SAMSIM

Generated by Doxygen 1.7.1

Mon May 26 2014 15:37:20

Contents

Chapter 1

SAMSIM Semi-Adaptive Multi-phase Sea-Ice Model V1.0

This model was developed from scratch by Philipp Griewank during and after his PhD at Max Planck Institute of Meteorology from 2010-2014. Most elements of the model are described in my PhD thesis (A 1D model study of brine dynamics in sea ice, 2013, Hamburg) and in the publication "Insights into brine dynamics and sea ice desalination from a 1-D model study of gravity drainage" by Griewank & Notz 2013 JGR: Oceans. Important changes from the version 1.0 to the previous version are:

- the switch to a harmonic mean permeability for gravity drainage
- A few important bug fixes, which have a large affect on flushing

SAMSIM.f90 is the root program of the SAMSIM, the 1D thermodynamic Semi-Adaptive Multi-phase Sea-Ice Model. The code is intended to be understandable and subroutines, modules, functions, parameters, and global variables all have doxygen compatible descriptions. However, in SAMSIM.f90 only the testcase and description thread are specified, which are then passed on to mo_grotz, which is where most of the actual work is done, including timestepping.

WARNING: SAMSIM was developed and used for scientific purposes. It surely contains at least some undetected bugs, can easily be crashed by using non-logical input settings, and some of the descriptions and comments may be outdated. Always check the plausibility of the model results!

Getting started.

- A number of testcases are implemented in SAMSIM. Testcases 1, 2, 3, and 4 are intended as standard testcases which should give a first time user a feel for the model capabilities and serve as a basis to set up custom testcases. To familiarize yourself with the model I suggest running testcases 1-3 and plotting the output with the python plotting scripts provided. The details of each testcase are commented in mo_init.f90, and each plot script begins with a list of steps required.

Running SAMSIM the first times.

- Make sure that all .f90 files are located in the same folder with the makefile.
- Open the makefile with your editor of choice and choose the compiler and flags of choice.
- Open SAMSIM.f90, set a testcase from 1-3, and edit the description string to fit your purpose.
- Use make to compile the code, which produces the executable samsim.x .

- Make sure a folder "output" is located in the folder with samsim.x .
- Execute SAMSIM by running samsim.x .
- Go into output folder
- Copy the plot script from plotscripts to output
- Follow the directions written in the plotscripts to plot the output.

Running testcase 4. In contrast to testcase 1-3, testcase four requires input files. Input data for testcase is provided in the input folder. Choose one of the subfolders from input/ERA-interim/, copy the ∗.input files into the folder with the code, and run the executable .samsim.x .

Following modules have a good documentation (both in the code and refman.pdf)

- mo_heat_fluxes.f90
- mo_layer_dynamics.f90
- mo_init.f90

Biogeochemical tracers can be activated with bgc_flag=2.

- Warning! This feature is still relatively new, and has not been standardized yet.
- The model will track Nbgc number of individual tracer.
- Especially if you are interested in dissolved gases, you should first make yourself familiar with the bgc_advection subroutine in mo_mass.f90.

Know issues/Tips and Tricks:

- If code changes have no effect, run "make clean" and then "make", for unknown reasons this is often needed when making changes to mo_parameters.f90
- When bug hunting increase thick_0 and dt, this way the model runs faster, and the output is easier to sort through.
- Use debug_flag= 2 to output data of each layer at each timestep. Be careful, the output size can become very large!
- Check dat_settings to keep track of runs, and use the description variable to keep track of experiments.
- Contact me :)

Revision History

Started by Philipp Griewank 2014-05-05

Chapter 2

Modules Index

2.1 Modules List

Here is a list of all modules with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Module Documentation

4.1 mo_data Module Reference

Sets data and contains all flag descriptions.

Variables

- REAL(wp), dimension(:), allocatable H

  Enthalpy [J].

- REAL(wp), dimension(:), allocatable H_abs

  specific Enthalpy [J/kg]

- REAL(wp), dimension(:), allocatable Q

  Heat in layer [J].

- REAL(wp), dimension(:), allocatable fl_Q

  Heat flux between layers [J/s].

- REAL(wp), dimension(:), allocatable T

  Temperature [C].

- REAL(wp), dimension(:), allocatable S_bu

  Bulk Salinity [g/kg].

- REAL(wp), dimension(:), allocatable fl_S

  Salinity flux [(g/s].

- REAL(wp), dimension(:), allocatable S_abs

  Absolute Salinity [g].

- REAL(wp), dimension(:), allocatable S_br

  Brine salinity [g/kg].

- REAL(wp), dimension(:), allocatable thick

Layer thickness [m].

- REAL(wp), dimension(:), allocatable m

  Mass [kg].

- REAL(wp), dimension(:), allocatable fl_m

  Mass fluxes between layers [kg].

- REAL(wp), dimension(:), allocatable V_s

  Volume [m^3] of solid.

- REAL(wp), dimension(:), allocatable V_l

  Volume [m^3] of liquid.

- REAL(wp), dimension(:), allocatable V_g

  Volume [m^3] of gas.

- REAL(wp), dimension(:), allocatable V_ex

  Volume of brine due expelled due to freezing [m^3] of solid, gas & liquid.

- REAL(wp), dimension(:), allocatable phi

  Solid mass fraction.

- REAL(wp), dimension(:), allocatable psi_s

  Solid volume fraction.

- REAL(wp), dimension(:), allocatable psi_l

  Liquid volume fraction.

- REAL(wp), dimension(:), allocatable psi_g

  Gas volume fraction.

- REAL(wp), dimension(:), allocatable ray

  Rayleigh number of each layer.

- REAL(wp), dimension(:), allocatable perm

  Permeability [?].

- REAL(wp) dt

  Timestep [s].

- REAL(wp) thick_0

  Initial layer thickness [m].

- REAL(wp) time

  Time [s].

- REAL(wp) freeboard

  Height of ice surface above (or below) waterlevel [m].

- REAL(wp) T_freeze

  Freezing temperature [C].

- INTEGER Nlayer

  Number of layers.

- INTEGER N_bottom

  Number of bottom layers.

- INTEGER N_middle

  Number of middle layers.

- INTEGER N_top

  Number of top layers.

- INTEGER N_active

  Number of Layers active in the present.

- INTEGER i

  Index, normally used for time.

- INTEGER k

  Index, normally used for layer.

- REAL(wp) time_out

  Time between outputs [s].

- REAL(wp) time_total

  Time of simulation [s].

- INTEGER i_time

  Number of timesteps.

- INTEGER i_time_out

  Number of timesteps between each output.

- INTEGER n_time_out

  Counts number of timesteps between output.

- CHARACTER ∗12000 format_T
- CHARACTER ∗12000 format_psi
- CHARACTER ∗12000 format_thick
- CHARACTER ∗12000 format_snow
- CHARACTER ∗12000 format_integer
- CHARACTER ∗12000 format_T2m_top
- CHARACTER ∗12000 format_bgc

  Format strings for output.

- REAL(wp) T_bottom

  Temperature of water beneath the ice [C].

- REAL(wp) T_top

  Temperature at the surface [C].

- REAL(wp) S_bu_bottom

  Salinity beneath the ice [g/kg].

- REAL(wp) T2m

  Two meter Temperature [C].

- REAL(wp) fl_q_bottom

  *Bottom heat flux [J*s].*

- REAL(wp) psi_s_snow

  Solid volume fraction of snow layer.

- REAL(wp) psi_l_snow

  Liquid volume fraction of snow layer.

- REAL(wp) psi_g_snow

  Gas volume fraction of snow layer.

- REAL(wp) phi_s

  Solid mass fraction of snow layer.

- REAL(wp) S_abs_snow

  Absolute salinity of snow layer [g].

- REAL(wp) H_abs_snow

  Absolute enthalpy of snow layer [J].

- REAL(wp) m_snow

  Mass of snow layer [kg].

- REAL(wp) T_snow

  Temperature of snow layer [C].

- REAL(wp) thick_snow

  Thickness of snow layer [m].

- REAL(wp) liquid_precip

  Liquid precip, [meter of water/s].

- REAL(wp) solid_precip

  Solid precip, [meter of water /s].

- REAL(wp) fl_q_snow

  flow of heat into the snow layer

- REAL(wp) energy_stored

Total amount of energy stored, control is freezing point temperature of S_bu_bottom [J].

- REAL(wp) total_resist

    Thermal resistance of the whole column [].

- REAL(wp) surface_water

    Percentage of water fraction in the top 5cm [%].

- REAL(wp) freshwater

    Meters of freshwater stored in column [m].

- REAL(wp) thickness

    Meters of ice [m].

- REAL(wp) bulk_salin

    Salt/Mass [ppt].

- REAL(wp) thick_min

    Parameter for snow, determines when snow is in thermal equilibrium with the ice and when it is totally neglected.

- REAL(wp), save T_test

    First guess for getT subroutine.

- REAL(wp) albedo

    Amount of short wave radiation which is reflected at the top surface.

- REAL(wp) fl_sw

    *Incoming shortwave radiation [W/m**2].*

- REAL(wp) fl_lw

    *Incoming longwave radiation [W/m**2].*

- REAL(wp) fl_sen

    *Sensitive heat flux [W/m**2].*

- REAL(wp) fl_lat

    *Latent heat flux [W/m**2].*

- REAL(wp) fl_rest

    *Bundled longwave,sensitive and latent heat flux [W/m**2].*

- REAL(wp), dimension(:), allocatable fl_rad

    Energy flux of absorbed sw radiation of each layer [J/s].

- REAL(wp) grav_drain

    brine flux of gravity drainage between two outputs [kg/s]

- REAL(wp) grav_salt

    *salt flux moved by gravity drainage between two outputs [kg*ppt/s]*

- REAL(wp) grav_temp

  average temperature of gravity drainage brine between two outputs [T]

- REAL(wp) melt_thick

  thickness of fully liquid part of top layer [m]

- REAL(wp) alpha_flux_instable

  Proportionality constant which determines energy flux by the temperature difference T_top>T2m [W/C].

- REAL(wp) alpha_flux_stable

  Proportionality constant which determines energy flux by the temperature difference T_top<T2m [W/C].

- INTEGER atmoflux_flag

  1: Use mean climatology of Notz, 2: Use imported reanalysis data, 3: use fixed values defined in mo_init

- INTEGER grav_flag

  1: no gravity drainage, 2: Gravity drainage, 3: Simple Drainage

- INTEGER prescribe_flag

  1: nothing happens, 2: prescribed Salinity profile is prescribed at each timestep (does not disable brine dynamics, just overwrites the salinity!)

- INTEGER grav_heat_flag

  1: nothing happens, 2: compensates heatfluxes in grav_flag = 2

- INTEGER flush_heat_flag

  1: nothing happens, 2: compensates heatfluxes in flush_flag = 5

- INTEGER turb_flag

  1: No bottom turbulence, 2: Bottom mixing

- INTEGER salt_flag

  1: Sea salt, 2: NaCL

- INTEGER boundflux_flag

  1: top and bottom cooling plate, 2:top Notz fluxes, bottom cooling plate 3: top flux=a∗(T-T_s)

- INTEGER flush_flag

  1: no flushing, 4:meltwater is removed artificially, 5:vert and horiz flushing, 6: simplified

- INTEGER flood_flag

  1: no flooding, 2:normal flooding, 3:simple flooding

- INTEGER bottom_flag

  1: nothing changes, 2: deactivates all bottom layer dynamics, useful for some debugging and idealized tests

- INTEGER debug_flag

  1: no raw layer output, 2: each layer is output at every timestep (warning, file size can be very large)

- INTEGER precip_flag

  0: solid and liquid precipitation, 1:phase determined by T2m

- INTEGER harmonic_flag

  1: minimal permeability is used to calculate Rayleigh number, 2:harmonic mean is used for Rayleigh number

- INTEGER tank_flag

  1: nothing, 2: S_bu_bottom and bgc_bottom are calculated as if the experiment is conducted in a tank

- INTEGER albedo_flag

  1: simple albedo, 2: normal albedo, see func_albedo for details

- INTEGER Length_Input

  Sets the input length for atmoflux_flag==2, common value of 13169.

- REAL(wp), dimension(8280) Tinput

  used to read in top temperature for field experiment tests, dimension needs to be set in the code

- REAL(wp), dimension(:), allocatable fl_sw_input

  Used to read in sw fluxes from ERA for atmoflux_flag==2.

- REAL(wp), dimension(:), allocatable fl_lw_input

  Used to read in lw fluxes from ERA for atmoflux_flag==2.

- REAL(wp), dimension(:), allocatable T2m_input

  Used to read in 2Tm from ERA for atmoflux_flag==2.

- REAL(wp), dimension(:), allocatable precip_input

  Used to read in precipitation from ERA for atmoflux_flag==2.

- REAL(wp), dimension(:), allocatable time_input

  Used to read in time from ERA for atmoflux_flag==2.

- INTEGER time_counter

  Keeps track of input data.

- INTEGER bgc_flag

  1: no bgc, 2:bgc

- INTEGER N_bgc

  Number of chemicals.

- REAL(wp), dimension(:,:), allocatable fl_brine_bgc

  Brine fluxes in a matrix, [kg/s], first index is the layer of origin, and the second index is the layer of arrival.

- REAL(wp), dimension(:,:), allocatable bgc_abs

  Absolute amount of chemicals [kmol] for each tracer.

- REAL(wp), dimension(:,:), allocatable bgc_bu

Bulk amounts of chemicals [kmol/kg].

- REAL(wp), dimension(:,:), allocatable bgc_br

  Brine concentrations of chems [kmol/kg].

- REAL(wp), dimension(:), allocatable bgc_bottom

  Bulk concentrations of chems below the ice [kmol/kg].

- REAL(wp), dimension(:), allocatable bgc_total

  Total of chems, for lab experiments with a fixed total amount.

- REAL(wp) m_total

  Total initial water mass, for lab experiments with a fixed total amount.

- REAL(wp) S_total

  Total initial salt mass, for lab experiments with a fixed total amount.

- REAL(wp) tank_depth

  water depth in meters, used to calculate concentrations below ice for tank experiments

4.1.1 Detailed Description

Sets data and contains all flag descriptions. All data needed by mo_grotz are set in this module. Most arrays are allocated after the needed dimension is specified for each testcase in mo_init.f90.

Author

Philipp Griewank

Revision History

Initialized by Philipp Griewank, IMPRS (2010-07-14)

4.1.2 Variable Documentation

4.1.2.1 REAL(wp) mo_data::albedo

Amount of short wave radiation which is reflected at the top surface.

4.1.2.2 INTEGER mo_data::albedo_flag

1: simple albedo, 2: normal albedo, see func_albedo for details

4.1.2.3 REAL(wp) mo_data::alpha_flux_instable

Proportionality constant which determines energy flux by the temperature difference T_top>T2m [W/C].

4.1.2.4 REAL(wp) mo_data::alpha_flux_stable

Proportionality constant which determines energy flux by the temperature difference T_top<T2m [W/C].

4.1.2.5 INTEGER mo_data::atmoflux_flag

1: Use mean climatology of Notz, 2: Use imported reanalysis data, 3: use fixed values defined in mo_init

4.1.2.6 REAL(wp),dimension(:,:),allocatable mo_data::bgc_abs

Absolute amount of chemicals [kmol] for each tracer.

4.1.2.7 REAL(wp),dimension(:),allocatable mo_data::bgc_bottom

Bulk concentrations of chems below the ice [kmol/kg].

4.1.2.8 REAL(wp),dimension(:,:),allocatable mo_data::bgc_br

Brine concentrations of chems [kmol/kg].

4.1.2.9 REAL(wp),dimension(:,:),allocatable mo_data::bgc_bu

Bulk amounts of chemicals [kmol/kg].

4.1.2.10 INTEGER mo_data::bgc_flag

1: no bgc, 2:bgc

4.1.2.11 REAL(wp),dimension(:),allocatable mo_data::bgc_total

Total of chems, for lab experiments with a fixed total amount.

4.1.2.12 INTEGER mo_data::bottom_flag

1: nothing changes, 2: deactivates all bottom layer dynamics, useful for some debugging and idealized tests

4.1.2.13 INTEGER mo_data::boundflux_flag

1: top and bottom cooling plate, 2:top Notz fluxes, bottom cooling plate 3: top flux=a∗(T-T_s)

4.1.2.14 REAL(wp) mo_data::bulk_salin

Salt/Mass [ppt].

4.1.2.15 INTEGER mo_data::debug_flag

1: no raw layer output, 2: each layer is output at every timestep (warning, file size can be very large)

4.1.2.16 REAL(wp) mo_data::dt

Timestep [s].

4.1.2.17 REAL(wp) mo_data::energy_stored

Total amount of energy stored, control is freezing point temperature of S_bu_bottom [J].

4.1.2.18 REAL(wp),dimension(:,:),allocatable mo_data::fl_brine_bgc

Brine fluxes in a matrix, [kg/s], first index is the layer of origin, and the second index is the layer of arrival.

4.1.2.19 REAL(wp) mo_data::fl_lat

Latent heat flux [W/m**2].

4.1.2.20 REAL(wp) mo_data::fl_lw

Incoming longwave radiation [W/m**2].

4.1.2.21 REAL(wp),dimension(:),allocatable mo_data::fl_lw_input

Used to read in lw fluxes from ERA for atmoflux_flag==2.

4.1.2.22 REAL(wp),dimension(:),allocatable mo_data::fl_m

Mass fluxes between layers [kg].

4.1.2.23 REAL(wp),dimension(:),allocatable mo_data::fl_Q

Heat flux between layers [J/s].

4.1.2.24 REAL(wp) mo_data::fl_q_bottom

Bottom heat flux [J*s].

4.1.2.25 REAL(wp) mo_data::fl_q_snow

flow of heat into the snow layer

4.1.2.26 REAL(wp),dimension(:),allocatable mo_data::fl_rad

Energy flux of absorbed sw radiation of each layer [J/s].

4.1.2.27 REAL(wp) mo_data::fl_rest

Bundled longwave,sensitive and latent heat flux [W/m**2].

4.1.2.28 REAL(wp),dimension(:),allocatable mo_data::fl_S

Salinity flux [(g/s].

4.1.2.29 REAL(wp) mo_data::fl_sen

Sensitive heat flux [W/m**2].

4.1.2.30 REAL(wp) mo_data::fl_sw

Incoming shortwave radiation [W/m**2].

4.1.2.31 REAL(wp),dimension(:),allocatable mo_data::fl_sw_input

Used to read in sw fluxes from ERA for atmoflux_flag==2.

4.1.2.32 INTEGER mo_data::flood_flag

1: no flooding, 2:normal flooding, 3:simple flooding

4.1.2.33 INTEGER mo_data::flush_flag

1: no flushing, 4:meltwater is removed artificially, 5:vert and horiz flushing, 6: simplified

4.1.2.34 INTEGER mo_data::flush_heat_flag

1: nothing happens, 2: compensates heatfluxes in flush_flag = 5

4.1.2.35 CHARACTER*12000 mo_data::format_bgc

Format strings for output.

4.1.2.36 CHARACTER∗12000 mo_data::format_integer

4.1.2.37 CHARACTER∗12000 mo_data::format_psi

4.1.2.38 CHARACTER∗12000 mo_data::format_snow

4.1.2.39 CHARACTER∗12000 mo_data::format_T

4.1.2.40 CHARACTER∗12000 mo_data::format_T2m_top

4.1.2.41 CHARACTER∗12000 mo_data::format_thick

4.1.2.42 REAL(wp) mo_data::freeboard

Height of ice surface above (or below) waterlevel [m].

4.1.2.43 REAL(wp) mo_data::freshwater

Meters of freshwater stored in column [m].

4.1.2.44 REAL(wp) mo_data::grav_drain

brine flux of gravity drainage between two outputs [kg/s]

4.1.2.45 INTEGER mo_data::grav_flag

1: no gravity drainage, 2: Gravity drainage, 3: Simple Drainage

4.1.2.46 INTEGER mo_data::grav_heat_flag

1: nothing happens, 2: compensates heatfluxes in grav_flag = 2

4.1.2.47 REAL(wp) mo_data::grav_salt

salt flux moved by gravity drainage between two outputs [kg∗ppt/s]

4.1.2.48 REAL(wp) mo_data::grav_temp

average temperature of gravity drainage brine between two outputs [T]

4.1.2.49 REAL(wp),dimension(:),allocatable mo_data::H

Enthalpy [J].

4.1.2.50 REAL(wp),dimension(:),allocatable mo_data::H_abs

specific Enthalpy [J/kg]

4.1.2.51 REAL(wp) mo_data::H_abs_snow

Absolute enthalpy of snow layer [J].

4.1.2.52 INTEGER mo_data::harmonic_flag

1: minimal permeability is used to calculate Rayleigh number, 2:harmonic mean is used for Rayleigh number

4.1.2.53 INTEGER mo_data::i

Index, normally used for time.

4.1.2.54 INTEGER mo_data::i_time

Number of timesteps.

4.1.2.55 INTEGER mo_data::i_time_out

Number of timesteps between each output.

4.1.2.56 INTEGER mo_data::k

Index, normally used for layer.

4.1.2.57 INTEGER mo_data::Length_Input

Sets the input length for atmoflux_flag==2, common value of 13169.

4.1.2.58 REAL(wp) mo_data::liquid_precip

Liquid precip, [meter of water/s].

4.1.2.59 REAL(wp),dimension(:),allocatable mo_data::m

Mass [kg].

4.1.2.60 REAL(wp) mo_data::m_snow

Mass of snow layer [kg].

4.1.2.61 REAL(wp) mo_data::m_total

Total initial water mass, for lab experiments with a fixed total amount.

4.1.2.62 REAL(wp) mo_data::melt_thick

thickness of fully liquid part of top layer [m]

4.1.2.63 INTEGER mo_data::N_active

Number of Layers active in the present.

4.1.2.64 INTEGER mo_data::N_bgc

Number of chemicals.

4.1.2.65 INTEGER mo_data::N_bottom

Number of bottom layers.

4.1.2.66 INTEGER mo_data::N_middle

Number of middle layers.

4.1.2.67 INTEGER mo_data::n_time_out

Counts number of timesteps between output.

4.1.2.68 INTEGER mo_data::N_top

Number of top layers.

4.1.2.69 INTEGER mo_data::Nlayer

Number of layers.

4.1.2.70 REAL(wp),dimension(:),allocatable mo_data::perm

Permeability [?].

4.1.2.71 REAL(wp),dimension(:),allocatable mo_data::phi

Solid mass fraction.

4.1.2.72 REAL(wp) mo_data::phi_s

Solid mass fraction of snow layer.

4.1.2.73 INTEGER mo_data::precip_flag

0: solid and liquid precipitation, 1:phase determined by T2m

4.1.2.74 REAL(wp),dimension(:),allocatable mo_data::precip_input

Used to read in precipitation from ERA for atmoflux_flag==2.

4.1.2.75 INTEGER mo_data::prescribe_flag

1: nothing happens, 2: prescribed Salinity profile is prescribed at each timestep (does not disable brine dynamics, just overwrites the salinity!)

4.1.2.76 REAL(wp),dimension(:),allocatable mo_data::psi_g

Gas volume fraction.

4.1.2.77 REAL(wp) mo_data::psi_g_snow

Gas volume fraction of snow layer.

4.1.2.78 REAL(wp),dimension(:),allocatable mo_data::psi_l

Liquid volume fraction.

4.1.2.79 REAL(wp) mo_data::psi_l_snow

Liquid volume fraction of snow layer.

4.1.2.80 REAL(wp),dimension(:),allocatable mo_data::psi_s

Solid volume fraction.

4.1.2.81 REAL(wp) mo_data::psi_s_snow

Solid volume fraction of snow layer.

4.1.2.82 REAL(wp),dimension(:),allocatable mo_data::Q

Heat in layer [J].

4.1.2.83 REAL(wp),dimension(:),allocatable mo_data::ray

Rayleigh number of each layer.

4.1.2.84 REAL(wp),dimension(:),allocatable mo_data::S_abs

Absolute Salinity [g].

4.1.2.85 REAL(wp) mo_data::S_abs_snow

Absolute salinity of snow layer [g].

4.1.2.86 REAL(wp),dimension(:),allocatable mo_data::S_br

Brine salinity [g/kg].

4.1.2.87 REAL(wp),dimension(:),allocatable mo_data::S_bu

Bulk Salinity [g/kg].

4.1.2.88 REAL(wp) mo_data::S_bu_bottom

Salinity beneath the ice [g/kg].

4.1.2.89 REAL(wp) mo_data::S_total

Total initial salt mass, for lab experiments with a fixed total amount.

4.1.2.90 INTEGER mo_data::salt_flag

1: Sea salt, 2: NaCL

4.1.2.91 REAL(wp) mo_data::solid_precip

Solid precip, [meter of water /s].

4.1.2.92 REAL(wp) mo_data::surface_water

Percentage of water fraction in the top 5cm [%].

4.1.2.93 REAL(wp),dimension(:),allocatable mo_data::T

Temperature [C].

4.1.2.94 REAL(wp) mo_data::T2m

Two meter Temperature [C].

4.1.2.95 REAL(wp),dimension(:),allocatable mo_data::T2m_input

Used to read in 2Tm from ERA for atmoflux_flag==2.

4.1.2.96 REAL(wp) mo_data::T_bottom

Temperature of water beneath the ice [C].

4.1.2.97 REAL(wp) mo_data::T_freeze

Freezing temperature [C].

4.1.2.98 REAL(wp) mo_data::T_snow

Temperature of snow layer [C].

4.1.2.99 REAL(wp),save mo_data::T_test

First guess for getT subroutine.

4.1.2.100 REAL(wp) mo_data::T_top

Temperature at the surface [C].

4.1.2.101 REAL(wp) mo_data::tank_depth

water depth in meters, used to calculate concentrations below ice for tank experiments

4.1.2.102 INTEGER mo_data::tank_flag

1: nothing, 2: S_bu_bottom and bgc_bottom are calculated as if the experiment is conducted in a tank

4.1.2.103 REAL(wp),dimension(:),allocatable mo_data::thick

Layer thickness [m].

4.1.2.104 REAL(wp) mo_data::thick_0

Initial layer thickness [m].

4.1.2.105 REAL(wp) mo_data::thick_min

Parameter for snow, determines when snow is in thermal equilibrium with the ice and when it is totally neglected.

4.1.2.106 REAL(wp) mo_data::thick_snow

Thickness of snow layer [m].

4.1.2.107 REAL(wp) mo_data::thickness

Meters of ice [m].

4.1.2.108 REAL(wp) mo_data::time

Time [s].

4.1.2.109 INTEGER mo_data::time_counter

Keeps track of input data.

4.1.2.110 REAL(wp),dimension(:),allocatable mo_data::time_input

Used to read in time from ERA for atmoflux_flag==2.

4.1.2.111 REAL(wp) mo_data::time_out

Time between outputs [s].

4.1.2.112 REAL(wp) mo_data::time_total

Time of simulation [s].

4.1.2.113 REAL(wp),dimension(8280) mo_data::Tinput

used to read in top temperature for field experiment tests, dimension needs to be set in the code

4.1.2.114 REAL(wp) mo_data::total_resist

Thermal resistance of the whole column [].

4.1.2.115 INTEGER mo_data::turb_flag

1: No bottom turbulence, 2: Bottom mixing

4.1.2.116 REAL(wp),dimension(:),allocatable mo_data::V_ex

Volume of brine due expelled due to freezing [m^3] of solid, gas & liquid.

4.1.2.117 REAL(wp),dimension(:),allocatable mo_data::V_g

Volume [m^3] of gas.

4.1.2.118 REAL(wp),dimension(:),allocatable mo_data::V_l

Volume [m^3] of liquid.

4.1.2.119 REAL(wp),dimension(:),allocatable mo_data::V_s

Volume [m^3] of solid.

4.2 mo_flood Module Reference

Computes the fluxes caused by liquid flooding the snow layer.

Functions/Subroutines

- subroutine flood (freeboard, psi_s, psi_l, S_abs, H_abs, m, T, thick, dt, Nlayer, N_active, T_bottom, S_bu_bottom, H_abs_snow, m_snow, thick_snow, psi_g_snow, debug_flag, fl_brine_bgc)

  Subroutine for calculating flooding.

- subroutine flood_simple (freeboard, S_abs, H_abs, m, thick, T_bottom, S_bu_bottom, H_abs_snow, m_snow, thick_snow, psi_g_snow, Nlayer, N_active, debug_flag)

  Subroutine for calculating flooding.

4.2.1 Detailed Description

Computes the fluxes caused by liquid flooding the snow layer. Water floods the snow layer instantly transforming it to ice which is added to the top layer. As long as the negative freeboard is smaller then a certain parameter (neg_free) the flood strength is limited by the harmonic mean permeability of the whole ice layer driven by the freeboard. When this parameter is exceed, instant flooding is assumed. Based on Ted Maksyms work, brine is moved from the ocean to the snow without interacting with the ice in between. Very little of the process is well understood, so this parametrisation is ID mostly speculation. Ratio_flood is a very important parameter, as it regulates how much wicking into the snow layer occurs during melting which dilutes the flooded snow. Ratio of two should lead to the snow pack being reduced twice as much as the top layer grows.

Author

<Philipp griewank>="">

Revision History

Copy and pasted into existence by Philipp Griewank, IMPRS (2011-01-21)

4.2.2 Function/Subroutine Documentation

4.2.2.1 subroutine mo_flood::flood (REAL(wp),intent(in) *freeboard*, REAL(wp),dimension(nlayer),intent(in) *psi_s*, REAL(wp),dimension(nlayer),intent(in) *psi_l*, REAL(wp),dimension(nlayer),intent(inout) *S_abs*, REAL(wp),dimension(nlayer),intent(inout) *H_abs*, REAL(wp),dimension(nlayer),intent(inout) *m*, REAL(wp),dimension(nlayer),intent(in) *T*, REAL(wp),dimension(nlayer),intent(inout) *thick*, REAL(wp),intent(in) *dt*, INTEGER,intent(in) *Nlayer*, INTEGER,intent(in) *N_active*, REAL(wp),intent(in) *T_bottom*, REAL(wp),intent(in) *S_bu_bottom*, REAL(wp),intent(inout) *H_abs_snow*, REAL(wp),intent(inout) *m_snow*, REAL(wp),intent(inout) *thick_snow*, REAL(wp),intent(in) *psi_g_snow*, INTEGER,intent(in) *debug_flag*, REAL(wp),dimension(nlayer+1,nlayer+1),intent(inout),optional *fl_brine_bgc*)

Subroutine for calculating flooding.

Details explained in module description.

Revision History

Formed by Philipp Griewank, IMPRS (2011-01-21) Cleaned and commented by Philipp Griewank, (2014-04-19)

4.2.2.2 subroutine mo_flood::flood_simple (REAL(wp),intent(in) *freeboard*, REAL(wp),dimension(nlayer),intent(inout) *S_abs*, REAL(wp),dimension(nlayer),intent(inout) *H_abs*, REAL(wp),dimension(nlayer),intent(inout) *m*, REAL(wp),dimension(nlayer),intent(inout) *thick*, REAL(wp),intent(in) *T_bottom*, REAL(wp),intent(in) *S_bu_bottom*, REAL(wp),intent(inout) *H_abs_snow*, REAL(wp),intent(inout) *m_snow*, REAL(wp),intent(inout) *thick_snow*, REAL(wp),intent(in) *psi_g_snow*, INTEGER,intent(in) *Nlayer*, INTEGER,intent(in) *N_active*, INTEGER,intent(in) *debug_flag*)

Subroutine for calculating flooding.

Simplified version of flood. Flooding occurs instantly to fill the negative freeboard until it reaches neg_free with underlying ocean water.

Revision History

Formed by Philipp Griewank, IMPRS (2012-07-16) Added neg_free limitation.

4.3 mo_flush Module Reference

Contains various subroutines for flushing.

Functions/Subroutines

- subroutine flush3 (freeboard, psi_l, thick, thick_0, S_abs, H_abs, m, T, dt, Nlayer, N_active, T_bottom, S_bu_bottom, melt_thick, debug_flag, flush_heat_flag, fl_brine_bgc)

    Subroutine for complex flushing.

- subroutine flush4 (psi_l, thick, T, thick_0, S_abs, H_abs, m, dt, Nlayer, N_active, N_top, N_middle, N_bottom, melt_thick, debug_flag)

  An alternative subroutine for calculating flushing.

4.3.1 Detailed Description

Contains various subroutines for flushing. Which subroutine is called is determined by flush_flag.

Author

<Philipp Griewank, IMPRS>

Revision History

Sang into existence for very 1D column by Philipp Griewank, IMPRS (2010-10-20) First stable release by Philipp Griewank, IMPRS (2010-11-27) Freeboard calculation outsourced to mo_functions by Philipp Griewank, IMPRS (2010-11-27) Drainage through cracks is added by Philipp Griewank, IMPRS (2011-02-24)

4.3.2 Function/Subroutine Documentation

4.3.2.1 subroutine mo_flush::flush3 (REAL(wp),intent(in) *freeboard,* REAL(wp),dimension(nlayer),intent(in) *psi_l,* REAL(wp),dimension(nlayer),intent(inout) *thick,* REAL(wp),intent(in) *thick_0,* REAL(wp),dimension(nlayer),intent(inout) *S_abs,* REAL(wp),dimension(nlayer),intent(inout) *H_abs,* REAL(wp),dimension(nlayer),intent(inout) *m,* REAL(wp),dimension(nlayer),intent(in) *T,* REAL(wp),intent(in) *dt,* INTEGER,intent(in) *Nlayer,* INTEGER,intent(inout) *N_active,* REAL(wp),intent(in) *T_bottom,* REAL(wp),intent(in) *S_bu_bottom,* REAL(wp),intent(inout) *melt_thick,* INTEGER,intent(in) *debug_flag,* INTEGER,intent(in) *flush_heat_flag,* REAL(wp),dimension(nlayer+1,nlayer+1),intent(inout),optional *fl_brine_bgc*)

Subroutine for complex flushing.

Each layer splits the flushing brine into a fraction that moves downward, and a fraction that leaves the ice. A fraction of the top layer is considered melt water. This approach uses hydraulic resistivity R = mu$*$thick/perm . The hydraulic head is assumed to be the freeboard. The vertical resistance R_v of each layer is a determined by its viscosity $*$ thickness divided by it's permeability. Additionally, each layer is given horizontal resistivity R_h. It is assumed that there is an average length horizontally which brine needs to flow to reach a drainage feature in the ice. We assume this length is a linear function of the ice thickness. The only tuning parameter is para_flush_horiz. The total resistance of layer i to the bottom is R.

For flush_heat_flag==2 the amount of heat which leaves by dynamics from the lowest layer is added to the lowest layer to keep results comparable to the other approaches. See PhD Griewank for details

Revision History

Invented by Philipp Griewank, IMPRS (2012-06-15) Trying to add brine fluxes by Philipp Griewank, IMPRS (2014-02-01)

4.3.2.2 subroutine mo_flush::flush4 (REAL(wp),dimension(nlayer),intent(in) *psi_l*, REAL(wp),dimension(nlayer),intent(inout) *thick*, REAL(wp),dimension(nlayer),intent(in) *T*, REAL(wp),intent(in) *thick_0*, REAL(wp),dimension(nlayer),intent(inout) *S_abs*, REAL(wp),dimension(nlayer),intent(inout) *H_abs*, REAL(wp),dimension(nlayer),intent(inout) *m*, REAL(wp),intent(in) *dt*, INTEGER,intent(in) *Nlayer*, INTEGER,intent(in) *N_active*, INTEGER,intent(in) *N_top*, INTEGER,intent(in) *N_middle*, INTEGER,intent(in) *N_bottom*, REAL(wp),intent(inout) *melt_thick*, INTEGER,intent(in) *debug_flag*)

An alternative subroutine for calculating flushing.

Simplified approach. Melt_thick of top layer is simply removed with brine salinity. Salinity of a layer is reduced if the solid fraction is lower than that of the layer above it. Flushing stops as soon as a layer has a higher solid fraction than the layer below it.

Revision History

Invented by Philipp Griewank, IMPRS (2012-07-9)

4.4 mo_functions Module Reference

Module houses functions which have no home :(.

Functions/Subroutines

- REAL(wp) func_density (T, S)

  Calculates the physical density for given S and T.

- REAL(wp) func_freeboard (N_active, Nlayer, psi_s, psi_g, m, thick, m_snow)

  Calculates the freeboard of the 1d ice column.

- REAL(wp) func_albedo (thick_snow, T_snow, psi_l, thick_min, albedo_flag)

  Calculates the albedo.

- REAL(wp) func_sat_O2 (T, S_bu)

  Calculates the oxygen saturation as a function of salinity and temperature.

- REAL(wp) func_T_freeze (S_bu, salt_flag)

  Calculates the freezing temperature. Salt_flag determines if either ocean salt or NAcl is used.

- subroutine sub_notzflux (time, fl_sw, fl_rest)

  Calculates the incoming shortwave and other fluxes according to p. 193-194 PhD Notz.

- subroutine sub_input (length_input, fl_sw_input, fl_lw_input, T2m_input, precip_input, time_-input)

  Reads in data for atmoflux_flag ==2.

- subroutine sub_turb_flux (T_bottom, S_bu_bottom, T, S_abs, m, dt, N_bgc, bgc_bottom, bgc_abs)

  Calculates salt and tracer mixing between lowest layer and underlying water.

- subroutine sub_melt_thick (psi_l, psi_s, psi_g, T, T_freeze, T_top, fl_Q, thick_snow, dt, melt_thick, thick, thick_min)

  Calculates the thickness of the meltwater film.

- subroutine sub_melt_snow (melt_thick, thick, thick_snow, H_abs, H_abs_snow, m, m_snow, psi_-g_snow)

  Calculates how the meltwater film interacts with snow.

4.4.1 Detailed Description

Module houses functions which have no home :(. Created because I wanted to calculate the freeboard separately and didn't know where to put it.

Revision History

Ribbon cut by Philipp Griewank 2011-01-07

4.4.2 Function/Subroutine Documentation

4.4.2.1 REAL(wp) mo_functions::func_albedo (REAL(wp),intent(in) *thick_snow,* REAL(wp),intent(in) *T_snow,* REAL(wp),intent(in) *psi_l,* REAL(wp),intent(in) *thick_min,* INTEGER,intent(in) *albedo_flag*)

Calculates the albedo.

Calculates the albedo according to top conditions. This is not a good albedo scheme! It is only a quick approach. Non-continuous switching between wet and dry ice. Linear change from wet ice to water. Linear change from ice_dry snow for snow thinner than 30cm.

psi_l(1)> 0.75 water psi_l(1)> 0.6 linear change from wet ice to water psi_l(1)> 0.2 wet ice psi_l(1)< 0.2 -> dry ice T_snow = 0 -> wet snow T_snow < 0 -> dry snow

Revision History

Built to spill by Philipp Griewank (2011-02-12)

4.4.2.2 REAL(wp) mo_functions::func_density (REAL(wp),intent(in) *T,* REAL(wp),intent(in) *S*)

Calculates the physical density for given S and T.

Although the model treats Salinity as a massless tracer, sometimes it is necessary to determine the exact density for specific purposes. First implemented to calculate simple turbulence between liquid layer and ocean. Uses following simplification of Frank J. Millero and Alain Poisson 1981: Density = density_0 +A*S+B*S**1.5

Revision History

Started by Philipp Griewank (2011-02-24)

4.4.2.3 REAL(wp) mo_functions::func_freeboard (INTEGER,intent(in) *N_active*, INTEGER,intent(in) *Nlayer*, REAL(wp),dimension(nlayer),intent(in) *psi_s*, REAL(wp),dimension(nlayer),intent(in) *psi_g*, REAL(wp),dimension(nlayer),intent(in) *m*, REAL(wp),dimension(nlayer),intent(in) *thick*, REAL(wp),intent(in) *m_snow*)

Calculates the freeboard of the 1d ice column.

The freeboard is calculated by first finding out which layer is at water level, and then finding out how deep the layer is submerged. For the correct freeboard the mass above water equals the buoyancy of the submerged part. Since the density of each layer is constant, step two can be calculated explicitly. The freeboard is the distance from the top of the ice to the water level. If snow pushes the ice underwater the freeboard becomes negative

Revision History

Built to spill by Philipp Griewank (2011-01-07) Negative freeboard included by Philipp Griewank (2011-01-09) Patched bug by Philipp Griewank (2011-03-10)

4.4.2.4 REAL(wp) mo_functions::func_sat_O2 (REAL(wp),intent(in) *T*, REAL(wp),intent(in) *S_bu*)

Calculates the oxygen saturation as a function of salinity and temperature.

Calculates the concentration of oxygen dissolved in freshwater and seawater in equilibrium with the atmosphere The value should be umol/kg. I switched to the solubility of nitrogen, oxygen and argon in water and sea wate from Weiss R.F. 1970 because I couldn't get the other one to work out

Revision History

Written by Dr. Philipp Griewank (2014-02-25)

4.4.2.5 REAL(wp) mo_functions::func_T_freeze (REAL(wp),intent(in) *S_bu*, INTEGER,intent(in) *salt_flag*)

Calculates the freezing temperature. Salt_flag determines if either ocean salt or NAcl is used.

Revision History

Written to procrastinate by Philipp Griewank (2011-05-05)

4.4.2.6 subroutine mo_functions::sub_input (INTEGER,intent(in) *length_input*, REAL(wp),dimension(:),intent(out),allocatable *fl_sw_input*, REAL(wp),dimension(:),intent(out),allocatable *fl_lw_input*, REAL(wp),dimension(:),intent(out),allocatable *T2m_input*, REAL(wp),dimension(:),intent(out),allocatable *precip_input*, REAL(wp),dimension(:),intent(out),allocatable *time_input*)

Reads in data for atmoflux_flag ==2.

Standard setup used for testcase 4 and all Griewank & Notz 2013/14 reanalysis forced runs is 4.5 years of three hourly values of shortwave incoming, longwave incoming, two meter T, and total precipitation. Data is read from ascii files and stored in long 1D arrays. ERA-interim derived input files in the standard length for various Arctic locations are located under /input/ERA/ Latent and sensible heat fluxes are not included, but could be added if needed.

Revision History

Moved here from mo_grotz by Philipp Griewank (2014-04-20)

4.4.2.7 subroutine mo_functions::sub_melt_snow (REAL(wp),intent(inout) *melt_thick,* REAL(wp),intent(inout) *thick,* REAL(wp),intent(inout) *thick_snow,* REAL(wp),intent(inout) *H_abs,* REAL(wp),intent(inout) *H_abs_snow,* REAL(wp),intent(inout) *m,* REAL(wp),intent(inout) *m_snow,* REAL(wp),intent(inout) *psi_g_snow*)

Calculates how the meltwater film interacts with snow.

Is activated when a thin snow layer (thinner then thick_min) is on top of meltwater. The snow is flooded and turned into ice.

Revision History

Put together by Philipp Griewank (2011-10-17)

4.4.2.8 subroutine mo_functions::sub_melt_thick (REAL(wp),intent(in) *psi_l,* REAL(wp),intent(in) *psi_s,* REAL(wp),intent(in) *psi_g,* REAL(wp),intent(in) *T,* REAL(wp),intent(in) *T_freeze,* REAL(wp),intent(in) *T_top,* REAL(wp),intent(in) *fl_Q,* REAL(wp),intent(in) *thick_snow,* REAL(wp),intent(in) *dt,* REAL(wp),intent(out) *melt_thick,* REAL(wp),intent(inout) *thick,* REAL(wp),intent(in) *thick_min*)

Calculates the thickness of the meltwater film.

If the top ice layer is being melted (T_top>T_freeze) it is assumed that a thin meltwater film appears at the top. The thickness of this film is determined by the amount of incoming heat and diffusive transport. The incoming heat is an input (fl_q(1)) and the diffusive heat is (T(1)-T_freeze)/R. See the thermodynamics section for R. The thickness of the meltlayer is determined by dividing the heat intake of the meltwater film by the amount of latent heat needed to melt the solid fraction of the top layer. If the solid fractions sinks below a given threshold (psi_s_top_min) a different approach is used. The melt thickness is then calculated by assuming that the ice below the meltwater film has a solid fraction of psi_s_top_min. Although the thickness can be reduced, variations of mass, salinity and enthalpy are calculated in the flushing subroutine.

Revision History

Introduced by Philipp Griewank (2011-05-09)

4.4.2.9 subroutine mo_functions::sub_notzflux (REAL(wp),intent(in) *time,* REAL(wp),intent(out) *fl_sw,* REAL(wp),intent(out) *fl_rest*)

Calculates the incoming shortwave and other fluxes according to p. 193-194 PhD Notz.

Simplified version of the Untersteiner Fluxes. Returns only two fluxes as a function of time. Simplified Year, 12 months of 30 days. fl_sw is set to zero for November till February Returns fluxes for day with day zero being 1. Jan. Depending on when the run starts the time should be modified when calling

Revision History

Ripped from Dirk by Philipp Griewank (2011-02-13)

4.4.2.10 subroutine mo_functions::sub_turb_flux (REAL(wp),intent(in) *T_bottom*, REAL(wp),intent(in) *S_bu_bottom*, REAL(wp),intent(in) *T*, REAL(wp),intent(inout) *S_abs*, REAL(wp),intent(in) *m*, REAL(wp),intent(in) *dt*, INTEGER,intent(in) *N_bgc*, REAL(wp),dimension(n_bgc),intent(in),optional *bgc_bottom*, REAL(wp),dimension(n_bgc),intent(inout),optional *bgc_abs*)

Calculates salt and tracer mixing between lowest layer and underlying water.

Very simple turbulence assumption which mixes the lowest layer with the underlying water. Based on assumption that there is a constant amount of turbulence A. This turbulence is amplified when the lowest layer is denser then the ocean mixed layer. And also dampened when the lowest layer is less dense then the mixed layer. Assumption; turb=A∗exp(B(density_layer-density_ocean)) A and B set in parameters.i A = turb_A , B = turb_B

Revision History

Moved from grotz by Philipp Griewank (2014-04-2)

4.5 mo_grav_drain Module Reference

Computes the Salt fluxes caused by gravity drainage.

Functions/Subroutines

- subroutine fl_grav_drain (S_br, S_bu, psi_l, psi_s, psi_g, thick, S_abs, H_abs, T, m, dt, Nlayer, N_-active, ray, T_bottom, S_bu_bottom, grav_drain, grav_temp, grav_salt, grav_heat_flag, harmonic_-flag, fl_brine_bgc)

  Calculates fluxes caused by gravity drainage.

- subroutine fl_grav_drain_simple (psi_s, psi_l, thick, S_abs, S_br, Nlayer, N_active, ray, grav_drain, harmonic_flag)

  Calculates salinity to imitate the effects gravity drainage.

4.5.1 Detailed Description

Computes the Salt fluxes caused by gravity drainage.

Author

<Philipp Griewank, IMPRS>

Revision History

Injected with life by Philipp Griewank, IMPRS (<2010-08-27>)

4.5.2 Function/Subroutine Documentation

4.5.2.1 subroutine mo_grav_drain::fl_grav_drain (REAL(wp),dimension(nlayer),intent(in) *S_br*, REAL(wp),dimension(nlayer),intent(in) *S_bu*, REAL(wp),dimension(nlayer),intent(in) *psi_l*, REAL(wp),dimension(nlayer),intent(in) *psi_s*, REAL(wp),dimension(nlayer),intent(in) *psi_g*, REAL(wp),dimension(nlayer),intent(in) *thick*, REAL(wp),dimension(nlayer),intent(inout) *S_abs*, REAL(wp),dimension(nlayer),intent(inout) *H_abs*, REAL(wp),dimension(nlayer),intent(in) *T*, REAL(wp),dimension(nlayer),intent(inout) *m*, REAL(wp),intent(in) *dt*, INTEGER,intent(in) *Nlayer*, INTEGER,dimension(n_active),intent(in) *N_active*, REAL(wp),dimension(nlayer-1),intent(out) *ray*, REAL(wp),intent(in) *T_bottom*, REAL(wp),intent(in) *S_bu_bottom*, REAL(wp),intent(inout) *grav_drain*, REAL(wp),intent(inout) *grav_temp*, REAL(wp),intent(inout) *grav_salt*, INTEGER,intent(in) *grav_heat_flag*, INTEGER,intent(in) *harmonic_flag*, REAL(wp),dimension(nlayer+1,nlayer+1),intent(inout),optional *fl_brine_bgc*)

Calculates fluxes caused by gravity drainage.

If the Rayleigh number of a layer is higher then the critical value, brine leaves the layer by a theoretical brine channel. The discharged brine flows downward through all layers directly into the underlying ocean. To preserve mass the same amount of water flows upwards through all lower layers. In contrast to the downward flux the upward flux is assumed to be in thermal equilibrium thus moving salt and heat to each layer. The upward flux is a standard upwind advection. The downward flux of a layer over the timestep is = x∗(Ray-Ray_crit)∗dt∗thick. ______________________________________ | ->__|__<- |_______________ | v |__________________ | ^ ->| |<- ^ |_______________| |__________________ | ^ ^ ->|||||<- ^ ^ |_______________|vvv|__________________ ^ ^ ^ ^ ^ ^ This superb ascii art is supposed to show how the assumed fluxes flow downward through a brine channel and upwards through the layer. x and r are passed on to enable easy optimization. The effect of the upward moving brine is calculated in mass_transfer.

IMPORTANT: The height assumptions are special. The bottom of the ice edge is assumed to be at psi_s(N_active)/psi_s_min ∗thick_0

The first approach assumed that brine drainage occurred between two layers but performed poorly.

If grav_heat_flag is set to 2 the amount of heat transported out of the ice will be compensated in the lowest layer

Revision History

created by Philipp Griewank, IMPRS (2010-08-27) Completely revised to assume brine channels by Philipp Griewank , IMPRS (2010-11-05) Mass_transfer is used to advect H and S by Philipp Griewank, IMPRS (2010-11-05) Added condition S_br(k)>S_br(k+1) by Philipp Griewank. IMPRS (2011-04-29) Added harmonic mean for permeability by Philipp Griewank (2014-01-05)

Parameters

ray Rayleigh number

4.5.2.2 subroutine mo_grav_drain::fl_grav_drain_simple (REAL(wp),dimension(nlayer),intent(in) *psi_s*, REAL(wp),dimension(nlayer),intent(in) *psi_l*, REAL(wp),dimension(nlayer),intent(in) *thick*, REAL(wp),dimension(nlayer),intent(inout) *S_abs*, REAL(wp),dimension(nlayer),intent(in) *S_br*, INTEGER,intent(in) *Nlayer*, INTEGER,intent(in) *N_active*, REAL(wp),dimension(nlayer-1),intent(out) *ray*, REAL(wp),intent(inout) *grav_drain*, INTEGER,intent(in) *harmonic_flag*)

Calculates salinity to imitate the effects gravity drainage.

Based on the assumption that super critical Rayleigh numbers are quickly reduced below the critical Rayleigh number. Proposed as a very simplified parametrisation of gravity drainage. Includes no fluxes of any kind, instead bulk salinity is simply reduced when ever the Rayleigh number is above the critical values. The parametrization begins from the bottom layers and moves upward.

Revision History

created by Philipp Griewank, IMPRS (2012-01-01)

Parameters

ray Rayleigh number

4.6 mo_grotz Module Reference

SAMSIM Semi-Adaptive Multi-phase Sea-Ice Model.

Functions/Subroutines

- subroutine grotz (testcase, description)
  Main subroutine of SAMSIM, a 1D thermodynamic seaice model. A semi-adaptive grid is used which is managed by mo_layer_dynamics. To many things happen in this subroutine to describe in this description, you'll just have to go through it.

4.6.1 Detailed Description

SAMSIM Semi-Adaptive Multi-phase Sea-Ice Model. This model was developed from scratch by Philipp Griewank during and after his PhD at Max Planck Institute of Meteorology from 2010-2014. The code is intended to be understandable and is subroutines, modules, functions, parameters, and global variables all have doxygen compatible descriptions. In addition to the doxygen generated description a getting started document helps first time users get the model running, and some python plotscripts are available to plot model output.

The module mo_grotz is the most important subroutine in which the timestepping occurs. (Named after GRiewank nOTZ) Mo_grotz is called by SAMSIM.f90. SAMSIM.f90's only purpose is to set the testcase number and description string.

Author

Philipp Griewank (Max Planck Institute of Meteorology)

Date

2012-08-28

4.6.2 Function/Subroutine Documentation

4.6.2.1 subroutine mo_grotz::grotz (INTEGER,intent(in) *testcase*, CHARACTER∗12000,intent(in) *description*)

Main subroutine of SAMSIM, a 1D thermodynamic seaice model. A semi-adaptive grid is used which is managed by mo_layer_dynamics. To many things happen in this subroutine to describe in this description, you'll just have to go through it.

IMPORTANT: To get the correct freshwater amount make sure the freshwater is calculated using a salinity value to compare against.

Common errors leading to termination are: too small timestep, bad programming

Revision History

Basic thermodynamics and layer_dynamics for fixed boundaries seem stable, backup made. by griewank (2010-08-10)

Parameters

description String to describes simulation which is output into dat_settings

4.7 mo_heat_fluxes Module Reference

Computes all heat fluxes.

Functions/Subroutines

- subroutine sub_heat_fluxes ()

  Computes surface temperature and heatfluxes.

4.7.1 Detailed Description

Computes all heat fluxes. Everything important happens in sub_heat_fluxes.

Author

Philipp Griewank

Revision History

Copy and pasted into existence by Philipp Griewank (2014-04-02)

4.7.2 Function/Subroutine Documentation

4.7.2.1 subroutine mo_heat_fluxes::sub_heat_fluxes ( )

Computes surface temperature and heatfluxes.

Major subroutine, calculates all atmospheric energy fluxes and applies both atmospheric and oceanic fluxes. Is one of the only subroutines to directly use mo_data because so many variables are needed.

There are three different ways to calculate atmospheric heat fluxes implemented which are defined using boundflux_flag.

- Boundflux_flag: 1 imitates top cooling plate by setting a fixed surface temperature, heat flux is derived from the T gradient from the surface to the top layer
- Boundflux_flag: 2 balances incoming and outgoing radiation to determine the surface temperature, heat flux is then calculated as in boundflux_flag 1. Some of the ice penetrates into the ice as is absorbed according to Beer's law. Optical properties are defined by the parameters emissivity_ice, emissivity_snow, extinct, and penetr.
- Boundflux_flag: 3 assumes the atmospheric heat flux is proportional to the difference between the top layer temperature and the air temperature.

For 1 and 2 the surface temperature in turn determines the atmospheric heat flux into the snow or ice. Atmoflux_flag is important for boundflux_flag 2, as it determines which atmospheric fluxes are used.

- Atmoflux_flag: 1 Mean climatology fluxes of Notz are used (see sub_notz)
- Atmoflux_flag: 2 Imported values are used, see sub_input for more info on reading in data.
- Atmoflux_flag: 3 Prescribed values are used (e.g. testcase 5).

Melting occurs when the surface T is above the melting temperature of the top layer

- Boundflux_flag: 1 atmospheric flux is limited by the parameter max_flux_plate which represents the maximum heating capacity of the plate
- Boundflux_flag: 2 the atmospheric heat flux is given by the difference between incoming and outgoing radiation
- Boundflux_flag: 3 works the same during melt and freezing, but a different proportionality parameter is used (alpha_flux_stable) because the air above the ice is assumed to be stably stratified.

Boundflux_flag 1 and 3 are not made to work with snow. If you need snow you'll have to implement snow cover yourself. For a detailed look at what is happening see the source code.

The snow layer is treated differently based on the snow thickness.

- If the snow layer is thinner than thick_min/100 it is simply ignored.
- If the snow layer is thinner than thick_min but thicker than thick_min/100 the snow and top ice layer are assumed to have the same temperature and are coupled using snow_coupling.
- If the snow layer is thicker than thick_min it is treated totally separately.

Revision History

First version by Philipp Griewank (2014-04-02)

4.8 mo_init Module Reference

Allocates Arrays and sets initial data for a given testcase for SAMSIM.

Functions/Subroutines

- subroutine init (testcase)

  Sets initial conditions according to which testcase is chosen.

- subroutine sub_allocate (Nlayer)

  Allocates Arrays.

- subroutine sub_allocate_bgc (Nlayer, N_bgc)

  Allocates BGC Arrays.

- subroutine sub_deallocate

  Deallocates Arrays.

4.8.1 Detailed Description

Allocates Arrays and sets initial data for a given testcase for SAMSIM.

Author

Philipp Griewank, IMPRS-ESM

Revision History

first version created to deal with first multi-layer tests. by Philipp Griewank, IMPRS (2010-07-22)

4.8.2 Function/Subroutine Documentation

4.8.2.1 subroutine mo_init::init (INTEGER,intent(in) *testcase*)

Sets initial conditions according to which testcase is chosen.

For different initial conditions the Arrays are allocated and the initial values are set. Following must always be: 1. Nlayer = N_top+N_middle+N_bottom 2. N_active is set correctly, N_active <= Nlayer 3. fl_q_-bottom >= 0 4. T_bottom > freezing point of for S_bu_bottom 5. A too high dt for a too small thick_0 leads to numerical thermodynamic instability. For a conservative guess dt [s] should be smaller than 250000 ∗ (dz [m])∗∗2

Testcase 1

- Testcase 1 is a replication of lab experiments conducted in tanks cooled from above by a cooling plate using the boundflux_flag 1.
- In this testcase the cooling plate Temperature T_top changes every 12 hours to imitate the experiments Dirk Notz conducted in his PhD.
- This testcase was used to optimize the free parameters of the gravity drainage parametrization (see Griewank Notz 2013/14).
- Can also be run with bgc tracers.

Testcase 2

- Testcase is an example of how to simulate ice growth and melt in cooling chambers.
- Boundflux_flag 3 is used, which uses T2m as the air temperature in the cooling chamber.
- The surface flux heat flux is proportional to the ice-air temperature difference (T_top-T2m).
- When reproducing cooling chamber experiments the alpha flux parameters need to be tuned, and a module in mo_testcase_specifics is needed to set/ T2m over time.
- The heat flux in the water from below (fl_q_bottom) for such experiments can be very hard to reproduce if the heat input is not carefully measured from all pumps or similar devices used.

Testcase 3

- Uses interpolated climate mean forcing from Notz and a constant oceanic heat flux (fl_q_bottom) to grow idealized arctic sea ice.
- Is generally intended as a numerically cheap testcase to check for effects of code changes.
- Is also useful when runs over many years are needed.
- The amount of liquid and solid precipitation is set in sub_test3 of mo_testcase specifics.

Testcase 4

- Uses three hourly reanalysis forcing over 4.5 years.
- Is set up to start in July.
- Prescribes annual cycle of oceanic heat flux.
- Requires the proper input data to be copied into the executable folder (see sub_input).
- Is more computer intensive
- Was used a lot for Griewank & Notz 2013/2014

Revision History

First set up by Philipp Griewank, IMPRS (2010-07-22>)

4.8.2.2 subroutine mo_init::sub_allocate (INTEGER,intent(in) *Nlayer*)

Allocates Arrays.

For a given number of layers Nlayers all arrays are allocated

Parameters

Nlayer number of layers

4.8.2.3 subroutine mo_init::sub_allocate_bgc (INTEGER,intent(in) *Nlayer,* INTEGER,intent(in) *N_bgc*)

Allocates BGC Arrays.

4.8.2.4 subroutine mo_init::sub_deallocate ( )

Deallocates Arrays.

4.9 mo_layer_dynamics Module Reference

Mo_layer_dynamics contains all subroutines for the growth and shrinking of layer thickness.

Functions/Subroutines

- subroutine layer_dynamics (phi, N_active, Nlayer, N_bottom, N_middle, N_top, m, S_abs, H_abs, thick, thick_0, T_bottom, S_bu_bottom, bottom_flag, debug_flag, N_bgc, bgc_abs, bgc_bottom)

    Organizes the Semi-Adaptive grid SAMSIM uses.

- subroutine top_melt (Nlayer, N_active, N_bottom, N_middle, N_top, thick_0, m, S_abs, H_abs, thick, N_bgc, bgc_abs)
- subroutine top_grow (Nlayer, N_active, N_bottom, N_middle, N_top, thick_0, m, S_abs, H_abs, thick, N_bgc, bgc_abs)

    Top grow subroutine.

4.9.1 Detailed Description

Mo_layer_dynamics contains all subroutines for the growth and shrinking of layer thickness. The middle layers have flexible thickness in contrast to the lower and upper layers which have static thickness. The details are provided in the separate subroutines.

Author

Philipp Griewank, IMPRS

Revision History

Shrinking and growth at the bottom are started by Philipp Griewank, IMPRS (2010-07-28)

4.9.2 Function/Subroutine Documentation

4.9.2.1 subroutine mo_layer_dynamics::layer_dynamics (REAL(wp),dimension(nlayer),intent(in) *phi,* INTEGER,intent(inout) *N_active,* INTEGER,intent(in) *Nlayer,* INTEGER,intent(in) *N_bottom,* INTEGER,intent(in) *N_middle,* INTEGER,intent(in) *N_top,* REAL(wp),dimension(nlayer),intent(inout) *m,* REAL(wp),dimension(nlayer),intent(inout) *S_abs,* REAL(wp),dimension(nlayer),intent(inout) *H_abs,* REAL(wp),dimension(nlayer),intent(inout) *thick,* REAL(wp),intent(in) *thick_0,* REAL(wp),intent(in) *T_bottom,* REAL(wp),intent(in) *S_bu_bottom,* INTEGER,intent(in) *bottom_flag,* INTEGER,intent(in) *debug_flag,* INTEGER,intent(in) *N_bgc,* REAL(wp),dimension(nlayer,n_bgc),intent(inout),optional *bgc_abs,* REAL(wp),dimension(n_bgc),intent(in),optional *bgc_bottom*)

Organizes the Semi-Adaptive grid SAMSIM uses.

Modifies the grid and all core variables due to growth or melt. Calls the different subroutines according to current conditions. All subroutines can be called with or without biogeochemical tracers active, which is triggered by providing bgc_abs when calling the subroutine. See Griewank PhD thesis for a full description of the grid.

Conditions under which following layer dynamics subroutines are called:

- bottom_melt: lowest layer is ice free, second lowest layer has a solid fraction smaller than phi_s_-min/2, and all Nlayer layers are active.
- bottom_melt_simple: lowest layer is ice free, second lowest layer has a solid fraction smaller than phi_s_min/2, and not all Nlayer layers are active.
- bottom_melt_simple: lowest layer is ice free, second lowest layer has a solid fraction smaller than phi_s_min/2, all Nlayer layers are active, and the thickness of the middle layers equals thick_0
- bottom_growth_simple: lowest layer has a solid fraction higher than psi_s_min, and not all Nlayer layers are active
- bottom_growth: lowest layer has a solid fraction higher than psi_s_min, and all Nlayer layers are active
- top_grow: top layer thicker than 3/2 ∗ thick_0
- top_melt: top layer thinner than 1/2 ∗ thick_0

If debug_flag is set to 2 the layer values will be written into the debug output (thermoXX.dat) before and after layer dynamics with a string to identify which subroutine was called

Revision History

created by Philipp Griewank, IMPRS (2010-07-29) first complete and hopefully stable version by Philipp Griewank, IMPRS (2010-08-10)

4.9.2.2 subroutine mo_layer_dynamics::top_grow (INTEGER,intent(in) *Nlayer,* INTEGER,intent(inout) *N_active,* INTEGER,intent(in) *N_bottom,* INTEGER,intent(in) *N_middle,* INTEGER,intent(in) *N_top,* REAL(wp),intent(in) *thick_0,* REAL(wp),dimension(nlayer),intent(inout) *m,* REAL(wp),dimension(nlayer),intent(inout) *S_abs,* REAL(wp),dimension(nlayer),intent(inout) *H_abs,* REAL(wp),dimension(nlayer),intent(inout) *thick,* INTEGER,intent(in) *N_bgc,* REAL(wp),dimension(nlayer,n_bgc),intent(inout),optional *bgc_abs*)

Top grow subroutine.

Should be called when the top layer is thicker then 1.5 ∗thick_0. If N_active=Nlayer middle layers are expanded by thick_0/N_middle and top layers are moved one down. IF N_active<Nlayer then N_active=N_-active+1 and all layers are shifted downwards.

Revision History

Started by Philipp Griewank, IMPRS (2011-05-10>)

4.9.2.3 subroutine mo_layer_dynamics::top_melt (INTEGER,intent(in) *Nlayer,* INTEGER,intent(inout) *N_active,* INTEGER,intent(in) *N_bottom,* INTEGER,intent(in) *N_middle,* INTEGER,intent(in) *N_top,* REAL(wp),intent(in) *thick_0,* REAL(wp),dimension(nlayer),intent(inout) *m,* REAL(wp),dimension(nlayer),intent(inout) *S_abs,* REAL(wp),dimension(nlayer),intent(inout) *H_abs,* REAL(wp),dimension(nlayer),intent(inout) *thick,* INTEGER,intent(in) *N_bgc,* REAL(wp),dimension(nlayer,n_bgc),intent(inout),optional *bgc_abs*)

4.10 mo_mass Module Reference

Regulates mass transfers and their results.

Functions/Subroutines

- subroutine mass_transfer (Nlayer, N_active, T, H_abs, S_abs, S_bu, T_bottom, S_bu_bottom, fl_-m)

  Calculates the effects of mass transfers on H_abs and S_abs.

- subroutine expulsion_flux (thick, V_ex, Nlayer, N_active, psi_g, fl_m, m)

  Generates the fluxes caused by expulsion.

- subroutine bgc_advection (Nlayer, N_active, N_bgc, fl_brine_bgc, bgc_abs, psi_l, T, S_abs, m, thick, bgc_bottom)

  Calculates how the brine fluxes stored in fl_brine_bgc advect bgc tracers.

4.10.1 Detailed Description

Regulates mass transfers and their results. Ultimately all processes which involve a mass flux should be stored here.

Author

Philipp Griewank, IMPRS

Revision History

Begin implementing Expulsion by Philipp Griewank, IMPRS (2010-08-24)

4.10.2 Function/Subroutine Documentation

4.10.2.1 subroutine mo_mass::bgc_advection (INTEGER,intent(in) *Nlayer,* INTEGER,intent(in) *N_active,* INTEGER,intent(in) *N_bgc,* REAL(wp),dimension(nlayer+1,nlayer+1),intent(in) *fl_brine_bgc,* REAL(wp),dimension(nlayer,n_bgc),intent(inout) *bgc_abs,* REAL(wp),dimension(nlayer),intent(in) *psi_l,* REAL(wp),dimension(nlayer),intent(in) *T,* REAL(wp),dimension(nlayer),intent(in) *S_abs,* REAL(wp),dimension(nlayer),intent(in) *m,* REAL(wp),dimension(nlayer),intent(in) *thick,* REAL(wp),dimension(n_-bgc),intent(in) *bgc_bottom*)

Calculates how the brine fluxes stored in fl_brine_bgc advect bgc tracers.

A very simple upwind strategy is employed. To avoid negative tracer densities, the maximum amount of advection is restricted to the current tracer content in a layer divided by three. Three is chosen as a limit as currently each layer can have a maximum of three flows leaving the layer (to the layer above, the layer below, and the lowest layer). The advection scheme is likely overly diffusive, but given the limitations we are working with (e.g. changing brine volumes) nothing more sophisticated can be applied easily.

For gases it might make sense to limit the brine density to saturation value in advecting brine, to take bubble formation into account. This needs to be specified in bgc_advection, and is a first attempt (both scientifically and code wise) which should be used with caution!

Revision History

Brought to life by Philipp Griewank, IMPRS (2014-02-10)

4.10.2.2 subroutine mo_mass::expulsion_flux (REAL(wp),dimension(nlayer),intent(in) *thick*, REAL(wp),dimension(nlayer),intent(in) *V_ex*, INTEGER,intent(in) *Nlayer*, INTEGER,intent(in) *N_active*, REAL(wp),dimension(nlayer),intent(inout) *psi_g*, REAL(wp),dimension(nlayer+1),intent(out) *fl_m*, REAL(wp),dimension(nlayer),intent(inout) *m*)

Generates the fluxes caused by expulsion.

Brine displaced by expansion of a freezing mushy layer lead to a mass, enthalpy and salt flux. This subroutine calculates the amount of brine which moves between the layers caused by V_ex and how the mass in the layers changes. Vary basic assumptions are made. Brine always moves downward (negative), no horizontal movement are allowed and gas pockets can be filled. The upper boundary layer is not permeable but the bottom one is. This subroutine was started as a quick and dirty way to simulate the bottom freezing experiment described in Notz 2005 p. 85

Revision History

Brought to life by Philipp Griewank, IMPRS (2010-08-24) Simplified by Philipp Griewank, IMPRS (2010-11-27)

4.10.2.3 subroutine mo_mass::mass_transfer (INTEGER,intent(in) *Nlayer*, INTEGER,intent(in) *N_active*, REAL(wp),dimension(nlayer),intent(in) *T*, REAL(wp),dimension(nlayer),intent(inout) *H_abs*, REAL(wp),dimension(nlayer),intent(inout) *S_abs*, REAL(wp),dimension(nlayer),intent(in) *S_bu*, REAL(wp),intent(in) *T_bottom*, REAL(wp),intent(in) *S_bu_bottom*, REAL(wp),dimension(nlayer+1),intent(in) *fl_m*)

Calculates the effects of mass transfers on H_abs and S_abs.

The effects of brine displaced by expulsion, flushing or drainage expansion lead to changes in mass, salt ans enthalpy. This subroutine calculates the effects on S_abs and H_abs. A very simple upwind strategy is employed, Brine from below has T and S_br of the lower layer, and brine from above T and S_br of the upper layer. To avoid negative salinity, the maximum amount of advective salt is the total salt content of the layer. The amount of mass transfered is calculated in other subroutines.

This subroutine was started as a quick and dirty way to simulate the bottom freezing experiment described in Notz 2005 p. 85 IMPORTANT: Before this subroutine expelled brine was removed from the system and its effects were determined in subroutine expulsion. S_bu must be up to date!

Revision History

Brought to life by Philipp Griewank, IMPRS (2010-08-24) Modified to work with all processes by Philipp Griewank, IMPRS (2010-11-27)

4.11 mo_output Module Reference

All things output.

Functions/Subroutines

- subroutine output_settings (description, testcase, N_top, N_bottom, Nlayer, fl_q_bottom, T_bottom, S_bu_bottom, thick_0, time_out, time_total, dt, boundflux_flag, atmoflux_flag, albedo_flag, grav_flag, flush_flag, flood_flag, grav_heat_flag, flush_heat_flag, harmonic_flag, prescribe_flag, salt_flag, turb_flag, bottom_flag, tank_flag, precip_flag, bgc_flag, N_bgc)

  Settings output.

- subroutine output (Nlayer, T, psi_s, psi_l, thick, S_bu, ray, format_T, format_psi, format_thick, format_snow, freeboard, thick_snow, T_snow, psi_l_snow, psi_s_snow, energy_stored, freshwater, total_resist, thickness, bulk_salin, grav_drain, grav_salt, grav_temp, T2m, T_top)

  Standard output.

- subroutine output_bgc (Nlayer, N_active, bgc_bottom, N_bgc, bgc_abs, psi_l, thick, m, format_bgc)

  Standard bgc output.

- subroutine output_raw (Nlayer, N_active, time, T, thick, S_bu, psi_s, psi_l, psi_g)

  Output for debugging purposes.

- subroutine output_raw_snow (time, T_snow, thick_snow, S_abs_snow, m_snow, psi_s_snow, psi_l_snow, psi_g_snow)

  Output for debugging purposes.

- subroutine output_raw_lay (Nlayer, N_active, H_abs, m, S_abs, thick, string)

  Output for debugging layer dynamics..

- subroutine output_begin (Nlayer, debug_flag, format_T, format_psi, format_thick, format_snow, format_T2m_top)

  Output files are opened and format strings are created.

- subroutine output_begin_bgc (Nlayer, N_bgc, format_bgc)

  Output files for bgc are opened and format strings are created.

4.11.1 Detailed Description

All things output. Used to clean up root.f90 and make it easier to implement changes to the output.

Author

<Philipp Griewank, IMPRS>

Revision History

Brought from the womb by Philipp Griewank, IMPRS (<2010-10-11>)

4.11.2 Function/Subroutine Documentation

4.11.2.1 subroutine mo_output::output (INTEGER,intent(in) *Nlayer,* REAL(wp),dimension(nlayer),intent(in) *T,* REAL(wp),dimension(nlayer),intent(in) *psi_s,* REAL(wp),dimension(nlayer),intent(in) *psi_l,* REAL(wp),dimension(nlayer),intent(in) *thick,* REAL(wp),dimension(nlayer),intent(in) *S_bu,* REAL(wp),dimension(nlayer-1),intent(in) *ray,* CHARACTER∗12000,intent(in) *format_T,* CHARACTER∗12000,intent(in) *format_psi,* CHARACTER∗12000,intent(in) *format_thick,* CHARACTER∗12000,intent(in) *format_snow,* REAL(wp),intent(in) *freeboard,* REAL(wp),intent(in) *thick_snow,* REAL(wp),intent(in) *T_snow,* REAL(wp),intent(in) *psi_l_snow,* REAL(wp),intent(in) *psi_s_snow,* REAL(wp),intent(in) *energy_stored,* REAL(wp),intent(in) *freshwater,* REAL(wp),intent(in) *total_resist,* REAL(wp),intent(in) *thickness,* REAL(wp),intent(in) *bulk_salin,* REAL(wp),intent(in) *grav_drain,* REAL(wp),intent(in) *grav_salt,* REAL(wp),intent(in) *grav_temp,* REAL(wp),intent(in) *T2m,* REAL(wp),intent(in) *T_top*)

Standard output.

For time=n∗time_out data is exported.

Revision History

created by Philipp Griewank, IMPRS (2010-10-11)

4.11.2.2 subroutine mo_output::output_begin (INTEGER,intent(in) *Nlayer,* INTEGER,intent(in) *debug_flag,* CHARACTER∗12000,intent(out) *format_T,* CHARACTER∗12000,intent(out) *format_psi,* CHARACTER∗12000,intent(out) *format_thick,* CHARACTER∗12000,intent(out) *format_snow,* CHARACTER∗12000,intent(out) *format_T2m_top*)

Output files are opened and format strings are created.

Format strings are defined according to the number of layers used which define the output format. Files are opened.

Revision History

created by Philipp Griewank, IMPRS (2010-10-11) moved by Philipp Griewank, IMPRS (2011-03-09)

4.11.2.3 subroutine mo_output::output_begin_bgc (INTEGER,intent(in) *Nlayer,* INTEGER,intent(in) *N_bgc,* CHARACTER∗12000,intent(out) *format_bgc*)

Output files for bgc are opened and format strings are created.

Same thing as out_begin but for bgc Each tracer is outputted in bulk and in brine concentration in a separate file.

Revision History

created by Dr. Philipp Griewank, MPI (2014-02-07)

4.11.2.4 subroutine mo_output::output_bgc (INTEGER,intent(in) *Nlayer,* INTEGER,intent(in) *N_active,* REAL(wp),dimension(n_bgc),intent(in) *bgc_bottom,* INTEGER,intent(in) *N_bgc,* REAL(wp),dimension(nlayer,n_bgc),intent(in) *bgc_abs,* REAL(wp),dimension(nlayer),intent(in) *psi_l,* REAL(wp),dimension(nlayer),intent(in) *thick,* REAL(wp),dimension(nlayer),intent(in) *m,* CHARACTER∗12000,intent(in) *format_bgc*)

Standard bgc output.

For time=n∗time_out data is exported.

Revision History

created by Philipp Griewank, IMPRS (2014-02-06)

4.11.2.5 subroutine mo_output::output_raw (INTEGER,intent(in) *Nlayer,* INTEGER,intent(in) *N_active,* REAL(wp),intent(in) *time,* REAL(wp),dimension(nlayer),intent(in) *T,* REAL(wp),dimension(nlayer),intent(in) *thick,* REAL(wp),dimension(nlayer),intent(in) *S_bu,* REAL(wp),dimension(nlayer),intent(in) *psi_s,* REAL(wp),dimension(nlayer),intent(in) *psi_l,* REAL(wp),dimension(nlayer),intent(in) *psi_g*)

Output for debugging purposes.

Data for each layer is written out each time step to aid in finding errors or understanding model behavior.

Revision History

created by Philipp Griewank, IMPRS (2010-10-11)

4.11.2.6 subroutine mo_output::output_raw_lay (INTEGER,intent(in) *Nlayer,* INTEGER,intent(in) *N_active,* REAL(wp),dimension(nlayer),intent(in) *H_abs,* REAL(wp),dimension(nlayer),intent(in) *m,* REAL(wp),dimension(nlayer),intent(in) *S_abs,* REAL(wp),dimension(nlayer),intent(in) *thick,* CHARACTER∗6,intent(in) *string*)

Output for debugging layer dynamics..

Is used when debug_flag = 2 to track when which layer dynamics occur (see mo_layer_dynamics).

4.11.2.7 subroutine mo_output::output_raw_snow (REAL(wp),intent(in) *time,* REAL(wp),intent(in) *T_snow,* REAL(wp),intent(in) *thick_snow,* REAL(wp),intent(in) *S_abs_snow,* REAL(wp),intent(in) *m_snow,* REAL(wp),intent(in) *psi_s_snow,* REAL(wp),intent(in) *psi_l_snow,* REAL(wp),intent(in) *psi_g_snow*)

Output for debugging purposes.

Data of snow layer is written out at each time step to aid in finding errors or understanding model behavior.

Revision History

created by Philipp Griewank, IMPRS (2010-10-11)

4.11.2.8 subroutine mo_output::output_settings (CHARACTER*12000,intent(in) *description*, INTEGER,intent(in) *testcase*, INTEGER,intent(in) *N_top*, INTEGER,intent(in) *N_bottom*, INTEGER,intent(in) *Nlayer*, REAL(wp),intent(in) *fl_q_bottom*, REAL(wp),intent(in) *T_bottom*, REAL(wp),intent(in) *S_bu_bottom*, REAL(wp),intent(in) *thick_0*, REAL(wp),intent(in) *time_out*, REAL(wp),intent(in) *time_total*, REAL(wp),intent(in) *dt*, INTEGER,intent(in) *boundflux_flag*, INTEGER,intent(in) *atmoflux_flag*, INTEGER,intent(in) *albedo_flag*, INTEGER,intent(in) *grav_flag*, INTEGER,intent(in) *flush_flag*, INTEGER,intent(in) *flood_flag*, INTEGER,intent(in) *grav_heat_flag*, INTEGER,intent(in) *flush_heat_flag*, INTEGER,intent(in) *harmonic_flag*, INTEGER,intent(in) *prescribe_flag*, INTEGER,intent(in) *salt_flag*, INTEGER,intent(in) *turb_flag*, INTEGER,intent(in) *bottom_flag*, INTEGER,intent(in) *tank_flag*, INTEGER,intent(in) *precip_flag*, INTEGER,intent(in) *bgc_flag*, INTEGER,intent(in) *N_bgc*)

Settings output.

Writes important values to latter identify run.

Revision History

created by Philipp Griewank, IMPRS (2011-02-12)

4.12 mo_parameters Module Reference

Module determines physical constants to be used by the SAMSIM Seaice model.

Variables

- INTEGER, parameter wp = SELECTED_REAL_KIND(12, 307)

  set working precision _wp

- REAL, parameter pi = 3.1415_wp
- REAL, parameter grav = 9.8061_wp

  gravitational constant [m/s^2]

- REAL(wp), parameter k_s = 2.2_wp

  solid heat conductivity [J / m s K] 2.2

- REAL(wp), parameter k_l = 0.523_wp

  liquid heat conductivity [J / m s K] 0.523

- REAL(wp), parameter c_s = 2020.0_wp

  solid heat capacity [J/ kg K]

- REAL(wp), parameter c_s_beta = 7.6973_wp

  *linear solid heat capacity approximation [J/ kg K^2] c_s = c_s+c_s_beta*T*

- REAL(wp), parameter c_l = 3400._wp

  liquid heat capacity [J/ kg K]

- REAL(wp), parameter rho_s = 920._wp

  density of solid [kg / m^3]

- REAL(wp), parameter rho_l = 1028.0_wp

  density of liquid [kg / m^3]

- REAL(wp), parameter latent_heat = 333500._wp

  latent heat release [J/kg]

- REAL(wp), parameter zeroK = 273.15_wp

  Zero degrees Celsius in Kelvin [K].

- REAL(wp), parameter bbeta = 0.8_wp∗1e-3

  concentration expansion coefficient [kg / (m^3 ppt)]

- REAL(wp), parameter mu = 2.55_wp∗1e-3

  dynamic viscosity [kg /m s]

- REAL(wp), parameter kappa_l = k_l/rho_l/c_l

  heat diffusivity of water

- REAL(wp), parameter sigma = 5.6704_wp∗1e-8

  Stefan Boltzmann constant [W/(m^2∗K^4)].

- REAL(wp), parameter psi_s_min = 0.05_wp

  The amount of ice that the lowest layer can have before it counts as an ice layer.

- REAL(wp), parameter neg_free = -0.05_wp

  The distance the freeboard can be below 0 before water starts flooding through cracks.

- REAL(wp), parameter x_grav = 0.000584_wp
- REAL(wp), parameter ray_crit = 4.89_wp
- REAL(wp), parameter para_flush_horiz = 1.00_wp

  determines relationship of horizontal flow distance in during flushing (guess 1)

- REAL(wp), parameter para_flush_gamma = 0.9_wp

  Strength of desalination per timestep (guess).

- REAL(wp), parameter psi_s_top_min = 0.40_wp

  if psi_s is below this value meltwater forms (guess) 0.4

- REAL(wp), parameter ratio_flood = 1.50_wp

  Ratio of flooded to dissolve snow, plays an important role in subroutine flood.

- REAL(wp), parameter ref_salinity = 34._wp

  Reference salinity [g/kg] used to calculate freshwater column.

- REAL(wp), parameter rho_snow = 330_wp

  *density of new snow [kg/m**3]*

- REAL(wp), parameter gas_snow_ice = 0.10_wp

  volume of gas percentage in new snow ice due to flooding, no longer used

- REAL(wp), parameter gas_snow_ice2 = 0.20_wp

  volume of gas percentage in new snow ice due to snow melting (Eicken 95)

- REAL(wp), parameter emissivity_ice = 0.95_wp

  Emissivity of water and ice.

- REAL(wp), parameter emissivity_snow = 1.00_wp

  Emissivity of Snow.

- REAL(wp), parameter penetr = 0.30_wp

  Amount of penetrating sw radiation.

- REAL(wp), parameter extinc = 2.00_wp

  Extinction coefficient of ice.

- REAL(wp), parameter Turb_A = 0.1_wp*0.05_wp*rho_l/86400._wp

  Standard turbulence [kg/s] WARNING no source, just set so that 5cm of water are overturned each day.

- REAL(wp), parameter Turb_B = 0.05_wp

  *Exponential turbulence slope [m**3/kg] WARNING no source, simple guess.*

- REAL(wp) max_flux_plate = 50.0

  Maximal heating rate of a heating plate.

4.12.1 Detailed Description

Module determines physical constants to be used by the SAMSIM Seaice model. Many values are taken from Notz 2005, Table 5.2.

Revision History

Started by Philipp Griewank 2010-07-08

4.12.2 Variable Documentation

4.12.2.1 REAL(wp),parameter mo_parameters::bbeta = 0.8_wp*1e-3

concentration expansion coefficient [kg / (m^3 ppt)]

4.12.2.2 REAL(wp),parameter mo_parameters::c_l = 3400._wp

liquid heat capacity [J/ kg K]

4.12.2.3 REAL(wp),parameter mo_parameters::c_s = 2020.0_wp

solid heat capacity [J/ kg K]

4.12.2.4 REAL(wp),parameter mo_parameters::c_s_beta = 7.6973_wp

linear solid heat capacity approximation [J/ kg K^2] c_s = c_s+c_s_beta∗T

4.12.2.5 REAL(wp),parameter mo_parameters::emissivity_ice = 0.95_wp

Emissivity of water and ice.

4.12.2.6 REAL(wp),parameter mo_parameters::emissivity_snow = 1.00_wp

Emissivity of Snow.

4.12.2.7 REAL(wp),parameter mo_parameters::extinc = 2.00_wp

Extinction coefficient of ice.

4.12.2.8 REAL(wp),parameter mo_parameters::gas_snow_ice = 0.10_wp

volume of gas percentage in new snow ice due to flooding, no longer used

4.12.2.9 REAL(wp),parameter mo_parameters::gas_snow_ice2 = 0.20_wp

volume of gas percentage in new snow ice due to snow melting (Eicken 95)

4.12.2.10 REAL,parameter mo_parameters::grav = 9.8061_wp

gravitational constant [m/s^2]

4.12.2.11 REAL(wp),parameter mo_parameters::k_l = 0.523_wp

liquid heat conductivity [J / m s K] 0.523

4.12.2.12 REAL(wp),parameter mo_parameters::k_s = 2.2_wp

solid heat conductivity [J / m s K] 2.2

4.12.2.13 REAL(wp),parameter mo_parameters::kappa_l = k_l/rho_l/c_l

heat diffusivity of water

4.12.2.14 REAL(wp),parameter mo_parameters::latent_heat = 333500._wp

latent heat release [J/kg]

4.12.2.15 REAL(wp) mo_parameters::max_flux_plate = 50.0

Maximal heating rate of a heating plate.

4.12.2.16 REAL(wp),parameter mo_parameters::mu = 2.55_wp*1e-3

dynamic viscosity [kg /m s]

4.12.2.17 REAL(wp),parameter mo_parameters::neg_free = -0.05_wp

The distance the freeboard can be below 0 before water starts flooding through cracks.

4.12.2.18 REAL(wp),parameter mo_parameters::para_flush_gamma = 0.9_wp

Strength of desalination per timestep (guess).

4.12.2.19 REAL(wp),parameter mo_parameters::para_flush_horiz = 1.00_wp

determines relationship of horizontal flow distance in during flushing (guess 1)

4.12.2.20 REAL(wp),parameter mo_parameters::penetr = 0.30_wp

Amount of penetrating sw radiation.

4.12.2.21 REAL,parameter mo_parameters::pi = 3.1415_wp

4.12.2.22 REAL(wp),parameter mo_parameters::psi_s_min = 0.05_wp

The amount of ice that the lowest layer can have before it counts as an ice layer.

4.12.2.23 REAL(wp),parameter mo_parameters::psi_s_top_min = 0.40_wp

if psi_s is below this value meltwater forms (guess) 0.4

4.12.2.24 REAL(wp),parameter mo_parameters::ratio_flood = 1.50_wp

Ratio of flooded to dissolve snow, plays an important role in subroutine flood.

4.12.2.25 REAL(wp),parameter mo_parameters::ray_crit = 4.89_wp

4.12.2.26 REAL(wp),parameter mo_parameters::ref_salinity = 34._wp

Reference salinity [g/kg] used to calculate freshwater column.

4.12.2.27 REAL(wp),parameter mo_parameters::rho_l = 1028.0_wp

density of liquid [kg / m^3]

4.12.2.28 REAL(wp),parameter mo_parameters::rho_s = 920._wp

density of solid [kg / m^3]

4.12.2.29 REAL(wp),parameter mo_parameters::rho_snow = 330_wp

density of new snow [kg/m**3]

4.12.2.30 REAL(wp),parameter mo_parameters::sigma = 5.6704_wp*1e-8

Stefan Boltzmann constant [W/(m^2*K^4)].

4.12.2.31 REAL(wp),parameter mo_parameters::Turb_A = 0.1_wp*0.05_wp*rho_l/86400._wp

Standard turbulence [kg/s] WARNING no source, just set so that 5cm of water are overturned each day.

4.12.2.32 REAL(wp),parameter mo_parameters::Turb_B = 0.05_wp

Exponential turbulence slope [m**3/kg] WARNING no source, simple guess.

4.12.2.33 INTEGER,parameter mo_parameters::wp = SELECTED_REAL_KIND(12, 307)

set working precision _wp

4.12.2.34 REAL(wp),parameter mo_parameters::x_grav = 0.000584_wp

4.12.2.35 REAL(wp),parameter mo_parameters::zeroK = 273.15_wp

Zero degrees Celsius in Kelvin [K].

4.13 mo_snow Module Reference

Module contains all things directly related to snow.

Functions/Subroutines

- subroutine snow_coupling (H_abs_snow, phi_s, T_snow, H_abs, H, phi, T, m_snow, S_abs_snow, m, S_bu)

  Subroutine to couple a thin snow layer to the upper ice layer.

- subroutine snow_precip (m_snow, H_abs_snow, thick_snow, psi_s_snow, dt, liquid_precip_in, T2m, solid_precip_in)

  Subroutine for calculating precipitation on an existing snow cover.

- subroutine snow_precip_0 (H_abs, S_abs, m, T, dt, liquid_precip_in, T2m, solid_precip_in)

  Subroutine for calculating precipitation into the ocean.

- subroutine snow_thermo (psi_l_snow, psi_s_snow, psi_g_snow, thick_snow, S_abs_snow, H_abs_-snow, m_snow, T_snow, m, thick, H_abs)

  Subroutine for calculating snow thermodynamics.

- subroutine sub_fl_Q_0_snow_thin (m_snow, thick_snow, T_snow, psi_s, psi_l, psi_g, thick, T_-bound, fl_Q_snow)

  Determines conductive Heat flux for combined top ice and snow layer.

- subroutine sub_fl_Q_snow (m_snow, thick_snow, T_snow, psi_s_2, psi_l_2, psi_g_2, thick_2, T_2, fl_Q)

  Determines conductive Heat flux between Snow and top ice layer.

- subroutine sub_fl_Q_0_snow (m_snow, thick_snow, T_snow, T_bound, fl_Q)

  Determines conductive Heat between snow layer and upper boundary layer. A limiting factor is added to increase stability of layers thinner then thick_min.

- REAL(wp) func_k_snow (m_snow, thick_snow)

  Calculates the thermal conductivity of the snow layer as a function of the density.

4.13.1 Detailed Description

Module contains all things directly related to snow.

Revision History

Provided for by Philipp Griewank 2010-12-13

4.13.2 Function/Subroutine Documentation

4.13.2.1 REAL(wp) mo_snow::func_k_snow (REAL(wp),intent(in) *m_snow,* REAL(wp),intent(in) *thick_snow*)

Calculates the thermal conductivity of the snow layer as a function of the density.

Based on the Sturm et al 1997 data fit for densities greater then 0.156 g/cm∗∗3. Warning, Sturm et al use g/cm∗∗3, I use kg/m∗∗3 Snow density probability functions can be included lated to raise the effective conductivity. Warning!: added 0.15 to the thermal conductivity.

Revision History

Forged by Philipp Griewank (2010-12-13)

4.13.2.2 subroutine mo_snow::snow_coupling (REAL(wp),intent(inout) *H_abs_snow,* REAL(wp),intent(inout) *phi_s,* REAL(wp),intent(inout) *T_snow,* REAL(wp),intent(inout) *H_abs,* REAL(wp),intent(inout) *H,* REAL(wp),intent(inout) *phi,* REAL(wp),intent(inout) *T,* REAL(wp),intent(in) *m_snow,* REAL(wp),intent(in) *S_abs_snow,* REAL(wp),intent(in) *m,* REAL(wp),intent(in) *S_bu*)

Subroutine to couple a thin snow layer to the upper ice layer.

Subroutine is activated when thick_snow<thick_min. The enthalpies of the two layers are adjusted until both layers have the same temperatures. The following approach is used. 1. The enthalpies are adjusted so T_snow=0, and phi_s=1. 2. The temperatures are calculated. 3. If the ice temperature is greater 0 the balanced enthalpies are calculated directly. ELSE they are calculated iteratively.

Revision History

Written by Philipp Griewank, IMPRS (2011-01-20)

4.13.2.3 subroutine mo_snow::snow_precip (REAL(wp),intent(inout) *m_snow,* REAL(wp),intent(inout) *H_abs_snow,* REAL(wp),intent(inout) *thick_snow,* REAL(wp),intent(inout) *psi_s_snow,* REAL(wp),intent(in) *dt,* REAL(wp),intent(in) *liquid_precip_in,* REAL(wp),intent(in) *T2m,* REAL(wp),intent(in),optional *solid_precip_in*)

Subroutine for calculating precipitation on an existing snow cover.

Can optionally deal with separate solid and liquid precipitation or a single liquid input. The 2 meter temperature determines the temperature of the precipitation. In case of single input the 2 meter temperature determines if snow or rain falls. Snow makes the thickness grow according to the density of new snow(rho_-snow), while rain falls into the snow without increasing snow depth. It is necessary to calculate the new psi_s_snow to ensure proper melting in snow_thermo.

Revision History

Sired by Philipp Griewank, IMPRS (2010-12-14)

4.13.2.4 subroutine mo_snow::snow_precip_0 (REAL(wp),intent(inout) *H_abs,* REAL(wp),intent(inout) *S_abs,* REAL(wp),intent(in) *m,* REAL(wp),intent(in) *T,* REAL(wp),intent(in) *dt,* REAL(wp),intent(in) *liquid_precip_in,* REAL(wp),intent(in) *T2m,* REAL(wp),intent(in),optional *solid_precip_in*)

Subroutine for calculating precipitation into the ocean.

Can optionally deal with separate solid and liquid precipitation or a single liquid input. The 2 meter temperature determines the temperature of the precipitation. In case of single input the 2 meter temperature determines if snow or rain falls. It is important, that the mass, energy and salt leaving the upper layer must be outputted. This is not the case. Temp!

Revision History

Copy and Pasted by Philipp Griewank, IMPRS (2011-01-10)

4.13.2.5 subroutine mo_snow::snow_thermo (REAL(wp),intent(inout) *psi_l_snow,* REAL(wp),intent(inout) *psi_s_snow,* REAL(wp),intent(inout) *psi_g_snow,* REAL(wp),intent(inout) *thick_snow,* REAL(wp),intent(inout) *S_abs_snow,* REAL(wp),intent(inout) *H_abs_snow,* REAL(wp),intent(inout) *m_snow,* REAL(wp),intent(inout) *T_snow,* REAL(wp),intent(inout) *m,* REAL(wp),intent(inout) *thick,* REAL(wp),intent(inout) *H_abs*)

Subroutine for calculating snow thermodynamics.

Behaves similar to mushy layer sea ice. Important differences are: 1. no expulsion, thick_snow is raised if the volume expands. 2. The liquid fraction is limited. 3. When the liquid fraction exceeds it's limit the thickness of the snow layer is reduced. This is done as follows: Only applies if the fluid fraction is above the irreducible water content as defined in Coleuo-Lasaffre 98. thick_snow=thick_snow∗(1._wp-(psi_s_-old-psi_s_snow)/psi_s_old) Warning: the formula for liquid water content in Coleuo-Lasaffre contains 2 typos When the water exceeds the limit water runs down to the bottom of the snow layer. The saturated lower layer is added to the top ice layer.

Revision History

Fabricated by Philipp Griewank, IMPRS (2010-12-14) Major redo, water saturated bottom snow added to top ice layer by Philipp Griewank (2010-12-14)

Parameters

H_abs Top ice layer variables

4.13.2.6 subroutine mo_snow::sub_fl_Q_0_snow (REAL(wp),intent(in) *m_snow,* REAL(wp),intent(in) *thick_snow,* REAL(wp),intent(in) *T_snow,* REAL(wp),intent(in) *T_bound,* REAL(wp),intent(out) *fl_Q*)

Determines conductive Heat between snow layer and upper boundary layer. A limiting factor is added to increase stability of layers thinner then thick_min.

Revision History

first version by Philipp Griewank (2010-12-15) Artificial limitation introduced by Philipp Griewank (2011-01-17)

Parameters

T_bound T_bound temperature of boundary layer

4.13.2.7 subroutine mo_snow::sub_fl_Q_0_snow_thin (REAL(wp),intent(in) *m_snow,* REAL(wp),intent(in) *thick_snow,* REAL(wp),intent(in) *T_snow,* REAL(wp),intent(in) *psi_s,* REAL(wp),intent(in) *psi_l,* REAL(wp),intent(in) *psi_g,* REAL(wp),intent(in) *thick,* REAL(wp),intent(in) *T_bound,* REAL(wp),intent(out) *fl_Q_snow*)

Determines conductive Heat flux for combined top ice and snow layer.

When thick_snow<thick_min.

Revision History

first version by Philipp Griewank (2011-01-19)

4.13.2.8 subroutine mo_snow::sub_fl_Q_snow (REAL(wp),intent(in) *m_snow*, REAL(wp),intent(in) *thick_snow*, REAL(wp),intent(in) *T_snow*, REAL(wp),intent(in) *psi_s_2*, REAL(wp),intent(in) *psi_l_2*, REAL(wp),intent(in) *psi_g_2*, REAL(wp),intent(in) *thick_2*, REAL(wp),intent(in) *T_2*, REAL(wp),intent(out) *fl_Q*)

Determines conductive Heat flux between Snow and top ice layer.

Standard approach.

Revision History

first version by Philipp Griewank (2010-12-15)

4.14 mo_testcase_specifics Module Reference

Module contains changes specific testcases require during the main timeloop.

Functions/Subroutines

- subroutine sub_test1 (time, T_top)

  Subroutine for changing T_top for testcase 1.

- subroutine sub_test2 (time, T2m)

  Subroutine for changing T_top for testcase 2.

- subroutine sub_test3 (time, liquid_precip, solid_precip)

  Subroutine for setting snow for testcase 3.

- subroutine sub_test4 (time, fl_q_bottom)

  Subroutine for setting snow for testcase 4.

- subroutine sub_test6 (time, T2m)

  Subroutine for changing T_top for testcase 6 which seeks to reproduce lab measurements of Roni Glud.

4.14.1 Detailed Description

Module contains changes specific testcases require during the main timeloop. Most settings related to the testcases are defined in mo_init, but if changes to the code need to applied after the timestepping has begun they are located here. Changes were initially simply implemented in the main timeloop, but things got confusing.

Author

<Philipp Griewank, IMPRS>

Revision History

Removed from mo_grotz by Philipp Griewank, IMPRS (2014-04-16)

4.14.2 Function/Subroutine Documentation

4.14.2.1 subroutine mo_testcase_specifics::sub_test1 (REAL(wp),intent(in) *time*, REAL(wp),intent(inout) *T_top*)

Subroutine for changing T_top for testcase 1.

Revision History

Formed by Philipp Griewank, IMPRS (2014-04-16)

4.14.2.2 subroutine mo_testcase_specifics::sub_test2 (REAL(wp),intent(in) *time*, REAL(wp),intent(inout) *T2m*)

Subroutine for changing T_top for testcase 2.

T2m is adjusted over time.

Revision History

Formed by Philipp Griewank, IMPRS (2014-04-17)

4.14.2.3 subroutine mo_testcase_specifics::sub_test3 (REAL(wp),intent(in) *time*, REAL(wp),intent(inout) *liquid_precip*, REAL(wp),intent(inout) *solid_precip*)

Subroutine for setting snow for testcase 3.

Precipitation rates are set

Revision History

Formed by Philipp Griewank, (2014-04-18)

4.14.2.4 subroutine mo_testcase_specifics::sub_test4 (REAL(wp),intent(in) *time*, REAL(wp),intent(inout) *fl_q_bottom*)

Subroutine for setting snow for testcase 4.

Revision History

Formed by Philipp Griewank, (2014-04-18)

4.14.2.5 subroutine mo_testcase_specifics::sub_test6 (REAL(wp),intent(in) *time*, REAL(wp),intent(inout) *T2m*)

Subroutine for changing T_top for testcase 6 which seeks to reproduce lab measurements of Roni Glud.

Revision History

Formed by Philipp Griewank, IMPRS (2014-04-38)

Chapter 5

File Documentation

5.1 minpack.f90 File Reference

Functions/Subroutines

- subroutine chkder (m, n, x, fvec, fjac, ldfjac, xp, fvecp, mode, err)
- subroutine dogleg (n, r, lr, diag, qtb, delta, x)
- real(kind=8) enorm (n, x)
- real(kind=8) enorm2 (n, x)
- subroutine fdjac1 (fcn, n, x, fvec, fjac, ldfjac, iflag, ml, mu, epsfcn)
- subroutine fdjac2 (fcn, m, n, x, fvec, fjac, ldfjac, iflag, epsfcn)
- subroutine hybrd (fcn, n, x, fvec, xtol, maxfev, ml, mu, epsfcn, diag, mode, factor, nprint, info, nfev, fjac, ldfjac, r, lr, qtf)
- subroutine hybrd1 (fcn, n, x, fvec, tol, info)
- subroutine hybrj (fcn, n, x, fvec, fjac, ldfjac, xtol, maxfev, diag, mode, factor, nprint, info, nfev, njev, r, lr, qtf)
- subroutine hybrj1 (fcn, n, x, fvec, fjac, ldfjac, tol, info)
- subroutine lmder (fcn, m, n, x, fvec, fjac, ldfjac, ftol, xtol, gtol, maxfev, diag, mode, factor, nprint, info, nfev, njev, ipvt, qtf)
- subroutine lmder1 (fcn, m, n, x, fvec, fjac, ldfjac, tol, info)
- subroutine lmdif (fcn, m, n, x, fvec, ftol, xtol, gtol, maxfev, epsfcn, diag, mode, factor, nprint, info, nfev, fjac, ldfjac, ipvt, qtf)
- subroutine lmdif1 (fcn, m, n, x, fvec, tol, info)
- subroutine lmpar (n, r, ldr, ipvt, diag, qtb, delta, par, x, sdiag)
- subroutine lmstr (fcn, m, n, x, fvec, fjac, ldfjac, ftol, xtol, gtol, maxfev, diag, mode, factor, nprint, info, nfev, njev, ipvt, qtf)
- subroutine lmstr1 (fcn, m, n, x, fvec, fjac, ldfjac, tol, info)
- subroutine qform (m, n, q, ldq)
- subroutine qrfac (m, n, a, lda, pivot, ipvt, lipvt, rdiag, acnorm)
- subroutine qrsolv (n, r, ldr, ipvt, diag, qtb, x, sdiag)
- subroutine r1mpyq (m, n, a, lda, v, w)
- subroutine r1updt (m, n, s, ls, u, v, w, sing)
- subroutine r8vec_print (n, a, title)
- subroutine rwupdt (n, r, ldr, w, b, alpha, c, s)
- subroutine timestamp ()

5.1.1 Function Documentation

5.1.1.1 subroutine chkder (integer (kind = 4) *m*, integer (kind = 4) *n*, real (kind = 8),dimension(n) *x*, real (kind = 8),dimension(m) *fvec*, real (kind = 8),dimension(ldfjac,n) *fjac*, integer (kind = 4) *ldfjac*, real (kind = 8),dimension(n) *xp*, real (kind = 8),dimension(m) *fvecp*, integer (kind = 4) *mode*, real (kind = 8),dimension(m) *err*)

5.1.1.2 subroutine dogleg (integer (kind = 4) *n*, real (kind = 8),dimension(lr) *r*, integer (kind = 4) *lr*, real (kind = 8),dimension(n) *diag*, real (kind = 8),dimension(n) *qtb*, real (kind = 8) *delta*, real (kind = 8),dimension(n) *x*)

5.1.1.3 real (kind = 8) enorm (integer (kind = 4) *n*, real (kind = 8),dimension(n) *x*)

5.1.1.4 real (kind = 8) enorm2 (integer (kind = 4) *n*, real (kind = 8),dimension(n) *x*)

5.1.1.5 subroutine fdjac1 (fcn , integer (kind = 4) *n*, real (kind = 8),dimension(n) *x*, real (kind = 8),dimension(n) *fvec*, real (kind = 8),dimension(ldfjac,n) *fjac*, integer (kind = 4) *ldfjac*, integer (kind = 4) *iflag*, integer (kind = 4) *ml*, integer (kind = 4) *mu*, real (kind = 8) *epsfcn*)

5.1.1.6 subroutine fdjac2 (fcn , integer (kind = 4) *m*, integer (kind = 4) *n*, real (kind = 8),dimension(n) *x*, real (kind = 8),dimension(m) *fvec*, real (kind = 8),dimension(ldfjac,n) *fjac*, integer (kind = 4) *ldfjac*, integer (kind = 4) *iflag*, real (kind = 8) *epsfcn*)

5.1.1.7 subroutine hybrd (fcn , integer (kind = 4) *n*, real (kind = 8),dimension(n) *x*, real (kind = 8),dimension(n) *fvec*, real (kind = 8) *xtol*, integer (kind = 4) *maxfev*, integer (kind = 4) *ml*, integer (kind = 4) *mu*, real (kind = 8) *epsfcn*, real (kind = 8),dimension(n) *diag*, integer (kind = 4) *mode*, real (kind = 8) *factor*, integer (kind = 4) *nprint*, integer (kind = 4) *info*, integer (kind = 4) *nfev*, real (kind = 8),dimension(ldfjac,n) *fjac*, integer (kind = 4) *ldfjac*, real (kind = 8),dimension(lr) *r*, integer (kind = 4) *lr*, real (kind = 8),dimension(n) *qtf*)

5.1.1.8 subroutine hybrd1 (fcn , integer (kind = 4) *n*, real (kind = 8),dimension(n) *x*, real (kind = 8),dimension(n) *fvec*, real (kind = 8) *tol*, integer (kind = 4) *info*)

5.1.1.9 subroutine hybrj (fcn , integer (kind = 4) *n*, real (kind = 8),dimension(n) *x*, real (kind = 8),dimension(n) *fvec*, real (kind = 8),dimension(ldfjac,n) *fjac*, integer (kind = 4) *ldfjac*, real (kind = 8) *xtol*, integer (kind = 4) *maxfev*, real (kind = 8),dimension(n) *diag*, integer (kind = 4) *mode*, real (kind = 8) *factor*, integer (kind = 4) *nprint*, integer (kind = 4) *info*, integer (kind = 4) *nfev*, integer (kind = 4) *njev*, real (kind = 8),dimension(lr) *r*, integer (kind = 4) *lr*, real (kind = 8),dimension(n) *qtf*)

5.1.1.10 subroutine hybrj1 (fcn , integer (kind = 4) *n*, real (kind = 8),dimension(n) *x*, real (kind = 8),dimension(n) *fvec*, real (kind = 8),dimension(ldfjac,n) *fjac*, integer (kind = 4) *ldfjac*, real (kind = 8) *tol*, integer (kind = 4) *info*)

5.1.1.11 subroutine lmder (fcn , integer (kind = 4) *m*, integer (kind = 4) *n*, real (kind = 8),dimension(n) *x*, real (kind = 8),dimension(m) *fvec*, real (kind = 8),dimension(ldfjac,n) *fjac*, integer (kind = 4) *ldfjac*, real (kind = 8) *ftol*, real (kind = 8) *xtol*, real (kind = 8) *gtol*, integer (kind = 4) *maxfev*, real (kind = 8),dimension(n) *diag*, integer (kind = 4) *mode*, real (kind = 8) *factor*, integer (kind = 4) *nprint*, integer (kind = 4) *info*, integer (kind = 4) *nfev*, integer (kind = 4) *njev*, integer (kind = 4),dimension(n) *ipvt*, real (kind = 8),dimension(n) *qtf*)

5.1.1.12 subroutine lmder1 (fcn , integer (kind = 4) *m*, integer (kind = 4) *n*, real (kind = 8),dimension(n) *x*, real (kind = 8),dimension(m) *fvec*, real (kind = 8),dimension(ldfjac,n) *fjac*, integer (kind = 4) *ldfjac*, real (kind = 8) *tol*, integer (kind = 4) *info*)

5.1.1.13 subroutine lmdif (fcn , integer (kind = 4) *m*, integer (kind = 4) *n*, real (kind = 8),dimension(n) *x*, real (kind = 8),dimension(m) *fvec*, real (kind = 8) *ftol*, real (kind =

Sets data and contains all flag descriptions.

Variables

- REAL(wp), dimension(:), allocatable mo_data::H
  Enthalpy [J].
- REAL(wp), dimension(:), allocatable mo_data::H_abs
  specific Enthalpy [J/kg]
- REAL(wp), dimension(:), allocatable mo_data::Q
  Heat in layer [J].
- REAL(wp), dimension(:), allocatable mo_data::fl_Q
  Heat flux between layers [J/s].
- REAL(wp), dimension(:), allocatable mo_data::T
  Temperature [C].
- REAL(wp), dimension(:), allocatable mo_data::S_bu
  Bulk Salinity [g/kg].
- REAL(wp), dimension(:), allocatable mo_data::fl_S
  Salinity flux [(g/s].
- REAL(wp), dimension(:), allocatable mo_data::S_abs
  Absolute Salinity [g].
- REAL(wp), dimension(:), allocatable mo_data::S_br
  Brine salinity [g/kg].
- REAL(wp), dimension(:), allocatable mo_data::thick
  Layer thickness [m].
- REAL(wp), dimension(:), allocatable mo_data::m
  Mass [kg].
- REAL(wp), dimension(:), allocatable mo_data::fl_m
  Mass fluxes between layers [kg].
- REAL(wp), dimension(:), allocatable mo_data::V_s
  Volume [m^3] of solid.
- REAL(wp), dimension(:), allocatable mo_data::V_l
  Volume [m^3] of liquid.
- REAL(wp), dimension(:), allocatable mo_data::V_g
  Volume [m^3] of gas.

- REAL(wp), dimension(:), allocatable mo_data::V_ex
  Volume of brine due expelled due to freezing [m^3] of solid, gas & liquid.
- REAL(wp), dimension(:), allocatable mo_data::phi
  Solid mass fraction.
- REAL(wp), dimension(:), allocatable mo_data::psi_s
  Solid volume fraction.
- REAL(wp), dimension(:), allocatable mo_data::psi_l
  Liquid volume fraction.
- REAL(wp), dimension(:), allocatable mo_data::psi_g
  Gas volume fraction.
- REAL(wp), dimension(:), allocatable mo_data::ray
  Rayleigh number of each layer.
- REAL(wp), dimension(:), allocatable mo_data::perm
  Permeability [?].
- REAL(wp) mo_data::dt
  Timestep [s].
- REAL(wp) mo_data::thick_0
  Initial layer thickness [m].
- REAL(wp) mo_data::time
  Time [s].
- REAL(wp) mo_data::freeboard
  Height of ice surface above (or below) waterlevel [m].
- REAL(wp) mo_data::T_freeze
  Freezing temperature [C].
- INTEGER mo_data::Nlayer
  Number of layers.
- INTEGER mo_data::N_bottom
  Number of bottom layers.
- INTEGER mo_data::N_middle
  Number of middle layers.
- INTEGER mo_data::N_top
  Number of top layers.
- INTEGER mo_data::N_active
  Number of Layers active in the present.

- INTEGER mo_data::i

    Index, normally used for time.

- INTEGER mo_data::k

    Index, normally used for layer.

- REAL(wp) mo_data::time_out

    Time between outputs [s].

- REAL(wp) mo_data::time_total

    Time of simulation [s].

- INTEGER mo_data::i_time

    Number of timesteps.

- INTEGER mo_data::i_time_out

    Number of timesteps between each output.

- INTEGER mo_data::n_time_out

    Counts number of timesteps between output.

- CHARACTER ∗12000 mo_data::format_T
- CHARACTER ∗12000 mo_data::format_psi
- CHARACTER ∗12000 mo_data::format_thick
- CHARACTER ∗12000 mo_data::format_snow
- CHARACTER ∗12000 mo_data::format_integer
- CHARACTER ∗12000 mo_data::format_T2m_top
- CHARACTER ∗12000 mo_data::format_bgc

    Format strings for output.

- REAL(wp) mo_data::T_bottom

    Temperature of water beneath the ice [C].

- REAL(wp) mo_data::T_top

    Temperature at the surface [C].

- REAL(wp) mo_data::S_bu_bottom

    Salinity beneath the ice [g/kg].

- REAL(wp) mo_data::T2m

    Two meter Temperature [C].

- REAL(wp) mo_data::fl_q_bottom

    Bottom heat flux [J∗s].

- REAL(wp) mo_data::psi_s_snow

    Solid volume fraction of snow layer.

- REAL(wp) mo_data::psi_l_snow

Liquid volume fraction of snow layer.

- REAL(wp) mo_data::psi_g_snow

  Gas volume fraction of snow layer.

- REAL(wp) mo_data::phi_s

  Solid mass fraction of snow layer.

- REAL(wp) mo_data::S_abs_snow

  Absolute salinity of snow layer [g].

- REAL(wp) mo_data::H_abs_snow

  Absolute enthalpy of snow layer [J].

- REAL(wp) mo_data::m_snow

  Mass of snow layer [kg].

- REAL(wp) mo_data::T_snow

  Temperature of snow layer [C].

- REAL(wp) mo_data::thick_snow

  Thickness of snow layer [m].

- REAL(wp) mo_data::liquid_precip

  Liquid precip, [meter of water/s].

- REAL(wp) mo_data::solid_precip

  Solid precip, [meter of water /s].

- REAL(wp) mo_data::fl_q_snow

  flow of heat into the snow layer

- REAL(wp) mo_data::energy_stored

  Total amount of energy stored, control is freezing point temperature of S_bu_bottom [J].

- REAL(wp) mo_data::total_resist

  Thermal resistance of the whole column [].

- REAL(wp) mo_data::surface_water

  Percentage of water fraction in the top 5cm [%].

- REAL(wp) mo_data::freshwater

  Meters of freshwater stored in column [m].

- REAL(wp) mo_data::thickness

  Meters of ice [m].

- REAL(wp) mo_data::bulk_salin

  Salt/Mass [ppt].

- REAL(wp) mo_data::thick_min

  Parameter for snow, determines when snow is in thermal equilibrium with the ice and when it is totally neglected.

- REAL(wp), save mo_data::T_test

  First guess for getT subroutine.

- REAL(wp) mo_data::albedo

  Amount of short wave radiation which is reflected at the top surface.

- REAL(wp) mo_data::fl_sw

  *Incoming shortwave radiation [W/m**2].*

- REAL(wp) mo_data::fl_lw

  *Incoming longwave radiation [W/m**2].*

- REAL(wp) mo_data::fl_sen

  *Sensitive heat flux [W/m**2].*

- REAL(wp) mo_data::fl_lat

  *Latent heat flux [W/m**2].*

- REAL(wp) mo_data::fl_rest

  *Bundled longwave,sensitive and latent heat flux [W/m**2].*

- REAL(wp), dimension(:), allocatable mo_data::fl_rad

  Energy flux of absorbed sw radiation of each layer [J/s].

- REAL(wp) mo_data::grav_drain

  brine flux of gravity drainage between two outputs [kg/s]

- REAL(wp) mo_data::grav_salt

  *salt flux moved by gravity drainage between two outputs [kg*ppt/s]*

- REAL(wp) mo_data::grav_temp

  average temperature of gravity drainage brine between two outputs [T]

- REAL(wp) mo_data::melt_thick

  thickness of fully liquid part of top layer [m]

- REAL(wp) mo_data::alpha_flux_instable

  Proportionality constant which determines energy flux by the temperature difference T_top>T2m [W/C].

- REAL(wp) mo_data::alpha_flux_stable

  Proportionality constant which determines energy flux by the temperature difference T_top<T2m [W/C].

- INTEGER mo_data::atmoflux_flag

  1: Use mean climatology of Notz, 2: Use imported reanalysis data, 3: use fixed values defined in mo_init

- INTEGER mo_data::grav_flag

1: no gravity drainage, 2: Gravity drainage, 3: Simple Drainage

- INTEGER mo_data::prescribe_flag

  1: nothing happens, 2: prescribed Salinity profile is prescribed at each timestep (does not disable brine dynamics, just overwrites the salinity!)

- INTEGER mo_data::grav_heat_flag

  1: nothing happens, 2: compensates heatfluxes in grav_flag = 2

- INTEGER mo_data::flush_heat_flag

  1: nothing happens, 2: compensates heatfluxes in flush_flag = 5

- INTEGER mo_data::turb_flag

  1: No bottom turbulence, 2: Bottom mixing

- INTEGER mo_data::salt_flag

  1: Sea salt, 2: NaCL

- INTEGER mo_data::boundflux_flag

  1: top and bottom cooling plate, 2:top Notz fluxes, bottom cooling plate 3: top flux=a∗(T-T_s)

- INTEGER mo_data::flush_flag

  1: no flushing, 4:meltwater is removed artificially, 5:vert and horiz flushing, 6: simplified

- INTEGER mo_data::flood_flag

  1: no flooding, 2:normal flooding, 3:simple flooding

- INTEGER mo_data::bottom_flag

  1: nothing changes, 2: deactivates all bottom layer dynamics, useful for some debugging and idealized tests

- INTEGER mo_data::debug_flag

  1: no raw layer output, 2: each layer is output at every timestep (warning, file size can be very large)

- INTEGER mo_data::precip_flag

  0: solid and liquid precipitation, 1:phase determined by T2m

- INTEGER mo_data::harmonic_flag

  1: minimal permeability is used to calculate Rayleigh number, 2:harmonic mean is used for Rayleigh number

- INTEGER mo_data::tank_flag

  1: nothing, 2: S_bu_bottom and bgc_bottom are calculated as if the experiment is conducted in a tank

- INTEGER mo_data::albedo_flag

  1: simple albedo, 2: normal albedo, see func_albedo for details

- INTEGER mo_data::Length_Input

  Sets the input length for atmoflux_flag==2, common value of 13169.

- REAL(wp), dimension(8280) mo_data::Tinput *used to read in top temperature for field experiment tests, dimension needs to be set in the code*

- REAL(wp), dimension(:), allocatable mo_data::fl_sw_input

  Used to read in sw fluxes from ERA for atmoflux_flag==2.

- REAL(wp), dimension(:), allocatable mo_data::fl_lw_input

  Used to read in lw fluxes from ERA for atmoflux_flag==2.

- REAL(wp), dimension(:), allocatable mo_data::T2m_input

  Used to read in 2Tm from ERA for atmoflux_flag==2.

- REAL(wp), dimension(:), allocatable mo_data::precip_input

  Used to read in precipitation from ERA for atmoflux_flag==2.

- REAL(wp), dimension(:), allocatable mo_data::time_input

  Used to read in time from ERA for atmoflux_flag==2.

- INTEGER mo_data::time_counter

  Keeps track of input data.

- INTEGER mo_data::bgc_flag

  1: no bgc, 2:bgc

- INTEGER mo_data::N_bgc

  Number of chemicals.

- REAL(wp), dimension(:,:), allocatable mo_data::fl_brine_bgc

  Brine fluxes in a matrix, [kg/s], first index is the layer of origin, and the second index is the layer of arrival.

- REAL(wp), dimension(:,:), allocatable mo_data::bgc_abs

  Absolute amount of chemicals [kmol] for each tracer.

- REAL(wp), dimension(:,:), allocatable mo_data::bgc_bu

  Bulk amounts of chemicals [kmol/kg].

- REAL(wp), dimension(:,:), allocatable mo_data::bgc_br

  Brine concentrations of chems [kmol/kg].

- REAL(wp), dimension(:), allocatable mo_data::bgc_bottom

  Bulk concentrations of chems below the ice [kmol/kg].

- REAL(wp), dimension(:), allocatable mo_data::bgc_total

  Total of chems, for lab experiments with a fixed total amount.

- REAL(wp) mo_data::m_total

  Total initial water mass, for lab experiments with a fixed total amount.

- REAL(wp) mo_data::S_total

  Total initial salt mass, for lab experiments with a fixed total amount.

- REAL(wp) mo_data::tank_depth

  water depth in meters, used to calculate concentrations below ice for tank experiments

5.3 mo_flood.f90 File Reference

Modules

- module mo_flood

  Computes the fluxes caused by liquid flooding the snow layer.

Functions/Subroutines

- subroutine mo_flood::flood (freeboard, psi_s, psi_l, S_abs, H_abs, m, T, thick, dt, Nlayer, N_active, T_bottom, S_bu_bottom, H_abs_snow, m_snow, thick_snow, psi_g_snow, debug_flag, fl_brine_-bgc)

  Subroutine for calculating flooding.

- subroutine mo_flood::flood_simple (freeboard, S_abs, H_abs, m, thick, T_bottom, S_bu_bottom, H_abs_snow, m_snow, thick_snow, psi_g_snow, Nlayer, N_active, debug_flag)

  Subroutine for calculating flooding.

5.4 mo_flush.f90 File Reference

Modules

- module mo_flush

  Contains various subroutines for flushing.

Functions/Subroutines

- subroutine mo_flush::flush3 (freeboard, psi_l, thick, thick_0, S_abs, H_abs, m, T, dt, Nlayer, N_-active, T_bottom, S_bu_bottom, melt_thick, debug_flag, flush_heat_flag, fl_brine_bgc)

  Subroutine for complex flushing.

- subroutine mo_flush::flush4 (psi_l, thick, T, thick_0, S_abs, H_abs, m, dt, Nlayer, N_active, N_top, N_middle, N_bottom, melt_thick, debug_flag)

  An alternative subroutine for calculating flushing.

5.5 mo_functions.f90 File Reference

Modules

- module mo_functions
  Module houses functions which have no home :(.

Functions/Subroutines

- REAL(wp) mo_functions::func_density (T, S)
  Calculates the physical density for given S and T.
- REAL(wp) mo_functions::func_freeboard (N_active, Nlayer, psi_s, psi_g, m, thick, m_snow)
  Calculates the freeboard of the 1d ice column.
- REAL(wp) mo_functions::func_albedo (thick_snow, T_snow, psi_l, thick_min, albedo_flag)
  Calculates the albedo.
- REAL(wp) mo_functions::func_sat_O2 (T, S_bu)
  Calculates the oxygen saturation as a function of salinity and temperature.
- REAL(wp) mo_functions::func_T_freeze (S_bu, salt_flag)
  Calculates the freezing temperature. Salt_flag determines if either ocean salt or NAcl is used.
- subroutine mo_functions::sub_notzflux (time, fl_sw, fl_rest)
  Calculates the incoming shortwave and other fluxes according to p. 193-194 PhD Notz.
- subroutine mo_functions::sub_input (length_input, fl_sw_input, fl_lw_input, T2m_input, precip_-input, time_input)
  Reads in data for atmoflux_flag ==2.
- subroutine mo_functions::sub_turb_flux (T_bottom, S_bu_bottom, T, S_abs, m, dt, N_bgc, bgc_-bottom, bgc_abs)
  Calculates salt and tracer mixing between lowest layer and underlying water.
- subroutine mo_functions::sub_melt_thick (psi_l, psi_s, psi_g, T, T_freeze, T_top, fl_Q, thick_snow, dt, melt_thick, thick, thick_min)
  Calculates the thickness of the meltwater film.
- subroutine mo_functions::sub_melt_snow (melt_thick, thick, thick_snow, H_abs, H_abs_snow, m, m_snow, psi_g_snow)
  Calculates how the meltwater film interacts with snow.

5.6 mo_grav_drain.f90 File Reference

Modules

- module mo_grav_drain

Computes the Salt fluxes caused by gravity drainage.

Functions/Subroutines

- subroutine mo_grav_drain::fl_grav_drain (S_br, S_bu, psi_l, psi_s, psi_g, thick, S_abs, H_abs, T, m, dt, Nlayer, N_active, ray, T_bottom, S_bu_bottom, grav_drain, grav_temp, grav_salt, grav_heat_-flag, harmonic_flag, fl_brine_bgc)

  Calculates fluxes caused by gravity drainage.

- subroutine mo_grav_drain::fl_grav_drain_simple (psi_s, psi_l, thick, S_abs, S_br, Nlayer, N_active, ray, grav_drain, harmonic_flag)

  Calculates salinity to imitate the effects gravity drainage.

5.7 mo_grotz.f90 File Reference

Modules

- module mo_grotz

  SAMSIM Semi-Adaptive Multi-phase Sea-Ice Model.

Functions/Subroutines

- subroutine mo_grotz::grotz (testcase, description)

  Main subroutine of SAMSIM, a 1D thermodynamic seaice model. A semi-adaptive grid is used which is managed by mo_layer_dynamics. To many things happen in this subroutine to describe in this description, you'll just have to go through it.

5.8 mo_heat_fluxes.f90 File Reference

Modules

- module mo_heat_fluxes

  Computes all heat fluxes.

Functions/Subroutines

- subroutine mo_heat_fluxes::sub_heat_fluxes ()

  Computes surface temperature and heatfluxes.

5.9 mo_init.f90 File Reference

Modules

- module mo_init

  Allocates Arrays and sets initial data for a given testcase for SAMSIM.

Functions/Subroutines

- subroutine mo_init::init (testcase)

  Sets initial conditions according to which testcase is chosen.

- subroutine mo_init::sub_allocate (Nlayer)

  Allocates Arrays.

- subroutine mo_init::sub_allocate_bgc (Nlayer, N_bgc)

  Allocates BGC Arrays.

- subroutine mo_init::sub_deallocate

  Deallocates Arrays.

5.10 mo_layer_dynamics.f90 File Reference

Modules

- module mo_layer_dynamics

  Mo_layer_dynamics contains all subroutines for the growth and shrinking of layer thickness.

Functions/Subroutines

- subroutine mo_layer_dynamics::layer_dynamics (phi, N_active, Nlayer, N_bottom, N_middle, N_top, m, S_abs, H_abs, thick, thick_0, T_bottom, S_bu_bottom, bottom_flag, debug_flag, N_bgc, bgc_abs, bgc_bottom)

  Organizes the Semi-Adaptive grid SAMSIM uses.

- subroutine mo_layer_dynamics::top_melt (Nlayer, N_active, N_bottom, N_middle, N_top, thick_0, m, S_abs, H_abs, thick, N_bgc, bgc_abs)
- subroutine mo_layer_dynamics::top_grow (Nlayer, N_active, N_bottom, N_middle, N_top, thick_0, m, S_abs, H_abs, thick, N_bgc, bgc_abs)

  Top grow subroutine.

5.11 mo_mass.f90 File Reference

Modules

- module mo_mass

  Regulates mass transfers and their results.

Functions/Subroutines

- subroutine mo_mass::mass_transfer (Nlayer, N_active, T, H_abs, S_abs, S_bu, T_bottom, S_bu_bottom, fl_m)

  Calculates the effects of mass transfers on H_abs and S_abs.

- subroutine mo_mass::expulsion_flux (thick, V_ex, Nlayer, N_active, psi_g, fl_m, m)

  Generates the fluxes caused by expulsion.

- subroutine mo_mass::bgc_advection (Nlayer, N_active, N_bgc, fl_brine_bgc, bgc_abs, psi_l, T, S_abs, m, thick, bgc_bottom)

  Calculates how the brine fluxes stored in fl_brine_bgc advect bgc tracers.

5.12 mo_output.f90 File Reference

Modules

- module mo_output

  All things output.

Functions/Subroutines

- subroutine mo_output::output_settings (description, testcase, N_top, N_bottom, Nlayer, fl_q_bottom, T_bottom, S_bu_bottom, thick_0, time_out, time_total, dt, boundflux_flag, atmoflux_flag, albedo_flag, grav_flag, flush_flag, flood_flag, grav_heat_flag, flush_heat_flag, harmonic_flag, prescribe_flag, salt_flag, turb_flag, bottom_flag, tank_flag, precip_flag, bgc_flag, N_bgc)

  Settings output.

- subroutine mo_output::output (Nlayer, T, psi_s, psi_l, thick, S_bu, ray, format_T, format_psi, format_thick, format_snow, freeboard, thick_snow, T_snow, psi_l_snow, psi_s_snow, energy_stored, freshwater, total_resist, thickness, bulk_salin, grav_drain, grav_salt, grav_temp, T2m, T_top)

  Standard output.

- subroutine mo_output::output_bgc (Nlayer, N_active, bgc_bottom, N_bgc, bgc_abs, psi_l, thick, m, format_bgc)

  Standard bgc output.

- subroutine mo_output::output_raw (Nlayer, N_active, time, T, thick, S_bu, psi_s, psi_l, psi_g)

Output for debugging purposes.

- subroutine mo_output::output_raw_snow (time, T_snow, thick_snow, S_abs_snow, m_snow, psi_s_-snow, psi_l_snow, psi_g_snow)

  Output for debugging purposes.

- subroutine mo_output::output_raw_lay (Nlayer, N_active, H_abs, m, S_abs, thick, string)

  Output for debugging layer dynamics..

- subroutine mo_output::output_begin (Nlayer, debug_flag, format_T, format_psi, format_thick, format_snow, format_T2m_top)

  Output files are opened and format strings are created.

- subroutine mo_output::output_begin_bgc (Nlayer, N_bgc, format_bgc)

  Output files for bgc are opened and format strings are created.

5.13 mo_parameters.f90 File Reference

Modules

- module mo_parameters

  Module determines physical constants to be used by the SAMSIM Seaice model.

Variables

- INTEGER, parameter mo_parameters::wp = SELECTED_REAL_KIND(12, 307)

  set working precision _wp

- REAL, parameter mo_parameters::pi = 3.1415_wp
- REAL, parameter mo_parameters::grav = 9.8061_wp

  gravitational constant [m/s^2]

- REAL(wp), parameter mo_parameters::k_s = 2.2_wp

  solid heat conductivity [J / m s K] 2.2

- REAL(wp), parameter mo_parameters::k_l = 0.523_wp

  liquid heat conductivity [J / m s K] 0.523

- REAL(wp), parameter mo_parameters::c_s = 2020.0_wp

  solid heat capacity [J/ kg K]

- REAL(wp), parameter mo_parameters::c_s_beta = 7.6973_wp

  *linear solid heat capacity approximation [J/ kg K^2] c_s = c_s+c_s_beta*T*

- REAL(wp), parameter mo_parameters::c_l = 3400._wp

  liquid heat capacity [J/ kg K]

- REAL(wp), parameter mo_parameters::rho_s = 920._wp

  density of solid [kg / m^3]

- REAL(wp), parameter mo_parameters::rho_l = 1028.0_wp

  density of liquid [kg / m^3]

- REAL(wp), parameter mo_parameters::latent_heat = 333500._wp

  latent heat release [J/kg]

- REAL(wp), parameter mo_parameters::zeroK = 273.15_wp

  Zero degrees Celsius in Kelvin [K].

- REAL(wp), parameter mo_parameters::bbeta = 0.8_wp∗1e-3

  concentration expansion coefficient [kg / (m^3 ppt)]

- REAL(wp), parameter mo_parameters::mu = 2.55_wp∗1e-3

  dynamic viscosity [kg /m s]

- REAL(wp), parameter mo_parameters::kappa_l = k_l/rho_l/c_l

  heat diffusivity of water

- REAL(wp), parameter mo_parameters::sigma = 5.6704_wp∗1e-8

  Stefan Boltzmann constant [W/(m^2∗K^4)].

- REAL(wp), parameter mo_parameters::psi_s_min = 0.05_wp

  The amount of ice that the lowest layer can have before it counts as an ice layer.

- REAL(wp), parameter mo_parameters::neg_free = -0.05_wp

  The distance the freeboard can be below 0 before water starts flooding through cracks.

- REAL(wp), parameter mo_parameters::x_grav = 0.000584_wp
- REAL(wp), parameter mo_parameters::ray_crit = 4.89_wp
- REAL(wp), parameter mo_parameters::para_flush_horiz = 1.00_wp

  determines relationship of horizontal flow distance in during flushing (guess 1)

- REAL(wp), parameter mo_parameters::para_flush_gamma = 0.9_wp

  Strength of desalination per timestep (guess).

- REAL(wp), parameter mo_parameters::psi_s_top_min = 0.40_wp

  if psi_s is below this value meltwater forms (guess) 0.4

- REAL(wp), parameter mo_parameters::ratio_flood = 1.50_wp

  Ratio of flooded to dissolve snow, plays an important role in subroutine flood.

- REAL(wp), parameter mo_parameters::ref_salinity = 34._wp

  Reference salinity [g/kg] used to calculate freshwater column.

- REAL(wp), parameter mo_parameters::rho_snow = 330_wp

  density of new snow [kg/m∗∗3]

- REAL(wp), parameter mo_parameters::gas_snow_ice = 0.10_wp

  volume of gas percentage in new snow ice due to flooding, no longer used

- REAL(wp), parameter mo_parameters::gas_snow_ice2 = 0.20_wp

  volume of gas percentage in new snow ice due to snow melting (Eicken 95)

- REAL(wp), parameter mo_parameters::emissivity_ice = 0.95_wp

  Emissivity of water and ice.

- REAL(wp), parameter mo_parameters::emissivity_snow = 1.00_wp

  Emissivity of Snow.

- REAL(wp), parameter mo_parameters::penetr = 0.30_wp

  Amount of penetrating sw radiation.

- REAL(wp), parameter mo_parameters::extinc = 2.00_wp

  Extinction coefficient of ice.

- REAL(wp), parameter mo_parameters::Turb_A = 0.1_wp*0.05_wp*rho_l/86400._wp

  Standard turbulence [kg/s] WARNING no source, just set so that 5cm of water are overturned each day.

- REAL(wp), parameter mo_parameters::Turb_B = 0.05_wp

  *Exponential turbulence slope [m**3/kg] WARNING no source, simple guess.*

- REAL(wp) mo_parameters::max_flux_plate = 50.0

  Maximal heating rate of a heating plate.

5.14 mo_snow.f90 File Reference

Modules

- module mo_snow

  Module contains all things directly related to snow.

Functions/Subroutines

- subroutine mo_snow::snow_coupling (H_abs_snow, phi_s, T_snow, H_abs, H, phi, T, m_snow, S_-abs_snow, m, S_bu)

  Subroutine to couple a thin snow layer to the upper ice layer.

- subroutine mo_snow::snow_precip (m_snow, H_abs_snow, thick_snow, psi_s_snow, dt, liquid_-precip_in, T2m, solid_precip_in)

  Subroutine for calculating precipitation on an existing snow cover.

- subroutine mo_snow::snow_precip_0 (H_abs, S_abs, m, T, dt, liquid_precip_in, T2m, solid_precip_-in)

Subroutine for calculating precipitation into the ocean.

- subroutine mo_snow::snow_thermo (psi_l_snow, psi_s_snow, psi_g_snow, thick_snow, S_abs_-snow, H_abs_snow, m_snow, T_snow, m, thick, H_abs)

  Subroutine for calculating snow thermodynamics.

- subroutine mo_snow::sub_fl_Q_0_snow_thin (m_snow, thick_snow, T_snow, psi_s, psi_l, psi_g, thick, T_bound, fl_Q_snow)

  Determines conductive Heat flux for combined top ice and snow layer.

- subroutine mo_snow::sub_fl_Q_snow (m_snow, thick_snow, T_snow, psi_s_2, psi_l_2, psi_g_2, thick_2, T_2, fl_Q)

  Determines conductive Heat flux between Snow and top ice layer.

- subroutine mo_snow::sub_fl_Q_0_snow (m_snow, thick_snow, T_snow, T_bound, fl_Q)

  Determines conductive Heat between snow layer and upper boundary layer. A limiting factor is added to increase stability of layers thinner then thick_min.

- REAL(wp) mo_snow::func_k_snow (m_snow, thick_snow)

  Calculates the thermal conductivity of the snow layer as a function of the density.

5.15 mo_testcase_specifics.f90 File Reference

Modules

- module mo_testcase_specifics

  Module contains changes specific testcases require during the main timeloop.

Functions/Subroutines

- subroutine mo_testcase_specifics::sub_test1 (time, T_top)

  Subroutine for changing T_top for testcase 1.

- subroutine mo_testcase_specifics::sub_test2 (time, T2m)

  Subroutine for changing T_top for testcase 2.

- subroutine mo_testcase_specifics::sub_test3 (time, liquid_precip, solid_precip)

  Subroutine for setting snow for testcase 3.

- subroutine mo_testcase_specifics::sub_test4 (time, fl_q_bottom)

  Subroutine for setting snow for testcase 4.

- subroutine mo_testcase_specifics::sub_test6 (time, T2m)

  Subroutine for changing T_top for testcase 6 which seeks to reproduce lab measurements of Roni Glud.

5.16 mo_thermo_functions.f90 File Reference

5.17 SAMSIM.f90 File Reference

Functions/Subroutines

- program SAMSIM

5.17.1 Function Documentation

5.17.1.1 program SAMSIM ( )

Index