sxy(1,jr)=g*rho*sbxy(j)/dconv2(iu)
          Pass_Syy(1,jr)=g*rho*sbyy(j)/dconv2(iu)

          Pass_Sxx_body(1,jr)=g*rho*sxxbody(j)/dconv2(iu)
          Pass_Syy_body(1,jr)=g*rho*syybody(j)/dconv2(iu)
          Pass_Sxy_body(1,jr)=g*rho*sxybody(j)/dconv2(iu)
         Pass_Sxx_surf(1,jr)=Pass_Sxx(1,jr)-d(1,jr)*Pass_Sxx_body(1,jr)
         Pass_Syy_surf(1,jr)=Pass_Syy(1,jr)-d(1,jr)*Pass_Syy_body(1,jr)
         Pass_Sxy_surf(1,jr)=Pass_Sxy(1,jr)-d(1,jr)*Pass_Sxy_body(1,jr)

          Pass_Height(1,jr)=h13(j)/dconv(iu)
        enddo

        do j=1,nr
          Pass_Theta(1,j)=thet(j)
          Pass_Diss(1,j)=0.
          Pass_ubott(1,j)=0.
          Pass_MassFluxU(1,j)=0.
          Pass_MassFluxV(1,j)=0.
        enddo

      endif
      do 11 ii=1,nii
      do 12 j=1,n
      az(j,ii)=a(1,j,ii)
12     continue
11     continue

C  Calculate constants for first row.
      if(ir.EQ.1)then
      call con(ir,1,1)
      endif

C  Line printer output for first row.
      if(ir.EQ.1)then

C   Line printer output.
      mm1=m-1
      write(*,205)ir,mm1
      write(*,202)x(1)/dconv(iu)
      endif

C  For next row, evaluate constants.
      do 9 icount=2,m
      call con(ir,icount,2)
C  Perform finite differencing calculations.
      call fdcalc(ir,icount)

C  Roll back values to row 1.
      call rbcon
9      continue

C  Write out reference grid data on disk file iun(3).
      write(iun(3),*)x(m)/dconv(iu)
      write(iun(3),*)(d(m,j)/dconv(iu),j=1,n,nd)
      if(iopt.EQ.1)then
!wer      write(36,*)x(m)/dconv(iu),psibar(icomp) !wer icomp is undefined, so I remo
      write(36,*)(d(m,j)/dconv(iu),j=1,n,nd)
!wer      write(36,*)(a(1,j,icomp)/dconv(iu),j=1,n,nd) !wer icomp is undefined, so I
      endif
```

```
  134   continue

  100   continue

          if(ipeak.ne.0.)then
          Pass_period=2.*pi/fpeak
          else
          print*,'fpeak = 0. '
          endif

        call calculate_wave_forcing

        return

  202   format(' x=',f10.2)
  203   format(500f20.4)
  205   format(' grid row ir=',i3,',  ',i3,' x-direction subdivisions',' u
       &sed')
  206   format(' Significant Wave Height')
        end
```

6 GRID.

Interpolate the depth and current grids for reference grid block ir.

⟨*⟩+≡

```
      subroutine grid(ir)
      IMPLICIT NONE
      include 'param.h'
      include 'common.h'
      include 'pass.h'
      integer i,j

      integer npts(ncomp)
      real*8 dref(iy),sumk(ncomp),xu(iy),xk(iy,ncomp)

      real*8 dconv,dconv2,dr,ur,vr,dxr,dyr,xr,yr,x,y,d,u,v,dx,dy,q,p,sig
     $     ,dd,an,anl,freqs,fpeak,amp,dir,tide,gam,b,sg,psibar,h13,sp,so

      integer ir,mr,nr,ispace,nd,md,iu,iff,icur,ibc,iun,iinput,ioutput
     $     ,iopt,isd,m,n,ntype,iwave,nfreqs,nwavs,istore,nii,j,jj,jjj
     $     ,ifreq,np,i,jr,js,jf,irol,idecay,irolsij

      real*8 pi,eps,big,alw,anw

C  Constants.
c       pi=3.1415927
      pi=2.0*dacos(0.D0)
      eps=1.0e-05

C  Perform |y|-interpolation on reference grid.

C  Interpolate first row.
      do 10 j=1,n,nd
      d(1,j)=dr(ir,((j-1)/nd+1))+Intp_eta_Wave(ir,((j-1)/nd+1))
      u(1,j)=ur(ir,((j-1)/nd+1))
      v(1,j)=vr(ir,((j-1)/nd+1))
10    continue
      if(nd.GT.1)then
      do 12 jj=2,nr
      do 11 j=1,(nd-1)
      jjj=nd*(jj-2)+(j+1)
      d(1,jjj)=(dr(ir,jj)-dr(ir,jj-1))*y(jjj)/dyr+(yr(jj)*dr(ir,jj-1)-yr
     &(jj-1)*dr(ir,jj))/dyr
      u(1,jjj)=(ur(ir,jj)-ur(ir,jj-1))*y(jjj)/dyr+(yr(jj)*ur(ir,jj-1)-yr
     &(jj-1)*ur(ir,jj))/dyr
      v(1,jjj)=(vr(ir,jj)-vr(ir,jj-1))*y(jjj)/dyr+(yr(jj)*vr(ir,jj-1)-yr
     &(jj-1)*vr(ir,jj))/dyr
11    continue
12    continue
      endif

C  Set number of |x| points and define |x| values.
```

```
      if(ispace.EQ.0)then

C  |ispace|=0, program sets subdivisions.
      do 13 j=1,n
      dref(j)=d(1,j)+tide
      if(dref(j).LT.0.001)dref(j)=0.001
13    continue
      do 27 j=1,n
      xu(j)=u(1,j)
27    continue
      call vwnum(dref,xu,freqs,xk,eps,nfreqs,n)
      do 29 ifreq=1,nfreqs
      do 28 j=1,n
      k(1,j,ifreq)=xk(j,ifreq)
28    continue
29    continue
      do 14 ifreq=1,nfreqs
      npts(ifreq)=0
      sumk(ifreq)=0.
14    continue
      do 15 ifreq=1,nfreqs
      do 16 j=1,n
      if(d(1,j).GT.0.05)then
      sumk(ifreq)=sumk(ifreq)+k(1,j,ifreq)
      npts(ifreq)=npts(ifreq)+1
      endif
16    continue
15    continue
      do 17 ifreq=1,nfreqs
      if(npts(ifreq).EQ.0)then
      kb(1,ifreq)=k(1,1,ifreq)
      else
      kb(1,ifreq)=sumk(ifreq)/dfloat(npts(ifreq))
      endif
17    continue

C  Find maximum wavenumber.
      big=kb(1,1)
      if(nfreqs.EQ.1)go to 40
      do 18 ifreq=2,nfreqs
      if(kb(1,ifreq).GT.big)big=kb(1,ifreq)
18    continue

C  Compute spacing.
40    alw=2.*pi/big
      anw=dxr/alw
!wer  IFIX changed to IDINT
      np=IDINT(5.*anw)
      if(np.LT.1)np=1
      md(ir)=min((ix-1),np)
      if(np.GT.(ix-1))write(*,100)np,ir
      else

C  |ispace|=1, user specified subdivision.
```

```
      endif
      m=md(ir)+1
      dx=dxr/dfloat(md(ir))
      do 19 i=1,m
      x(i)=xr(ir)+dfloat(i-1)*dx
19    continue

C  Interpolate values on |m| row.
      do 20 j=1,n,nd
      d(m,j)=dr(ir+1,((j-1)/nd+1))
      u(m,j)=ur(ir+1,((j-1)/nd+1))
      v(m,j)=vr(ir+1,((j-1)/nd+1))
20    continue
      if(nd.GT.1)then
      do 21 jj=2,nr
      do 22 j=1,(nd-1)
      jjj=nd*(jj-2)+(j+1)
      d(m,jjj)=(dr(ir+1,jj)-dr(ir+1,jj-1))*y(jjj)/dyr+(yr(jj)*dr(ir+1,jj
     &-1)-yr(jj-1)*dr(ir+1,jj))/dyr
      u(m,jjj)=(ur(ir+1,jj)-ur(ir+1,jj-1))*y(jjj)/dyr+(yr(jj)*ur(ir+1,jj
     &-1)-yr(jj-1)*ur(ir+1,jj))/dyr
      v(m,jjj)=(vr(ir+1,jj)-vr(ir+1,jj-1))*y(jjj)/dyr+(yr(jj)*vr(ir+1,jj
     &-1)-yr(jj-1)*vr(ir+1,jj))/dyr
22    continue
21    continue
      endif

C  Interpolate values in |x|-direction.
      do 23 i=2,m-1
      do 24 j=1,n
      d(i,j)=(d(m,j)-d(1,j))*x(i)/dxr+(x(m)*d(1,j)-x(1)*d(m,j))/dxr
      u(i,j)=(u(m,j)-u(1,j))*x(i)/dxr+(x(m)*u(1,j)-x(1)*u(m,j))/dxr
      v(i,j)=(v(m,j)-v(1,j))*x(i)/dxr+(x(m)*v(1,j)-x(1)*v(m,j))/dxr
24    continue
23    continue

C  Add in user specified grid subdivisions (read from unit iun(2)).
      do 30 jr=1,nr-1
      if(isd(ir,jr).EQ.1)then
      js=nd*jr+(1-nd)
      jf=js+nd
      read(iun(2),101)((d(i,j),j=js,jf),i=1,m)
      if(icur.EQ.1)then
      read(iun(2),101)((u(i,j),j=js,jf),i=1,m)
      read(iun(2),101)((v(i,j),j=js,jf),i=1,m)
      endif
      do 31 i=1,m
      do 32 j=js,jf
      d(i,j)=d(i,j)*dconv(iu)
      u(i,j)=u(i,j)*dconv(iu)
      v(i,j)=v(i,j)*dconv(iu)
32    continue
31    continue
      end if
```

```
 30     continue

 C  Add tidal offset to all rows and establish thin film.
        do 33 i=1,m
        do 34 j=1,n
        d(i,j)=d(i,j)+tide
        if(d(i,j).LT.0.001)d(i,j)=0.001
 34     continue
 33     continue

 C  Interpolation complete, return to model.
        return
 100    format(’ model tried to put more spaces (md=’,i4,’) than allowed i
       &n grid block ’,i3)
 101    format(16f8.4)
        end
```

7 CON.

This subroutine calculates constants for row ij in reference grid block ir.

⟨*⟩+≡

```
        subroutine con(ir,icount,ij)
        IMPLICIT NONE
        include 'param.h'
        include 'common.h'

        real*8 akd(2,iy,ncomp),xd(iy),xu(iy),xk(iy,ncomp),npts(ncomp),s
       &umk(ncomp)

        real*8 dconv,dconv2,dr,ur,vr,dxr,dyr,xr,yr,x,y,d,u,v,dx,dy,q,p,sig
       $     ,dd,an,anl,freqs,fpeak,amp,dir,tide,gam,b,sg,psibar,h13,sp,so

        integer ir,icount,ij,mr,nr,ispace,nd,md,iu,iff,icur,ibc,iun,iinput
       $     ,ioutput,iopt,isd,m,n,ntype,iwave,nfreqs,nwavs,istore,nii,j
       $     ,ifreq,irol,idecay,irolsij

        real*8 eps

 C  Constants.
        eps=1.0e-05

 C  Calculated constants.
        do 1 j=1,n
        xd(j)=d(icount,j)
        xu(j)=u(icount,j)
 1      continue
        call vwnum(xd,xu,freqs,xk,eps,nfreqs,n)
        do 5 ifreq=1,nfreqs
        do 2 j=1,n
        k(ij,j,ifreq)=xk(j,ifreq)
 2      continue
 5      continue
        do 3 j=1,n
        do 4 ifreq=1,nfreqs
        sig(ij,j,ifreq)=freqs(ifreq)-k(ij,j,ifreq)*u(icount,j)
        akd(ij,j,ifreq)=k(ij,j,ifreq)*d(icount,j)
        q(ij,j,ifreq)=(1.+akd(ij,j,ifreq)/(dsinh(akd(ij,j,ifreq))
       &*dcosh(akd(
       &ij,j,ifreq))))/2.
        p(ij,j,ifreq)=q(ij,j,ifreq)*9.80621*tanh(akd(ij,j,ifreq))
       &/k(ij,j,i
       &freq)
        dd(ij,j,ifreq)=(dcosh(4.*akd(ij,j,ifreq))+8.
       &-2.*(tanh(akd(ij,j,ifre
       &q))**2))/(8.*(dsinh(akd(ij,j,ifreq))**4.))
 4      continue
 3      continue
```

```
C  Calculate the dissipation term |w|.
      call diss(ir,icount,ij)

C  Calculate the mean |kb| on each row.
      do 6 ifreq=1,nfreqs
      npts(ifreq)=0
      sumk(ifreq)=0.
6     continue
      do 7 ifreq=1,nfreqs
      do 8 j=1,n
      if(d(icount,j).GT.0.05)then
      sumk(ifreq)=sumk(ifreq)+k(ij,j,ifreq)
      npts(ifreq)=npts(ifreq)+1
      endif
8     continue
      if(npts(ifreq).EQ.0)then
      kb(ij,ifreq)=k(ij,1,ifreq)
      else
!wer..REAL used rather than DFLOAT
      kb(ij,ifreq)=sumk(ifreq)/REAL(npts(ifreq))
      endif
7     continue
      return
      end
```

8 FDCALC.

Perform the Crank-Nicolson finite-difference calculations on grid block ir. The method used is the implicit-implicit iteration used by Kirby and Dalrymple (1983).

⟨*⟩+≡

```
      subroutine fdcalc(ir,icount)
      IMPLICIT NONE
      include 'param.h'
      include 'common.h'
      include 'pass.h'
      integer i,j

      common/rolbk/phsp(2,iy),thm(2,iy),er(2,iy),disp(2,iy)
      external qb

!      real*8 ksth1,ksth2
      complex*16 first,second,third,fourth,fifth,sixth
      complex*16 c1,c2,c3,cp1,cp2,cp3,ci,damp
      complex*16 ac(iy,nnii),bc(iy,nnii),cc(iy,nnii),rhs(iy,nnii)
     &,sol(iy,nnii),rhso(iy,nnii)

      real*8 ucp1(iy),uc(iy),vc(iy),vcp1(iy),dc(iy),dcp1(iy)
     $     ,ksth1(nnii),ksth2(nnii),s(iy),thet(iy,nnii),sxy(iy)
     $     ,sxx(iy),syy(iy)
     $     ,sxxbody(iy),sxybody(iy),syybody(iy)
     $     ,urs(iy,nnii),beta(iy),rolmod(iy),qsu(iy)
     $     ,qsv(iy),hb(iy),qru(iy),qrv(iy),tmean(iy),qutot(iy),qvtot(iy)
     $     ,ubsig(iy),wl(iy),ga(iy),sxxr(iy),sxyr(iy),syyr(iy)

      real*8 kp(iyr),theta(iyr),sbxx(iy),sbxy(iy),sbyy(iy),alpha(iy),wk
     $mean(iy),area(iy),hm,hp

      real*8 thm,er,phsp,bett,qb,disp

      real*8 dconv,dconv2,dr,ur,vr,dxr,dyr,xr,yr,x,y,d,u,v,dx,dy,q,p,sig
     $     ,dd,an,anl,freqs,fpeak,amp,dir,tide,gam,b,sg,psibar,h13,sp,
     $     cab1,cab2,al,so,sumft

      integer ir,icount,mr,nr,ispace,nd,md,iu,iff,icur,ibc,iun,iinput
     $     ,ioutput,iopt,isd,m,n,ntype,iwave,nfreqs,nwavs,istore,nii
     $     ,ifreq,ifilt,iii,it,jh,jn,jj,jr,mm1,ii,irol,idecay,
     $      irolsij

      real*8 delta1,cdamp,a1,b1,pi,a0,delta2,u2,g,rho,eps,akx2,akx1,akx
     $     ,aky2,aky1,aky,sum1,sum2,rmm,topp,bott,hbbb,one,two
     $     ,hbb,ompeak,qp,arg,sumu,sumv,sum3,ubottom,sigbar,termm

      real*8 cg,pv,pvp1,bet,dv,deltap,hrms,alpha1,f1,f1p1,f2,f2p1,alpha
     $2,rolcoef,rolcoefm1,hr,hmax
```

```
c     define new variables for breaking term correction. Fengyan 04/15/2002
```

8.1 FDCALC statement functions.

The following code provides the statement functions used in establishing the tridiagonal matrix structure used in $fdcalc$.

⟨*⟩+≡

```
       cg(i,j,ifreq)=sqrt(p(i,j,ifreq)*q(i,j,ifreq))

       pv(i,j,ifreq)=p(i,j,ifreq)-vc(j)*vc(j)

       pvp1(i,j,ifreq)=p(i+1,j,ifreq)-vcp1(j)*vcp1(j)

       bet(i,j,ifreq)=-4.*(k(i+1,j,ifreq)-k(i,j,ifreq))/(dx*((k(i+1,j,ifr
      &eq)+k(i,j,ifreq))**2))-4.*(k(i+1,j,ifreq)*(p(i+1,j,ifreq)-ucp1(j)*
      &*2)-k(i,j,ifreq)*(p(i,j,ifreq)-uc(j)**2))/(dx*((k(i+1,j,ifreq)+k(i
      &,j,ifreq))**2.)*(p(i+1,j,ifreq)+p(i,j,ifreq)-(ucp1(j)**2.+uc(j)**2
      &)))

       dv(i,j,ifreq)=(cg(i+1,j,ifreq)+ucp1(j))/sig(i+1,j,ifreq)-(cg(i,j,i
      &freq)+uc(j))/sig(i,j,ifreq)+(-delta1)*dx*((vcp1(j+1)/sig(i+1,j+1,i
      &freq))+(vc(j+1)/sig(i,j+1,ifreq))-(vcp1(j-1)/sig(i+1,j-1,ifreq))-(
      &vc(j-1)/sig(i,j-1,ifreq)))/(2.*dy)

       damp(i,j,ifreq)=2.*ci*cdamp*((cg(i+1,j,ifreq)+ucp1(j))+(cg(i,j,ifr
      &eq)+uc(j)))/(dy*dy*(k(i+1,j,ifreq)**2+k(i,j,ifreq)**2))

       deltap(i,j,ifreq)=a1-b1*kb(i,ifreq)/k(i,j,ifreq)

       hrms(j)=h13(j)/(sqrt(2.))

       hr(i,j)=hrms(j)

       hmax(j)=0.88*dtanh(ga(j)*wkmean(j)*dcp1(j)/0.88)/wkmean(j)


       first(i,j,ifreq)=(cg(i+1,j,ifreq)+ucp1(j))
      &*dcmplx(1.,dx*(kb(i+1,ifr
      &eq)-a0*k(i+1,j,ifreq)))+dcmplx(1.,0.)*(cg(i,j,ifreq)+uc(j)
      &+dv(i,j,i
      &freq)*(sig(i+1,j,ifreq)+sig(i,j,ifreq))/4.)
      &+2.*freqs(ifreq)*dcmplx(
      &0.,1.)*(-b1)*bet(i,j,ifreq)*(ucp1(j)+uc(j))/sig(i+1,j,ifreq)+4.*fr
      &eqs(ifreq)*(-b1)*dcmplx(0.,1.)*(3.*(ucp1(j)-uc(j))/dx+(vcp1(j+1)
      &+vc
      &(j+1)-vcp1(j-1)-vc(j-1))/(4.*dy))/(sig(i+1,j,ifreq)*(k(i+1,j,ifreq
      &)+k(i,j,ifreq)))+dcmplx(-2.*(-b1)/(dy*dy*(k(i+1,j,ifreq)
      &+k(i,j,ifre
      &q)))+b1*bet(i,j,ifreq)*dx/(2.*dy*dy),(-deltap(i,j,ifreq))*dx/(2.*d
      &y*dy))*(pvp1(i,j+1,ifreq)+2.*pvp1(i,j,ifreq)+pvp1(i,j-1,ifreq))/si
      &g(i+1,j,ifreq)

       cp1(i,j,ifreq)=first(i,j,ifreq)
      &-dcmplx(1.,0.)*freqs(ifreq)*delta2*(
      &3.*ucp1(j)+uc(j))/(2.*sig(i+1,j,ifreq))+ci*freqs(ifreq)*(a0-1.)*k(
      &i+1,j,ifreq)*ucp1(j)*dx/sig(i+1,j,ifreq)+2.*ifilt*damp(i,j,ifreq)+
      &dcmplx(1.,0.)*2.*beta(j)*alpha(j)*dx
 c    add breaking term correction
      &*cg(i,j,ifreq)

       second(i,j,ifreq)=dcmplx((-delta1)*dx*(vcp1(j)+vc(j))/(2.*dy)
      &+b1*u2
      &*bet(i,j,ifreq)*(ucp1(j)*vcp1(j)+uc(j)*vc(j))/(dy*sig(i+1,j+1,ifre
      &q)),(-delta1*u2)*(ucp1(j+1)*vcp1(j+1)+uc(j+1)*vc(j+1)+2.*ucp1(j)*v
      &cp1(j))/(2.*dy*sig(i+1,j+1,ifreq))+dx*(-b1)*bet(i,j,ifreq)*(sig(i+
      &1,j,ifreq)*vcp1(j)+sig(i,j,ifreq)*vc(j))/(2.*dy*sig(i+1,j+1,ifreq)
      &))+dcmplx(2.*(-b1)/(dy*dy*(k(i+1,j,ifreq)+k(i,j,ifreq)))
      &+(-b1)*bet(
      &i,j,ifreq)*dx/(2.*dy*dy),-(-deltap(i,j,ifreq)*dx)/(2.*dy*dy))*(pvp
      &1(i,j+1,ifreq)+pvp1(i,j,ifreq))/sig(i+1,j+1,ifreq)
      &+4.*dcmplx(0.,1.)
      &*(-b1)*sig(i+1,j,ifreq)*vcp1(j)/(dy*sig(i+1,j+1,ifreq)*(k(i+1,j,if
      &req)+k(i,j,ifreq)))

       cp2(i,j,ifreq)=second(i,j,ifreq)-ifilt*damp(i,j,ifreq)

       third(i,j,ifreq)=dcmplx(-(-delta1)*dx*(vcp1(j)+vc(j))/(2.*dy)
      &+(-b1)
      &*u2*bet(i,j,ifreq)*(ucp1(j)*vcp1(j)+uc(j)*vc(j))/(dy*sig(i+1,j-1,i
      &freq)),-(-delta1)*u2*(ucp1(j-1)*vcp1(j-1)+uc(j-1)*vc(j-1)+2.*ucp1(
      &j)*vcp1(j))/(2.*dy*sig(i+1,j-1,ifreq))-dx*(-b1)*bet(i,j,ifreq)*(si
      &g(i+1,j,ifreq)*vcp1(j)+sig(i,j,ifreq)*vc(j))/(2.*dy*sig(i+1,j-1,if
      &req)))+dcmplx(2.*(-b1)/(dy*dy*(k(i+1,j,ifreq)+k(i,j,ifreq)))
      &-b1*bet
      &(i,j,ifreq)*dx/(2.*dy*dy),-(-deltap(i,j,ifreq)*dx)/(2.*dy*dy))*(pv
      &p1(i,j,ifreq)+pvp1(i,j-1,ifreq))/sig(i+1,j-1,ifreq)-4.*dcmplx(0.,1.
      &)*(-b1)*sig(i+1,j,ifreq)*vcp1(j)/(dy*sig(i+1,j-1,ifreq)*(k(i+1,j,i
      &freq)+k(i,j,ifreq)))

       cp3(i,j,ifreq)=third(i,j,ifreq)-ifilt*damp(i,j,ifreq)

       fourth(i,j,ifreq)=dcmplx(cg(i+1,j,ifreq)+ucp1(j)
      &-dv(i,j,ifreq)*(sig
      &(i+1,j,ifreq)+sig(i,j,ifreq))/4.,0.)+dcmplx(1.,-dx*(kb(i,ifreq)
      &-a0*
      &k(i,j,ifreq)))*(cg(i,j,ifreq)+uc(j))
      &+2.*dcmplx(0.,1.)*freqs(ifreq)*
      &(-b1)*bet(i,j,ifreq)*(ucp1(j)+uc(j))/sig(i,j,ifreq)+4.*dcmplx(0.,1.
      &)*freqs(ifreq)*(-b1)*(3.*(ucp1(j)-uc(j))/dx+(vcp1(j+1)+vc(j+1)-vcp
      &1(j-1)-vc(j-1))/(4.*dy))/(sig(i,j,ifreq)*(k(i+1,j,ifreq)+k(i,j,ifr
      &eq)))+dcmplx(-2.*(-b1)/(dy*dy*(k(i+1,j,ifreq)
      &+k(i,j,ifreq)))+(-b1)*
      &bet(i,j,ifreq)*dx/(2.*dy*dy),-(-deltap(i,j,ifreq)*dx)/(2.*dy*dy))*
      &(pv(i,j+1,ifreq)+2.*pv(i,j,ifreq)+pv(i,j-1,ifreq))/sig(i,j,ifreq)

       c1(i,j,ifreq)=fourth(i,j,ifreq)
      &-dcmplx(1.,0.)*freqs(ifreq)*delta2*(
      &3.*ucp1(j)+uc(j))/(2.*sig(i,j,ifreq))-ci*freqs(ifreq)*(a0-1.)*k(i,
      &j,ifreq)*uc(j)*dx/sig(i,j,ifreq)+2.*ifilt*damp(i,j,ifreq)-dcmplx(1.
      &,0.)*2.*(1-beta(j))*alpha(j)*dx
 c    add breaking term correction -- Fengyan 04/15/2002
      &*cg(i,j,ifreq)

       fifth(i,j,ifreq)=dcmplx(-(-delta1)*dx*(vcp1(j)+vc(j))/(2.*dy)
      &+b1*u2
      &*bet(i,j,ifreq)*(ucp1(j)*vcp1(j)+uc(j)*vc(j))/(dy*sig(i,j+1,ifreq)
      &),(-delta1)*u2*(ucp1(j+1)*vcp1(j+1)+uc(j+1)*vc(j+1)+2.*uc(j)*vc(j)
      &)/(2.*dy*sig(i,j+1,ifreq))+4.*(-b1)*sig(i,j,ifreq)*vc(j)/(dy*(k(i+
      &1,j,ifreq)+k(i,j,ifreq))*sig(i,j+1,ifreq))-dx*(-b1)*bet(i,j,ifreq)
      &*(sig(i+1,j,ifreq)*vcp1(j)+sig(i,j,ifreq)*vc(j))/(2.*dy*sig(i,j+1,
      &ifreq)))+dcmplx(2.*(-b1)/(dy*dy*(k(i+1,j,ifreq)+k(i,j,ifreq)))
      &+b1*b
      &et(i,j,ifreq)*dx/(2.*dy*dy),(-deltap(i,j,ifreq))*dx/(2.*dy*dy))*(p
      &v(i,j+1,ifreq)+pv(i,j,ifreq))/sig(i,j+1,ifreq)

       c2(i,j,ifreq)=fifth(i,j,ifreq)-ifilt*damp(i,j,ifreq)

       sixth(i,j,ifreq)=dcmplx((-delta1)*dx*(vcp1(j)+vc(j))/(2.*dy)
      &+(-b1)*
      &u2*bet(i,j,ifreq)*(ucp1(j)*vcp1(j)+uc(j)*vc(j))/(dy*sig(i,j-1,ifre
      &q)),-(-delta1)*u2*(ucp1(j-1)*vcp1(j-1)+uc(j-1)*vc(j-1)+2.*uc(j)*vc
      &(j))/(2.*dy*sig(i,j-1,ifreq))-4.*(-b1)*sig(i,j,ifreq)*vc(j)/(dy*(k
      &(i+1,j,ifreq)+k(i,j,ifreq))*sig(i,j-1,ifreq))+dx*(-b1)*bet(i,j,ifr
      &eq)*(sig(i+1,j,ifreq)*vcp1(j)+sig(i,j,ifreq)*vc(j))/(2.*dy*sig(i,j
      &-1,ifreq)))+dcmplx(2.*(-b1)/(dy*dy*(k(i+1,j,ifreq)+k(i,j,ifreq)))
      &-(
      &-b1)*bet(i,j,ifreq)*dx/(2.*dy*dy),(-deltap(i,j,ifreq))*dx/(2.*dy*d
      &y))*(pv(i,j,ifreq)+pv(i,j-1,ifreq))/sig(i,j-1,ifreq)

       c3(i,j,ifreq)=sixth(i,j,ifreq)-ifilt*damp(i,j,ifreq)

       f1(i,j,ifreq)=tanh(k(i,j,ifreq)*dc(j))**5.

       f1p1(i,j,ifreq)=tanh(k(i+1,j,ifreq)*dcp1(j))**5.

       f2(i,j,ifreq)=(k(i,j,ifreq)*dc(j)/dsinh(k(i,j,ifreq)*dc(j)))**4.

       f2p1(i,j,ifreq)=(k(i+1,j,ifreq)*dcp1(j)/dsinh(k(i+1,j,ifreq)
      &*dcp1(j)))**4.

 c
 c    fixed error in finite differencing of Stive and deVriend roller
 c
 cjmk 8/22/02
 c
       rolcoef(i,j)=(2./dx)*(phsp(i+1,j)*dcos(thm(i+1,j)))
      &+(g*dsin(bett)/2.)*
      &((1./phsp(i+1,j))+(1./phsp(i,j)))
       rolcoefm1(i,j)=(2./dx)*(phsp(i,j)*dcos(thm(i,j)))
      &-(g*dsin(bett)/2.)*
      &((1./phsp(i+1,j))+(1./phsp(i,j)))
```

c c I think there’s a booboo with the original REFDIFS. c c For a centered difference in x, such as Crank-Nicholson, the c diffence equation looks something like: c c (1/dx)[A(i+1)-A(i)] = -0.5[alpha(i+1)A(i+1)+ alpha(i)A(i)] c c or: c c [(1/dx)+0.5alpha(i+1)]A(i+1) = [-0.5alpha(i)+(1/dx)]A(i) c c This implies that the solution for A(i+1) is dependent on c alpha(i+1). But if alpha(i+1) is dependent on Hrms(i+1), c which is dependent on A(i+1), then what do you do? c c Iteration would seem to be key here. Use Hrms(i) to calculate c alpha(i), call it alpha(i+1) to start, calculate A(i+1) and c then Hrms(i+1), use this to recalculate A(i+1). c c this doesn’t appear to be done here, because in the statement c functions above, Hrms is calculated from H13, which isn’t c calculated until the end of a step. Thus it appears that the c Hrms is never updated. c c this needs to be fixed, particularly as it appears that iteration c occurs anyway even if linear theory is selected. c c jmk 12/4/01 c c under further review, have decided to hold off on heavy revision c until later - require confirmation of original problem from c other sources. c c jmk 12/4/01

⟨*⟩+≡

```
C  Booij coefficients.
      a0=1.
      a1=-0.75
      b1=-0.25

C  Constants.
      do 1 j=1,n
      dc(j)=d(icount-1,j)
      uc(j)=u(icount-1,j)
      vc(j)=v(icount-1,j)
      dcp1(j)=d(icount,j)
      ucp1(j)=u(icount,j)
      vcp1(j)=v(icount,j)
      beta(j)=0.5+0.5*((0.001/dc(j))**3)
1     continue
      u2=1.0
      g=9.80621
c      pi=3.1415927
      pi=2.0*dacos(0.D0)
      rho=1000.
      ci=dcmplx(0.,1.)
      cdamp=0.025
c      bett=0.1
      bett=(5.71*pi/180.)
      do j=1,n
      ga(j)=0.5+0.4*dtanh(33.*so(j))
      enddo
```

c c 3/25/97 c c previously we had difficulties with REFDIFS losing energy even c with Thornton and Guza breaking wave dissipation turned off c (commented out). The parameter ”ifilt” is now set to 0. c The parameter appears to trigger a numerical filter. This c filter is triggered in REFDIF1 when breaking starts. In this case c dissipation is on all the time so it is not clear at present c whether this should be 1 or 0. c c jmk c

⟨*⟩+≡

```
c        ifilt=1
      ifilt=0
      delta1=a1-b1
      delta2=1+2.*a1-2.*b1
      nii=nfreqs*nwavs
      it=0
      eps=0.00001
      al=1.8
c     notice: al=1 in JMK’s version. 1.8 is good for Duck94 case.

c
c     set up alpha
c
c     first step - calculate wkmean
c
      do j=1,n
       sum1=0.
       sum2=0.
       do ii=1,nii
        sum1=sum1+kb(2,istore(ii))*cdabs(a(1,j,ii))**2
        sum2=sum2+cdabs(a(1,j,ii))**2
       enddo
      continue
      wkmean(j)=sum1/sum2
      enddo
c
```

c first guess on dissipation - use hrms from c back row. c fake for now. Will make it iterative later. c c jmk 12-4-01 c c may not be that bad, even if uniterated. c c in subroutine model, disp(1,j) is calculated before c first call to fdcalc c c then fdcalc calculates disp(2,j) using hrms(j) from prior row c c iterates without updating, realizing that it's wrong c c then in rbcon, disp(1,j)=disp(2,j) and rollback occurs c c jmk 12-4-01 c c in preparation of fixing the problem, we've pulled alpha out c of the statement functions c c use the Eldeberky and Battjes result: c c c c and Dtot is the Battjes and Janssen dissipation mechanism. c c PRoblem - alpha then becomes a function of frequency because of c the Cg. Try using sqrt(gh) for now.

C — breaking term correction, Fengyan 04/11/2002 c As same as Jim K mentioned above in calculation of Battjes and Janssen, c calculations are not consistent between refdifs and energy equation) c c c In the correction, we define c c c and c c An equation becomes c

c c jmk 1/23/02 c

⟨*⟩+≡

```
        if(irol.eq.2.and.idecay.eq.1)then
         do j=1,nr
          hm=hmax(j)
          hp=h13(j)
          if (hm.lt.hp) hm=hp
          disp(2,j)=(0.25*al*fpeak*qb(hrms(j),hmax(j),dcp1(j))*hm
      &*hm)*rho*g
         enddo
        endif

        if(idecay.eq.2)then
         do j=1,nr
          hm=hmax(j)
          hp=h13(j)
          if (hm.lt.hp) hm=hp
          disp(2,j)=(0.25*al*fpeak*qb(hrms(j),hmax(j),dcp1(j))*hm
      &*hm)*rho*g
         enddo
         do j=1,nr
          sumft=0.
          do ifreq=1,nfreqs
           do iwave=1,nwavs
            ii=nwavs*(ifreq-1)+iwave
            sumft=sumft+cg(1,j,ifreq)*cdabs(a(1,j,ii))**2
           enddo
          enddo

          alpha(j)=0.5*(disp(2,j)/(rho*g))/sumft

          if(j.eq.20)then
          write(97,*) i,qb(hrms(j),hmax(j),dcp1(j)),
      1   hmax(j),hrms(j),dlog(qb(hrms(j),hmax(j),dcp1(j))),alpha(j)
          endif

         enddo
         else
```

```
c --- Thornton & Guza’s formulation
       do j=1,nr
        disp(2,j)=0.1875*sqrt(pi)*fpeak*(b**3)*hrms(j)**7/(gam**4)
     &            /(dcp1(j)**5)
       enddo
       do j=1,nr
        sumft=0.
        do ifreq=1,nfreqs
         do iwave=1,nwavs
          ii=nwavs*(ifreq-1)+iwave
          sumft=sumft+cg(1,j,ifreq)*cdabs(a(1,j,ii))**2
         enddo
        enddo
        alpha(j)=disp(2,j)/sumft
       enddo
      endif

C Setup right hand side of matrix equation.
      do 2 ii=1,nii
      rhs(1,ii)=dcmplx(0.,0.)
      do 3 j=2,(n-1)
      rhso(j,ii)=c1(1,j,istore(ii))*a(1,j,ii)+c2(1,j,istore(ii))*a(1,j+1
     &,ii)+c3(1,j,istore(ii))*a(1,j-1,ii)
      rhs(j,ii)=rhso(j,ii)-dx*w(1,j,ii)*a(1,j,ii)/2.
     &-dx*dcmplx(0.,1.)*an*
     &anl*sig(1,j,istore(ii))*k(1,j,istore(ii))*k(1,j,istore(ii))*dd(1,j
     &,istore(ii))*(cdabs(a(1,j,ii))**2.)*a(1,j,ii)/2.
     &-dx*dcmplx(0.,1.)*an
     &*(1.-anl)*sig(1,j,istore(ii))*((1.+f1(1,j,istore(ii))*k(1,j,istore
     &(ii))*k(1,j,istore(ii))*(cdabs(a(1,j,ii))**2.)
     &*dd(1,j,istore(ii)))*
     &tanh(k(1,j,istore(ii))*dc(j)+f2(1,j,istore(ii))*k(1,j,istore(ii))*
     &0.5*h13(j))/tanh(k(1,j,istore(ii))*dc(j))-1.)*a(1,j,ii)/2.
3     continue
      rhs(n,ii)=dcmplx(0.,0.)
2     continue

C  Return here for iterations.
20    it=it+1
      write(*,*) it,ir
      if(it.EQ.1)iii=1
      if(it.EQ.2)iii=2

C  Establish boundary conditions.
      if(ibc.EQ.1)then
      do 4 ii=1,nii
      cab1=cdabs(a(1,1,ii))+cdabs(a(1,2,ii))
      cab2=cdabs(a(1,n,ii))+cdabs(a(1,n-1,ii))
      if(cab1.eq.0.)then
      ksth1(ii)=1d-06
      else
      ksth1(ii)=dble((2.*(a(1,2,ii)-a(1,1,ii))/((a(1,2,ii)+a(1,1,ii))*dy
     &))*dcmplx(0.,-1.))
      endif
      if(cab2.eq.0.)then
      ksth2(ii)=1d-06
      else
      ksth2(ii)=dble((2.*(a(1,n,ii)-a(1,n-1,ii))/((a(1,n,ii)+a(1,n-1,ii)
     &)*dy))*dcmplx(0.,-1.))
      endif
      bc(1,ii)=dcmplx(1.,ksth1(ii)*dy/2.)
      cc(1,ii)=-dcmplx(1.,-ksth1(ii)*dy/2.)
      bc(n,ii)=-dcmplx(1.,-ksth2(ii)*dy/2.)
      ac(n,ii)=dcmplx(1.,ksth2(ii)*dy/2.)
4     continue
      else
      do 5 ii=1,nii
      bc(1,ii)=dcmplx(1.,0.)
      cc(1,ii)=-bc(1,ii)
      bc(n,ii)=dcmplx(1.,0.)
      ac(n,ii)=-bc(n,ii)
5     continue
      endif

C  Coefficients for forward row.
      do 6 ii=1,nii
      do 7 j=2,(n-1)
      ac(j,ii)=cp3(1,j,istore(ii))
      bc(j,ii)=cp1(1,j,istore(ii))+(dx/2.)*(w(2,j,ii))
     &+dcmplx(0.,an*anl)*
     &sig(2,j,istore(ii))*k(2,j,istore(ii))*k(2,j,istore(ii))*dd(2,j,ist
     &ore(ii))*(cdabs(a(iii,j,ii))**2.)*(dx/2.)
     &+dcmplx(0.,an*(1.-anl))*sig
     &(2,j,istore(ii))*(dx/2.)*((1.+f1p1(1,j,istore(ii))*k(2,j,istore(ii
     &))*k(2,j,istore(ii))*(cdabs(a(iii,j,ii))**2.)
     &*dd(2,j,istore(ii)))*t
     &anh(k(2,j,istore(ii))*dcp1(j)+f2p1(1,j,istore(ii))*k(2,j,istore(ii
     &))*0.5*h13(j))/tanh(k(2,j,istore(ii))*dcp1(j))-1.)
      cc(j,ii)=cp2(1,j,istore(ii))
7     continue
6     continue

C  Update solution one step.
      call vtrida(1,n,ac,bc,cc,rhs,sol,nii)
      do 8 ii=1,nii
      do 9 j=1,n
      a(2,j,ii)=sol(j,ii)
      sol(j,ii)=dcmplx(0.,0.)
9     continue
8     continue
      if(it.EQ.1)go to 20

C  Check Ursell parameter for Stokes solution.
      if(ntype.EQ.2)then
      do 23 j=1,n
      do 24 ii=1,nii
      urs(j,ii)=(cdabs(a(2,j,ii))/dcp1(j))
     &/((k(2,j,istore(ii))*dcp1(j))**
     &2)
      if(urs(j,ii).GT.0.5)write(*,204)urs(j,ii),icount,j,ii
24    continue
23    continue
      endif

C  Calculate reference phase function.
      do 10 ifreq=1,nfreqs
      psibar(ifreq)=psibar(ifreq)+(kb(2,ifreq)+kb(1,ifreq))*dx/2.
10    continue

C  Calculate significant waveheight |h13|.
      if(nii.EQ.1)then
      do 33 j=1,n
      h13(j)=2.*cdabs(a(2,j,1))
33    continue
      else
      do 25 j=1,n
      s(j)=0.
      do 26 ii=1,nii
      s(j)=s(j)+(cdabs(a(2,j,ii))**2)
26    continue
      h13(j)=sqrt(8.*s(j))
25    continue
      endif
      if(icount.EQ.m)then

C Calculate wave angles at reference grid rows.   Note:  angles are not well
C  defined in a directional, multicomponent sea, or where waves become short
C  crested.  This routine was heavily modified by Raul Medina, University of
C  Cantabria. It is further modified by Arun Chawla to take out a one-sided
C derivative that introduced an asymmetry bias.
c
cjmk
c
c  there was a problem with the way Chawla did the central
c  differencing. We re-dood it
c
cjmk
      do 16 ii=1,nii
      do 15 j=1,n
      if(a(2,j,ii).EQ.(0.,0.))then
      akx2=0.
      else
      akx2=dimag(cdlog(a(2,j,ii)))
      endif
      if(a(1,j,ii).EQ.(0.,0.))then
      akx1=0.
      else
      akx1=dimag(cdlog(a(1,j,ii)))
      endif
      if(abs(akx2-akx1).GT.pi)then
      akx=sign((2.*pi-(abs(akx1)+abs(akx2)))/dx,akx1)
      else
      akx=(akx2-akx1)/dx
      endif
      if(j.EQ.1)then
      if(a(2,j+1,ii).EQ.(0.,0.))then
      aky2=0.
      else
      aky2=dimag(cdlog(a(2,j+1,ii)))
      endif
      if(a(2,j,ii).EQ.(0.,0.))then
      aky1=0.
      else
      aky1=dimag(cdlog(a(2,j,ii)))
      endif
      elseif(j.EQ.n)then
      if(a(2,j,ii).EQ.(0.,0.))then
      aky2=0.
      else
      aky2=dimag(cdlog(a(2,j,ii)))
      endif
      if(a(2,j-1,ii).EQ.(0.,0.))then
      aky1=0.
      else
      aky1=dimag(cdlog(a(2,j-1,ii)))
      endif
      else
      if(a(2,j+1,ii).EQ.(0.,0.))then
      aky2=0.
      else
      aky2=dimag(cdlog(a(2,j+1,ii)))
      endif
      if(a(2,j-1,ii).EQ.(0.,0.))then
      aky1=0.
      else
      aky1=dimag(cdlog(a(2,j-1,ii)))
      endif
      endif
c
c     revision by jmk 10/11/96
c
c     fix finite difference scheme. move divide by 2’s from
c     individual aky definitions to here. centered difference in
c     interior of domain, forward or backward difference on lateral
c     boundaries
c
      if(j.eq.1.or.j.eq.n)then
       if(abs(aky2-aky1).GT.pi)then
        aky=sign((2.*pi-(abs(aky1)+abs(aky2)))/(dy),aky1)
       else
        aky=(aky2-aky1)/(dy)
       endif
      else
       if(abs(aky2-aky1).GT.pi)then
        aky=sign((2.*pi-(abs(aky1)+abs(aky2)))/(2.*dy),aky1)
       else
```

```
          aky=(aky2-aky1)/(2.*dy)
         endif
        endif
        thet(j,ii)=atan2(aky,(akx+kb(2,istore(ii))))
15      continue
16      continue

C  Estimation of radiation stresses.
c
cjmk
c
c      looks like a rho*g is missing from rad stress calc.
c
cjmk 12-26-00
c
c      adding the rho*g seems to screw up the angle calculation
c      we put it at writout
c
c
c      adding roller effects to radiation stress calculation
c      For time being, simply making H of Svendsen's roller
c      term = H_rms for random waves.
c
c      The real answer would be to integrate Hˆ2 through the
c      Rayleigh probability distribution. Maybe later.
c
cjmk  1-10-01

c
c      first calculate some statistical quantities
c

        do 222 j=1,n
        sum1=0.
        sum2=0.
        do 223 ii=1,nii
         sum1=sum1+sig(2,j,istore(ii))*cdabs(a(2,j,ii))**2
         sum2=sum2+cdabs(a(2,j,ii))**2
223     continue
        sigbar=sum1/sum2
        rmm=(hrms(j)**2)/(gam*dcp1(j))**2
        topp=(gam*dcp1(j))**4
        bott=((hrms(j)**2)+((gam*dcp1(j))**2))**2
        hbbb=rmm*(hrms(j)**2)*(1-(topp/bott))
        one=1+(hrms(j)/(gam*dcp1(j)))**2
        two=1-(1/(one**(5/2)))
        hbb=(3.*sqrt(pi)/(4.*(gam*dcp1(j))**2))*hrms(j)**5*two
cjmk
c
c      calculate roller energy density for roller effect on
c      radiation stress
c
c      note!! this isn't quite correct. the mean angle thm is
c      one row back from where things are calculated. however,
c     the problem is that sxy is needed to calculate mean angle,
c     but mean angle is needed to calculate sxy with roller.
c     you can see where this can make you crazy
c
c     might try refracting via snell’s law to forward row at some
c     point in the future.
c
c     jmk 11-28-01
c
cjmk
c
c     have determined that original idea sketched above comprise the
c     ravings of a madman, and that the radiation stress should be
c     unaffected by the roller when the angle is calculated. this
c     allows us to use the most recent theta mean to calculate the
c     roller effect.  ergo the roller effect calculation has been
c     moved back to where it was before.
c
c     jmk 12/2/01
c
cjmk
222   continue


      do 17 j=1,n
      sxx(j)=0.
      syy(j)=0.
      sxy(j)=0.
      sxxbody(j)=0.
      syybody(j)=0.
      sxybody(j)=0.
17    continue
      do 22 j=1,n
      do 18 ii=1,nii

!old      sxx(j)=sxx(j)+(cdabs(a(1,j,ii))**2)*((q(1,j,istore(ii))*
!old     &((1+rolmod(j))*(cos(thet(j,ii))**2))+1.)-0.5)

        sxx(j)=sxx(j)+(cdabs(a(2,j,ii))**2)*(q(2,j,istore(ii))
     $       *((1)*dcos(thet(j,ii))**2+1.)-0.5) ! changed

        sxxbody(j)=sxxbody(j)+(cdabs(a(2,j,ii))**2)*(q(2,j,istore(ii))
     $       *((1)*dcos(thet(j,ii))**2))/dcp1(j) ! bodyforcing

!old      syy(j)=syy(j)+(cdabs(a(1,j,ii))**2)*((q(1,j,istore(ii))*
!old     &((1)*(sin(thet(j,ii))**2))+1.)-0.5)

        syy(j)=syy(j)+(cdabs(a(2,j,ii))**2)*(q(2,j,istore(ii))
     $       *((1)*dsin(thet(j,ii))**2+1.)-0.5) ! changed

        syybody(j)=syybody(j)+(cdabs(a(2,j,ii))**2)*(q(2,j,istore(ii))
     $       *((1)*dsin(thet(j,ii))**2))/dcp1(j) ! bodyforcing

      sxy(j)=sxy(j)+q(2,j,istore(ii))*(cdabs(a(2,j,ii))**2)*
     &(1)*dsin(2.*thet(j,ii))

      sxybody(j)=sxybody(j)+q(2,j,istore(ii))*(cdabs(a(2,j,ii))**2)*
     &(1)*dsin(2.*thet(j,ii))/dcp1(j)  !bodyforcing

18    continue
      sxx(j)=(sxx(j)/2.)
      syy(j)=(syy(j)/2.)
      sxy(j)=(sxy(j)/4.)

      sxxbody(j)=sxxbody(j)/2.
      syybody(j)=syybody(j)/2.
      sxybody(j)=sxybody(j)/4.

22    continue

C  Smooth estimate of radiation stresses when subdivisions are used.
      if(nd.NE.1)then
      jh=dint(dfloat(nd)/2.d0)
      do j=1,n,nd
      sbxx(j)=0
      sbyy(j)=0
      sbxy(j)=0
      jn=(j-1)*nd+1
      if(j.EQ.1)then
      do jj=1,1+jh
      sbxx(1)=sbxx(1)+sxx(jj)
      sbxy(1)=sbxy(1)+sxy(jj)
      sbyy(1)=sbyy(1)+syy(jj)
      end do
      sbxx(1)=sbxx(1)/(jh+1)
      sbxy(1)=sbxy(1)/(jh+1)
      sbyy(1)=sbyy(1)/(jh+1)
      endif
      if(j.EQ.n)then
      do jj=n-jh,n
      sbxx(n)=sbxx(n)+sxx(jj)
      sbxy(n)=sbxy(n)+sxy(jj)
      sbyy(n)=sbyy(n)+syy(jj)
      end do
      sbxx(n)=sbxx(j)/(jh+1)
      sbxy(n)=sbxy(j)/(jh+1)
      sbyy(n)=sbyy(j)/(jh+1)
      endif
      if((j.GT.1).AND.(j.LT.n))then
      do jj=j-jh,j+jh
      sbxx(j)=sbxx(j)+sxx(jj)
      sbxy(j)=sbxy(j)+sxy(jj)
      sbyy(j)=sbyy(j)+syy(jj)
      end do
      sbxx(j)=sbxx(j)/(2*jh+1)
      sbxy(j)=sbxy(j)/(2*jh+1)
      sbyy(j)=sbyy(j)/(2*jh+1)
      endif
      end do
      endif
      if(nd.EQ.1)then
      do j=1,n
      sbxx(j)=sxx(j)
      sbxy(j)=sxy(j)
      sbyy(j)=syy(j)
      end do
      endif

C  Compute an average angle at the reference grid locations.
      ompeak=fpeak*2.*pi
      do jr=1,nr
      j=(jr-1)*nd+1
      call wvnum(dcp1(j),ucp1(j),ompeak,kp(jr),eps)
      qp=(1./2.)*(1.+2.*kp(jr)*dcp1(j)/dsinh(2.*kp(jr)*dcp1(j)))
c      arg=32.*(sbxy(j))/(qp*h13(jr)*h13(jr))
      arg=16.*(sbxy(j))/(qp*hrms(jr)*hrms(jr))
      if(abs(arg).GT.1.)then
c      if(abs(arg).lt.1.)then
      arg=0.
      write(*,*)'angle calculation failed at',icount,',',jr
      endif
      theta(jr)=(1./2.)*dasin(arg)
      thm(2,jr)=theta(jr)
      theta(jr)=180.*theta(jr)/pi
      enddo
c
c     calculate Stive and deVriend roller
c
c     jmk 12/2/01
c
      if(irol.eq.2)then
      do j=1,n
       wl(j)=2*pi/wkmean(j)
       phsp(2,j)=sigbar/wkmean(j)
       disp(2,j)=(0.25*al*fpeak*qb(hrms(j),hmax(j),dcp1(j))*hmax(j)
     1 *hmax(j))*rho*g
       er(2,j)=rolcoefm1(1,j)*er(1,j)/rolcoef(1,j)+((disp(2,j)+
     1 disp(1,j))/2.)/(rolcoef(1,j))
       if(j.eq.20)then
       write(99,1010) er(2,j),rolcoefm1(1,j),er(1,j),rolcoef(1,j),disp
     1 (2,j),disp(1,j),phsp(2,j),phsp(1,j),ir
       endif
1010   format(8f15.6,1x,i6)
      enddo
      else
      do j=1,n
       wl(j)=2*pi/wkmean(j)
       phsp(2,j)=sigbar/wkmean(j)
       one=1+(hrms(j)/(gam*dcp1(j)))**2
       two=1-(1/(one**(5/2)))
       hb(j)=(3.*sqrt(pi)/(4.*(gam*dcp1(j))**2))*hrms(j)**5*two
       area(j)=(b**3*hb(j))/(4*dcp1(j)*tan(pi*sg/180.))
```

```
         er(2,j)=(rho*area(j)*phsp(2,j)**2/(2*wl(j)))
        enddo
        endif
c
c     calculate roller contribution to radiation stress
c
c     jmk 12/2/01
c
        if(irolsij.eq.1)then
        do j=1,n
         sxxr(j)=2.*(er(2,j)/(rho*g))*dcos(thm(2,j))**2
         syyr(j)=2.*(er(2,j)/(rho*g))*dsin(thm(2,j))**2
         sxyr(j)=2.*(er(2,j)/(rho*g))*dcos(thm(2,j))*dsin(thm(2,j))
        enddo
        else
        do j=1,n
         sxxr(j)=0.
         syyr(j)=0.
         sxyr(j)=0.
        enddo
        endif
c
c     add roller effect to radiation stress
c
c     jmk 12/2/01
c
        do j=1,n
         sbxx(j)=sbxx(j)+sxxr(j)
         sbyy(j)=sbyy(j)+syyr(j)
         sbxy(j)=sbxy(j)+sxyr(j)
        enddo

C  Line printer output.
        mm1=m-1
        write(*,205)(ir+1),mm1
        write(*,202)x(m)/dconv(iu)

C  Pass the radiation stresses, wave height and angle -- Fengyan (01/25/02)

          do j=1,n,nd
            jr=(j-1)/nd +1
            Pass_Sxx(ir+1,jr)=g*rho*sbxx(j)/dconv2(iu)
            Pass_Sxy(ir+1,jr)=g*rho*sbxy(j)/dconv2(iu)
            Pass_Syy(ir+1,jr)=g*rho*sbyy(j)/dconv2(iu)

            Pass_Sxx_body(ir+1,jr)=g*rho*sxxbody(j)/dconv2(iu)
            Pass_Sxy_body(ir+1,jr)=g*rho*sxybody(j)/dconv2(iu)
            Pass_Syy_body(ir+1,jr)=g*rho*syybody(j)/dconv2(iu)

            Pass_Sxx_surf(ir+1,jr)=Pass_Sxx(ir+1,jr)
     &                         -d(ir+1,jr)*Pass_Sxx_body(ir+1,jr)
            Pass_Syy_surf(ir+1,jr)=Pass_Syy(ir+1,jr)
     &                         -d(ir+1,jr)*Pass_Syy_body(ir+1,jr)
            Pass_Sxy_surf(ir+1,jr)=Pass_Sxy(ir+1,jr)
     &                         -d(ir+1,jr)*Pass_Sxy_body(ir+1,jr)

            Pass_Height(ir+1,jr)=h13(j)/dconv(iu)
          enddo

          do j=1,nr
            Pass_Theta(ir+1,j)=theta(j)
          enddo


C  Output of wave angle, wave height and radiation stresses.
        write(12,203)(h13(j)/dconv(iu),j=1,n,nd)
        if(fname8.NE.'  ')then
        write(9,203)(theta(jr),jr=1,nr)
        endif
        if(fname12.NE.'  ')then
        write(13,203)(g*rho*sbxx(j)/dconv2(iu),j=1,n,nd)
        write(14,203)(g*rho*sbxy(j)/dconv2(iu),j=1,n,nd)
        write(15,203)(g*rho*sbyy(j)/dconv2(iu),j=1,n,nd)
        endif
        write(16,203)(d(m,j)/dconv(iu),j=1,n,nd)

!wer  I want to include the stuff below in the "if(icount.EQ.m)then" stucture.
!wer  With my dx=5m planar beach simulation, RDS was subdividing quite a lot, so thi
!wer      endif   ! wer I'm moving this endif to a location below

CJMK 12/11/00
C
C     calculate that mass flux
c
C
CJMK 12/11/00

        do jr=1,nr
         j=(jr-1)*nd+1
         sumu=0.
         sumv=0.
         do ii=1,nii
          sumu=sumu+g*kb(2,istore(ii))*cdabs(a(2,j,ii))**2/
     1  (2.*sig(2,j,istore(ii)))*
     1  dcos(thet(j,ii))
          sumv=sumv+g*kb(2,istore(ii))*cdabs(a(2,j,ii))**2/
     1  (2.*sig(2,j,istore(ii)))*
     1  dsin(thet(j,ii))
         enddo
         qsu(j)=sumu
         qsv(j)=sumv
        enddo
c
c     now calculate mass flux from rollers
c
c
c     first calculate mean period (and while we're
c     at it, mean bottom velocity)
```

```
c
      do jr=1,nr
       j=(jr-1)*nd+1
```

!wer...This was calculating ubottom as an average of ubottom over !wer...spectral components with a weighting by amplitude. !wer...Actually, it should be a summation, not an average. !wer...I'm changing it to use a different method. !wer...Essentially, I use the analogy !wer...amplitude(i) ¡==¿ Umax(i) !wer...ampRMS ¡==¿ Urms !wer...Umax is the maximum deviation of U from the mean. !wer...amplitude is the maximum deviation of eta from the mean. !wer.......and I calculate Urms in the same manner as ampRMS is calculated.

!wer...One concern is the directionality. It may be more correct to !wer.......calculate Urms and Vrms separately (using wave angle) and combine them afterwards. !wer.......I'm seeking a second opinion before doing this.

!wer...Note that SC's definition of u0 is essentially Umax. !wer...However, there is no Umax when you have multiple waves. !wer...Usig is the next best thing, I think. !wer...So I will convert from Urms to Usig and provide Usig to SC.

!wer sum1=0. !wer sum2=0. !wer sum3=0. !wer do ii=1,nii !wer sum1=sum1+sig(2,j,istore(ii))*cdabs(a(2,j,ii))**2 !wer sum2=sum2+cdabs(a(2,j,ii))**2 !wer ubottom=cdabs(a(2,j,ii))*sig(2,j,istore(ii))/ !wer 1 sinh(kb(2,istore(ii))*dcp1(j)) !wer sum3=sum3+cdabs(a(2,j,ii))**2*(ubottom**2) !wer enddo !wer sigbar=sum1/sum2 !wer ubrms(j)=sqrt(sum3/sum2)

⟨*⟩+≡

```
        sum1=0.
        sum2=0.
        sum3=0.
        do ii=1,nii
         sum1=sum1+sig(2,j,istore(ii))*cdabs(a(2,j,ii))**2
         sum2=sum2+cdabs(a(2,j,ii))**2
         ubottom=cdabs(a(2,j,ii))*sig(2,j,istore(ii))/
      1  dsinh(kb(2,istore(ii))*dcp1(j))
         sum3=sum3+(ubottom**2)
        enddo
        sigbar=sum1/sum2
        tmean(j)=2.*pi/sigbar
        ubsig(j)=sqrt(sum3*2.0) !wer...Without the 2.0, you get ubrms

cjmk
c
c      incorporate Stive and deVriend roller description if
c      desired -
c
cjmk 4/24/01
c       if(irol.eq.2)then
c        wl(j)=2*pi/wkmean(j)
c        phsp(2,j)=sigbar/wkmean(j)
c        er(2,j)=rolcoefm1(1,j)*er(1,j)/rolcoef(1,j)+((disp(2,j)+
c     1  disp(1,j))/2.)/(rolcoef(1,j))
c       endif
cjmk
c
c      roller energy density calculated above for
c      calculating roller effect on radiation stress
c
cjmk 11/28/01
```

```
         if(irol.eq.2)then
          area(j)=2*wl(j)*er(2,j)/(rho*phsp(2,j)**2)
          qru(j)=(area(j)/tmean(j))*dcos(pi*theta(jr)/180.)
          qrv(j)=(area(j)/tmean(j))*dsin(pi*theta(jr)/180.)
          if(j.eq.50)then
          write(98,3456) er(2,j),area(j),qru(j),qrv(j)
3456      format(4f20.10)
          endif
         else
          one=1+(hrms(j)/(gam*dcp1(j)))**2
          two=1-(1/(one**(5/2)))
          hb(j)=(3.*sqrt(pi)/(4.*(gam*dcp1(j))**2))*hrms(j)**5*two
          qru(j)=b**3*hb(j)*dcos(pi*theta(jr)/180.)/(4*dcp1(j)*tmean(j)
     1    *tan(pi*sg/180.))
          qrv(j)=b**3*hb(j)*dsin(pi*theta(jr)/180.)/(4*dcp1(j)*tmean(j)
     1    *tan(pi*sg/180.))
c         write(*,*) qru(j),qrv(j)
         endif
        enddo
c
c       add stokes drift and roller flux for total flux
c
        do jr=1,nr
         j=(jr-1)*nd+1
         qutot(j)=qsu(j)+qru(j)
         qvtot(j)=qsv(j)+qrv(j)
        enddo

C   Pass mass flux, dissip -- Fengyan 01/25/02

          do j=1,n,nd
            jr=(j-1)/nd +1
            Pass_MassFluxU(ir+1,jr)=qutot(j)
            Pass_MassFluxV(ir+1,jr)=qvtot(j)
            Pass_Diss(ir+1,jr)=alpha(j)
            Pass_ubott(ir+1,jr)=ubsig(j)
          enddo


c
c       write out
c
        if(fname16.NE.'  ')then
        write(17,203)(qsu(j),j=1,n,nd)
        endif
        if(fname17.NE.'  ')then
        write(18,203)(qsv(j),j=1,n,nd)
        endif
        if(fname22.ne.'  ')then
        write(22,203)(qru(j),j=1,n,nd)
        endif
        if(fname23.ne.'  ')then
        write(23,203)(qrv(j),j=1,n,nd)
```

```
      endif
c
c     write out ub_rms !wer...I changed it to ubsig.
c
      if(fname7.ne.'  ')then
      write(19,203)(ubsig(j),j=1,n,nd)
      endif
c
c     write out alpha, the dissipation
c
      if(fname18.ne.'  ')then
      write(20,203) (alpha(j),j=1,n,nd)
      endif
      write(71,203)(er(2,j),j=1,n,nd)
      write(72,203)(area(j),j=1,n,nd)
      if(irol.eq.2)then
      write(73,203)(qb(hrms(j),hmax(j),dc(j)),j=1,n,nd)
      endif
      endif !wer now put in the "endif" (corresponds to ... if(icount.EQ.m)then

C  Return control back to |model|.
      return
202   format(' x=',f10.2)
203   format(500f20.4)
204   format(' '//' Warning: Ursell number =',f10.4,' encountered at','g
     &          rid location',i6,',',i6/' during computation of wave com
     &ponent',i3,'should be using Stokes-Hedges model          (ntype=1)
     & due to shallow','water')
205   format(' grid row ir=',i3,',  ',i3,' x-direction subdivisions',' u
     &sed')
      end
```

9 RBCON.

Roll back constants and solution to row 1.

⟨*⟩+≡

```
      subroutine rbcon
      IMPLICIT NONE
      include 'param.h'
      include 'common.h'

       common/rolbk/phsp(2,iy),thm(2,iy),er(2,iy),disp(2,iy)

      real*8 dconv,dconv2,dr,ur,vr,dxr,dyr,xr,yr,x,y,d,u,v,dx,dy,q,p,sig
     $     ,dd,an,anl,freqs,fpeak,amp,dir,tide,gam,b,sg,psibar,h13,sp,
     $      phsp,thm,er,so,disp

      real*8 alpha2(2,iy)

      integer mr,nr,ispace,nd,md,iu,iff,icur,ibc,iun,iinput,ioutput,iopt
     $     ,isd,m,n,ntype,iwave,nfreqs,nwavs,istore,nii,j,ifreq,ii,irol,
     $     idecay,irolsij

      do 1 j=1,n

       alpha2(1,j)=alpha2(2,j)
       disp(1,j)=disp(2,j)
       phsp(1,j)=phsp(2,j)
       thm(1,j)=thm(2,j)
       er(1,j)=er(2,j)

C    Move the forward row values to the backward row.
      do 2 ifreq=1,nfreqs
      k(1,j,ifreq)=k(2,j,ifreq)
      sig(1,j,ifreq)=sig(2,j,ifreq)
      q(1,j,ifreq)=q(2,j,ifreq)
      p(1,j,ifreq)=p(2,j,ifreq)
      dd(1,j,ifreq)=dd(2,j,ifreq)
2     continue

C  Now zero out the forward row values.
      do 3 ifreq=1,nfreqs
      k(2,j,ifreq)=0.
      sig(2,j,ifreq)=0.
      q(2,j,ifreq)=0.
      p(2,j,ifreq)=0.
      dd(2,j,ifreq)=0.
3     continue
1     continue
      do 4 ifreq=1,nfreqs
      kb(1,ifreq)=kb(2,ifreq)
      kb(2,ifreq)=0.
4     continue
```

```
C  Roll back solution.
      do 5 j=1,n
      do 6 ii=1,nii
      a(1,j,ii)=a(2,j,ii)
      a(2,j,ii)=dcmplx(0.,0.)
      w(1,j,ii)=w(2,j,ii)
      w(2,j,ii)=0.
6     continue
5     continue
      return
      end
```

10 VWVNUM.

Vectorized wavenumber calculations. Variable definitions are same as in subroutine $wnum$. The wavenumber k is calculated according to:

$$s * s - 2 * s * k * u + k * k * u * u = g * k * \tanh(k * d) \tag{1}$$

where

- d = local water depth
- s = absolute frequency
- g = gravitational acceleration constant
- u = x-component of ambient current
- eps = tolerance for iteration convergence
- i, j = indices in finite-difference grid

Solution is by Newton-Raphson iteration using Eckart's approximation as a seed value.

⟨*⟩+≡

```
      subroutine vwnum(dt,u,freqs,k,eps,nfreqs,n)
      IMPLICIT NONE
      include 'param.h'
      real*8 dt(iy),u(iy),freqs(ncomp),f(iy,ncomp),fp(iy,ncomp)
      real*8 k(iy,ncomp),kn(iy,ncomp)

      integer nfreqs,n,ifreq,j,i

      real*8 eps,g,pi

C  Constants.
      g=9.806
c      pi=3.1415927
      pi=2.0*dacos(0.D0)
C  Calculate first guess.
      do 1 ifreq=1,nfreqs
      do 2 j=1,n
      k(j,ifreq)=freqs(ifreq)*freqs(ifreq)/(g*sqrt(tanh(freqs(ifreq)*fre
     &qs(ifreq)*dt(j)/g)))
2     continue
1     continue

C  Iteration.
      do 4 ifreq=1,nfreqs
      do 5 j=1,n
      do 3 i=1,40
      f(j,ifreq)=freqs(ifreq)**2-2.*freqs(ifreq)*k(j,ifreq)*u(j)+(k(j,if
     &req)*u(j))**2-g*k(j,ifreq)*tanh(k(j,ifreq)*dt(j))
      fp(j,ifreq)=-2.*freqs(ifreq)*u(j)+2.*k(j,ifreq)*(u(j)**2)-g*tanh(k
     &(j,ifreq)*dt(j))-g*k(j,ifreq)*dt(j)/(dcosh(k(j,ifreq)*dt(j))**2.)
      kn(j,ifreq)=k(j,ifreq)-f(j,ifreq)/fp(j,ifreq)
      if((abs(kn(j,ifreq)-k(j,ifreq))/kn(j,ifreq)).LT.eps)go to 10
      k(j,ifreq)=kn(j,ifreq)
3     continue
      write(*,*)'Wavenumber failed to converge for frequency component'
     &,ifreq,'on column',j
10    k(j,ifreq)=kn(j,ifreq)
5     continue
4     continue
      return
      end
```

11 WVNUM.

Single wavenumber calculations. The wavenumber k is calculated according to:

$$s * s - 2 * s * k * u + k * k * u * u = g * k * \tanh(k * d) \qquad (2)$$

where

- d = local water depth
- s = absolute frequency
- g = gravitational acceleration constant
- u = x-component of ambient current
- eps = tolerance for iteration convergence

Solution is by Newton-Raphson iteration using Eckart's approximation as a seed value.

⟨*⟩+≡

```
      subroutine wvnum(d,u,freqs,k,eps)
      IMPLICIT NONE
      include 'param.h'
      real*8 k,kn,d,u,freqs,eps,pi,g,f,fp
      integer i

C  Constants.
      g=9.806
c      pi=3.1415927
      pi=2.0*dacos(0.D0)

C  Calculate first guess.
      k=freqs*freqs/(g*sqrt(tanh(freqs*freqs*d/g)))

C  Iteration.
      do 3 i=1,40
      f=freqs**2-2.*freqs*k*u+(k*u)**2-g*k*tanh(k*d)
      fp=-2.*freqs*u+2.*k*(u**2)-g*tanh(k*d)-g*k*d/(dcosh(k*d)**2.)
      kn=k-f/fp
      if((abs(kn-k)/kn).LT.eps)go to 10
      k=kn
3     continue
      write(*,*)'Wavenumber failed to converge at peak frequency'
      write(*,*)' depth = ',d,', current =',u,', angular frequency =',f
     &reqs
      write(*,*)'  eps =',eps
10    k=kn
5     continue
4     continue
      return
      end
```

12 VTRIDA.

Vectorized tridiagonal matrix solution by double sweep algorithm. The present subroutine is adopted from the subroutine described in Carnahan, Luther and Wilkes, *Applied Numerical Methods*, Wiley, 1969, modified to handle complex array coefficients and solution values. Input and output are:

1. a,b,c = coefficients of row in tridiagonal matrix.
2. d = right hand side vector of matrix equation
3. v = solution vector
4. if,l = beginning and end indices of positions in the dimensioned range of the column vector to be considered.

$\langle$*$\rangle$+≡

```
      subroutine vtrida(i1,l,a,b,c,d,v,mm)
      IMPLICIT NONE
      include 'param.h'
      complex*16 a(iy,nnii),b(iy,nnii),c(iy,nnii),d(iy,nnii)
     $     ,v(iy,nnii),beta(iy,nnii),gamma(iy,nnii)
      integer i1,l,mm,j,i1p1,i,last,k

C  Compute intermediate vectors |beta| and |gamma|.
      do 1 j=1,mm
      beta(i1,j)=b(i1,j)
      gamma(i1,j)=d(i1,j)/beta(i1,j)
1     continue
      i1p1=i1+1
      do 2 i=i1p1,l
      do 3 j=1,mm
      beta(i,j)=b(i,j)-a(i,j)*c(i-1,j)/beta(i-1,j)
      gamma(i,j)=(d(i,j)-a(i,j)*gamma(i-1,j))/beta(i,j)
3     continue
2     continue

C  Compute solution vector |v|.
      do 4 j=1,mm
      v(l,j)=gamma(l,j)
4     continue
      last=l-i1
      do 5 k=1,last
      i=l-k
      do 6 j=1,mm
      v(i,j)=gamma(i,j)-c(i,j)*v(i+1,j)/beta(i,j)
6     continue
5     continue
      return
      end
```

13 DISS.

Subroutine calculates the dissipation at a single grid point based on values of the switch iw at that point.

⟨*⟩+≡

```
       subroutine diss(ir,icount,i)
       IMPLICIT NONE
       include 'param.h'
       include 'common.h'

       real*8 nu,cp,kd(iy,nnii)

       real*8 dconv,dconv2,dr,ur,vr,dxr,dyr,xr,yr,x,y,d,u,v,dx,dy,q,p,sig
      $     ,dd,an,anl,freqs,fpeak,amp,dir,tide,gam,b,sg,psibar,h13,sp,so

       real*8 g,pi,f

       integer ir,icount,i,mr,nr,ispace,nd,md,iu,iff,icur,ibc,iun,iinput
      $     ,ioutput,iopt,isd,m,n,ntype,iwave,nfreqs,nwavs,istore,nii,j
      $     ,ifreq,ii,irol,idecay,irolsij

       real*8 sq

 C  Statement function.
       sq(i,j,ifreq)=sqrt(nu/(2.*sig(i,j,ifreq)))

 C  Constants.
       nu=1.3e-06
       cp=4.5e-11
       g=9.80621
 c      pi=3.1415927
       pi=2.0*dacos(0.D0)

 C  Value of |f| here is value assuming |tau=(f/8)*u**2|.
       f=0.01*4.0
       do 1 j=1,n
       do 2 ii=1,nfreqs
       w(i,j,ii)=dcmplx(0.,0.)
       kd(j,ii)=k(i,j,ii)*d(icount,j)

 C  If |iff(1) = 1|, use turbulent boundary layer damping.
       if(iff(1).EQ.1)w(i,j,ii)=2.*f*cdabs(az(j,ii))
      &*sig(i,j,istore(ii))*k
      &(i,j,istore(ii))/(dsinh(2.*kd(j,ii))*dsinh(kd(j,ii))*3.*pi)

 C  If |iff(2) = 1|, add porous bottom damping.
       if(iff(2).EQ.1)w(i,j,ii)=w(i,j,ii)+(g*k(i,j,istore(ii))*cp/(nu*(co
      &sh(kd(j,ii))**2)))*dcmplx(1.,0.)

 C  If |iff(3) = 1|, add boundary layer damping.
       if(iff(3).EQ.1)w(i,j,ii)=w(i,j,ii)+2.*k(i,j,istore(ii))*sig(i,j,is
      &tore(ii))*sq(i,j,istore(ii))*(1.+(dcosh(kd(j,ii))**2))
      &*dcmplx(1.,-1.
     &)/dsinh(2.*kd(j,ii))
 2      continue
 1      continue
        return
        end
c
c       include function for breaking probability
c
        function qb(hrmss,hmaxx,dep)
        IMPLICIT NONE
        include 'param.h'
        real*8 hmaxx,hrmss,dep,be,z,b2e,q0,qb
c        write(*,*) 'in qb'

        if ((hmaxx.gt.0).and.(hrmss.gt.0))then
         be=dsqrt(hrmss**2/hmaxx**2)
        else
         be=0
        endif
c
c       solutions use approximations
c
        if(be.le.0.5)then
          q0=0.
        else if (be.le.1.0)then
          q0=(2.*be-1.)**2
        endif
c
        if(be.le.0.2)then
         qb=0.
        else if (be.lt.1.0)then
         b2e=be*be
         z=dexp((q0-1.)/b2e)
         qb=q0-b2e*(q0-z)/(b2e-z)
        else
         qb=1
        endif
        return
        end

          subroutine calculate_wave_forcing
          include 'param.h'
          include 'common.h'
          include 'pass.h'
          integer i,j
c ---  depth-integerated short wave forcing

          do j=2,ny_wave-1
          do i=2,nx_wave-1
            Pass_Wave_Fx(i,j)=(Pass_Sxx(i+1,j)-Pass_Sxx(i-1,j))/2./dxr
     *                  +(Pass_Sxy(i,j+1)-Pass_Sxy(i,j-1))/2./dyr
            Pass_Wave_Fy(i,j)=(Pass_Sxy(i+1,j)-Pass_Sxy(i-1,j))/2./dxr
     *                  +(Pass_Syy(i,j+1)-Pass_Syy(i,j-1))/2./dyr
          enddo
```

```
      enddo

      do i=2,nx_wave-1
        j=1
        Pass_Wave_Fx(i,j)=(Pass_Sxx(i+1,j)-Pass_Sxx(i-1,j))/2./dxr
   *                +(Pass_Sxy(i,j+1)-Pass_Sxy(i,j))/1./dyr
        Pass_Wave_Fy(i,j)=(Pass_Sxy(i+1,j)-Pass_Sxy(i-1,j))/2./dxr
   *                +(Pass_Syy(i,j+1)-Pass_Syy(i,j))/1./dyr
      enddo
      do i=2,nx_wave-1
        j=ny_wave
        Pass_Wave_Fx(i,j)=(Pass_Sxx(i+1,j)-Pass_Sxx(i-1,j))/2./dxr
   *                +(Pass_Sxy(i,j)-Pass_Sxy(i,j-1))/1./dyr
        Pass_Wave_Fy(i,j)=(Pass_Sxy(i+1,j)-Pass_Sxy(i-1,j))/2./dxr
   *                +(Pass_Syy(i,j)-Pass_Syy(i,j-1))/1./dyr
      enddo

      do j=2,ny_wave-1
        i=1
        Pass_Wave_Fx(i,j)=(Pass_Sxx(i+1,j)-Pass_Sxx(i,j))/1./dxr
   *                +(Pass_Sxy(i,j+1)-Pass_Sxy(i,j-1))/2./dyr
        Pass_Wave_Fy(i,j)=(Pass_Sxy(i+1,j)-Pass_Sxy(i,j))/1./dxr
   *                +(Pass_Syy(i,j+1)-Pass_Syy(i,j-1))/2./dyr
      enddo

      do j=2,ny_wave-1
        i=nx_wave
        Pass_Wave_Fx(i,j)=(Pass_Sxx(i,j)-Pass_Sxx(i-1,j))/1./dxr
   *                +(Pass_Sxy(i,j+1)-Pass_Sxy(i,j-1))/2./dyr
        Pass_Wave_Fy(i,j)=(Pass_Sxy(i,j)-Pass_Sxy(i-1,j))/1./dxr
   *                +(Pass_Syy(i,j+1)-Pass_Syy(i,j-1))/2./dyr
      enddo

      i=1
      j=1
        Pass_Wave_Fx(i,j)=(Pass_Sxx(i+1,j)-Pass_Sxx(i,j))/1./dxr
   *                +(Pass_Sxy(i,j+1)-Pass_Sxy(i,j))/1./dyr
        Pass_Wave_Fy(i,j)=(Pass_Sxy(i+1,j)-Pass_Sxy(i,j))/1./dxr
   *                +(Pass_Syy(i,j+1)-Pass_Syy(i,j))/1./dyr
      i=nx_wave
      j=1
        Pass_Wave_Fx(i,j)=(Pass_Sxx(i,j)-Pass_Sxx(i-1,j))/1./dxr
   *                +(Pass_Sxy(i,j+1)-Pass_Sxy(i,j))/1./dyr
        Pass_Wave_Fy(i,j)=(Pass_Sxy(i,j)-Pass_Sxy(i-1,j))/1./dxr
   *                +(Pass_Syy(i,j+1)-Pass_Syy(i,j))/1./dyr
      i=1
      j=ny_wave
        Pass_Wave_Fx(i,j)=(Pass_Sxx(i+1,j)-Pass_Sxx(i,j))/1./dxr
   *                +(Pass_Sxy(i,j)-Pass_Sxy(i,j-1))/1./dyr
        Pass_Wave_Fy(i,j)=(Pass_Sxy(i+1,j)-Pass_Sxy(i,j))/1./dxr
   *                +(Pass_Syy(i,j)-Pass_Syy(i,j-1))/1./dyr

      i=nx_wave
      j=ny_wave
```

```
          Pass_Wave_Fx(i,j)=(Pass_Sxx(i,j)-Pass_Sxx(i-1,j))/1./dxr
     *                +(Pass_Sxy(i,j)-Pass_Sxy(i,j-1))/1./dyr
          Pass_Wave_Fy(i,j)=(Pass_Sxy(i,j)-Pass_Sxy(i-1,j))/1./dxr
     *                +(Pass_Syy(i,j)-Pass_Syy(i,j-1))/1./dyr

c ---  - and / rho

        do j=1,ny_wave
        do i=1,nx_wave
          Pass_Wave_Fx(i,j)=-Pass_Wave_Fx(i,j)/rho
          Pass_Wave_Fy(i,j)=-Pass_Wave_Fy(i,j)/rho
        enddo
        enddo

        return
        end
```