MASTER PROGRAM

FOR NEARSHORE COMMUNITY MODEL

Version x.x

Documentation and User's Manual

Fengyan Shi, James T. Kirby, Priscilla Newberger, and Kevin Haas

NOPP Nearshore Community Model Research Group

July 16, 2003

Contents

1 Master Program Outline

1.1 Functions of Master Program

The master program plays a role of "backbone" in the Nearshore Community Model. It is an interface, handling module coupling control, internal data transfer and interpolation/extrapolation between modules, as well as data input and output. The functions of the master program can be summarized as following

1. connection of three modules: i.e., wave module, circulation module and sediment transport module
2. time stepping control and model coupling control
3. model interaction control: one-way coupling or two-way coupling
4. internal data transfer between modules, interpolation or extrapolation between different computational grids if needed
5. data input and output

The master program does not handle input of data and parameters needed by specific modules. Each module should have its own input files.

1.2 Flow Chart

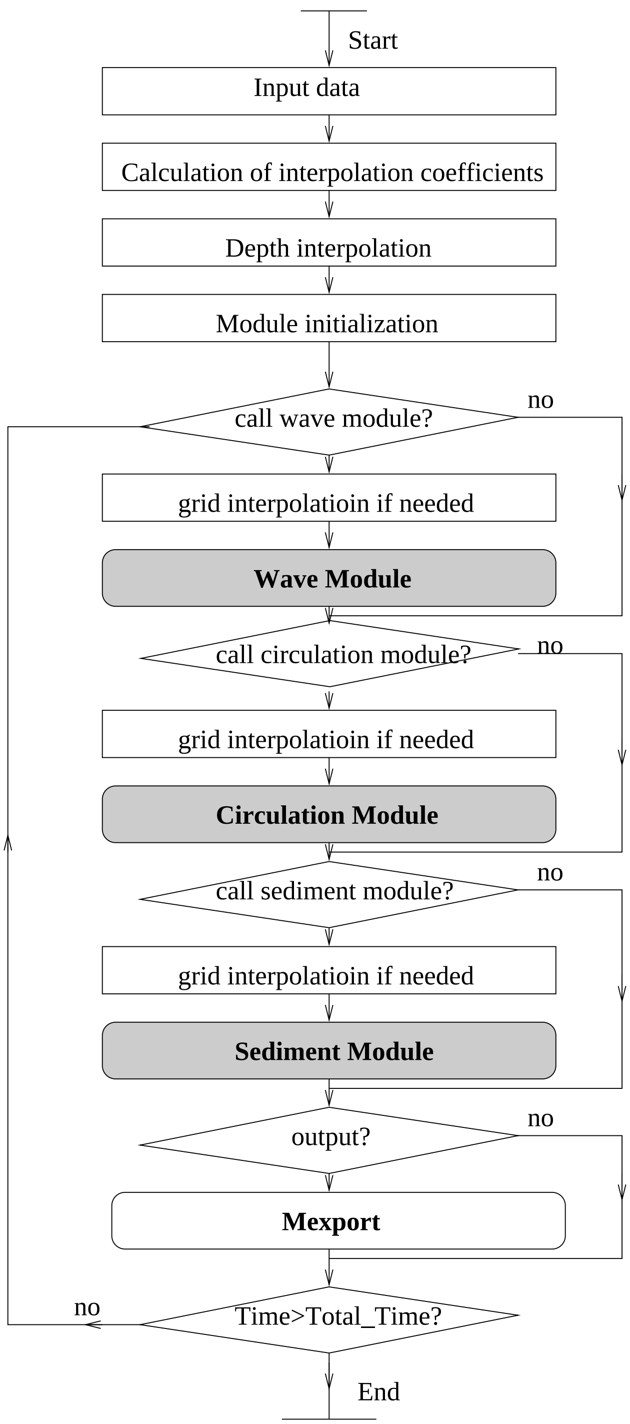
The flow chart of the master program is shown in Figure 1.

1.3 Parameters and Pass Variables Between Modules

The parameters and pass variables are included in 'pass.h'.

The definitions of the parameters and pass variables are described as following.

1. Model Dimension
   (a) Nx_Max and Ny_Max: maximum grid numbers for master grid and all three modules
   (b) Nx_Mast and Ny_Mast: grid dimension of master grid
   (c) Nx_Wave and Ny_Wave: grid dimension of wave grid
   (d) Nx_Circ and Ny_Circ: grid dimension of circulation grid
   (e) Nx_Sedi and Ny_Sedi: grid dimension of sediment grid
2. Wave Module
   (a) Pass_Sxx, Pass_Sxy, and Pass_Syy: radiation stresses
   (b) Pass_Sxx_body, Pass_Sxy_body, Pass_Syy_body: local radiation stresses (body part)



h

Figure 1: Flow Chart

(c) Pass_Sxx_surf, Pass_Sxy_surf, Pass_Syy_surf: local radiation stresses (surface part)

(d) Pass_Wave_Fx, Pass_Wave_Fy: wave forcing

(e) Pass_MassFluxU,Pass_MassFluxV: mass flux

(f) Pass_Diss: dissipation caused by wave breaking

(g) Pass_WaveNum: wave number

(h) Pass_Theta: wave angle

(i) Pass_ubott: bottom velocity

(j) Pass_Height: wave height

(k) Pass_C: phase velocity

(l) Pass_Cg: group velocity

(m) Pass_Period: wave period

(n) Intp_U_Wave and Intp_V_Wave: current velocity interpolated from the circulation module.

(o) Intp_eta_Wave: surface elevation interpolated from the circulation module.

(p) Pass_ibrk: wave breaking index (1 - breaking, 0 - nonbreaking)

3. Circulation Module

(a) Pass_U and Pass_V: depth-averaged velocity

(b) Pass_Ub and Pass_Vb: bottom current velocity

(c) Pass_eta: surface elevation (time-averaged)

(d) Pass_d11, Pass_d12, Pass_e11, Pass_e12, Pass_f11 and Pass_f12: coefficients for calculation of vertical velocity profile (SHORECIRC)

(e) Pass_fw: bottom friction coefficient

(f) Intp_Fx_Circ and Intp_Fy_Circ: short wave forcing interpolated from the wave module

(g) Intp_ubott_Circ: bottom velocity interpolated from the wave module

(h) Intp_Theta_Circ: wave angle interpolated from the wave module

(i) Intp_Sxx_Circ, Intp_Sxy_Circ, and Intp_Syy_Circ: radiation stresses interpolated from the wave module

(j) Intp_Sxx_Surf, Intp_Sxy_Surf, and Intp_Syy_Surf: local radiation stresses (surface part) interpolated from the wave module

(k) Intp_Sxx_Body, Intp_Sxy_Body, and Intp_Syy_Body: local radiation stresses (body part) interpolated from the wave module

(l) Intp_Height_Circ: wave height interpolated from the wave module

(m) Intp_Theta_Circ: wave angle interpolated from the wave module

(n) Intp_WaveNum_Circ: wave number interpolated from the wave module

(o) Intp_C_Circ: wave phase velocity interpolated from the wave module

(p) Intp_MassFluxU_Circ and Intp_MassFluxU_Circ: mass flux interpolated from the wave module

(q) Intp_Diss_Circ: energy dissipation interpolated from the wave module

(r) Intp_ibrk_Circ: wave breaking index interpolated from the wave module

4. Sediment Module

   (a) Pass_Dupdated: updated water depth

   (b) Intp_U_Sedi and Intp_V_Sedi: current velocity (depth averaged) interpolated from the circulation module.

   (c) Intp_Ub_Sedi and Intp_Vb_Sedi: bottom current velocity interpolated from the circulation module.

   (d) Intp_ubott_Sedi: bottom wave velocity interpolated from the wave module

   (e) Intp_eta_Sedi: surface elevation interpolated from the circulation module

   (f) Intp_fw_Sedi: bottom friction coefficient interpolated from the circulation module

   (g) Intp_Theta_Sedi: wave angle interpolated from the wave module

   (h) Intp_Height_Sedi: wave Height interpolated from the wave module

   (i) Intp_ibrk_Sedi: wave breaking index interpolated from the wave module

5. Coordinate System, Water Depth

   (a) Depth_Mast: water depth on Mast_Grid

   (b) Depth_Wave: water depth on Wave_Grid

   (c) Depth_Circ: water depth on Circ_Grid

   (d) Depth_Sedi: water depth on Sedi_Grid

   (e) X_Mast and Y_Mast: (x,y) of Mast_Grid

   (f) X_Wave and Y_Wave: (x,y) of Wave_Grid

   (g) X_Circ and Y_Circ: (x,y) of Circ_Grid

   (h) X_Sedi and Y_Sedi: (x,y) of Sedi_Grid

6. Other Variables

   (a) U_wind_Mast and V_wind_Mast: wind speed on Mast_Grid

   (b) U_wind_Circ and V_wind_Circ: wind speed on Circ_Grid

   (c) U_wind_Wave and V_wind_Wave: wind speed on Wave_Grid

7. Control Parameters

   (a) N_Interval_CallWave: step interval for calling wave module. If N_Interval_CallWave = -1, the model will never be called, and if N_Interval_CallWave = 0 for only initialization.
   (b) N_Interval_CallCirc: step interval for calling circulation module. If N_Interval_CallCirc = -1, the model will never be called, and if N_Interval_CallCirc = 0 for only initialization.
   (c) N_Interval_CallSedi: step interval for calling sediment module. If N_Interval_CallSedi = -1, the model will never be called, and if N_Interval_CallSedi = 0 for only initialization.
   (d) N_Delay_CallSedi: time (steps) delay for calling sediment module.
   (e) N_Interval_Output: step interval for output
   (f) Total_Time: total simulation time
   (g) Master_dt: time step in the master program
   (h) Grid_Mast_Wave_Same: logical parameter indicating whether or not Mast_Grid and Wave_Grid are the same grid system. If the input file name is a null string, the master program will set the Wave_Grid as the same as the Mast_Grid.
   (i) Grid_Mast_Circ_Same: logical parameter indicating whether or not Mast_Grid and Circ_Grid are the same grid system. If the input file name is a null string, the master program will set the Circ_Grid as the same as the Mast_Grid.
   (j) Grid_Mast_Sedi_Same: logical parameter indicating whether or not Mast_Grid and Sedi_Grid are the same grid system. If the input file name is a null string, the master program will set the Sedi_Grid as the same as the Mast_Grid.
   (k) Grid_Wave_Circ_Same, Grid_Wave_Sedi_Same, and Grid_Circ_Sedi_Same: logical parameters indicating relations between Wave_Grid and Circ_Grid, Wave_Grid and Sedi_Grid, and Circ_Grid and Sedi_Grid. Usually, if two grids or more than two grids are the same grid system, the Mast_Grid will be the same grid as the grid system. Grid_Wave_Circ_Same, Grid_Wave_Sedi_Same, and Grid_Circ_Sedi_Same will be judged out based on Grid_Mast_Wave_Same, Grid_Mast_Circ_Same, and Grid_Mast_Sedi_Same within the master program.
   (l) Wave_Staggered, Circ_Staggered, and Sedi_Staggered: logical parameters indicating whether or not grid systems are staggered.
   (m) Wave_Structured, Circ_Structured, and Sedi_Structured: logical parameters indicating whether or not grid system is structured.

(n) Grid_Extrapolation: logical parameter indicating whether or not value extrapolation is allowed between two grid systems. If Grid_Extrapolation = .false., the value which is supposed to be extrapolated will equal to the value at the nearest grid point.

(o) Wave_Curr_Interact, Wave_Bed_Interact, Curr_Bed_Interact: logical parameters indicating whether or not two-way coupling is needed between modules.

8. Control Parameters for Passing variables

(a) Wave_To_Circ_Height: pass wave heights from the wave module to the circulation module

(b) Wave_To_Circ_Angle: pass wave angles (degree) from the wave module to the circulation module

(c) Wave_To_Circ_WaveNum: pass wave numbers from the wave module to the circulation module

(d) Wave_To_Circ_C: pass wave phase velocities from the wave module to the circulation module

(e) Wave_To_Circ_Radiation: pass radiation stresses from the wave module to the circulation module

(f) Wave_To_Circ_Rad_Surf: pass local radiation stresses (surface part) from the wave module to the circulation module

(g) Wave_To_Circ_Rad_Body:pass local radiation stresses (body part) from the wave module to the circulation module

(h) Wave_To_Circ_Forcing: pass short wave forcing from the wave module to the circulation module

(i) Wave_To_Circ_MassFlux: pass mass flux from the wave module to the circulation module

(j) Wave_To_Circ_Dissipation: pass wave energy dissipation from the wave module to the circulation module

(k) Wave_To_Circ_BottomUV: pass Bottom velocity from the wave module to the circulation module

(l) Wave_To_Circ_Brkindex: pass wave breaking index from the wave module to the circulation module

(m) Circ_To_Wave_UV: pass current velocity (depth averaged) from the circulation module to the wave module

(n) Circ_To_Wave_eta: pass surface elevation from the circulation module to the wave module

(o) Wave_To_Sedi_Height: pass wave height from the wave module to the sediment module

(p) Wave_To_Sedi_Angle: pass wave angle from the wave module to the sediment module

(q) Wave_To_Sedi_BottomUV: pass bottom velocity from the wave module to the sediment module

(r) Circ_To_Sedi_UV: pass current velocity from the circulation module to the sediment module

(s) Circ_To_Sedi_UVb: pass bottom current velocity from the circulation module to the sediment module

(t) Circ_To_Sedi_eta: pass surface elevation from the circulation module to the sediment module

(u) Circ_To_Sedi_UV3D: pass 3D current velocity from the circulation module to the sediment module

(v) Circ_To_Sedi_fw: pass bottom friction coefficient from the circulation module to the sediment module

(w) Circ_To_Sedi_UVquasi3D: pass coefficients of quasi-3D current profiles from the circulation module to the sediment module

(x) Sedi_To_Wave_Depth: pass updated water depth from the sediment module to the wave module

(y) Sedi_To_Circ_Depth: pass updated water depth from the sediment module to the circulation module

9. Vector Rotation

   (a) Circ_Rotate_Angle: angle (°) between the vector reference direction on Circ_Grid and the geographic reference direction

   (b) Wave_Rotate_Angle: angle (°) between the vector reference direction on Wave_Grid and the geographic reference direction

   (c) Sedi_Rotate_Angle: angle (°) between the vector reference direction on Sedi_Grid and the geographic reference direction

1.4 Subroutines in Master Program

1. *readfile* reads in file names, grid dimensions, and model control parameters provided by 'minput.dat'. It also reads in water depth and (x,y) of grid points from file names given in 'minput.dat'. *readfile* is called by *master*.

2. *get_interpolation_coef* computes interpolation coefficients and save them in arrays in 'interp.h'. *get_interpolation_coef* is called by *master*. It calls *interpsame* and *interpolation*.

3. *interp_depth* interpolates water depth from Master-Grid to Wave-Grid, Circ-Grid, and Sedi-Grid. *interp_depth* is called by *master*. *interp_depth* calls *grid1_to_grid2*.

4. *interp_circ_wave* interpolates variables from Circ-Grid to Wave-Grid. *interp_circ_wave* is called by *master*. and it calls *grid1_to_grid2*.

5. *interp_sedi_wave* interpolates variables from Sedi-Grid to Wave-Grid. *interp_sedi_wave* is called by *master* and calls *grid1_to_grid2*.

6. *interp_wave_circ* interpolates variables from Wave-Grid to Circ-Grid. *interp_wave_circ* is called by *master* and calls *grid1_to_grid2*.

7. *interp_sedi_circ* interpolates variables from Sedi-Grid to Circ-Grid. *interp_sedi_circ* is called by *master* and calls *grid1_to_grid2*.

8. *interp_wave_sedi* interpolates variables from Wave-Grid to Sedi-Grid. *interp_wave_sedi* is called by *master* and calls *grid1_to_grid2*.

9. *interp_circ_sedi* interpolates variables from Circ-Grid to Sedi-Grid. *interp_circ_sedi* is called by *master* and calls *grid1_to_grid2*

10. *interpsame* calculates interpolation coefficients when two module grids are same. *interpsame* is called by *get_interpolation_coef*.

11. *interpolation* is used to get coefficients of interpolation or extrapolation between two grid systems. *interpolation* is called by *get_interpolation_coef*.

12. *grid1_to_grid2* makes interpolation/extrapolation from grid1 to grid2 based on interpolation coefficients.

    grid1_to_grid2 is called by *interp_depth*, *interp_circ_wave*, *interp_circ_wave*, *interp_sedi_wave*, *interp_wave_circ*, *interp_sedi_circ*, *interp_wave_sedi*, and *interp_circ_sedi*.

13. *SediModule* is the Sediment module. *SediModule* is called by *Master*.

14. *WaveModule* is the Wave module. *WaveModule* is called by *master*

15. *CircModule* is the Circulation module. *CircModule* is called by *master*

16. *Mexport* is for model output. *Mexport* is called by *Master*.

17. *MasterInit* initializes all pass variables. *MasterInit* is called by *master*.

1.5 Interpolation/Extrapolation between model grids

Interpolation or extrapolation is usually employed between a curvilinear grid and a rectangular grid. The program used here can also handle interpolation/extrapolation between two different curvilinear grids or two different rectangular grids.

A linear interpolation/extrapolation method is used in the program. We assume two grid systems, grid-1 and grid-2, which can be curvilinear grids or rectangular grids. As shown in Figure 2, the interpolation value at point A in grid-1 is evaluated by the values at three points, $1, 2$ and 3, of a triangle in grid-2 which surrounds point A. For extrapolation, point A is out of the triangle. Four triangle areas $S_{\alpha\beta\gamma}$, i.e., $S_{123}, S_{12A}, S_{31A}$ and S_{23A} are calculated using the following formula:

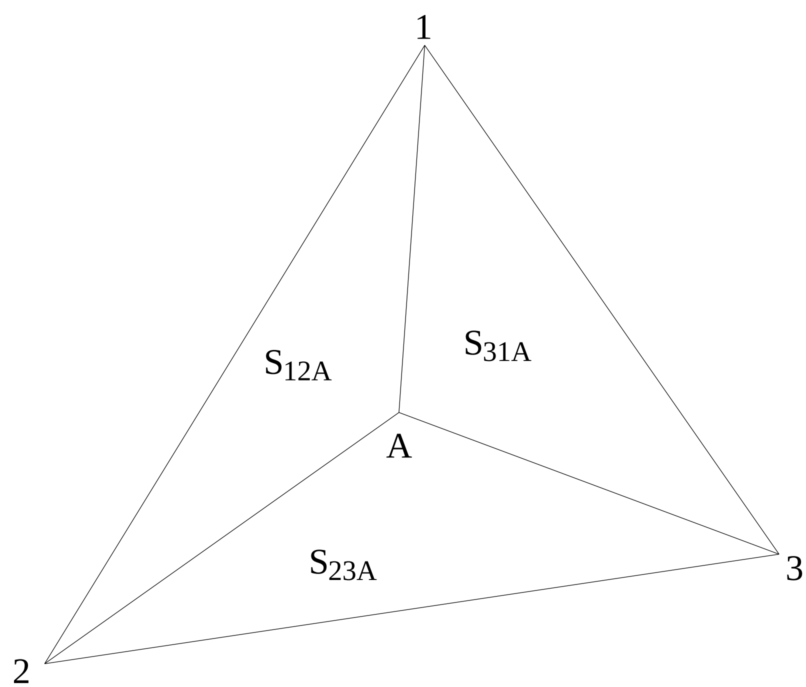


Figure 2: Interpolation triangle.

$$S_{\alpha\beta\gamma} = \begin{vmatrix} x_\alpha & y_\alpha & 1 \\ x_\beta & y_\beta & 1 \\ x_\gamma & y_\gamma & 1 \end{vmatrix} \tag{1}$$

where (x_α, y_α) represents coordinates of point $1, 2, 3$ and A. For interpolation, (α, β, γ) are counterclock wise for all the four triangles and thus $S_{\alpha\beta\gamma}$ are positive. For extrapolation, clock wise (α, β, γ) results in negative $S_{\alpha\beta\gamma}$. The following formula is used for both interpolation and extrapolation:

$$F_A = (F_1 S_{23A} + F_2 S_{31A} + F_3 S_{12A})/S_{123} \tag{2}$$

where F_1, F_2, F_3 and F_A represent any converted variables at point $1, 2, 3$ and A, respectively.

It should be mentioned that the extrapolation is usually used at domain boundaries and is only suitable for the case that domain boundaries are close to each other. To save computational time for interpolation/extrapolation, $S_{\alpha\beta\gamma}$ are stored when subroutine *get_interpolation_coef* is called.

1.6 Input for Master Program

1. F_NAMES: definition of input and output file names.
   (a) f_depth - input file name for depth on Mast_Grid
   (b) f_xymast - input file name for (x,y) of Mast_Grid
   (c) f_xywave - input file name for (x,y) of Wave_Grid
   (d) f_xycirc - input file name for (x,y) of Circ_Grid
   (e) f_xysedi - input file name for (x,y) of Sedi_Grid
   (f) ...
2. GRIDIN: grid information.
   (a) Nx_Mast, Ny_Mast: Mast_Grid dimension
   (b) Nx_Wave, Ny_Wave: Wave_Grid dimension
   (c) Nx_Circ, Ny_Circ: Circ_Grid dimension
   (d) Nx_Sedi, Ny_Sedi: Sedi_Grid dimension
   (e) Grid_Mast_Wave_Same - logical parameter indicating whether Wave_Grid is the same as Mast_Grid
   (f) Grid_Mast_Circ_Same - logical parameter indicating whether Circ_Grid is the same as Mast_Grid
   (g) Grid_Mast_Sedi_Same - logical parameter indicating whether Sedi_Grid is the same as Mast_Grid
   (h) Wave_Staggered - logical parameter indicating whether Wave_Grid is staggered

   (i) Circ_Staggered - logical parameter indicating whether Circ_Grid is staggered
   (j) Sedi_Staggered - logical parameter indicating whether Sedi_Grid is staggered
   (k) Wave_Structured - logical parameter indicating whether or not Wave_Grid is structured.
   (l) Circ_Structured - logical parameter indicating whether or not Circ_Grid is structured.
   (m) Sedi_Structured - logical parameters indicating whether or not Sedi_Grid is structured
   (n) Grid_Extrapolation - logical parameter indicating whether or not value extrapolation is allowed between two grid systems.

3. INTERACTION: control parameters for one-way or two-way coupling between modules
   (a) Wave_Curr_Interact - logical parameter for two-way coupling between the wave and circulation modules
   (b) Wave_Bed_Interact - logical parameter for two-way coupling between the wave and sediment modules
   (c) Curr_Bed_Interact - logical parameter for two-way coupling between the circulation and sediment modules

4. PASSVARIABLES
   (a) Wave_To_Circ_Height - pass wave height from the wave module to the circulation module
   (b) Wave_To_Circ_Angle - pass wave angle from the wave module to the circulation module
   (c) Wave_To_Circ_WaveNum - pass wave numbers from the wave module to the circulation module
   (d) Wave_To_Circ_Radiation - pass wave phase velocity from the wave module to the circulation module
   (e) Wave_To_Circ_Radiation - pass radiation stresses from the wave module to the circulation module
   (f) Wave_To_Circ_Rad_Surf - pass local radiation stresses (surface part) from the wave module to the circulation module
   (g) Wave_To_Circ_Rad_Body -pass local radiation stresses (body part) from the wave module to the circulation module
   (h) Wave_To_Circ_Forcing - pass short wave forcing from the wave module to the circulation module
   (i) Wave_To_Circ_MassFlux - pass mass flux from the wave module to the circulation module

(j) Wave_To_Circ_Dissipation - pass wave energy dissipation from the wave module to the circulation module

(k) Wave_To_Circ_BottomUV - pass Bottom velocity from the wave module to the circulation module

(l) Wave_To_Circ_Brkindex - pass wave breaking index from the wave module to the circulation module

(m) Circ_To_Wave_UV - pass current velocity (depth averaged) from the circulation module to the wave module

(n) Circ_To_Wave_eta - pass surface elevation from the circulation module to the wave module

(o) Wave_To_Sedi_Height - pass wave height from the wave module to the sediment module

(p) Wave_To_Sedi_Angle - pass wave angle from the wave module to the sediment module

(q) Wave_To_Sedi_BottomUV - pass bottom velocity from the wave module to the sediment module

(r) Circ_To_Sedi_UV - pass current velocity from the circulation module to the sediment module

(s) Circ_To_Sedi_UVb - pass bottom current velocity from the circulation module to the sediment module

(t) Circ_To_Sedi_eta - pass surface elevation from the circulation module to the sediment module

(u) Circ_To_Sedi_UV3D - pass 3D current velocity from the circulation module to the sediment module

(v) Circ_To_Sedi_fw - pass bottom friction coefficient from the circulation module to the sediment module

(w) Circ_To_Sedi_UVquasi3D - pass coefficients of quasi-3D current profiles from the circulation module to the sediment module

(x) Sedi_To_Wave_Depth - pass updated water depth from the sediment module to the wave module

(y) Sedi_To_Circ_Depth - pass updated water depth from the sediment module to the circulation module

5. VECTORROTATE

(a) Circ_Rotate_Angle - angle (°) between the vector reference direction on Circ_Grid and the geographic reference direction

(b) Wave_Rotate_Angle - angle (°) between the vector reference direction on Wave_Grid and the geographic reference direction

(c) Sedi_Rotate_Angle - angle (°) between the vector reference direction on Sedi_Grid and the geographic reference direction

6. TIMEIN: time stepping control.
   (a) Total_Time - total computational time
   (b) Master_dt - time step in master program.
   (c) N_Interval_CallWave - interval steps for calling the wave module. N_Interval_CallWave=0 represents "never call the wave module" and N_Interval_CallWave¡0 for single initial call
   (d) N_Interval_CallCirc - interval steps for calling the circulation module. N_Interval_CallCirc=0 represents "never call the circulation module" and N_Interval_CallCirc¡0 for single initial call
   (e) N_Interval_CallSedi - interval steps for calling the sediment module N_Interval_CallSedi=0 represents "never call the sediment module" and N_Interval_CallSedi¡0 for single initial call
   (f) N_Delay_CallSedi - time (steps) delay for calling the sediment module
   (g) N_Interval_Output - interval steps for output

2 Link Master Program to Three Modules

2.1 Necessary Modifications for Three Modules

1. change the main program of each module into a subroutine
   (a) wave module - subroutine WaveModule()
   (b) circulation module - subroutine CircModule()
   (c) sediment module - subroutine SediModule()
2. include 'pass.h' in necessary subroutines.
3. split each module code into initialization part and time integration part if necessary.See the example blew.

   For example:

   ⟨*example*⟩≡

```
        subroutine CircModule()
          include 'pass.h'
          include 'arrays.h'

! ---  initialize module
          if (Master_Start.eq.1)then
            call init_shorecirc
          else

           ntime = Master_dt/Circ_dt
```

```
! time integration

          do 100 itstep = 1, ntime

            call predict_stage
            ...

  100      continue

         endif

           end
```

4. specify parameters in 'minput.dat'

5. assign module variables to pass variables in your modules. For example, if you want to pass depth-averaged current velocity (U, V) from your circulation module, specify

   Pass_U(i,j) = U(i,j)

   Pass_V(i,j) = V(i,j)

6. use variables passed and interpolated from other modules. For example, if you want to use depth-averaged current velocity in your sediment module, set

   Used(i,j) = Intp_U_Sedi(i,j)

   Vsed(i,j) = Intp_V_Sedi(i,j)

   where (Used, Vsed) represnets current velocity vector used in your sediment module.

2.2 Units

The International System of Units (SI) is used for all variables defined in pass.h. If a particular module was developed in other unit systems rather than the SI system, unit conversions are needed when variables pass between the master program and the module. The unit conversions should be implemented in the particular module.

2.3 Coordinate systems

The grid point locations for all computational grids, including the master grid, are given by the Cartesian (x,y) (meters) in geographical reference coordinates. Figure 3 shows an example of two computational grid systems in which Grid1 is represented by the curvilinear solid lines and Grid2 by the dashed lines. All grid point locations in both Grid1 and Grid2 are given by (x,y) values wherever the first grid point (i=1,j=1) is defined on each grid. The directions of vector variables in each module could be defined based on its own reference direction such as (u_1, v_1) on grid1 or (u_2, v_2) on grid2 as shown in Figure 3. But users should specify, in "minput.dat", the angles between the module reference direction and the geographical x - direction, e.g., θ_1 and θ_2 in Figure 3. For example, in "minput.dat", $Circ\_Rotate\_Angle = 10.$, which means the vector reference direction in the circulation module rotates 10° (counterclockwise) from the geographical reference direction. It should be mentioned that the vector reference direction in each module should be fixed and should not vary spatially even in a curvilinear coordinate system. They should neither be normal or tangential directions of curvilinear coordinate lines, nor be covariant or contravariant base directions. The vectors based on normal/tangential directions or covariant/contravariant directions should be converted before they are transfered into other grid systems.

The interpolation/extrapolation can be carried out between non-staggered and staggered grids, or staggered grid systems with different Arakawa types. The logical paramters " xxxx_Staggered" are used to represent if a system is a staggered grid system or not. The integer parameters such as "xxxx_Stag_huv" are used to represent Arakawa grid types. xxxx_Stag_huv is an integer array with three elements. The first element gives the location of water depth h or surface elevation ζ. The second and the third elements give the locations of velocity components u and v, respectively. For a grid element, we use '0', '1', '2', and '3' to represent the positions where variables are calculated as shown in Figure 4 (a). Figure 4 (b) - (d) show the examples of Arakawa grid definitions. For Arakawa-A, xxxx_Stag_huv = (0 0 0); Arakawa-B:xxxx_Stag_huv = (0 1 1); and Arakawa-C:xxxx_Stag_huv = (1 2 3). You could not use the standard Arakawa definition for variable passing. For example, when you use the POM module (Arakawa - C) in a curvilinear coordinate system, current velocity vector may be first interpolated into the grid nodes (location "0") and then passed into another module. The pre-interpolated veocity components are actually located at "0" position rather than at "2" and "3" positions. In this case, you may specify Circ_Stag_huv = (1 0 0).

Usually, only water depth, surface elevation and vectors such as current velocity, short wave volume flux and short wave forcing are passed and interpolated between staggered grid systems. For other passing variables such as wave height, wave angle, wave phase velocity, wave breaking index, radiation stresses, and etc., interpolation/extrapolation are carried out at grid nodes (location "0").

2.4 Unstructured Grid

For unstructured grids, 2-D arrays are also used for coordinate information and pass variables, as the same as in structured grid systems. If only 1-D arrays are defined in the module with a unstructured grid, Ny = 1 should be used in pass variables.

3 *noweb* Documentation of the Master Program

3.1 Main Program

The program calls three modules: WaveModule(), CircModule() and SediModule(). It also includes data input, calculation of interpolation/extrapolation coefficients, Pass_variables initialization, module initializations, and data output.

Master calls the following subroutines

1. *readfile*
2. *get_interpolation_coef*
3. *interp_depth*

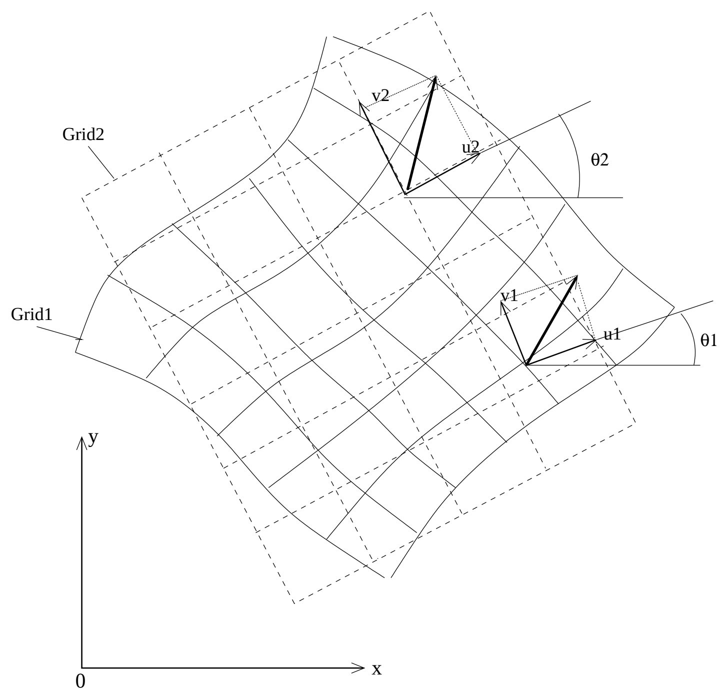


Figure 3: Example of module grids in geographical reference coordinates.

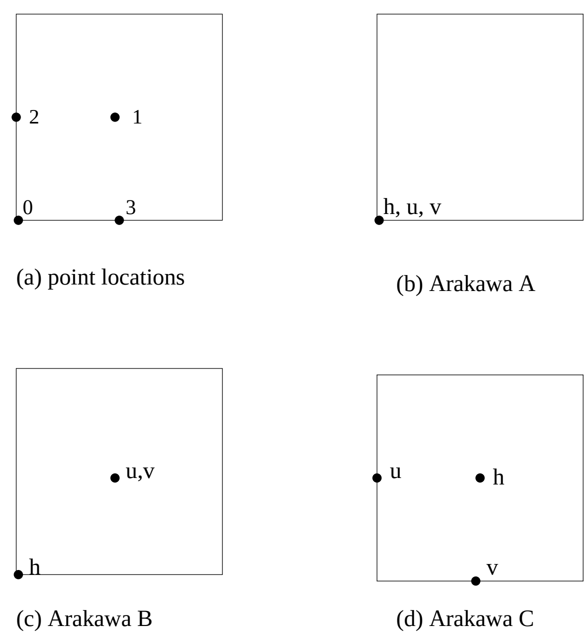


Figure 4: Staggered grid definition.

4. *interp_circ_wave*
5. *interp_sedi_wave*
6. *interp_wave_circ*
7. *interp_sedi_circ*
8. *interp_wave_sedi*
9. *interp_circ_sedi*
10. *MasterInit()*
11. *WaveModule()*
12. *CircModule()*
13. *SediModule()*
14. *Mexport()*

⟨*⟩≡
```
c -------------------------------------------------------------------
        program master
        implicit none
        include 'pass.h'
        include 'interp.h'

        integer master_steps
        real Time_Circ, Time_Wave, Time_Sedi, Time_Output
        data Time_Circ /0./, Time_Wave /0./, Time_Sedi /0./,
     &       Time_Output /0./

        Waveupdat_for_Circ = .false.
        Waveupdat_for_Sedi = .false.
        Circupdat_for_Wave = .false.
        Circupdat_for_Sedi = .false.
        Sediupdat_for_Wave = .false.
        Sediupdat_for_Circ = .false.

c --- read file

        call readfile

c --- calculate interpolation coefficients

        call get_interpolation_coef
```

```
c --- depth interplation/extrapolation

        call interp_depth

c ---   module Initialization and first call
c       the first call is for hot start

         Master_Start = 1

         call MasterInit()

c ---    wave
         if(N_Interval_CallWave.ge.0)call WaveModule()
         Master_Start=-1
         if(N_Interval_CallWave.ge.0)call WaveModule()

c ---    circulation
         Master_Start = 1
         call interp_wave_circ
         if(N_Interval_CallCirc.ge.0)call CircModule()
         Master_Start=-1
         if(N_Interval_CallCirc.ge.0)call CircModule()

c ---    sediment
         Master_Start = 1
         call interp_wave_sedi
         call interp_circ_sedi
         if(N_Interval_CallSedi.ge.0)call SediModule()
         Master_Start=-1
         if(N_Interval_CallSedi.ge.0)call SediModule()

c ---   get Master_steps and ...

        Master_steps = Total_Time / Master_dt

        if(N_Interval_CallCirc.gt.0)then
          nCirc = N_Interval_CallCirc
        else
          nCirc= Master_steps+1
        endif

        if(N_Interval_CallWave.gt.0)then
          nWave = N_Interval_CallWave
        else
          nWave= Master_steps+1
        endif
```

```
        if(N_Interval_CallSedi.gt.0)then
          nSedi = N_Interval_CallSedi
        else
          nSedi= Master_steps+1
        endif

        nOut = N_Interval_Output

c ---   Do timestepping

        do 100 istep = 1, Master_steps

        Time_Master= (istep-1)*Master_dt

        write(*,*) 'Time = ',Time_Master, 's'

c --- call wave module

         if (mod(istep,nWave).eq.0) then

            if(Circupdat_for_Wave)then
               call interp_circ_wave
            endif

            if(Sediupdat_for_Wave)then
               call interp_sedi_wave
            endif

            call WaveModule()

            Circupdat_for_Wave = .false.
            Sediupdat_for_Wave =.false.
            Waveupdat_for_Circ = .true.
            Waveupdat_for_Sedi = .true.

         end if

c ---  call circulation module

         if (mod(istep,nCirc).eq.0) then

         if(Waveupdat_for_Circ) then
            call interp_wave_circ
         endif
```

```
          if(Sediupdat_for_Circ) then
             call interp_sedi_circ
          endif

            call CircModule()

            Waveupdat_for_Circ = .false.
            Sediupdat_for_Circ = .false.
            Circupdat_for_Wave = .true.
            Circupdat_for_Sedi = .true.

          end if

c ---  call sediment module

          if (mod(istep,nSedi).eq.0.and.istep.ge.N_Delay_CallSedi) then

          if(Waveupdat_for_Sedi) then
             call interp_wave_sedi
          endif

          if(Circupdat_for_Sedi)then
             call interp_circ_Sedi
          endif

          call SediModule()

          Waveupdat_for_Sedi = .false.
          Circupdat_for_Sedi = .false.
          Sediupdat_for_Wave = .true.
          Sediupdat_for_Circ = .true.

          end if

c --- output

          if (mod(istep,nOut).eq.0) then
            call Mexport()
          end if

100       continue
c   ---   program end

         print*,'Program end'

        end
```

3.2 Subroutine readfile

This subroutine reads in file names, grid dimensions, and model control parameters provided by 'minput.dat'. It also reads in water depth and (x,y) at grid points, from file names given in 'minput.dat'.

readfile is called by

1. *master*

⟨*⟩+≡
```
c -----------------------------------------------------------------
        subroutine readfile
        implicit none
        include 'pass.h'

        namelist /f_names/ f_depth,f_xymast,f_xywave,f_xycirc,
     &                 f_xysedi,f_name6,
     &                 f_name7,
     &                 f_name8,f_name9,f_name10,f_name11,f_name12,
     &                 f_name13,f_name14,
     &                 f_name15,f_name16

     &          /gridin/ Nx_Mast, Ny_Mast, Nx_Circ,
     &                  Ny_Circ, Nx_Wave, Ny_Wave, Nx_Sedi, Ny_Sedi,
     &                  Grid_Mast_Wave_Same, Grid_Mast_Circ_Same,
     &                  Grid_Mast_Sedi_Same,
     &                  Wave_Staggered, Circ_Staggered,Sedi_Staggered,
     &                  Wave_Stag_huv,  Circ_Stag_huv, Sedi_Stag_huv,
     &                  Wave_structured, Circ_Structured,
     &                  Sedi_Structured,
     &                  Grid_Extrapolation

     &          /interaction/
     &                  Wave_Curr_Interact,
     &                  Wave_Bed_Interact,
     &                  Curr_Bed_Interact

     &          /passvariables/
     &              Wave_To_Circ_Height,
     &              Wave_To_Circ_Angle,
     &              Wave_To_Circ_WaveNum,
     &              Wave_To_Circ_C,
     &              Wave_To_Circ_Radiation,
     &              Wave_To_Circ_Rad_Surf,
     &              Wave_To_Circ_Rad_Body,
     &              Wave_To_Circ_Forcing,
     &              Wave_To_Circ_MassFlux,
```

```
     &              Wave_To_Circ_Dissipation,
     &              Wave_To_Circ_BottomUV,
     &              Wave_To_Circ_Brkindex,
     &              Circ_To_Wave_UV,
     &              Circ_To_Wave_eta,
     &              Wave_To_Sedi_Height,
     &              Wave_To_Sedi_Angle,
     &              Wave_To_Sedi_BottomUV,
     &              Circ_To_Sedi_UV,
     &              Circ_To_Sedi_UVb,
     &              Circ_To_Sedi_eta,
     &              Circ_To_Sedi_UV3D,
     &              Circ_To_Sedi_fw,
     &              Circ_To_Sedi_UVquasi3D,
     &              Sedi_To_Wave_Depth,
     &              Sedi_To_Circ_Depth

     &          /vectorrotate/
     &               Circ_Rotate_Angle, Wave_Rotate_Angle,
     &               Sedi_Rotate_Angle

     &          /timein/ Total_Time,Master_dt, N_Interval_CallWave,
     &                 N_Interval_CallCirc,N_Interval_CallSedi,
     &                 N_Delay_CallSedi,
     &                 N_Interval_Output


        open(1,file='minput.dat')

        read(1,nml=f_names)
        read(1,nml=gridin)
        read(1,nml=interaction)
        read(1,nml=passvariables)
        read(1,nml=vectorrotate)
        read(1,nml=timein)

        close(1)

c --- read initial depth

        open(1,file=f_depth)
        do j=1,Ny_Mast
          read(1,100)(Depth_Mast(i,j),i=1,Nx_Mast)
        enddo
        close(1)
```

```
c --- read xy of master grid

        open(1,file=f_xymast)
        do j=1,Ny_Mast
          read(1,100)(X_Mast(i,j),i=1,Nx_Mast)
        enddo
        do j=1,Ny_Mast
          read(1,100)(Y_Mast(i,j),i=1,Nx_Mast)
        enddo
        close(1)

c --- read xy of wave module

        if(f_xywave.ne.' ')then
        open(1,file=f_xywave)
        do j=1,Ny_Wave
          read(1,100)(X_Wave(i,j),i=1,Nx_Wave)
        enddo
        do j=1,Ny_Wave
          read(1,100)(Y_Wave(i,j),i=1,Nx_Wave)
        enddo
        close(1)
        else
        do j=1,Ny_Wave
        do i=1,Nx_Wave
          X_Wave(i,j)=X_Mast(i,j)
          Y_Wave(i,j)=Y_Mast(i,j)
        enddo
        enddo
        Grid_Mast_Wave_Same = .true.
        endif

c --- read xy of circulation module

        if(f_xycirc.ne.' ')then
        open(1,file=f_xycirc)
        do j=1,Ny_Circ
          read(1,100)(X_Circ(i,j),i=1,Nx_Circ)
        enddo
        do j=1,Ny_Circ
          read(1,100)(Y_Circ(i,j),i=1,Nx_Circ)
        enddo
        close(1)
        else
        do j=1,Ny_Circ
        do i=1,Nx_Circ
```

```
          X_Circ(i,j)=X_Mast(i,j)
          Y_Circ(i,j)=Y_Mast(i,j)
        enddo
        enddo
        Grid_Mast_Circ_Same = .true.
        endif

c --- read xy of sediment module

        if(f_xysedi.ne.' ')then
        open(1,file=f_xysedi)
        do j=1,Ny_Sedi
          read(1,100)(X_Sedi(i,j),i=1,Nx_Sedi)
        enddo
        do j=1,Ny_Sedi
          read(1,100)(Y_Sedi(i,j),i=1,Nx_Sedi)
        enddo
        close(1)
        else
        do j=1,Ny_Sedi
        do i=1,Nx_Sedi
          X_Sedi(i,j)=X_Mast(i,j)
          Y_Sedi(i,j)=Y_Mast(i,j)
        enddo
        enddo
        Grid_Mast_Sedi_Same = .true.
        endif

c --- grid relations between wave-circ, wave-sedi, and circ-sedi

        Grid_Wave_Circ_Same = .false.
        Grid_Wave_Sedi_Same = .false.
        Grid_Circ_Sedi_Same = .false.

        if(Grid_Mast_Wave_Same.and.Grid_Mast_Circ_Same)
     &     Grid_Wave_Circ_Same = .true.

        if(Grid_Mast_Wave_Same.and.Grid_Mast_Sedi_Same)
     &     Grid_Wave_Sedi_Same = .true.

        if(Grid_Mast_Circ_Same.and.Grid_Mast_Sedi_Same)
     &     Grid_Circ_Sedi_Same = .true.

100      format(800f16.8)

        return
```

```
    end
```

3.3 Subroutine get_interpolation_coef

The subroutine computes interpolation coefficients and save them in arrays in 'interp.h'.

get_interpolation_coef is called by

1. *master*

It calls

1. *interpsame*
2. *interpolation*

⟨*⟩+≡

```
c -------------------------------------------------------------

        subroutine get_interpolation_coef
        implicit none
        include 'pass.h'
        include 'interp.h'

c ---   Mast-Wave
        if(Grid_Mast_Wave_Same) then
          call interpsame(Nx_Wave,Ny_Wave,Sc_01,S1_01,S2_01,S3_01,
     .           nx1_01,ny1_01,nx2_01,ny2_01,nx3_01,ny3_01)
        else
         write(*,*)'Grid_Mast & Grid_Wave are different, calc coef...'
          call interpolation
     .    (Nx_Mast,Ny_Mast,X_Mast,Y_Mast,
     .     Nx_Wave,Ny_Wave,X_Wave,Y_Wave,Sc_01,S1_01,S2_01,S3_01,
     .     nx1_01,ny1_01,nx2_01,ny2_01,nx3_01,ny3_01)
        endif

c ---   Mast-Circ
        if(Grid_Mast_Circ_Same) then
          call interpsame(Nx_Circ,Ny_Circ,Sc_02,S1_02,S2_02,S3_02,
     .           nx1_02,ny1_02,nx2_02,ny2_02,nx3_02,ny3_02)
        else
         write(*,*)'Grid_Mast & Grid_Circ are different, calc coef...'

          call interpolation
     .    (Nx_Mast,Ny_Mast,X_Mast,Y_Mast,
     .     Nx_Circ,Ny_Circ,X_Circ,Y_Circ,Sc_02,S1_02,S2_02,S3_02,
     .     nx1_02,ny1_02,nx2_02,ny2_02,nx3_02,ny3_02)
        endif

c ---   Mast-Sedi
```

```
          if(Grid_Mast_Sedi_Same) then
            call interpsame(Nx_Sedi,Ny_Sedi,Sc_03,S1_03,S2_03,S3_03,
     .            nx1_03,ny1_03,nx2_03,ny2_03,nx3_03,ny3_03)
          else
           write(*,*)'Grid_Mast & Grid_Sedi are different, calc coef...'
            call interpolation
     .      (Nx_Mast,Ny_Mast,X_Mast,Y_Mast,
     .       Nx_Sedi,Ny_Sedi,X_Sedi,Y_Sedi,Sc_03,S1_03,S2_03,S3_03,
     .       nx1_03,ny1_03,nx2_03,ny2_03,nx3_03,ny3_03)
          endif

c ---     Circ-Wave
          if(Grid_Wave_Circ_Same) then
            call interpsame(Nx_Wave,Ny_Wave,Sc_21,S1_21,S2_21,S3_21,
     .            nx1_21,ny1_21,nx2_21,ny2_21,nx3_21,ny3_21)
          else
           write(*,*)'Grid_Wave & Grid_Circ are different, calc coef...'
            call interpolation
     .      (Nx_Circ,Ny_Circ,X_Circ,Y_Circ,
     .       Nx_Wave,Ny_Wave,X_Wave,Y_Wave,Sc_21,S1_21,S2_21,S3_21,
     .       nx1_21,ny1_21,nx2_21,ny2_21,nx3_21,ny3_21)
          endif

c ---     Sedi-Wave
          if(Grid_Wave_Sedi_Same) then
            call interpsame(Nx_Wave,Ny_Wave,Sc_31,S1_31,S2_31,S3_31,
     .            nx1_31,ny1_31,nx2_31,ny2_31,nx3_31,ny3_31)
          else
           write(*,*)'Grid_Wave & Grid_Sedi are different, calc coef...'
            call interpolation
     .      (Nx_Sedi,Ny_Sedi,X_Sedi,Y_Sedi,
     .       Nx_Wave,Ny_Wave,X_Wave,Y_Wave,Sc_31,S1_31,S2_31,S3_31,
     .       nx1_31,ny1_31,nx2_31,ny2_31,nx3_31,ny3_31)
          endif

c ---     Wave-Circ
          if(Grid_Wave_Circ_Same) then
            call interpsame(Nx_Circ,Ny_Circ,Sc_12,S1_12,S2_12,S3_12,
     .            nx1_12,ny1_12,nx2_12,ny2_12,nx3_12,ny3_12)
          else
           write(*,*)'Grid_Wave & Grid_Circ are different, calc coef...'
            call interpolation
     .      (Nx_Wave,Ny_Wave,X_Wave,Y_Wave,
     .       Nx_Circ,Ny_Circ,X_Circ,Y_Circ,Sc_12,S1_12,S2_12,S3_12,
     .       nx1_12,ny1_12,nx2_12,ny2_12,nx3_12,ny3_12)
          endif
```

```
c ---   Sedi-Circ
        if(Grid_Circ_Sedi_Same) then
          call interpsame(Nx_Circ,Ny_Circ,Sc_32,S1_32,S2_32,S3_32,
     .          nx1_32,ny1_32,nx2_32,ny2_32,nx3_32,ny3_32)
        else
         write(*,*)'Grid_Circ & Grid_Sedi are different, calc coef...'
          call interpolation
     .    (Nx_Sedi,Ny_Sedi,X_Sedi,Y_Sedi,
     .     Nx_Circ,Ny_Circ,X_Circ,Y_Circ,Sc_32,S1_32,S2_32,S3_32,
     .     nx1_32,ny1_32,nx2_32,ny2_32,nx3_32,ny3_32)
        endif

c ---   Wave-Sedi
        if(Grid_Wave_Sedi_Same) then
          call interpsame(Nx_Sedi,Ny_Sedi,Sc_13,S1_13,S2_13,S3_13,
     .          nx1_13,ny1_13,nx2_13,ny2_13,nx3_13,ny3_13)
        else
         write(*,*)'Grid_Wave & Grid_Sedi are different, calc coef...'
          call interpolation
     .    (Nx_Wave,Ny_Wave,X_Wave,Y_Wave,
     .     Nx_Sedi,Ny_Sedi,X_Sedi,Y_Sedi,Sc_13,S1_13,S2_13,S3_13,
     .     nx1_13,ny1_13,nx2_13,ny2_13,nx3_13,ny3_13)
        endif

c ---   Circ-Sedi
        if(Grid_Circ_Sedi_Same) then
          call interpsame(Nx_Sedi,Ny_Sedi,Sc_23,S1_23,S2_23,S3_23,
     .          nx1_23,ny1_23,nx2_23,ny2_23,nx3_23,ny3_23)
        else
         write(*,*)'Grid_Circ & Grid_Sedi are different, calc coef...'
          call interpolation
     .    (Nx_Circ,Ny_Circ,X_Circ,Y_Circ,
     .     Nx_Sedi,Ny_Sedi,X_Sedi,Y_Sedi,Sc_23,S1_23,S2_23,S3_23,
     .     nx1_23,ny1_23,nx2_23,ny2_23,nx3_23,ny3_23)
        endif

100     continue

        return
        end
```

3.4 Subroutine interp_depth

The subroutine interpolates water depth from Master-Grid to Wave-Grid, Circ-Grid, and Sedi-Grid.

interp_depth is called by

1. *master*

interp_depth calls

1. *grid1_to_grid2*

⟨*⟩+≡

```
c --------------------------------------------------------------

      subroutine interp_depth
      implicit none
      include 'pass.h'
      include 'interp.h'

      call grid1_to_grid2(Nx_Mast,Ny_Mast,Nx_Wave,Ny_Wave,
     &                    Sc_01,S1_01,S2_01,S3_01,
     &                    nx1_01,ny1_01,nx2_01,ny2_01,nx3_01,ny3_01,
     &                    Depth_Mast,Depth_Wave,
     &                    .false.,0,Wave_Staggered,Wave_Stag_huv(1))

      call grid1_to_grid2(Nx_Mast,Ny_Mast,Nx_Circ,Ny_Circ,
     &                    Sc_02,S1_02,S2_02,S3_02,
     &                    nx1_02,ny1_02,nx2_02,ny2_02,nx3_02,ny3_02,
     &                    Depth_Mast,Depth_Circ,
     &                    .false.,0,Circ_Staggered,Circ_Stag_huv(1))

      call grid1_to_grid2(Nx_Mast,Ny_Mast,Nx_Sedi,Ny_Sedi,
     &                    Sc_03,S1_03,S2_03,S3_03,
     &                    nx1_03,ny1_03,nx2_03,ny2_03,nx3_03,ny3_03,
     &                    Depth_Mast,Depth_Sedi,
     &                    .false.,0,Sedi_Staggered,Sedi_Stag_huv(1))


c --- test interpolatoin
c      call output(Nx_Mast,Ny_Mast,1,Depth_Mast)
c      call output(Nx_Circ,Ny_Circ,2,Depth_Circ)

      return
      end
```

3.5 Subroutine interp_circ_wave

The subroutine interpolates variables from Circ-Grid to Wave-Grid.

interp_circ_wave is called by

1. *master*

interp_circ_wave calls

1. *grid1_to_grid2*

⟨*⟩+≡
```
c ---------------------------------------------------------------

      subroutine interp_circ_wave
      implicit none
      include 'pass.h'
      include 'interp.h'
      real Tmp1, Tmp2, tht

      if (Wave_Curr_Interact) then

        if(Circ_To_Wave_UV) then


      call grid1_to_grid2(Nx_Circ,Ny_Circ,Nx_Wave,Ny_Wave,
   &                    Sc_21,S1_21,S2_21,S3_21,
   &                    nx1_21,ny1_21,nx2_21,ny2_21,nx3_21,ny3_21,
   &                    Pass_U,Intp_U_Wave,
   &                    Circ_Staggered,Circ_Stag_huv(2),
   &                    Wave_Staggered,Wave_Stag_huv(2))

      call grid1_to_grid2(Nx_Circ,Ny_Circ,Nx_Wave,Ny_Wave,
   &                    Sc_21,S1_21,S2_21,S3_21,
   &                    nx1_21,ny1_21,nx2_21,ny2_21,nx3_21,ny3_21,
   &                    Pass_V,Intp_V_Wave,
   &                    Circ_Staggered,Circ_Stag_huv(3),
   &                    Wave_Staggered,Wave_Stag_huv(3))


           if((Circ_Rotate_Angle-Wave_Rotate_Angle).ne.0)then
             do j=1,Ny_Wave
             do i=1,Nx_Wave
               Tmp1=Intp_U_Wave(i,j)
               Tmp2=Intp_V_Wave(i,j)
               tht=(Circ_Rotate_Angle-Wave_Rotate_Angle)*3.14159/180.
                Intp_U_Wave(i,j)=Tmp1*cos(tht)- Tmp2*sin(tht)
                Intp_V_Wave(i,j)=Tmp1*sin(tht)+ Tmp2*cos(tht)
```

```
          enddo
          enddo

        endif
      endif

      if(Circ_To_Wave_eta)then

     call grid1_to_grid2(Nx_Circ,Ny_Circ,Nx_Wave,Ny_Wave,
     &                    Sc_21,S1_21,S2_21,S3_21,
     &                    nx1_21,ny1_21,nx2_21,ny2_21,nx3_21,ny3_21,
     &                    Pass_eta,Intp_eta_Wave,
     &                    Circ_Staggered,Circ_Stag_huv(1),
     &                    Wave_Staggered,Wave_Stag_huv(1))

      endif

     endif

     return
     end
```

3.6 Subroutine interp_sedi_wave

The subroutine interpolates variables from Sedi-Grid to Wave-Grid.

interp_sedi_wave is called by

1. *master*

interp_sedi_wave calls

1. *grid1_to_grid2*

⟨*⟩+≡

```
c -------------------------------------------------------------

      subroutine interp_sedi_wave
      implicit none
      include 'pass.h'
      include 'interp.h'
      real Tmp1,Tmp2,tht


      if (Wave_Bed_Interact) then
        if(Sedi_To_Wave_Depth)then
      call grid1_to_grid2(Nx_Sedi,Ny_Sedi,Nx_Wave,Ny_Wave,
     &                    Sc_31,S1_31,S2_31,S3_31,
     &                    nx1_31,ny1_31,nx2_31,ny2_31,nx3_31,ny3_31,
     &                    Pass_Dupdated,Depth_Wave,
     &                    Sedi_Staggered,Sedi_Stag_huv(1),
     &                    Wave_Staggered,Sedi_Stag_huv(1))
        endif
      endif

      return
      end
```

3.7 Subroutine interp_wave_circ

The subroutine interpolates variables from Wave-Grid to Circ-Grid.

interp_wave_circ is called by

1. *master*

interp_wave_circ calls

1. *grid1_to_grid2*

⟨*⟩+≡
```
c -----------------------------------------------------------------

        subroutine interp_wave_circ
        implicit none
        include 'pass.h'
        include 'interp.h'
        real Tmp1, Tmp2, tht

c --- wave height

        if(Wave_To_Circ_Height)then

        call grid1_to_grid2(Nx_Wave,Ny_Wave,Nx_Circ,Ny_Circ,
     &                      Sc_12,S1_12,S2_12,S3_12,
     &                      nx1_12,ny1_12,nx2_12,ny2_12,nx3_12,ny3_12,
     &                      Pass_Height,Intp_Height_Circ,
     &                      Wave_Staggered,0,
     &                      Circ_Staggered,0)

        endif

c --- wave angle

        if(Wave_To_Circ_Angle)then

        call grid1_to_grid2(Nx_Wave,Ny_Wave,Nx_Circ,Ny_Circ,
     &                      Sc_12,S1_12,S2_12,S3_12,
     &                      nx1_12,ny1_12,nx2_12,ny2_12,nx3_12,ny3_12,
     &                      Pass_Theta,Intp_Theta_Circ,
     &                      Wave_Staggered,0,
     &                      Circ_Staggered,0)

        endif


c --- wave number
```

```
      if(Wave_To_Circ_WaveNum)then

      call grid1_to_grid2(Nx_Wave,Ny_Wave,Nx_Circ,Ny_Circ,
     &                    Sc_12,S1_12,S2_12,S3_12,
     &                    nx1_12,ny1_12,nx2_12,ny2_12,nx3_12,ny3_12,
     &                    Pass_WaveNum,Intp_WaveNum_Circ,
     &                    Wave_Staggered,0,
     &                    Circ_Staggered,0)

      endif

c --- wave C

      if(Wave_To_Circ_C)then

      call grid1_to_grid2(Nx_Wave,Ny_Wave,Nx_Circ,Ny_Circ,
     &                    Sc_12,S1_12,S2_12,S3_12,
     &                    nx1_12,ny1_12,nx2_12,ny2_12,nx3_12,ny3_12,
     &                    Pass_C,Intp_C_Circ,
     &                    Wave_Staggered,0,
     &                    Circ_Staggered,0)

      endif

c --- radiation stress (depth-average form)

      if(Wave_To_Circ_Radiation)then

      call grid1_to_grid2(Nx_Wave,Ny_Wave,Nx_Circ,Ny_Circ,
     &                    Sc_12,S1_12,S2_12,S3_12,
     &                    nx1_12,ny1_12,nx2_12,ny2_12,nx3_12,ny3_12,
     &                    Pass_Sxx,Intp_Sxx_Circ,
     &                    Wave_Staggered,0,
     &                    Circ_Staggered,0)

      call grid1_to_grid2(Nx_Wave,Ny_Wave,Nx_Circ,Ny_Circ,
     &                    Sc_12,S1_12,S2_12,S3_12,
     &                    nx1_12,ny1_12,nx2_12,ny2_12,nx3_12,ny3_12,
     &                    Pass_Sxy,Intp_Sxy_Circ,
     &                    Wave_Staggered,0,Circ_Staggered,0)

      call grid1_to_grid2(Nx_Wave,Ny_Wave,Nx_Circ,Ny_Circ,
     &                    Sc_12,S1_12,S2_12,S3_12,
     &                    nx1_12,ny1_12,nx2_12,ny2_12,nx3_12,ny3_12,
     &                    Pass_Syy,Intp_Syy_Circ,
```

```
     &                       Wave_Staggered,0,Circ_Staggered,0)

        endif

c --- radiation stress - surface

        if(Wave_To_Circ_Rad_Surf)then

        call grid1_to_grid2(Nx_Wave,Ny_Wave,Nx_Circ,Ny_Circ,
     &                       Sc_12,S1_12,S2_12,S3_12,
     &                       nx1_12,ny1_12,nx2_12,ny2_12,nx3_12,ny3_12,
     &                       Pass_Sxx_surf,Intp_Sxx_surf,
     &                       Wave_Staggered,0,Circ_Staggered,0)

        call grid1_to_grid2(Nx_Wave,Ny_Wave,Nx_Circ,Ny_Circ,
     &                       Sc_12,S1_12,S2_12,S3_12,
     &                       nx1_12,ny1_12,nx2_12,ny2_12,nx3_12,ny3_12,
     &                       Pass_Sxy_surf,Intp_Sxy_surf,
     &                       Wave_Staggered,0,Circ_Staggered,0)

        call grid1_to_grid2(Nx_Wave,Ny_Wave,Nx_Circ,Ny_Circ,
     &                       Sc_12,S1_12,S2_12,S3_12,
     &                       nx1_12,ny1_12,nx2_12,ny2_12,nx3_12,ny3_12,
     &                       Pass_Syy_surf,Intp_Syy_surf,
     &                       Wave_Staggered,0,Circ_Staggered,0)

        endif

c --- radiation stress - body

        if(Wave_To_Circ_Rad_Body)then

        call grid1_to_grid2(Nx_Wave,Ny_Wave,Nx_Circ,Ny_Circ,
     &                       Sc_12,S1_12,S2_12,S3_12,
     &                       nx1_12,ny1_12,nx2_12,ny2_12,nx3_12,ny3_12,
     &                       Pass_Sxx_body,Intp_Sxx_body,
     &                       Wave_Staggered,0,Circ_Staggered,0)

        call grid1_to_grid2(Nx_Wave,Ny_Wave,Nx_Circ,Ny_Circ,
     &                       Sc_12,S1_12,S2_12,S3_12,
     &                       nx1_12,ny1_12,nx2_12,ny2_12,nx3_12,ny3_12,
     &                       Pass_Sxy_body,Intp_Sxy_body,
     &                       Wave_Staggered,0,Circ_Staggered,0)

        call grid1_to_grid2(Nx_Wave,Ny_Wave,Nx_Circ,Ny_Circ,
     &                       Sc_12,S1_12,S2_12,S3_12,
```

```
     &                       nx1_12,ny1_12,nx2_12,ny2_12,nx3_12,ny3_12,
     &                       Pass_Syy_body,Intp_Syy_body,
     &                       Wave_Staggered,0,Circ_Staggered,0)

        endif


c --- short wave forcing

        if(Wave_To_Circ_Forcing)then

        call grid1_to_grid2(Nx_Wave,Ny_Wave,Nx_Circ,Ny_Circ,
     &                      Sc_12,S1_12,S2_12,S3_12,
     &                      nx1_12,ny1_12,nx2_12,ny2_12,nx3_12,ny3_12,
     &                      Pass_Wave_Fx,Intp_Fx_Circ,
     &                      Wave_Staggered,Wave_Stag_huv(2),
     &                      Circ_Staggered,Circ_Stag_huv(2))

        call grid1_to_grid2(Nx_Wave,Ny_Wave,Nx_Circ,Ny_Circ,
     &                      Sc_12,S1_12,S2_12,S3_12,
     &                      nx1_12,ny1_12,nx2_12,ny2_12,nx3_12,ny3_12,
     &                      Pass_Wave_Fy,Intp_Fy_Circ,
     &                      Wave_Staggered,Wave_Stag_huv(3),
     &                      Circ_Staggered,Circ_Stag_huv(3))

             if((Wave_Rotate_Angle-Circ_Rotate_Angle).ne.0)then
               do j=1,Ny_Circ
               do i=1,Nx_Circ
                 Tmp1=Intp_Fx_Circ(i,j)
                 Tmp2=Intp_Fy_Circ(i,j)
                 tht=(Wave_Rotate_Angle-CIrc_Rotate_Angle)*3.14159/180.
                  Intp_Fx_Circ(i,j)=Tmp1*cos(tht)- Tmp2*sin(tht)
                  Intp_Fy_Circ(i,j)=Tmp1*sin(tht)+ Tmp2*cos(tht)
               enddo
               enddo
             endif

        endif

c ---  mass flux

        if(Wave_To_Circ_MassFlux) then

        call grid1_to_grid2(Nx_Wave,Ny_Wave,Nx_Circ,Ny_Circ,
     &                      Sc_12,S1_12,S2_12,S3_12,
     &                      nx1_12,ny1_12,nx2_12,ny2_12,nx3_12,ny3_12,
```

```
     &                      Pass_MassFluxU,Intp_MassFluxU_Circ,
     &                      Wave_Staggered,Wave_Stag_huv(2),
     &                      Circ_Staggered,Circ_Stag_huv(2))

        call grid1_to_grid2(Nx_Wave,Ny_Wave,Nx_Circ,Ny_Circ,
     &                      Sc_12,S1_12,S2_12,S3_12,
     &                      nx1_12,ny1_12,nx2_12,ny2_12,nx3_12,ny3_12,
     &                      Pass_MassFluxV,Intp_MassFluxV_Circ,
     &                      Wave_Staggered,Wave_Stag_huv(3),
     &                      Circ_Staggered,Circ_Stag_huv(3))

             if((Wave_Rotate_Angle-Circ_Rotate_Angle).ne.0)then
               do j=1,Ny_Circ
               do i=1,Nx_Circ
                 Tmp1=Intp_MassFluxU_Circ(i,j)
                 Tmp2=Intp_MassFluxV_Circ(i,j)
                 tht=(Wave_Rotate_Angle-CIrc_Rotate_Angle)*3.14159/180.
                  Intp_MassFluxU_Circ(i,j)=Tmp1*cos(tht)- Tmp2*sin(tht)
                  Intp_MassFluxV_Circ(i,j)=Tmp1*sin(tht)+ Tmp2*cos(tht)
               enddo
               enddo
             endif

        endif

c --- wave dissipation

        if(Wave_To_Circ_Dissipation) then

        call grid1_to_grid2(Nx_Wave,Ny_Wave,Nx_Circ,Ny_Circ,
     &                      Sc_12,S1_12,S2_12,S3_12,
     &                      nx1_12,ny1_12,nx2_12,ny2_12,nx3_12,ny3_12,
     &                      Pass_Diss,Intp_Diss_Circ,
     &                      Wave_Staggered,0,Circ_Staggered,0)

        endif

c --- wave bottom velocity

        if(Wave_To_Circ_BottomUV) then

        call grid1_to_grid2(Nx_Wave,Ny_Wave,Nx_Circ,Ny_Circ,
     &                      Sc_12,S1_12,S2_12,S3_12,
     &                      nx1_12,ny1_12,nx2_12,ny2_12,nx3_12,ny3_12,
     &                      Pass_ubott,Intp_ubott_Circ,
     &                      Wave_Staggered,0,Circ_Staggered,0)
```

```
        endif

c ---  break index

        if(Wave_To_Circ_Dissipation) then

        call grid1_to_grid2(Nx_Wave,Ny_Wave,Nx_Circ,Ny_Circ,
     &                     Sc_12,S1_12,S2_12,S3_12,
     &                     nx1_12,ny1_12,nx2_12,ny2_12,nx3_12,ny3_12,
     &                     Pass_ibrk,Intp_ibrk_Circ,
     &                     Wave_Staggered,0,Circ_Staggered,0)

        endif

        return
        end
```

3.8 Subroutine interp_sedi_circ

The subroutine interpolates variables from Sedi-Grid to Circ-Grid.

interp_sedi_circ is called by

1. *master*

interp_sedi_circ calls

1. *grid1_to_grid2*

⟨*⟩+≡
```
c ---------------------------------------------------------------

      subroutine interp_sedi_circ
      implicit none
      include 'pass.h'
      include 'interp.h'
      real Tmp1,Tmp2,tht


      if (Curr_Bed_Interact) then

      if(Sedi_To_Circ_Depth)then

      call grid1_to_grid2(Nx_Sedi,Ny_Sedi,Nx_Circ,Ny_Circ,
     &                    Sc_32,S1_32,S2_32,S3_32,
     &                    nx1_32,ny1_32,nx2_32,ny2_32,nx3_32,ny3_32,
     &                    Pass_Dupdated,Depth_Circ,
     &                    Sedi_Staggered,Sedi_Stag_huv(1),
     &                    Circ_Staggered,Circ_Stag_huv(1))
      endif

      endif

      return
      end
```

3.9 Subroutine interp_wave_sedi

The subroutine interpolates variables from Wave-Grid to Sedi-Grid.
interp_wave_sedi is called by

1. *master*

interp_wave_sedi calls

1. *grid1_to_grid2*

⟨*⟩+≡
```
c -----------------------------------------------------------

      subroutine interp_wave_sedi
      implicit none
      include 'pass.h'
      include 'interp.h'
      real Tmp1,Tmp2,tht

      if(Wave_To_Sedi_Height)then

      call grid1_to_grid2(Nx_Wave,Ny_Wave,Nx_Sedi,Ny_Sedi,
   &                      Sc_13,S1_13,S2_13,S3_13,
   &                      nx1_13,ny1_13,nx2_13,ny2_13,nx3_13,ny3_13,
   &                      Pass_Height,Intp_Height_Sedi,
   &                      Wave_Staggered,0,Sedi_Staggered,0)

      endif

      if(Wave_To_Sedi_Angle)then

      call grid1_to_grid2(Nx_Wave,Ny_Wave,Nx_Sedi,Ny_Sedi,
   &                      Sc_13,S1_13,S2_13,S3_13,
   &                      nx1_13,ny1_13,nx2_13,ny2_13,nx3_13,ny3_13,
   &                      Pass_Theta,Intp_Theta_Sedi,
   &                      Wave_Staggered,0,Sedi_Staggered,0)

         if((Wave_Rotate_Angle-Sedi_Rotate_Angle).ne.0)then
           do j=1,Ny_Sedi
           do i=1,Nx_Sedi
             tht=Wave_Rotate_Angle-Sedi_Rotate_Angle
             Intp_Theta_Sedi(i,j)=Intp_Theta_Sedi(i,j)+tht
           enddo
           enddo
         endif

      endif
```

```
      if(Wave_To_Sedi_BottomUV)then

      call grid1_to_grid2(Nx_Wave,Ny_Wave,Nx_Sedi,Ny_Sedi,
     &                    Sc_13,S1_13,S2_13,S3_13,
     &                    nx1_13,ny1_13,nx2_13,ny2_13,nx3_13,ny3_13,
     &                    Pass_ubott,Intp_ubott_Sedi,
     &                    Wave_Staggered,0,Sedi_Staggered,0)


      endif

      return
      end
```

3.10 Subroutine interp_circ_sedi

The subroutine interpolates variables from Circ-Grid to Sedi-Grid.
interp_circ_sedi is called by

1. *master*

interp_circ_sedi calls

1. *grid1_to_grid2*

⟨*⟩+≡
```
c --------------------------------------------------------------

      subroutine interp_circ_sedi
      implicit none
      include 'pass.h'
      include 'interp.h'
      real Tmp1,Tmp2,tht


c --- UV
      if(Circ_To_Sedi_UV)then

      call grid1_to_grid2(Nx_Circ,Ny_Circ,Nx_Sedi,Ny_Sedi,
     &                    Sc_23,S1_23,S2_23,S3_23,
     &                    nx1_23,ny1_23,nx2_23,ny2_23,nx3_23,ny3_23,
     &                    Pass_U,Intp_U_Sedi,
     &                    Circ_Staggered,Circ_Stag_huv(2),
     &                    Sedi_Staggered,Sedi_Stag_huv(2))


      call grid1_to_grid2(Nx_Circ,Ny_Circ,Nx_Sedi,Ny_Sedi,
     &                    Sc_23,S1_23,S2_23,S3_23,
     &                    nx1_23,ny1_23,nx2_23,ny2_23,nx3_23,ny3_23,
     &                    Pass_V,Intp_V_Sedi,
     &                    Circ_Staggered,Circ_Stag_huv(3),
     &                    Sedi_Staggered,Sedi_Stag_huv(3))

          if((Circ_Rotate_Angle-Sedi_Rotate_Angle).ne.0)then
            do j=1,Ny_Sedi
            do i=1,Nx_Sedi
              Tmp1=Intp_U_Sedi(i,j)
              Tmp2=Intp_V_Sedi(i,j)
              tht=(Circ_Rotate_Angle-Sedi_Rotate_Angle)*3.14159/180.
               Intp_U_Sedi(i,j)=Tmp1*cos(tht)- Tmp2*sin(tht)
               Intp_V_Sedi(i,j)=Tmp1*sin(tht)+ Tmp2*cos(tht)
            enddo
```

```
                enddo
              endif
            endif

c -- Ub Vb
            if(Circ_To_Sedi_UVb)then

            call grid1_to_grid2(Nx_Circ,Ny_Circ,Nx_Sedi,Ny_Sedi,
     &                        Sc_23,S1_23,S2_23,S3_23,
     &                        nx1_23,ny1_23,nx2_23,ny2_23,nx3_23,ny3_23,
     &                        Pass_Ub,Intp_Ub_Sedi,
     &                        Circ_Staggered,Circ_Stag_huv(2),
     &                        Sedi_Staggered,Sedi_Stag_huv(2))


            call grid1_to_grid2(Nx_Circ,Ny_Circ,Nx_Sedi,Ny_Sedi,
     &                        Sc_23,S1_23,S2_23,S3_23,
     &                        nx1_23,ny1_23,nx2_23,ny2_23,nx3_23,ny3_23,
     &                        Pass_Vb,Intp_Vb_Sedi,
     &                        Circ_Staggered,Circ_Stag_huv(3),
     &                        Sedi_Staggered,Sedi_Stag_huv(3))

              if((Circ_Rotate_Angle-Sedi_Rotate_Angle).ne.0)then
                do j=1,Ny_Sedi
                do i=1,Nx_Sedi
                  Tmp1=Intp_Ub_Sedi(i,j)
                  Tmp2=Intp_Vb_Sedi(i,j)
                  tht=(Circ_Rotate_Angle-Sedi_Rotate_Angle)*3.14159/180.
                   Intp_Ub_Sedi(i,j)=Tmp1*cos(tht)- Tmp2*sin(tht)
                   Intp_Vb_Sedi(i,j)=Tmp1*sin(tht)+ Tmp2*cos(tht)
                enddo
                enddo
              endif
            endif

c --- eta

            if(Circ_To_Sedi_eta)then

            call grid1_to_grid2(Nx_Circ,Ny_Circ,Nx_Sedi,Ny_Sedi,
     &                        Sc_23,S1_23,S2_23,S3_23,
     &                        nx1_23,ny1_23,nx2_23,ny2_23,nx3_23,ny3_23,
     &                        Pass_eta,Intp_eta_Sedi,
     &                        Circ_Staggered,Circ_Stag_huv(1),
     &                        Sedi_Staggered,Sedi_Stag_huv(1))
```

```
      endif

c ---  fw

      if(Circ_To_Sedi_fw)then

      call grid1_to_grid2(Nx_Circ,Ny_Circ,Nx_Sedi,Ny_Sedi,
     &                    Sc_23,S1_23,S2_23,S3_23,
     &                    nx1_23,ny1_23,nx2_23,ny2_23,nx3_23,ny3_23,
     &                    Pass_fw,Intp_fw_Sedi,
     &                    Circ_Staggered,0,Sedi_Staggered,0)

      endif

      if(Circ_To_Sedi_UV3D)then
      print*,'not done yet'

      endif


      if(Circ_To_Sedi_UVquasi3D)then
      print*,'not done yet'

      endif



      return
      end
```

3.11 Subroutine interpsame

The subroutine calculates interpolation coefficients when two module grids are same.

interpsame is called by

1. *get_interpolation_coef*

⟨*⟩+≡

```
c ----------------------------------------------------------------

        subroutine interpsame
     .  (m_grid2,n_grid2,Sc,S1,S2,S3,
     .   nx1,ny1,nx2,ny2,nx3,ny3)

        implicit none
        include 'pass.h'

! ---   (i,j) of three points surrounded
        integer nx1(Nx_Max,Ny_Max)
     .         ,ny1(Nx_Max,Ny_Max)
     .         ,nx2(Nx_Max,Ny_Max)
     .         ,ny2(Nx_Max,Ny_Max)
     .         ,nx3(Nx_Max,Ny_Max)
     .         ,ny3(Nx_Max,Ny_Max)

! ---  areas of the four triangles, area will be negative
!      if an order is clockwise
!       Sc -- triangle 1,2,3
!       S1 -- triangle 2,3,c
!       S2 -- triangle 3,1,c
!       S3 -- triangle 1,2,c

        real Sc(Nx_Max,Ny_Max)
     .      ,S1(Nx_Max,Ny_Max)
     .      ,S2(Nx_Max,Ny_Max)
     .      ,S3(Nx_Max,Ny_Max)

! ---   m(x direction) and n(y direction)
        integer m_grid2,n_grid2

        do j=1,n_grid2
        do i=1,m_grid2
          Sc(i,j)=1.
          S1(i,j)=1.
          S2(i,j)=0.
```

```
  S3(i,j)=0.
  nx1(i,j)=i
  ny1(i,j)=j
  nx2(i,j)=1
  ny2(i,j)=1
  nx3(i,j)=1
  ny3(i,j)=1
enddo
enddo

return
end
```

3.12 Subroutine interpolation

This subroutine is used to get coefficients of interpolation or extrapolation between two structured grid systems. Any grid of two or both of two grids can be curvilinear or rectangular. The routine can deal with the case that one grid does not exactly overlap another grid. For the no-overlapped the grid points, extrapolations may be carried out if extrapolation is allowed, i.e., Grid_Extrapolation = .true.. To save time, all necessary arrays are stored in arrays in 'interp.h'.

The interpolation/extrapolation theory can be found in Section 1.2.

1. Formulas:
   (a) calculation of triangle area:

   $$S = 0.5 * (x1 * y2 - x2 * y1 + x2 * y3 - x3 * y2 + x3 * y1 - x1 * y3)$$

   if (1,2,3) is counterclock wise, $S > 0$, otherwise, $S < 0$

   (b) interpolation/extrapolation:

   $$var_c = (S1 * var_1 + S2 * var_2 + S3 * var_3)/Sc$$

2. Arguments
   (a) m_grid1 – grid number of grid1 in x direction
   (b) n_grid1 – grid number of grid1 in y direction
   (c) m_grid2 – grid number of grid2 in x direction
   (d) n_grid2 – grid number of grid2 in y direction
   (e) var_grid1 – variables in grid1
   (f) var_grid2 – variables converted in grid2

interpolation is called by

1. *get_interpolation_coef*

⟨*⟩+≡

```
! -------------------------------------------------------------------
        subroutine interpolation
     .  (m_grid1,n_grid1,x_grid1,y_grid1,
     .   m_grid2,n_grid2,x_grid2,y_grid2,Sc,S1,S2,S3,
     .   nx1,ny1,nx2,ny2,nx3,ny3)

        implicit none
        include 'pass.h'

! ---   types, interpolation -- 0, extrapolation -- 1
        integer ntype(Nx_Max,Ny_Max)
```

```
! ---   (i,j) of three points surrounded
        integer nx1(Nx_Max,Ny_Max)
     .         ,ny1(Nx_Max,Ny_Max)
     .         ,nx2(Nx_Max,Ny_Max)
     .         ,ny2(Nx_Max,Ny_Max)
     .         ,nx3(Nx_Max,Ny_Max)
     .         ,ny3(Nx_Max,Ny_Max)

! ---  areas of the four triangles, area will be negative
!      if an order is clockwise
!       Sc -- triangle 1,2,3
!       S1 -- triangle 2,3,c
!       S2 -- triangle 3,1,c
!       S3 -- triangle 1,2,c

        real Sc(Nx_Max,Ny_Max)
     .      ,S1(Nx_Max,Ny_Max)
     .      ,S2(Nx_Max,Ny_Max)
     .      ,S3(Nx_Max,Ny_Max)

! ---   m(x direction) and n(y direction)
        integer m_grid1,n_grid1,m_grid2,n_grid2

! ---   x, y and variables of grid1 and grid2
        real x_grid1(Nx_Max,Ny_Max),y_grid1(Nx_Max,Ny_Max)
     .      ,var_grid1(Nx_Max,Ny_Max)
     .      ,x_grid2(Nx_Max,Ny_Max),y_grid2(Nx_Max,Ny_Max)
     .      ,var_grid2(Nx_Max,Ny_Max)

        real x1,y1,x2,y2,x3,y3,area1,area2,area3,area,dist,
     .       dist_init

        integer ii,jj,nx_near,ny_near

! ---  control parameter, the initial -- 0
        integer Iconv
        data Iconv /0/

! --- find the triangle includes the points in grid2

        do j=1,n_grid2
        do i=1,m_grid2
          x1=x_grid2(i,j)
          y1=y_grid2(i,j)
          do jj=1,n_grid1-1
```

```
      do ii=1,m_grid1-1

        x2=x_grid1(ii+1,jj)
        y2=y_grid1(ii+1,jj)
        x3=x_grid1(ii,jj+1)
        y3=y_grid1(ii,jj+1)
        area1=0.5*(x1*y2-x2*y1+x2*y3-x3*y2+x3*y1-x1*y3)

        x2=x_grid1(ii,jj+1)
        y2=y_grid1(ii,jj+1)
        x3=x_grid1(ii,jj)
        y3=y_grid1(ii,jj)
        area2=0.5*(x1*y2-x2*y1+x2*y3-x3*y2+x3*y1-x1*y3)

        x2=x_grid1(ii,jj)
        y2=y_grid1(ii,jj)
        x3=x_grid1(ii+1,jj)
        y3=y_grid1(ii+1,jj)
        area3=0.5*(x1*y2-x2*y1+x2*y3-x3*y2+x3*y1-x1*y3)

        if(area1.ge.0.and.area2.ge.0.and.area3.ge.0)then
          ntype(i,j)=0
          nx1(i,j)=ii
          ny1(i,j)=jj
          nx2(i,j)=ii+1
          ny2(i,j)=jj
          nx3(i,j)=ii
          ny3(i,j)=jj+1
          S1(i,j)=area1
          S2(i,j)=area2
          S3(i,j)=area3

          x1=x_grid1(nx1(i,j),ny1(i,j))
          y1=y_grid1(nx1(i,j),ny1(i,j))
          x2=x_grid1(nx2(i,j),ny2(i,j))
          y2=y_grid1(nx2(i,j),ny2(i,j))
          x3=x_grid1(nx3(i,j),ny3(i,j))
          y3=y_grid1(nx3(i,j),ny3(i,j))
          Sc(i,j)=0.5*(x1*y2-x2*y1+x2*y3-x3*y2+x3*y1-x1*y3)

          goto 110
        endif

        x2=x_grid1(ii+1,jj)
        y2=y_grid1(ii+1,jj)
        x3=x_grid1(ii+1,jj+1)
```

```
          y3=y_grid1(ii+1,jj+1)
          area1=0.5*(x1*y2-x2*y1+x2*y3-x3*y2+x3*y1-x1*y3)

          x2=x_grid1(ii+1,jj+1)
          y2=y_grid1(ii+1,jj+1)
          x3=x_grid1(ii,jj+1)
          y3=y_grid1(ii,jj+1)
          area2=0.5*(x1*y2-x2*y1+x2*y3-x3*y2+x3*y1-x1*y3)

          x2=x_grid1(ii,jj+1)
          y2=y_grid1(ii,jj+1)
          x3=x_grid1(ii+1,jj)
          y3=y_grid1(ii+1,jj)
          area3=0.5*(x1*y2-x2*y1+x2*y3-x3*y2+x3*y1-x1*y3)

          if(area1.ge.0.and.area2.ge.0.and.area3.ge.0)then
            ntype(i,j)=0
            nx1(i,j)=ii
            ny1(i,j)=jj+1
            nx2(i,j)=ii+1
            ny2(i,j)=jj
            nx3(i,j)=ii+1
            ny3(i,j)=jj+1
            S1(i,j)=area1
            S2(i,j)=area2
            S3(i,j)=area3

            x1=x_grid1(nx1(i,j),ny1(i,j))
            y1=y_grid1(nx1(i,j),ny1(i,j))
            x2=x_grid1(nx2(i,j),ny2(i,j))
            y2=y_grid1(nx2(i,j),ny2(i,j))
            x3=x_grid1(nx3(i,j),ny3(i,j))
            y3=y_grid1(nx3(i,j),ny3(i,j))
            Sc(i,j)=0.5*(x1*y2-x2*y1+x2*y3-x3*y2+x3*y1-x1*y3)

            goto 110
          endif

          ntype(i,j)=1

        enddo
        enddo

110     continue

      enddo
```

```
        enddo


! --- find the nearest point in grid1 for grid2-points with ntype=1
!     these points will be used for extrapolation

        do j=1,n_grid2
        do i=1,m_grid2

          if (ntype(i,j).eq.1)then

!           -- find the nearest point

            x1=x_grid2(i,j)
            y1=y_grid2(i,j)
            x2=x_grid1(1,1)
            y2=y_grid1(1,1)
            dist_init=(x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)
            nx_near=1
            ny_near=1

            do jj=2,n_grid1-1
            do ii=2,m_grid1-1
              x2=x_grid1(ii,jj)
              y2=y_grid1(ii,jj)
              dist=(x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)
              if(dist.lt.dist_init)then
                dist_init=dist
                nx_near=ii
                ny_near=jj
              endif
            enddo
            enddo

!           -- calculate four areas -- S1, S2, S3, Sc
!              choose the nearest triangle by using the sign of the area

            x2=x_grid1(nx_near+1,ny_near)
            y2=y_grid1(nx_near+1,ny_near)
            x3=x_grid1(nx_near,ny_near+1)
            y3=y_grid1(nx_near,ny_near+1)
            area=0.5*(x1*y2-x2*y1+x2*y3-x3*y2+x3*y1-x1*y3)

            if(area.ge.0)then

              nx1(i,j)=nx_near
```

```
          ny1(i,j)=ny_near
          nx2(i,j)=nx_near+1
          ny2(i,j)=ny_near
          nx3(i,j)=nx_near
          ny3(i,j)=ny_near+1

        else

          nx1(i,j)=nx_near
          ny1(i,j)=ny_near+1
          nx2(i,j)=nx_near+1
          ny2(i,j)=ny_near
          nx3(i,j)=nx_near+1
          ny3(i,j)=ny_near+1

        endif

! --- if no extrapolation is allowed , evaluated variable will equal
!     to the variable at neast grid point

        if (Grid_Extrapolation) then
          x2=x_grid1(nx2(i,j),ny2(i,j))
          y2=y_grid1(nx2(i,j),ny2(i,j))
          x3=x_grid1(nx3(i,j),ny3(i,j))
          y3=y_grid1(nx3(i,j),ny3(i,j))
          S1(i,j)=0.5*(x1*y2-x2*y1+x2*y3-x3*y2+x3*y1-x1*y3)

          x2=x_grid1(nx3(i,j),ny3(i,j))
          y2=y_grid1(nx3(i,j),ny3(i,j))
          x3=x_grid1(nx1(i,j),ny1(i,j))
          y3=y_grid1(nx1(i,j),ny1(i,j))
          S2(i,j)=0.5*(x1*y2-x2*y1+x2*y3-x3*y2+x3*y1-x1*y3)

          x2=x_grid1(nx1(i,j),ny1(i,j))
          y2=y_grid1(nx1(i,j),ny1(i,j))
          x3=x_grid1(nx2(i,j),ny2(i,j))
          y3=y_grid1(nx2(i,j),ny2(i,j))
          S3(i,j)=0.5*(x1*y2-x2*y1+x2*y3-x3*y2+x3*y1-x1*y3)

          x1=x_grid1(nx1(i,j),ny1(i,j))
          y1=y_grid1(nx1(i,j),ny1(i,j))
          x2=x_grid1(nx2(i,j),ny2(i,j))
          y2=y_grid1(nx2(i,j),ny2(i,j))
          x3=x_grid1(nx3(i,j),ny3(i,j))
          y3=y_grid1(nx3(i,j),ny3(i,j))
          Sc(i,j)=0.5*(x1*y2-x2*y1+x2*y3-x3*y2+x3*y1-x1*y3)
```

```
    else

      S1(i,j)=1.
      S2(i,j)=0.
      S3(i,j)=0.
      Sc(i,j)=1.

    endif

  endif
enddo
enddo

return
end
```

3.13 Subroutine interpolation_nonstruc

This subroutine is used to get coefficients of interpolation or extrapolation between two grid systems in which one or two grids are non-structured grids. For non-structured grid, 2-D arrays are still used such as x(m_grid1,n_grid1), and n_grid1 = 1. The interpolation/extrapolation value is obtained from the values at three nearest points on grid1. To save time, all necessary arrays are stored in arrays in 'interp.h'.

The interpolation/extrapolation theory can be found in Section 1.2.

1. Formulas:

   (a) calculation of triangle area:

   $$S = 0.5 * (x1 * y2 - x2 * y1 + x2 * y3 - x3 * y2 + x3 * y1 - x1 * y3)$$

   if (1,2,3) is counterclock wise, $S > 0$, otherwise, $S < 0$

   (b) interpolation/extrapolation:

   $$var_c = (S1 * var_1 + S2 * var_2 + S3 * var_3)/Sc$$

2. Arguments

   (a) m_grid1 – grid number of grid1 in x direction
   (b) n_grid1 – grid number of grid1 in y direction
   (c) m_grid2 – grid number of grid2 in x direction
   (d) n_grid2 – grid number of grid2 in y direction
   (e) var_grid1 – variables in grid1
   (f) var_grid2 – variables converted in grid2

interpolation is called by

1. *get_interpolation_coef*

⟨*⟩+≡

```
! -------------------------------------------------------------------
        subroutine interpolation_nonstruc
     .  (m_grid1,n_grid1,x_grid1,y_grid1,
     .   m_grid2,n_grid2,x_grid2,y_grid2,Sc,S1,S2,S3,
     .   nx1,ny1,nx2,ny2,nx3,ny3)

        implicit none
        include 'pass.h'

! ---   (i,j) of three points surrounded
        integer nx1(Nx_Max,Ny_Max)
```

```
     .         ,ny1(Nx_Max,Ny_Max)
     .         ,nx2(Nx_Max,Ny_Max)
     .         ,ny2(Nx_Max,Ny_Max)
     .         ,nx3(Nx_Max,Ny_Max)
     .         ,ny3(Nx_Max,Ny_Max)

! ---  areas of the four triangles, area will be negative
!      if an order is clockwise
!       Sc -- triangle 1,2,3
!       S1 -- triangle 2,3,c
!       S2 -- triangle 3,1,c
!       S3 -- triangle 1,2,c

        real Sc(Nx_Max,Ny_Max)
     .      ,S1(Nx_Max,Ny_Max)
     .      ,S2(Nx_Max,Ny_Max)
     .      ,S3(Nx_Max,Ny_Max)

! ---   m(x direction) and n(y direction)
        integer m_grid1,n_grid1,m_grid2,n_grid2

! ---   x, y and variables of grid1 and grid2
        real x_grid1(Nx_Max,Ny_Max),y_grid1(Nx_Max,Ny_Max)
     .      ,var_grid1(Nx_Max,Ny_Max)
     .      ,x_grid2(Nx_Max,Ny_Max),y_grid2(Nx_Max,Ny_Max)
     .      ,var_grid2(Nx_Max,Ny_Max)

        real x1,y1,x2,y2,x3,y3,area1,area2,area3,area,dist,
     .       dist_init,dist_1,dist_2,dist_3

        integer ii,jj,nx_near,ny_near,nx_near_1,ny_near_1,
     .     nx_near_2,ny_near_2,nx_near_3,ny_near_3

! ---  control parameter, the initial -- 0
        Integer Iconv
        data Iconv /0/

! --- find the nearest three points on grid1
!     these points will be used for interpolation/extrapolation

        do j=1,n_grid2
        do i=1,m_grid2

!  --- find the farest point first

            x1=x_grid2(i,j)
```

```
        y1=y_grid2(i,j)
        x2=x_grid1(1,1)
        y2=y_grid1(1,1)
        dist_1=(x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)
        nx_near_1=1
        ny_near_1=1

        do jj=1,n_grid1
        do ii=1,m_grid1
          x2=x_grid1(ii,jj)
          y2=y_grid1(ii,jj)
          dist=(x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)
          if(dist.gt.dist_1)then
            dist_1=dist
            nx_near_1=ii
            ny_near_1=jj
          endif
        enddo
        enddo

        nx_near_2=nx_near_1
        ny_near_2=ny_near_1
        nx_near_3=nx_near_1
        ny_near_3=ny_near_1
        dist_2=dist_1
        dist_3=dist_1

! ---   find nearest three points

        do jj=1,n_grid1
        do ii=1,m_grid1
          x2=x_grid1(ii,jj)
          y2=y_grid1(ii,jj)
          dist=(x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)
          if(dist.lt.dist_1)then
            dist_1=dist
            nx_near_1=ii
            ny_near_1=jj
          elseif(dist.lt.dist_2)then
            dist_2=dist
            nx_near_2=ii
            ny_near_2=jj
          elseif(dist.lt.dist_3)then
            dist_3=dist
            nx_near_3=ii
            ny_near_3=jj
```

```
            endif
          enddo
          enddo

  !       -- calculate four areas -- S1, S2, S3, Sc


            nx1(i,j)=nx_near_1
            ny1(i,j)=ny_near_1
            nx2(i,j)=nx_near_2
            ny2(i,j)=ny_near_2
            nx3(i,j)=nx_near_3
            ny3(i,j)=ny_near_3


            x2=x_grid1(nx2(i,j),ny2(i,j))
            y2=y_grid1(nx2(i,j),ny2(i,j))
            x3=x_grid1(nx3(i,j),ny3(i,j))
            y3=y_grid1(nx3(i,j),ny3(i,j))
            S1(i,j)=0.5*(x1*y2-x2*y1+x2*y3-x3*y2+x3*y1-x1*y3)

            x2=x_grid1(nx3(i,j),ny3(i,j))
            y2=y_grid1(nx3(i,j),ny3(i,j))
            x3=x_grid1(nx1(i,j),ny1(i,j))
            y3=y_grid1(nx1(i,j),ny1(i,j))
            S2(i,j)=0.5*(x1*y2-x2*y1+x2*y3-x3*y2+x3*y1-x1*y3)

            x2=x_grid1(nx1(i,j),ny1(i,j))
            y2=y_grid1(nx1(i,j),ny1(i,j))
            x3=x_grid1(nx2(i,j),ny2(i,j))
            y3=y_grid1(nx2(i,j),ny2(i,j))
            S3(i,j)=0.5*(x1*y2-x2*y1+x2*y3-x3*y2+x3*y1-x1*y3)

            x1=x_grid1(nx1(i,j),ny1(i,j))
            y1=y_grid1(nx1(i,j),ny1(i,j))
            x2=x_grid1(nx2(i,j),ny2(i,j))
            y2=y_grid1(nx2(i,j),ny2(i,j))
            x3=x_grid1(nx3(i,j),ny3(i,j))
            y3=y_grid1(nx3(i,j),ny3(i,j))
            Sc(i,j)=0.5*(x1*y2-x2*y1+x2*y3-x3*y2+x3*y1-x1*y3)

        enddo
        enddo

        print*,'two grids are different, calculate interp coef..'
```

```
      return
      end
```

3.14 Subroutine grid1_to_grid2

The subroutine makes interpolation/extrapolation from grid1 to grid2 based on interpolation coefficients.

grid1_to_grid2 is called by

1. *interp_depth*
2. *interp_circ_wave*
3. *interp_circ_wave*
4. *interp_sedi_wave*
5. *interp_wave_circ*
6. *interp_sedi_circ*
7. *interp_wave_sedi*
8. *interp_circ_sedi*

⟨*⟩+≡

```
! -------------------------------------------------------------

      subroutine grid1_to_grid2
     .     (m_grid1,n_grid1,m_grid2,n_grid2,
     .      Sc,S1,S2,S3,nx1,ny1,nx2,ny2,nx3,ny3,
     .      var_grid1,var_grid2,grid1_stag,ntype_grid1,
     .      grid2_stag,ntype_grid2)


      implicit none
      include 'pass.h'
      real Sc(Nx_Max,Ny_Max)
     .     ,S1(Nx_Max,Ny_Max)
     .     ,S2(Nx_Max,Ny_Max)
     .     ,S3(Nx_Max,Ny_Max)
      integer nx1(Nx_Max,Ny_Max),ny1(Nx_Max,Ny_Max)
     .     ,nx2(Nx_Max,Ny_Max),ny2(Nx_Max,Ny_Max)
     .     ,nx3(Nx_Max,Ny_Max),ny3(Nx_Max,Ny_Max)
     .     ,ntype_grid1,ntype_grid2

! ---   x, y and variables of grid1 and grid2
      real var_grid1(Nx_Max,Ny_Max)
     .     ,var_grid2(Nx_Max,Ny_Max)
     .     ,tmp(Nx_Max,Ny_Max)

! ---   logical parameters for staggered grid
```

```
        logical grid1_stag, grid2_stag

! --- others
        integer m_grid1,n_grid1,m_grid2,n_grid2

        do j=1,n_grid1
        do i=1,m_grid1
          tmp(i,j)=var_grid1(i,j)
        enddo
        enddo

! ---   for staggered grid1

        if (grid1_stag.and.ntype_grid1.ne.0)then
! ---      ntype=1

          if(ntype_grid1.eq.1)then
            do j=2,n_grid1-1
            do i=2,m_grid1-1

             tmp(i,j)=0.25*(var_grid1(i-1,j-1)+var_grid1(i,j-1)
     &                     +var_grid1(i,j)+var_grid1(i-1,j))

            enddo
            enddo

            do i=2,m_grid1-1
              tmp(i,1)=2.*var_grid1(i,2)-var_grid1(i,3)
              tmp(i,n_grid1)=2.*var_grid1(i,n_grid1-1)
     &                         -var_grid1(i,n_grid1-2)
            enddo
            do j=1,n_grid1
              tmp(1,j)=2.*var_grid1(2,j)-var_grid1(3,j)
              tmp(m_grid1,j)=2.*var_grid1(m_grid1-1,j)
     &                         -var_grid1(m_grid1-2,j)
            enddo
          endif
! ---      ntype=2
          if(ntype_grid1.eq.2)then
            do j=2,n_grid1-1
            do i=1,m_grid1
             tmp(i,j)=0.5*(var_grid1(i,j)+var_grid1(i,j-1))
            enddo
            enddo

            do i=1,m_grid1
```

```
             tmp(i,1)=0.5*(3.*var_grid1(i,1)-var_grid1(i,2))
             tmp(i,n_grid1)=0.5*(3.*var_grid1(i,n_grid1-1)
     &                          -var_grid1(i,n_grid1-2))
           enddo
         endif
! ---     ntype=3
         if(ntype_grid1.eq.3)then
           do j=1,n_grid1
           do i=2,m_grid1-1
            tmp(i,j)=0.5*(var_grid1(i,j)+var_grid1(i-1,j))
           enddo
           enddo

           do j=1,n_grid1
             tmp(1,j)=0.5*(3.*var_grid1(1,j)-var_grid1(2,j))
             tmp(m_grid1,j)=0.5*(3.*var_grid1(m_grid1-1,j)
     &                          -var_grid1(m_grid1-2,j))
           enddo
         endif

       endif

! --- interpolation/extrapolation

       do j=1,n_grid2
       do i=1,m_grid2

           var_grid2(i,j)=(S1(i,j)*tmp(nx1(i,j),ny1(i,j))
     .                  +S2(i,j)*tmp(nx2(i,j),ny2(i,j))
     .                  +S3(i,j)*tmp(nx3(i,j),ny3(i,j)))
     .                  /Sc(i,j)

       enddo
       enddo

! ---  for staggered grid2

       if (grid2_stag.and.ntype_grid2.ne.0)then
       do j=1,n_grid2
       do i=1,m_grid2
         tmp(i,j)=var_grid2(i,j)
       enddo
       enddo

! ---  ntype_grid2 = 1
         if(ntype_grid2.eq.1)then
```

```
             do j=1,n_grid2-1
             do i=1,m_grid2-1
               var_grid2(i,j)=0.25*(tmp(i,j)+tmp(i+1,j)
     &                          +tmp(i+1,j+1)+tmp(i,j+1))
             enddo
             enddo
           endif
 ! ---  ntype_grid2 = 2
           if(ntype_grid2.eq.2)then
             do j=1,n_grid2-1
             do i=1,m_grid2
               var_grid2(i,j)=0.5*(tmp(i,j)+tmp(i,j+1))
             enddo
             enddo
           endif
 ! ---  ntype_grid2 = 3
           if(ntype_grid2.eq.3)then
             do j=1,n_grid2
             do i=1,m_grid2-1
               var_grid2(i,j)=0.5*(tmp(i,j)+tmp(i+1,j))
             enddo
             enddo
           endif

         endif

         return
         end
```

3.15 Subroutine output

The subroutine output a variable to the file named 'data'//file_name//'.dat'. It is used to test the code.

⟨*⟩+≡
```
! ---------------------------------------------------------------
        subroutine output(mp,np,num_file,varb)

        implicit none
        include 'pass.h'
        real varb(Nx_Max,Ny_Max)
        character*2 file_name
        integer nm_first,nm_second,nm_third,nm_fourth,np,mp,
     &          num_file

        nm_first=mod(num_file/1000,10)
        nm_second=mod(num_file/100,10)
        nm_third=mod(num_file/10,10)
        nm_fourth=mod(num_file,10)

        write(file_name(1:1),'(I1)')nm_third
        write(file_name(2:2),'(I1)')nm_fourth
c        write(file_name(3:3),'(I1)')nm_third
c        write(file_name(4:4),'(I1)')nm_fourth

        open(2,file='data'//file_name//'.dat')
        do j=1,np
        write(2,100)(varb(i,j),i=1,mp)
100     format(801f16.8)
        enddo
        close(2)

        return

        end
```

3.16 Subroutine SediModule

The subroutine is the Sediment module.

SediModule is called by

1. *Master*

⟨*⟩+≡

```
c -------------------------------------------------------------------
        subroutine SediModulesample()
        implicit none
        include 'pass.h'

        if(Master_Start.eq.1)then

          print*,'Sediment module initialization ...'
        else
         print*, 'call Sediment module ...'
        endif

        return
        end
```

3.17 Subroutine Mexport

The subroutine is for model output.

Mexport is called by

1. *Master*

⟨*⟩+≡

```
c ---------------------------------------------------------------------
        subroutine  Mexport()
        implicit none
        include 'pass.h'

        print*,'mexport routine'

        open(2,file=f_name13)
          do i=1,Nx_Wave
            write(2,111)(Pass_Height(i,j),j=1,Ny_Wave)
          enddo
        close(2)

        open(2,file=f_name14)
          do i=1,Nx_Circ
            write(2,111)(Pass_U(i,j),j=1,Ny_Circ)
          enddo
        close(2)

        open(2,file=f_name15)
          do i=1,Nx_Circ
            write(2,111)(Pass_V(i,j),j=1,Ny_Circ)
          enddo
        close(2)

        open(2,file=f_name16)
          do i=1,Nx_Circ
            write(2,111)(Pass_eta(i,j),j=1,Ny_Circ)
          enddo
        close(2)


111     format(500f16.6)


        return
       end
```

3.18 Subroutine WaveModule

The subroutine is the Wave module.
WaveModule is called by

1. *Master*

⟨*⟩+≡

```
c ----------------------------------------------------------
        subroutine WaveModulesample()
        implicit none
        include 'pass.h'

        if(Master_Start.eq.1)then

        print*,'wave module initialization ...'

c          write(*,*)'Do you want to run refdifs? Yes=1'
c         read(*,*)ikey
c         if(ikey.eq.1)then
c            call refdifs
c         else
c            call load_wave
c          endif

        else

        print*, 'call wave module ...'
c         call refdifs

        endif

        return
        end
```

3.19 Subroutine CircModule

The subroutine is the circulation module.
CircModule is called by

1. *Master*

⟨*⟩+≡

```
c -----------------------------------------------------------
      subroutine CircModulesample()
      implicit none
      include 'pass.h'

      if(Master_Start.eq.1)then

      print*,'circulation module initialization ...'

      else

      print*, 'call circulation module ...'

      endif

      return
      end
```

3.20 Subroutine MasterInit

The subroutine initializes all pass variables.
MasterInit is called by

1. *Master*

⟨*⟩+≡

```
c --------------------------------------------------------
        subroutine MasterInit
        implicit none
        include 'pass.h'

        do j=1,Ny_Max
        do i=1,Nx_Max
          Pass_Sxx(i,j)=0.
          Pass_Sxy(i,j)=0.
          Pass_Syy(i,j)=0.

          Pass_Sxx_body(i,j)=0.
          Pass_Sxy_body(i,j)=0.
          Pass_Syy_body(i,j)=0.

          Pass_Sxx_surf(i,j)=0.
          Pass_Sxy_surf(i,j)=0.
          Pass_Syy_surf(i,j)=0.

          Pass_Wave_Fx(i,j)=0.
          Pass_Wave_Fy(i,j)=0.

          Pass_MassFluxU(i,j)=0.
          Pass_MassFluxV(i,j)=0.
          Pass_MassFlux(i,j)=0.

          Pass_Diss(i,j)=0.
          Pass_WaveNum(i,j)=0.
          Pass_Theta(i,j)=0.
          Pass_ubott(i,j)=0.
          Pass_Height(i,j)=0.
          Pass_C(i,j)=0.
          Pass_Cg(i,j)=0.
          Intp_U_Wave(i,j)=0.
          Intp_V_Wave(i,j)=0.
          Intp_eta_Wave(i,j)=0.
          Pass_ibrk(i,j)=0

          Pass_U(i,j)=0.
```

```
    Pass_V(i,j)=0.
    Pass_Ub(i,j)=0.
    Pass_Vb(i,j)=0.
    Pass_eta(i,j)=0.

    Pass_d11(i,j)=0.
    Pass_d12(i,j)=0.
    Pass_e11(i,j)=0.
    Pass_e12(i,j)=0.
    Pass_f11(i,j)=0.
    Pass_f12(i,j)=0.

    Pass_fw(i,j)=0.

    Intp_Fx_Circ(i,j)=0.
    Intp_Fy_Circ(i,j)=0.
    Intp_ubott_Circ(i,j)=0.
    Intp_Theta_Circ(i,j)=0.
    Intp_Sxx_Circ(i,j)=0.
    Intp_Sxy_Circ(i,j)=0.
    Intp_Syy_Circ(i,j)=0.
    Intp_Sxx_Surf(i,j)=0.
    Intp_Sxy_Surf(i,j)=0.
    Intp_Syy_Surf(i,j)=0.
    Intp_Sxx_Body(i,j)=0.
    Intp_Sxy_Body(i,j)=0.
    Intp_Syy_Body(i,j)=0.
    Intp_MassFluxU_Circ(i,j)=0.
    Intp_MassFluxV_Circ(i,j)=0.
    Intp_Diss_Circ(i,j)=0.
    Intp_ibrk_Circ(i,j)=0.

    Pass_Dupdated(i,j)=Depth_Sedi(i,j)
    Intp_U_Sedi(i,j)=0.
    Intp_V_Sedi(i,j)=0.
    Intp_Ub_Sedi(i,j)=0.
    Intp_Vb_Sedi(i,j)=0.
    Intp_ubott_Sedi(i,j)=0.
    Intp_eta_Sedi(i,j)=0.
    Intp_fw_Sedi(i,j)=0.
    Intp_Theta_Sedi(i,j)=0.
    Intp_Height_Sedi(i,j)=0.
    Intp_ibrk_Sedi(i,j)=0.

  enddo
  enddo
```

```
 Pass_period = 1.

return
end
```

4 Frequently Asked Questions

1. If I have only two modules coupled, e.g., WaveModule and CircModule, how to set the model?

   Set N_Interval_CallSedi = -1 and make an empty subroutine SediModule as below

   ```
   subroutine SediModule()

   end
   ```

2. when three modules use three different grid systems, does the master program cost a lot of time during data transfer and interpolation?

   Because all the interpolation/extrapolation coefficients are obtained in the model initialization, the master program does not cost too much time during time integration. The interpolation/extrapolation is actually operated based on a simple formula given by Equ. (2).

3. Can I use a unstructured grid?

   Yes. See **Unstructured Grid** in 2.4.

4. If the three modules are in the same grid system, do I still need to provide three grid files?

   No. You may only provide Mast_Grid file and set null strings at module-grid-name locations in minput.dat. For example, set F_xycirc = ' '

5. How to pass a new variable from a module to another?

   We listed some possible passing variables in pass.h. The names for passing variables are standard long names such as 'Pass_MassFlux'. If you want to pass a new variable that is not in the list, you may add it in pass.h and modify the code in the corresponding subroutine interp_xxxx_xxxx. Please also inform us the modification for code updating.

6. How to tell the master program I want to pass a variable from one module to another?

   We use pass-control parameters, e.g., Wave_To_Circ_MassFlux, listed in minput.dat to control which variable will be transfer from a module to another. For example, Wave_To_Circ_MassFlux = .true. means the short wave flux will be passed (interpolated) from the wave module to the circulation module.

7. Where to input water depth?

   Water depth should be input in the master program and on Mast_Grid. The water depths on module grids are then obtained by interpolations from the Mast_Grid. We do not recommend reading water depth in a specific module since the water depth would be updated by the sediment module.

8. When passing a vector defined by tangential or normal direction in curvilinear coordinates, can the master program handle the vector rotation?

   No. You should rotate the vector into the reference geographic coordinate system before passing it. See **Coordinate systems** in 2.3.

9. Can we pass the contravariant radiation stresses when we use a curvilinear model?

   No. The master program does not handle the transformation of second-order tensors. It is suggested that the short wave forcing be calculated using the contravariant radiation stresses and then interpolated into other modules.