MASTER PROGRAM
FOR NEARSHORE COMMUNITY MODEL
Version 2005.4
Documentation and User’s Manual

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

NOPP Nearshore Community Model Research Group
April, 2005
Contents

1 Introduction 3

2 Master Program Outline 3
   2.1 Functions of Master Program 3
   2.2 Flow Chart and Pseudo-Code 4
   2.3 Parameters and Pass Variables Between Modules 4
   2.4 Subroutines in Master Program 10
   2.5 Interpolation/Extrapolation between model grids 11
   2.6 Input for Master Program 12

3 Link Master Program and Three Modules 15
   3.1 Necessary Modifications of Three Modules 15
   3.2 Units 24
   3.3 Coordinate systems 24
   3.4 One Dimensional Grid 25
   3.5 Unstructured Grid 26

4 noweb Documentation of the Master Program 26
   4.1 Main Program 26
   4.2 Subroutine readfile 31
   4.3 Subroutine get_interpolation_coef 36
   4.4 Subroutine interp_depth 39
   4.5 Subroutine interp_circ_wave 40
   4.6 Subroutine interp_sedi_wave 42
   4.7 Subroutine interp_wave_circ 43
   4.8 Subroutine interp_sedi_circ 49
   4.9 Subroutine interp_wave_sedi 50
   4.10 Subroutine interp_circ_sedi 52
   4.11 Subroutine interp_same 56
   4.12 Subroutine interpolation 58
   4.13 Subroutine interpolation_nonstruc 66
   4.14 Subroutine grid1_to_grid2 71
   4.15 Subroutine output 76
   4.16 Subroutine SediModule 77
   4.17 Subroutine Mexport 78
   4.18 Subroutine WaveModule 81
   4.19 Subroutine CircModule 82
   4.20 Subroutine MasterInit 83

5 Frequently Asked Questions 86


1 Introduction

The master program controls the interaction between the wave, circulation and sediment transport modules. It also handles the interpolation (and possibly extrapolation) between the grids used by the different modules. The frequency with which each module is called is determined within the master program.

The basic framework of the master program assumes that there is a wave module which provides forcing for a circulation module while both the wave and circulation module provide driving to the sediment transport module. It is possible to run the wave module with inactive circulation and sediment transport modules. If there is other forcing (i.e. wind or surface heating) in the circulation module, it is possible to run the circulation module with an inactive wave module. The sediment transport module requires input from at least one of the other modules.

There is provision within the master program for further optional interactions between the modules. The currents (depth integrated or 3-dimensional (3D)) from the circulation module may be returned to the wave module where they act to refract the wave field. The sediment transport module may return updated topography to either or both of the other modules.

The master program can handle interpolation to either a structured or unstructured grid. The variables on the structured grid may be staggered in space. Velocities passed to and from the master program by the modules are interpolated as scalars (U and V) so that rotation into the directions defined by the rectilinear master grid must be done within the modules.

Output of basic fields interpolated to the master grid is handled by the master program. Other output such as time series or derived fields such as momentum balances may optionally be done by the modules. This optional output is on the grid particular to the module in question.

2 Master Program Outline

2.1 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.

2.2 Flow Chart and Pseudo-Code

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

A pseudo-code is described in the following.

Program start
input depth, coordinates, and parameters needed by master program
calculate inter(extra)polation coefficients defined in “interp.h”
depth inter(extra)polation from mast_grid to module_grid
master program initialization
wave, circulation and sediment module initialization
time loop start
if it is time to call wave module
update inter(extra)polation info from circ and sedi module results if needed
call wave module
endif
if it is time to call circulation module
update inter(extra)polation info from wave and sedi module results if needed
call circulation module
endif
if it is time to call sediment module
update inter(extra)polation info from wave and circ module results if needed
call sediment module
endif
if it is time for output
call output subroutine
endif
time loop end
program end

2.3 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

- \( \text{Nx}_\text{Max} \) and \( \text{Ny}_\text{Max} \): maximum grid numbers for master grid and all three modules
- \( \text{Nx}_\text{Mast} \) and \( \text{Ny}_\text{Mast} \): grid dimension of master grid
- \( \text{Nx}_\text{Wave} \) and \( \text{Ny}_\text{Wave} \): grid dimension of wave grid
Figure 1: Flow Chart
2. Wave Module

- \( Pass_{Sxx}, Pass_{Sxy}, \) and \( Pass_{Syy} \): radiation stresses
- \( Pass_{Sxx}_{\text{body}}, Pass_{Sxy}_{\text{body}}, Pass_{Syy}_{\text{body}} \): local radiation stresses (body part)
- \( Pass_{Sxx}_{\text{surf}}, Pass_{Sxy}_{\text{surf}}, Pass_{Syy}_{\text{surf}} \): local radiation stresses (surface part)
- \( Pass_{\text{Wave}_Fx}, Pass_{\text{Wave}_Fy} \): wave forcing
- \( Pass_{\text{MassFluxU}}, Pass_{\text{MassFluxV}} \): mass flux
- \( Pass_{\text{Diss}} \): dissipation caused by wave breaking
- \( Pass_{\text{WaveNum}} \): wave number
- \( Pass_{\text{Theta}} \): wave angle
- \( Pass_{\text{ubott}} \): bottom velocity
- \( Pass_{\text{Height}} \): wave height
- \( Pass_{\text{C}} \): phase velocity
- \( Pass_{\text{Cg}} \): group velocity
- \( Pass_{\text{Period}} \): wave period
- \( Pass_{\text{ibrk}} \): wave breaking index (1 - breaking, 0 - nonbreaking)
- \( \text{Intp}_{U_{\text{Wave}}} \) and \( \text{Intp}_{V_{\text{Wave}}} \): current velocity interpolated from the circulation module.
- \( \text{Intp}_{\text{eta}_{\text{Wave}}} \): surface elevation interpolated from the circulation module.

3. Circulation Module

- \( Pass_{U} \) and \( Pass_{V} \): depth-averaged velocity
- \( Pass_{Ub} \) and \( Pass_{Vb} \): bottom current velocity
- \( Pass_{\eta}\text{ta} \): surface elevation (time-averaged)
- \( Pass_{f11}, Pass_{f12}, Pass_{f11}, Pass_{f12}, Pass_{f11} \) and \( Pass_{f12} \): coefficients for calculation of vertical velocity profile (SHORECIRC)
- \( Pass_{fw} \): bottom friction coefficient
- \( Pass_{vt} \): turbulent eddy viscosity coefficient
- \( \text{Intp}_{Fx_{\text{Circ}}} \) and \( \text{Intp}_{Fy_{\text{Circ}}} \): short wave forcing interpolated from the wave module
- \( \text{Intp}_{\text{ubott}_{\text{Circ}}} \): bottom velocity interpolated from the wave module
- \( \text{Intp}_{\text{Theta}_{\text{Circ}}} \): wave angle interpolated from the wave module

- \( Nx_{\text{Circ}} \) and \( Ny_{\text{Circ}} \): grid dimension of circulation grid
- \( Nx_{\text{Sedi}} \) and \( Ny_{\text{Sedi}} \): grid dimension of sediment grid
Intp Sxx_Circ, Intp Sxy_Circ, and Intp Syy_Circ: radiation stresses interpolated from the wave module

Intp Sxx_Surf, Intp Sxy_Surf, and Intp Syy_Surf: local radiation stresses (surface part) interpolated from the wave module

Intp Sxx_Body, Intp Sxy_Body, and Intp Syy_Body: local radiation stresses (body part) interpolated from the wave module

Intp Height_Circ: wave height interpolated from the wave module

Intp Theta_Circ: wave angle interpolated from the wave module

Intp WaveNum_Circ: wave number interpolated from the wave module

Intp C_Circ: wave phase velocity interpolated from the wave module

Intp MassFluxU_Circ and Intp MassFluxV_Circ: mass flux interpolated from the wave module

Intp Diss_Circ: energy dissipation interpolated from the wave module

Intp ibrk_Circ: wave breaking index interpolated from the wave module

4. Sediment Module

Pass Dupdated: updated water depth

Pass SediFluxcum_x: accumulated sediment flux (x component) calculated in the circulation module.

Pass SediFluxcum_y: accumulated sediment flux (y component) calculated in the circulation module.

Intp SediFluxcum_x: accumulated sediment flux (x component) interpolated from the circulation grid to sediment grid

Intp SediFluxcum_y: accumulated sediment flux (y component) interpolated from the circulation grid to sediment grid

Intp U_Sedi and Intp V_Sedi: current velocity (depth averaged) interpolated from the circulation module.

Intp Ub_Sedi and Intp Vb_Sedi: bottom current velocity interpolated from the circulation module.

Intp ubott_Sedi: bottom wave velocity interpolated from the wave module

Intp eta_Sedi: surface elevation interpolated from the circulation module

Intp fw_Sedi: bottom friction coefficient interpolated from the circulation module

Intp vt_Sedi: viscosity coefficient interpolated from the circulation module
- Intp_Theta_Sedi: wave angle interpolated from the wave module
- Intp_Height_Sedi: wave height interpolated from the wave module
- Intpibrk_Sedi: wave breaking index interpolated from the wave module

5. Coordinate System, Water Depth
- Depth_Mast: water depth on Mast_Grid, can be updated by the sediment module
- Depth_Wave: water depth on Wave_Grid, can be updated by the sediment module
- Depth_Circ: water depth on Circ_Grid, can be updated by the sediment module
- Depth_Sedi: water depth on Sedi_Grid, can be updated by the sediment module
- X_Mast and Y_Mast: (x,y) of Mast_Grid
- X_Wave and Y_Wave: (x,y) of Wave_Grid
- X_Circ and Y_Circ: (x,y) of Circ_Grid
- X_Sedi and Y_Sedi: (x,y) of Sedi_Grid

6. Other Variables
- U_wind_Mast and V_wind_Mast: wind speed on Mast_Grid
- U_wind_Circ and V_wind_Circ: wind speed on Circ_Grid
- U_wind_Wave and V_wind_Wave: wind speed on Wave_Grid

7. Control Parameters
- 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.
- 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.
- 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.
- N_Delay_CallSedi: time (steps) delay for calling sediment module.
- N_Interval_Output: step interval for output
- Total_Time: total simulation time (in seconds)
- Master_dt: time step in the master program
• 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.

• 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.

• 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.

• 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.

• Wave\_Staggered, Circ\_Staggered, and Sedi\_Staggered: logical parameters indicating whether or not grid systems are staggered.

• Wave\_Structured, Circ\_Structured, and Sedi\_Structured: logical parameters indicating whether or not grid system is structured.

• 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.

• 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

• Wave\_To\_Circ\_Height: pass wave heights from the wave module to the circulation module

• Wave\_To\_Circ\_Angle: pass wave angles (degree) from the wave module to the circulation module

• Wave\_To\_Circ\_WaveNum: pass wave numbers from the wave module to the circulation module

• Wave\_To\_Circ\_C: pass wave phase velocities from the wave module to the circulation module

• Wave\_To\_Circ\_Radiation: pass radiation stresses from the wave module to the circulation module
- Wave_To_Circ_Rad_Surf: pass local radiation stresses (surface part) from the wave module to the circulation module
- Wave_To_Circ_Rad.Body: pass local radiation stresses (body part) from the wave module to the circulation module
- Wave_To_Circ_Forcing: pass short wave forcing from the wave module to the circulation module
- Wave_To_Circ_MassFlux: pass mass flux from the wave module to the circulation module
- Wave_To_Circ_Dissipation: pass wave energy dissipation from the wave module to the circulation module
- Wave_To_Circ_BottomUV: pass bottom velocity from the wave module to the circulation module
- Wave_To_Circ_Brkindex: pass wave breaking index from the wave module to the circulation module
- Circ_To_Wave_UV: pass current velocity (depth averaged) from the circulation module to the wave module
- Circ_To_Wave_eta: pass surface elevation from the circulation module to the wave module
- Wave_To_Sedi_Height: pass wave height from the wave module to the sediment module
- Wave_To_Sedi_Angle: pass wave angle from the wave module to the sediment module
- Wave_To_Sedi_BottomUV: pass bottom velocity from the wave module to the sediment module
- Circ_To_Sedi_UV: pass current velocity from the circulation module to the sediment module
- Circ_To_Sedi_UVb: pass bottom current velocity from the circulation module to the sediment module
- Circ_To_Sedi_eta: pass surface elevation from the circulation module to the sediment module
- Circ_To_Sedi_UV3D: pass 3D current velocity from the circulation module to the sediment module
- Circ_To_Sedi_fw: pass bottom friction coefficient from the circulation module to the sediment module
- Circ_To_Sedi_Vquasi3D: pass coefficients of quasi-3D current profiles from the circulation module to the sediment module
- Sedi_To_Wave_Depth: pass updated water depth from the sediment module to the wave module
- Sedi_To_Circ_Depth: pass updated water depth from the sediment module to the circulation module
• Circ_POM: use POM as the circulation module, it helps to initialize logical variables
• Circ_SC: use SHORECIRC as the circulation module, it helps to initialize logical variables

9. Vector Rotation

• Circ_Rotate_Angle: angle (°) between the vector reference direction on Circ_Grid and the geographic reference direction
• Wave_Rotate_Angle: angle (°) between the vector reference direction on Wave_Grid and the geographic reference direction
• Sedi_Rotate_Angle: angle (°) between the vector reference direction on Sedi_Grid and the geographic reference direction

2.4 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`.

### 2.5 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:

\[
S_{\alpha\beta\gamma} = \begin{vmatrix}
  x_\alpha & y_\alpha & 1 \\
  x_\beta & y_\beta & 1 \\
  x_\gamma & y_\gamma & 1
\end{vmatrix}
\]  

(1)

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

\[
F_A = \frac{(F_1 S_{23A} + F_2 S_{31A} + F_3 S_{12A})/S_{123}}{S_{123}}
\]  

(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.
Figure 2: Interpolation triangle.
2.6 Input for Master Program

1. F_NAMES: definition of input and output file names.
   - f_depth - input file name for depth on Mast_Grid
   - f_xymast - input file name for (x,y) of Mast_Grid
   - f_xywave - input file name for (x,y) of Wave_Grid
   - f_xycirc - input file name for (x,y) of Circ_Grid
   - f_xysedi - input file name for (x,y) of Sedi_Grid
   - ...

2. GRIDIN: grid information.
   - Nx_Mast, Ny_Mast: Mast_Grid dimension
   - Nx_Wave, Ny_Wave: Wave_Grid dimension
   - Nx_Circ, Ny_Circ: Circ_Grid dimension
   - Nx_Sedi, Ny_Sedi: Sedi_Grid dimension
   - Grid_Mast_Wave_Same - logical parameter indicating whether Wave_Grid is the same as Mast_Grid
   - Grid_Mast_Circ_Same - logical parameter indicating whether Circ_Grid is the same as Mast_Grid
   - Grid_Mast_Sedi_Same - logical parameter indicating whether Sedi_Grid is the same as Mast_Grid
   - Wave_Staggered - logical parameter indicating whether Wave_Grid is staggered
   - Circ_Staggered - logical parameter indicating whether Circ_Grid is staggered
   - Sedi_Staggered - logical parameter indicating whether Sedi_Grid is staggered
   - Wave_Structured - logical parameter indicating whether or not Wave_Grid is structured.
   - Circ_Structured - logical parameter indicating whether or not Circ_Grid is structured.
   - Sedi_Structured - logical parameters indicating whether or not Sedi_Grid is structured
   - 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
   - Wave_Curr_Interact - logical parameter for two-way coupling between the wave and circulation modules
• Wave_Bed_Interact - logical parameter for two-way coupling between the wave and sediment modules
• Curr_Bed_Interact - logical parameter for two-way coupling between the circulation and sediment modules

4. PASSVARIABLES

• Wave_To_Circ_Height - pass wave height from the wave module to the circulation module
• Wave_To_Circ_Angle - pass wave angle from the wave module to the circulation module
• Wave_To_Circ_WaveNum - pass wave numbers from the wave module to the circulation module
• Wave_To_Circ_Radiation - pass wave phase velocity from the wave module to the circulation module
• Wave_To_Circ_Radiation - pass radiation stresses from the wave module to the circulation module
• Wave_To_Circ_Rad_Surf - pass local radiation stresses (surface part) from the wave module to the circulation module
• Wave_To_Circ_Rad_Body - pass local radiation stresses (body part) from the wave module to the circulation module
• Wave_To_Circ_Forceing - pass short wave forcing from the wave module to the circulation module
• Wave_To_Circ_MassFlux - pass mass flux from the wave module to the circulation module
• Wave_To_Circ_Dissipation - pass wave energy dissipation from the wave module to the circulation module
• Wave_To_Circ_BottomUV - pass Bottom velocity from the wave module to the circulation module
• Wave_To_Circ_Brkindex - pass wave breaking index from the wave module to the circulation module
• Circ_To_Wave_UV - pass current velocity (depth averaged) from the circulation module to the wave module
• Circ_To_Wave_eta - pass surface elevation from the circulation module to the wave module
• Wave_To_Sedi_Height - pass wave height from the wave module to the sediment module
• Wave_To_Sedi_Angle - pass wave angle from the wave module to the sediment module
• Wave_To_Sedi_BottomUV - pass bottom velocity from the wave module to the sediment module
- Circ_To_Sedi_SedFlux - pass sediment flux from the circulation module to the sediment module
- Circ_To_Sedi_UV - pass current velocity from the circulation module to the sediment module
- Circ_To_Sedi_UVb - pass bottom current velocity from the circulation module to the sediment module
- Circ_To_Sedi_eta - pass surface elevation from the circulation module to the sediment module
- Circ_To_Sedi_UV3D - pass 3D current velocity from the circulation module to the sediment module
- Circ_To_Sedi_fw - pass bottom friction coefficient from the circulation module to the sediment module
- Circ_To_Sedi_UVquasi3D - pass coefficients of quasi-3D current profiles from the circulation module to the sediment module
- Sedi_To_Wave_Depth - pass updated water depth from the sediment module to the wave module
- Sedi_To_Circ_Depth - pass updated water depth from the sediment module to the circulation module

5. VECTORROTATE

- Circ_Rotate_Angle - angle (°) between the vector reference direction on Circ_Grid and the geographic reference direction
- Wave_Rotate_Angle - angle (°) between the vector reference direction on Wave_Grid and the geographic reference direction
- Sedi_Rotate_Angle - angle (°) between the vector reference direction on Sedi_Grid and the geographic reference direction

6. TIMEIN: time stepping control.

- Total_Time - total computational time
- Master_dt - time step in master program.
- 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
- 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
- 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
- N_Delay_CallSedi - time (steps) delay for calling the sediment module
- N_Interval_Output - interval steps for output
3 Link Master Program and Three Modules

3.1 Necessary Modifications of 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 module subroutines in which you will use master-related variables such as Pass.xxx, Intp.xxx. Notice that, for some Fortran compilers, values of module variables may not be kept when returning to the master program. If it happens, try to include the module common blocks in the master program.

3. split each module code into initialization part and time integration part, if necessary.

   In the following example, init_shorecirc is the initialization subroutine of SHORECIRC. Master_Start is a switch parameter which varies between 1 and 0 in the master program. “Master_Start = 1” makes an initialization call and “Master_Start = 0” for a model time integration call. ntime is the internal loop number of the module and can be calculated by
   \[ ntime = \frac{\text{Master}\_dt}{\text{Module}\_dt} \]

   (example)\(\equiv\)

   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
A module initialization subroutine should include variable initializations, module parameter input and output, and passing grid information or water depth (if needed for initialization). The following example is modifications in SHORECIRC

```fortran
example:=

nx=Nx_Circ
ny=Ny_Circ

do j=1,ny
do i=1,nx
   x(i,j)=X_Circ(i,j)
   y(i,j)=Y_Circ(i,j)
enddo
enddo

do j=1,ny
do i=1,nx
   dr(i,j)=depth_circ(i,j)
enddo
enddo
```
Table 1: Names for input files.

<table>
<thead>
<tr>
<th>Namelist variable</th>
<th>type</th>
<th>default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>fnames</td>
<td>fdepth</td>
<td>' '</td>
</tr>
<tr>
<td></td>
<td>fxymast</td>
<td>' '</td>
</tr>
<tr>
<td></td>
<td>fxywave</td>
<td>' '</td>
</tr>
<tr>
<td></td>
<td>fxycirc</td>
<td>' '</td>
</tr>
<tr>
<td></td>
<td>fxsedi</td>
<td>' '</td>
</tr>
<tr>
<td></td>
<td>name6</td>
<td>' '</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>name16</td>
<td>' '</td>
</tr>
</tbody>
</table>

4. specify parameters in ‘minput.dat’

The file minput.dat is the input file that determines the behavior of the master program and thus the interactions between the various modules. The input files for the initial topography and each of the grids (master, wave, circulation and sediment transport) are specified as well as the description of these grids. The optional module interactions are determined and the variables to be passed to each module are listed. The frequency of module calls, the duration of the run and the output frequency are specified.

In this file a wild card “*” may be used in file names or variable names to indicate that the statement applies to all variables of the same form. For example fxy* refers to any of the variables fxymast, fxywave, fxycirc and fxsedi. In particular examples, the wild card “?” is used as well two indicate matching pairs as in: if fxy? = “ then Grid_Mast?_Same is set .true.

The file used the FORTRAN namelist input with six lists of variables . Names for input files as shown in Table 1, fdepth, water depth (topography) on master grid. fxy*, locations of the xy pairs for the master grid, location of x y pairs for the wave, circulation and sediment modules respectively. If the grids are identical to the master grid the additional file names do not need to be specified. If file names are not specified the master grid is used and if fxy? = ” then Grid_Mast?_Same is set .true..

Variables associated with computational grids are defined in Table 2. Nx_Mast, Ny_Mast, Nx_Wave etc. define the number of x and y points in the grid for the master program and each module (wave, circulation and sediment transport). The maximum value of the grid sizes Nx_Max and Ny_Max are set in the file pass.h. The logical variables Grid_Mast_Wave_Same define the relationship between the master grid (required to be rectilinear) and the grids for the modules. These are set to .true. if the respective grids are identical. The logical variables Wave_Staggered etc. are set true for grids in which different variables are at different points. The lo-
<table>
<thead>
<tr>
<th>Namelist</th>
<th>variable</th>
<th>type</th>
<th>default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>gridin</td>
<td>Nx_Mast</td>
<td>integer</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>Ny_Mast</td>
<td>integer</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>Nx_Circ</td>
<td>integer</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>Ny_Circ</td>
<td>integer</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>Nx_Wave</td>
<td>integer</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>Ny_Wave</td>
<td>integer</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>Nx_Sedi</td>
<td>integer</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>Ny_Sedi</td>
<td>integer</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>Grid_Mast_Wave_Same</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td></td>
<td>Grid_Mast_Circ_Same</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td></td>
<td>Grid_Mast_Sedi_Same</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td></td>
<td>Wave_Staggered</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td></td>
<td>Circ_Staggered</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td></td>
<td>Sedi_Staggered</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td></td>
<td>Wave_Stag_huv(3)</td>
<td>integer</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>Circ_Stag_huv(3)</td>
<td>integer</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>Sedi_Stag_huv(3)</td>
<td>integer</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>Wave_Structured</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td></td>
<td>Circ_Structured</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td></td>
<td>Sedi_Structured</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td></td>
<td>Grid_Extrapolation</td>
<td>logical</td>
<td>.false.</td>
</tr>
</tbody>
</table>

Table 2: Grid information
Table 3: Interaction control.

<table>
<thead>
<tr>
<th>Namelist interaction variable</th>
<th>type</th>
<th>default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wave_Bed_Interact</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Curr_Bed_Interact</td>
<td>logical</td>
<td>.false.</td>
</tr>
</tbody>
</table>

Definitions of variables are described by the integer arrays Wave_Stag_huv. Variables located at the center of the grid cell are indicated by a value of 1, center of the cell left side is 2 and center of the grid bottom is 3. Wave_Stag_huv(1)=n determines the location of the water depth variable (as well as surface elevation and scalar fields). Wave_Stag_huv(2)=n determines the location of the across shore velocity u, Wave_Stag_huv(3)=n determines the location of the along shore velocity v. For example, the Arakawa c grid is given by Wave_Stag_huv(1)=1, Wave_Stag_huv(2)=2, Wave_Stag_huv(3)=3.

The logical variables Wave_structured are set .true. if the grid is unstructured. Grid_Extrapolation is set .true. if extrapolation of values beyond the boundaries of a grid is desired.

Logical variables for model interaction control are defined as shown in Table 3. Wave_Curr_Interact is set .true. if the currents are passed to the wave module to include the refraction of the waves by the currents. Wave_Bed_Interact, Curr_Bed_Interact are set .true. if the water depth is updated by the sediment transport module and the results passed to the wave module or the circulation module respectively.

The variables shown in Table 4 define those that the master program is currently set up to pass for the first named module to the second. The last two are blanket definitions for the two currently implemented circulation modules. If either of these is set to true the appropriate list of Wave_to_Circ* variables are set true and consistency is checked that the circulation module can provide the Circ_to_* variables requested by the other modules. Note that not all modules of a particular either provides or needs the entire list of potential passed variables.

Table 5 shows the defined angle between the master grid and the grids of the various modules. This is spatially constant and is the angle between the direction of the vectors passed back to the master program by the circulation or wave modules and the direction given by the rectilinear master grid. These vector fields include velocities passed from the circulation and forcing terms and wave mass flux passed from the wave module. The wave direction array is changed by the appropriate constant value as it is passed to the other modules as well.

In Table 6, Total_Time is the duration of the model run in seconds. Master_dt is the time interval in seconds between calls to the most frequently called module. N_Interval_Call* is the number of Master_dt time steps be-
<table>
<thead>
<tr>
<th>Namelist variable</th>
<th>type</th>
<th>default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wave_to_Circ_Height</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Wave_to_Circ_Angle</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Wave_to_Circ_WaveNum</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Wave_to_Circ_C</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Wave_to_Circ_Cg</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Wave_to_Circ_Radiation</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Wave_to_Circ_Rad_Surf</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Wave_to_Circ_Rad_Body</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Wave_to_Circ_Forcing</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Wave_to_Circ_MassFlux</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Wave_to_Circ_Dissipation</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Wave_to_Circ_BottomUV</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Wave_to_Circ_Brkiindex</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Circ_to_Wave_UV</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Circ_to_Wave_eta</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Wave_to_Sedi_Height</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Wave_to_Sedi_Angle</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Wave_to_Sedi_BottomUV</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Circ_to_Sedi_UV</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Circ_to_Sedi_UVb</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Circ_to_Sedi_UV3D</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Circ_to_Sedi_fw</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Circ_to_Sedi_vt</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Circ_to_Sedi_UVquasi3D</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Sedi_to_Wave_Depth</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Sedi_to_Circ_Depth</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Waveupdat_for_Circ</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Waveupdat_for_Sedi</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Circupdat_for_Wave</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Circupdat_for_Sedi</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Sedipdat_for_Wave</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Sedipdat_for_Circ</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Circ_POM</td>
<td>logical</td>
<td>.false.</td>
</tr>
<tr>
<td>Circ_SC</td>
<td>logical</td>
<td>.false.</td>
</tr>
</tbody>
</table>

Table 4: Logical variables for passing variables
### Table 5: Vector rotation angles

<table>
<thead>
<tr>
<th>Namelist Variable</th>
<th>Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>vectorrotate Circ Rotate Angle</td>
<td>real</td>
<td>none</td>
</tr>
<tr>
<td>Wave Rotate Angle</td>
<td>real</td>
<td>none</td>
</tr>
<tr>
<td>Sedi Rotate Angle</td>
<td>real</td>
<td>none</td>
</tr>
</tbody>
</table>

### Table 6: Time control variables

<table>
<thead>
<tr>
<th>Namelist Variable</th>
<th>Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>timein Total Time</td>
<td>real</td>
<td>none</td>
</tr>
<tr>
<td>Master dt</td>
<td>real</td>
<td>none</td>
</tr>
<tr>
<td>N Interval CallWave</td>
<td>real</td>
<td>none</td>
</tr>
<tr>
<td>N Interval CallCirc</td>
<td>real</td>
<td>none</td>
</tr>
<tr>
<td>N Interval CallSedi</td>
<td>real</td>
<td>none</td>
</tr>
<tr>
<td>N Delay CallSedi</td>
<td>real</td>
<td>none</td>
</tr>
<tr>
<td>N Interval Output</td>
<td>real</td>
<td>none</td>
</tr>
</tbody>
</table>

tween calls to the corresponding module. N Delay CallSedi is the number of Master dt steps before the sediment is called. This allows the circulation and wave field to become established before the effect on sediment is calculated. N Interval Output is the number of Master dt steps between master program output. Note that the internal time step for modules such as the circulation modules POM and ShoreCirc are set in the input files for the module and are not known by the master program. The number of steps to be calculated at each call to any module that uses time stepping is calculated within the module and given by: nstep=N Interval Call dt / Master dt .

5. assign module variables to passing variables in your modules. The following are examples.

(a) wave modules

Pass Sxx(i,j) = Sxx(i,j)
Pass Sxy(i,j) = Sxy(i,j)
Pass Syy(i,j) = Syy(i,j)
Pass MassFluxU(i,j) = Uflux(i,j)
Pass MassFluxV(i,j) = Vflux(i,j)
Pass Diss(i,j) = disp(i,j)
Pass_ibrk(i,j) = indexbreak(i,j)
Pass_Theta(i,j) = angle(i,j)
Pass_Height(i,j) = height(i,j)
Pass_ubott(i,j) = ub(i,j)
Pass Cg(i,j) = Cg(i,j)
Pass C(i,j) = C(i,j)

(b) circulation modules
Pass U(i,j) = u(i,j)
Pass V(i,j) = v(i,j)
Pass fw(i,j) = fw(i,j)
Pass vt(i,j) = vt(i,j)

(c) Sediment modules
Pass Dupdated(i,j) = dep(i,j)

6. use variables passed and interpolated from other modules. The following are examples.

(a) Wave modules
ur(i,j) = Intp U Wave(i,j)
vr(i,j) = Intp V Wave(i,j)
dr(i,j) = Depth Wave(i,j) + Intp eta Wave(i,j)

(b) Circulation modules
depcirc(i,j) = Depth Circ(i,j)
sxx(i,j) = Intp Sxx Circ(i,j)
sxy(i,j) = Intp Sxy Circ(i,j)
syy(i,j) = Intp Syy Circ(i,j)
ub(i,j) = Intp ubott Circ(i,j)
wangle(i,j) = Intp Theta Circ(i,j)
wh(i,j) = Intp Height Circ(i,j)
Cw(i,j) = Intp C Circ(i,j)
Cgw(i,j) = Intp Cg Circ(i,j)
dissipation(i,j) = Intp Diss Circ(i,j)
ibrk(i,j) = Intp ibrk Circ(i,j)
fluxU(i,j) = Intp MassFluxU Circ(i,j)
fluxV(i,j) = Intp MassFluxV Circ(i,j)

(c) Sediment modules
currU(i,j) = Intp U Sedi(i,j)
currV(i,j) = Intp V Sedi(i,j)
bottU(i,j) = Intp Ub Sedi(i,j)
bottV(i,j) = Intp Vb Sedi(i,j)
wavebottu(i,j) = Intp ubott Sedi(i,j)
surface(i,j) = Intp eta Sedi(i,j)
bfric(i,j) = Intp fw Sedi(i,j)
vt(i,j) = Intp vt Sedi(i,j)
\[
\text{wave_angle}(i,j) = \text{Intp}_\Theta \text{Sedi}(i,j) \\
\text{wave_height}(i,j) = \text{Intp}_\text{Height} \text{Sedi}(i,j) \\
\text{wave}_\text{ibrk}(i,j) = \text{Intp}_\text{ibrk} \text{Sedi}(i,j)
\]

7. Makefile

There is a generic Makefile for the Nearshore Community Model. Before you use the Makefile, please make sure the master program, the computational modules and dependences in your model directory. In the Makefile, you may need to choose a Fortran compiler and flags to the Fortran compiler. The following is an example to choose the Intel Fortran Compiler as the compiler and setup flags to the compiler.

\[CFT = \text{ifc}
\]
\[\text{FFLAGS} = -O3 -w\]

For users who want to use the existing modules, you can choose one module (set true, see the following example) from each module group (i.e., wave, circulation, and sediment module groups). When you choose the FUNWAVE module as a circulation module, don’t choose any wave module from the wave module group (all wave modules should be set false). The following example shows a combination setup of REF/DIF-S, SHORECIRC, and BBB-sediment modules.

\[
\begin{align*}
\text{REFDIFS} & = \text{true} \\
\text{REFDIF1} & = \text{false} \\
\text{REFDIF}\_\text{SNL} & = \text{false} \\
\text{WAVE}\_\text{KENNEDY} & = \text{false} \\
\text{WAVE}\_\text{HERBERS} & = \text{false} \\
\text{WAVE}\_\text{TUBA} & = \text{false} \\
\text{SHORECIRC} & = \text{true} \\
\text{CURVCIRC} & = \text{false} \\
\text{POM} & = \text{false} \\
\text{FUNWAVE} & = \text{false} \\
\text{BBB} & = \text{true} \\
\text{HH} & = \text{false} \\
\text{RIB} & = \text{false} \\
\text{WAT} & = \text{false}
\end{align*}
\]

\[
\begin{align*}
\text{# --- master program, always true} \\
\text{MASTSC} & = \text{master.f}
\end{align*}
\]
For users who want to add a new module into the Nearshore Community Model, you need to modify the Makefile as shown in the following example. The new module is assumed as a wave module named “NewModule”.

(a) add the module name in the block of the wave module group and set “true”:

```
# | wave module group
NewModule = true
```

(b) list names of all the Fortran codes needed to be compiled (e.g., NewModule1.f NewModule2.f ... ) in the source code block (don’t list “.h”):

```
ifeq ($(NewModule),true)
    WAVESC = NewModule1.f NewModule2.f ...
endif
```

(c) list dependences (e.g., NewModule1.f, NewModule2.f, pass.h NewParam.h ...) in the dependences block:

```
# # # # # # # # # # # 
# # # # # # # # # # # 
# # # # # # # # # # # 
# NewModule1.o: NewModule1.f ... pass.h NewParam.h
```

3.2 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.

3.3 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., \(\theta_1\) and \(\theta_2\) in Figure 3. For example, in “minput.dat”, \(\text{Circ\_Rotate\_Angle} = 10\), 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 transferred 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 parameters “ 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 velocity 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”).

### 3.4 One Dimensional Grid

The master program allows to use one or more 1-D module/modules. The 1-D direction is assumed to be the x-direction (say onshore). For the 1-D module/modules, Ny_xxxx=1. When 1-D variables are interpolated/extrapolated into a 2-D grid, longshore uniform values are obtained on the 2-D grid. When 2-D variables are interpolated/extrapolated into a 1-D grid, longshore averaged values are used for the x-direction interpolation/extrapolation.

### 3.5 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
Figure 3: Example of module grids in geographical reference coordinates.
Figure 4: Staggered grid definition.
are defined in the module with a unstructured grid, Ny = 1 should be used in passing variables.

4 noweb Documentation of the Master Program

4.1 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
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()

(*=)

program master
implicit none
include ‘pass.h’
include ‘interp.h’
integer i,j
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./

c --- read file

call readfile

c --- calculate interpolation coefficients

call get_interpolation_coef

c --- depth interpolation/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()

c --- circulation
  if(N_Interval_CallCirc.ge.0)call CircModule()

c --- sediment
  if(N_Interval_CallSedi.ge.0)call SediModule()

c --- set initialization switch
  Master_Start=-1

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
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 (N_Interval_CallWave.ge.0.and.mod(istep-1,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.
   endif

   c --- call circulation module

   if (N_Interval_CallCirc.ge.0.and.mod(istep-1,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.

dend if

c --- call sediment module

if(N_Interval_CallSedi.ge.0 & .and.mod(istep-1,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.

dend if

c --- output

if (mod(istep-1,nOut).eq.0) then
   call Mexport()
endif

100 continue

c --- program end
print*, 'Program end'

end
4.2 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

```fortran
subroutine readfile
  implicit none
  include 'pass.h'
  integer i,j

  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,
```

(*)+≡

```fortran
```

---

```fortran
```
& 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_SedFlux,  &
& Circ_To_Sedi_UV,  &
& Circ_To_Sedi_UVb,  &
& Circ_To_Sedi_eta,  &
& Circ_To_Sedi_UV3D,  &
& Circ_To_Sedi_fw,  &
& Circ_To_Sedi_vt,  &
& Circ_To_Sedi_UVquasi3D,  &
& Sedi_To_Wave_Depth,  &
& Sedi_To_Circ_Depth,  &
& Circ_POM, Circ_SC

& /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

include 'ini_logical.f'

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 minimal consistency check
if(Circ_POM .and. Circ_SC) then
    write(0,*)' Specify only one circulation module'
    stop
endif
if(Circ_POM) then
  Wave_To_Circ_Height = .true.
  Wave_To_Circ_Angle = .true.
  Wave_To_Circ_WaveNum = .true.
  Wave_To_Circ_C = .true.
  Wave_To_Circ_Cg = .true.
  Wave_To_Circ_MassFlux = .true.
  Wave_To_Circ_Dissipation = .true.
  Wave_To_Circ_BottomUV = .true.
  Circ_Staggered = .true.
C check
  Circ_Stag_huv(1)=1
  Circ_Stag_huv(2)=2
  Circ_Stag_huv(3)=3
endif

C put definitions for other modules here so all pass flags are set
if(Circ_POM) then
  if(Circ_to_Sedi_vt) then
    write(0,*), 'vt not implemented in POM'
    stop
  endif
  if(Circ_to_Sedi_fw) then
    write(0,*), 'fw not implemented in POM'
    stop
  endif
  if(Circ_to_Sedi_UVquasi3D) then
    write(0,*), 'use UV3D with POM'
    stop
  endif
endif

c --- read initial depth

open(1, file=f_depth)
do j=1,Ny_Mast
  read(1,*),(Depth_Mast(i,j), i=1,Nx_Mast)
endo
close(1)

c --- read xy of master grid

open(1, file=f_xymast)
do j=1,Ny_Mast
  read(1,*),(X_Mast(i,j), i=1,Nx_Mast)
endo
do j=1,Ny_Mast
    read(1,*)(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,*)(X_Wave(i,j),i=1,Nx_Wave)
enddo
do j=1,Ny_Wave
    read(1,*)(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
dotoenddo
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,*)(X_Circ(i,j),i=1,Nx_Circ)
enddo
do j=1,Ny_Circ
    read(1,*)(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
doenddo
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,*)(X_Sedi(i,j),i=1,Nx_Sedi)
      enddo
      do j=1,Ny_Sedi
         read(1,*)(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.
    endif

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

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

100  format(800f16.8)

return
end
4.3 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

```fortran
subroutine get_interpolation_coef
  implicit none
  include 'pass.h'
  include 'interp.h'
  integer i,j
  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
4.4 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'
integer i,j

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 interpolation

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

return

end
4.5 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

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

```fortran
subroutine interp_circ_wave
implicit none
include 'pass.h'
include 'interp.h'
real Tmp1, Tmp2, tht
integer i,j

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)
        enddo
      enddo
    end if
  endif
endif
end subroutine interp_circ_wave
```

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

1. master

interp_circ_wave calls

1. grid1_to_grid2
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
4.6 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
integer i,j

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
4.7 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
integer i,j

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
if((Wave_Rotate_Angle-Circ_Rotate_Angle).ne.0)then
   do j=1,Ny_Circ
      do i=1,Nx_Circ
tht=Wave_Rotate_Angle-Circ_Rotate_Angle
Intp_Theta_Circ(i,j)=Intp_Theta_Circ(i,j)+tht
enddo
enddo
endif
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)

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)

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_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_Syy_surf,Intp_Syy_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_Sxx_body,Intp_Sxx_body,
  Wave_Staggered,0,Circ_Staggered,0)

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)
& 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
4.8 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*

```plaintext
subroutine interp_sedi_circ
implicit none
include 'pass.h'
include 'interp.h'
real Tmp1,Tmp2,tht
integer i,j

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
```
4.9 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*

```fortran
subroutine interp_wave_sedi
implicit none
include 'pass.h'
include 'interp.h'
real Tmp1,Tmp2,tht
integer i,j

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)
endif

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
4.10 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`

```fortran
subroutine interp_circ_sedi
  implicit none
  include 'pass.h'
  include 'interp.h'
  real Tmp1,Tmp2,tht
  integer i,j
  c --- Sediment Flux
  if(Circ_To_Sedi_SedFlux)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_SedFluxCum_x,Intp_SedFluxCum_x,
      Circ_Staggered,Circ_Stag_huv(1),
      Sedi_Staggered,Sedi_Stag_huv(1))
    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_SedFluxCum_y,Intp_SedFluxCum_y,
      Circ_Staggered,Circ_Stag_huv(1),
      Sedi_Staggered,Sedi_Stag_huv(1))
    if((Circ_Rotate_Angle-Sedi_Rotate_Angle).ne.0)then
      do j=1,Ny_Sedi
        do i=1,Nx_Sedi
          Tmp1=Intp_SedFluxCum_x(i,j)
          Tmp2=Intp_SedFluxCum_y(i,j)
          tht=(Circ_Rotate_Angle-Sedi_Rotate_Angle)*3.14159/180.
          Intp_SedFluxCum_x(i,j)=Tmp1*cos(tht)-Tmp2*sin(tht)
          Intp_SedFluxCum_y(i,j)=Tmp1*sin(tht)+Tmp2*cos(tht)
        enddo
      enddo
    endif
  endif
end subroutine interp_circ_sedi
```
enddo
endif
endif

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

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

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

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
c --- vt

if(Circ_To_Sedi_vt)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_vt,Intp_vt_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
4.11 Subroutine interpsame

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

interpsame is called by

1. get_interpolation_coef

\[
\text{(*)} \quad \equiv
\]

c -------------------------------------------------------------

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

         implicit none
         include 'pass.h'
         integer i,j

         ! --- (i,j) of three points surrounded
         ! --- 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.
x1(i,j)=i
y1(i,j)=j
x2(i,j)=1
y2(i,j)=1
x3(i,j)=1
y3(i,j)=1
enddo
enddo

return
end
4.12 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:
   \[ S = 0.5 \times (x_1 \times y_2 - x_2 \times y_1 + x_2 \times y_3 - x_3 \times y_2 + x_3 \times y_1 - x_1 \times y_3) \]
   if (1,2,3) is counterclock wise, \( S > 0 \), otherwise, \( S < 0 \)

   (b) interpolation/extrapolation:
   \[ var_c = \left( S \times var_1 + S_2 \times var_2 + S_3 \times var_3 \right) / Sc \]

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_

\( \langle \ast \rangle + \equiv \)

!--------------------------------------------------------------
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'
integer i,j

! --- 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/

! --- for 1-D case
if(n_grid2.eq.1.or.n_grid1.eq.1)then
  j=1
  do i=1,m_grid2
x1 = \text{x\_grid2}(i, j) \\
jj = 1 \\
do ii = 1, m\_grid1 - 1 \\
x2 = \text{x\_grid1}(ii, jj) \\
x3 = \text{x\_grid1}(ii + 1, jj) \\
area1 = x1 - x2 \\
area2 = x3 - x1 \\
if (area1 \geq 0 \text{ and } area2 \geq 0) then \\
x1(i, j) = ii + 1 \\
x2(i, j) = ii \\
S1(i, j) = area1 \\
S2(i, j) = area2 \\
Sc(i, j) = area1 + area2 \\
ntype(i, j) = 0 \\
goto 1100 \\
endif \\
enddo ! ii \\
enddo ! i \\
j = 1 \\
do i = 1, m\_grid2 \\
if (ntype(i, j) \text{ eq. } 1) then \\
jj = 1 \\
x1 = \text{x\_grid1}(1, jj) \\
x2 = \text{x\_grid1}(m\_grid1, jj) \\
x3 = \text{x\_grid2}(i, j) \\
if (abs(x3 - x1) \lt abs(x2 - x3)) then \\
x1(i, j) = 2 \\
x2(i, j) = 1 \\
S1 = x3 - x1 \\
S2 = \text{x\_grid1}(2, jj) - x3 \\
Sc = \text{x\_grid1}(2, jj) - x1 \\
else \\
x1(i, j) = m\_grid1 - 1 \\
x2(i, j) = m\_grid1 \\
S1 = x2 - x3 \\
S2 = x3 - \text{x\_grid1}(m\_grid1 - 1, jj) \\
S3 = x2 - \text{x\_grid1}(m\_grid1 - 1, jj) \\
endif \\
enddo ! ntype = 1
enddo ! i

else

! --- 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
    end if
    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,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+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)
x_near=1
y_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
    endif
  enddo
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
   ny3(i,j)=ny_near+1
endif

! --- if no extrapolation is allowed , evaluated variable will equal
! to the variable at nearest 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

endif ! for n_grid != 1

return

end
4.13 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:
   
   \[
   S = 0.5 \times (x_1 \times y_2 - x_2 \times y_1 + x_2 \times y_3 - x_3 \times y_2 + x_3 \times y_1 - x_1 \times y_3)
   \]
   if (1,2,3) is counterclock wise, $S > 0$, otherwise, $S < 0$

   (b) interpolation/extrapolation:
   
   \[
   var_c = \frac{(S1 \times var_1 + S2 \times var_2 + S3 \times var_3)}{Sc}
   \]

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`

```fortran
! *-----------------------------------------------------------------
! -- subroutine interpolation_nonstruc
! *-----------------------------------------------------------------
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'
integer i,j

! --- (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)
x_near_1=1
y_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
endoi
endoj

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
4.14 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'
integer i,j

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),tmpa(Nx_Max,Ny_Max),tmpb
! --- 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
  if(n_grid1.eq.1)then
    write(*,*)'stagered type defined wrong for 1D grid'
    stop
  endif

  ntype=1

  if(ntype_grid1.eq.1)then
    if(n_grid1.eq.1)then
      write(*,*)'stagered type defined wrong for 1D grid'
      stop
    endif
  endif

  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
    do i=1,m_grid1
      tmp(i,j)=var_grid1(i,j)
    enddo
  enddo
enddo
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
if(n_grid1.eq.1)then
write(*,'(a)')'stagered type defined wrong for 1D grid'
stop
endif
! --- interpolation/extrapolation
if(n_grid1.eq.1)then
do j=1,n_grid2
  do i=1,m_grid2
    var_grid2(i,j)=1./Sc(i,1)*(S1(i,1)*tmp(nx1(i,1),1) & # 
                          +S2(i,1)*tmp(nx2(i,1),1))
  enddo
enddo

elseif(n_grid2.eq.1)then
  do j=1,n_grid1
    do i=1,m_grid1
      tmpa(i,j)=tmp(i,j)
    enddo
  enddo
  do i=1,m_grid1
    tmpb=0.
    do j=1,n_grid1
      tmpb=tmpb+tmpa(i,j)
    enddo
    tmp(i,1)=tmpb/n_grid1
  enddo
  j=1
  do i=1,m_grid2
    var_grid2(i,j)=1./Sc(i,j)*(S1(i,j)*tmp(nx1(i,j),1) & # 
                               +S2(i,j)*tmp(nx2(i,j),1))
  enddo
else
  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
endif ! ngrid.eq.1
! --- 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
    if(n_grid2.eq.1)then
      write(*,*)'wrong staggered grid type for 1-D case'
      stop
    endif
    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
    if(n_grid2.eq.1)then
      write(*,*)'wrong staggered grid type for 1-D case'
      stop
    endif
    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
4.15 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'
integer i,j
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
```
4.16 Subroutine SediModule

The subroutine is the Sediment module.

SediModule is called by

1. Master

```fortran
(*)+==
c-------------------------------------------------------------------------------
subroutine SediModulesample()
  implicit none
  include 'pass.h'
  integer i,j

  if(Master_Start.eq.1)then
    print*, ’Sediment module initialization ...’
  else
    print*, ’call Sediment module ...’
  endif

  return
end
```
4.17 Subroutine Mexport

The subroutine is for model output. 

*Mexport* is called by

1. *Master*

```fortran
subroutine Mexport()
implicit none
include 'pass.h'
integer i,j
integer nm_first,nm_second,nm_third,nm_fourth,num_file
character*4 file_name

num_file=(istep-1)/N_Interval_Output
print*, 'mexport routine plot num=', 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_first
write(file_name(2:2),'(I1)')nm_second
write(file_name(3:3),'(I1)')nm_third
write(file_name(4:4),'(I1)')nm_fourth

if(f_name11.ne. ' ')then
  open(2,file=f_name11(1:2)//file_name//'.'out')
    do j=1,Ny_Circ
      write(2,111)(Pass_Theta(i,j),i=1,Nx_Circ)
    enddo
  close(2)
endif

if(f_name12.ne. ' ')then
  open(2,file=f_name12(1:2)//file_name//'.'out')
    do j=1,Ny_Circ
      write(2,111)(Pass_Height(i,j),i=1,Nx_Circ)
    enddo
  close(2)
endif
```

if(f_name13.ne. ' ')then
open(2, file=f_name13(1:2)//file_name//'.out')
do j=1,Ny_Circ
   write(2,111)(Depth_Circ(i,j),i=1,Nx_Circ)
enddo
close(2)
endif

if(f_name14.ne. ' ')then
open(2, file=f_name14(1:2)//file_name//'.out')
do j=1,Ny_Circ
   write(2,111)(Pass_U(i,j),i=1,Nx_Circ)
enddo
close(2)
endif

if(f_name15.ne. ' ')then
open(2, file=f_name15(1:2)//file_name//'.out')
do j=1,Ny_Circ
   write(2,111)(Pass_V(i,j),i=1,Nx_Circ)
enddo
close(2)
endif

if(f_name16.ne. ' ')then
open(2, file=f_name16(1:2)//file_name//'.out')
do j=1,Ny_Circ
   write(2,111)(Pass_eta(i,j),i=1,Nx_Circ)
enddo
close(2)
endif

if(f_name9.ne. ' ')then
open(2, file=f_name9(1:2)//file_name//'.out')
do j=1,Ny_Circ
   write(2,111)(Pass_Ub(i,j),i=1,Nx_Circ)
enddo
close(2)
endif

if(f_name10.ne. ' ')then
open(2, file=f_name10(1:2)//file_name//'.out')
do j=1,Ny_Circ
   write(2,111)(Pass_Vb(i,j),i=1,Nx_Circ)
enddo
close(2)
endif

if(f_name7.ne.' ')then
  open(2,file=f_name7(1:2)//file_name//'.'out')
    do j=1,Ny_Circ
      write(2,111)(Intp_MassFluxU_Circ(i,j)/depth_circ(i,j),
        i=1,Nx_Circ)
    enddo
  close(2)
endif

if(f_name8.ne.' ')then
  open(2,file=f_name8(1:2)//file_name//'.'out')
    do j=1,Ny_Circ
      write(2,111)(Intp_MassFluxV_Circ(i,j)/depth_circ(i,j),
        i=1,Nx_Circ)
    enddo
  close(2)
endif

111 format(500f16.6)

return
end
4.18 Subroutine WaveModule

The subroutine is the Wave module.

WaveModule is called by

1. Master

```fortran
(*)+==                       
  subroutine WaveModulesample()
  implicit none
  include 'pass.h'
  integer i,j

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

  return
end
```
4.19 Subroutine CircModule

The subroutine is the circulation module. CircModule is called by

1. Master

```fortran
subroutine CircModulesample()
  implicit none
  include 'pass.h'
  integer i,j

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

  return
end
```
4.20 Subroutine MasterInit

The subroutine initializes all pass variables.

MasterInit is called by

1. Master

\[
\begin{align*}
& \text{integer } i, j \\
& \text{do } j=1, \text{Ny}\_\text{Max} \\
& \quad \text{do } i=1, \text{Nx}\_\text{Max} \\
& \quad \quad \text{Pass}\_\text{Sxx}(i,j)=0. \\
& \quad \quad \text{Pass}\_\text{Sxy}(i,j)=0. \\
& \quad \quad \text{Pass}\_\text{Syy}(i,j)=0. \\
& \quad \quad \text{Pass}\_\text{Sxx}\_\text{body}(i,j)=0. \\
& \quad \quad \text{Pass}\_\text{Sxy}\_\text{body}(i,j)=0. \\
& \quad \quad \text{Pass}\_\text{Syy}\_\text{body}(i,j)=0. \\
& \quad \quad \text{Pass}\_\text{Sxx}\_\text{surf}(i,j)=0. \\
& \quad \quad \text{Pass}\_\text{Sxy}\_\text{surf}(i,j)=0. \\
& \quad \quad \text{Pass}\_\text{Syy}\_\text{surf}(i,j)=0. \\
& \quad \quad \text{Pass}\_\text{Wave}\_\text{Fx}(i,j)=0. \\
& \quad \quad \text{Pass}\_\text{Wave}\_\text{Fx}(i,j)=0. \\
& \quad \quad \text{Pass}\_\text{MassFluxU}(i,j)=0. \\
& \quad \quad \text{Pass}\_\text{MassFluxV}(i,j)=0. \\
& \quad \quad \text{Pass}\_\text{MassFlux}(i,j)=0. \\
& \quad \quad \text{Pass}\_\text{Diss}(i,j)=0. \\
& \quad \quad \text{Pass}\_\text{WaveNum}(i,j)=0. \\
& \quad \quad \text{Pass}\_\text{Theta}(i,j)=0. \\
& \quad \quad \text{Pass}\_\text{ubott}(i,j)=0. \\
& \quad \quad \text{Pass}\_\text{Height}(i,j)=0. \\
& \quad \quad \text{Pass}\_\text{C}(i,j)=0. \\
& \quad \quad \text{Pass}\_\text{C}(i,j)=0. \\
& \quad \quad \text{Intp}\_\text{U}\_\text{Wave}(i,j)=0. \\
& \quad \quad \text{Intp}\_\text{V}\_\text{Wave}(i,j)=0. \\
& \quad \quad \text{Intp}\_\text{eta}\_\text{Wave}(i,j)=0. \\
& \quad \quad \text{Pass}\_\text{ibrk}(i,j)=0
\end{align*}
\]
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.
pass_vt(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_vt_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
5 Frequently Asked Questions

1. If I have only two modules coupled, e.g., WaveModule and CircModule, how to set the model?
   Set $N_{\text{Interval\_Call\_Sedi}} = -1$ 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_{\text{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.