SAMSIM

Generated by Doxygen 1.8.13

Contents

Chapter 1

SAMSIM Semi-Adaptive Multi-phase Sea-Ice Model V2.0

V1.0 of 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 the two papers "Insights into brine dynamics and sea ice desalination from a 1-D model study of gravity drainage" and "A 1-D modelling study of Arctic sea-ice salinity" of Griewank and Notz" which are both included in the repository. V2.0 of SAMSIM is a minor expansion of V1.0 released in 2018. Most work was done by Niels Fuchs as part of his Master's thesis "The impact of snow on sea-ice salinity" at the Max Planck Institute of Meteorology from 2016-2017 (thesis also in repository). The biggest change is an improvement of the flushing parametrization, as well as the settings and forcings for a large amount of laboratory experiments Niels conducted, making it possible to run lab testcases with snow.

SAMSIM.f90 is the root program of the SAMSIM, the 1D thermodynamic Semi-Adaptive Multi-phase Sea-Ice Model. 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. The code is intended to be understandable and subroutines, modules, functions, parameters, and global variables all have (more or less) doxygen compatible descriptions. Both a pdf and html documentation generated via doxygen are included under documentation.

WARNING: SAMSIM was developed and was/is used for scientific purposes. It likely contains a few 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bugs, can easily be crashed by using non-logical input settings, and some of the descriptions and comments may be outdated.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 was implemented at the end of my PhD and not used much. As a result it has not been thouroughly tested.
- 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 you are using a mac and run into this error: dyld: Library not loaded: /libgfortran.3.dylib, this solution worked for Max Thomas and can hopefully help others: export DYLD_FALLBACK_LIBRARY_PATH="/↩ Users/max/opt/anaconda3/lib"
- 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 :)

Contacts:

- Philipp Griewank: philipp.griewank@uni-koeln.de
- Niels Fuchs: niels.fuchs@awi.de
- Dirk Notz: dirk.notz@mpimet.mpg.de
- Luise Zeigermann: zeigermannluise@gmail.com, when related to her Master Thesis

Author

Philipp Griewank

COPYRIGHT

Revision History

Started by Philipp Griewank 2014-05-05
nothing changed here by Niels Fuchs, MPIMET (2017-03-01)
License changed by Philipp Griewank 2018-05-22
V2.0 finalized by Philipp Griewank 2018-08-29

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
- real(wp), dimension(:), allocatable flush_v
- real(wp), dimension(:), allocatable flush_h
- real(wp), dimension(:), allocatable flush_v_old
- real(wp), dimension(:), allocatable flush_h_old

    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.
- integer styropor_flag
- 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
- character ∗12000 format_melt

  Format strings for output. Niels(2017) add: melt output.
- character ∗12000 format_perm

  Niels(2017) add: permeability 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
- real(wp) test

  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) melt_thick_snow
- real(wp) melt_thick_snow_old

  Niels(2017) add: thickness of excess fully liquid part from snow_melt_processes [m].
- real(wp), dimension(3) melt_thick_output

  Niels, 2017 add: output field of surface liquid meltwater sizes.
- 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 lab_snow_flag

  Niels, 2017 add: 0: lab setup without snow covers, 1: lab setup include snow influence on heat fluxes.

- integer freeboard_snow_flag

  Niels, 2017 add: 0: respect the mass of snow in the freeboard calculation, 1: don't.

- integer snow_flush_flag

  Niels, 2017 add: 0: all meltwater from snow forms slush, 1: meltwater partly leads to flushing, ratio defined by "k_snow_flush".

- integer snow_precip_flag

  Niels, 2017 add: 0: all precipitation is set to zero, 1: physical behaviour.

- integer length_input

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

- real(wp), dimension(:), allocatable tinput

  Niels, 2017 add: used to read in top temperature for field experiment tests, dimension needs to be set in the code.

- real(wp), dimension(:), allocatable precipinput

  Niels, 2017 add: used to read in precipation for field experiment tests, dimension needs to be set in the code.

- real(wp), dimension(:), allocatable ocean_t_input

  Niels, 2017 add: used to read in ocean temperature for field experiment tests, dimension needs to be set in the code.

- real(wp), dimension(:), allocatable ocean_flux_input

  Niels, 2017 add: used to read in oceanic heat flux for field experiment tests, dimension needs to be set in the code.

- real(wp), dimension(:), allocatable styropor_input

  Niels, 2017 add: if styropor is used in the lab on top of the ice to simulate snow heat fluxes.

- real(wp), dimension(:), allocatable ttop_input

Niels, 2017 add: used for testcase 111, comparison with greenland harp data, uppermost harp temperature is seen as Ttop.

- 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
- character ∗3 flush_question ='No!'

  Niels, 2017 add: used to indicate in stdout wether flushing occurs at this moment or not.
- real(wp) melt_err =0._wp

  Niels, 2017 add: used to check how much meltwater vanishes in flushing routine.
- integer length_input_lab

  Niels, 2017 add: used to allocate lab testcase input arrays in mo_init, set value in testcases.

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)
Add several variables by Niels Fuchs, MPIMET (2017-03-01)

4.1.2 Variable Documentation

4.1.2.1 albedo

`real(wp) mo_data::albedo`

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

4.1.2.2 albedo_flag

`integer mo_data::albedo_flag`

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

4.1.2.3 alpha_flux_instable

`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 alpha_flux_stable

```
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 atmoflux_flag

```
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 bgc_abs

```
real(wp), dimension(:,:), allocatable mo_data::bgc_abs
```

Absolute amount of chemicals [kmol] for each tracer.

4.1.2.7 bgc_bottom

```
real(wp), dimension(:), allocatable mo_data::bgc_bottom
```

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

4.1.2.8 bgc_br

```
real(wp), dimension(:,:), allocatable mo_data::bgc_br
```

Brine concentrations of chems [kmol/kg].

4.1.2.9 bgc_bu

```
real(wp), dimension(:,:), allocatable mo_data::bgc_bu
```

Bulk amounts of chemicals [kmol/kg].

4.1.2.10 bgc_flag

`integer mo_data::bgc_flag`

1: no bgc, 2:bgc

4.1.2.11 bgc_total

`real(wp), dimension(:), allocatable mo_data::bgc_total`

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

4.1.2.12 bottom_flag

`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 boundflux_flag

`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 bulk_salin

`real(wp) mo_data::bulk_salin`

Salt/Mass [ppt].

4.1.2.15 debug_flag

`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 dt

```
real(wp) mo_data::dt
```

Timestep [s].

4.1.2.17 energy_stored

```
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 fl_brine_bgc

```
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 fl_lat

```
real(wp) mo_data::fl_lat
```

Latent heat flux [W/m∗∗2].

4.1.2.20 fl_lw

```
real(wp) mo_data::fl_lw
```

Incoming longwave radiation [W/m∗∗2].

4.1.2.21 fl_lw_input

```
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 fl_m

`real(wp), dimension(:), allocatable mo_data::fl_m`

Mass fluxes between layers [kg].

4.1.2.23 fl_q

`real(wp), dimension(:), allocatable mo_data::fl_q`

Heat flux between layers [J/s].

4.1.2.24 fl_q_bottom

`real(wp) mo_data::fl_q_bottom`

Bottom heat flux [J∗s].

4.1.2.25 fl_q_snow

`real(wp) mo_data::fl_q_snow`

flow of heat into the snow layer

4.1.2.26 fl_rad

`real(wp), dimension(:), allocatable mo_data::fl_rad`

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

4.1.2.27 fl_rest

`real(wp) mo_data::fl_rest`

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

4.1.2.28 fl_s

```
real(wp), dimension(:), allocatable mo_data::fl_s
```

Salinity flux [(g/s].

4.1.2.29 fl_sen

```
real(wp) mo_data::fl_sen
```

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

4.1.2.30 fl_sw

```
real(wp) mo_data::fl_sw
```

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

4.1.2.31 fl_sw_input

```
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 flood_flag

```
integer mo_data::flood_flag
```

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

4.1.2.33 flush_flag

```
integer mo_data::flush_flag
```

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

4.1.2.34 flush_h

```
real(wp), dimension(:), allocatable mo_data::flush_h
```

4.1.2.35 flush_h_old

```
real(wp), dimension(:), allocatable mo_data::flush_h_old
```

Permeability [?].

4.1.2.36 flush_heat_flag

```
integer mo_data::flush_heat_flag
```

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

4.1.2.37 flush_question

```
character*3 mo_data::flush_question ='No!'
```

Niels, 2017 add: used to indicate in stdout wether flushing occurs at this moment or not.

4.1.2.38 flush_v

```
real(wp), dimension(:), allocatable mo_data::flush_v
```

4.1.2.39 flush_v_old

```
real(wp), dimension(:), allocatable mo_data::flush_v_old
```

4.1.2.40 format_bgc

```
character*12000 mo_data::format_bgc
```

4.1.2.41 format_integer

```
character*12000 mo_data::format_integer
```

4.1.2.42 format_melt

```
character*12000 mo_data::format_melt
```

Format strings for output. Niels(2017) add: melt output.

4.1.2.43 format_perm

```
character*12000 mo_data::format_perm
```

Niels(2017) add: permeability output.

4.1.2.44 format_psi

```
character*12000 mo_data::format_psi
```

4.1.2.45 format_snow

```
character*12000 mo_data::format_snow
```

4.1.2.46 format_t

```
character*12000 mo_data::format_t
```

4.1.2.47 format_t2m_top

```
character*12000 mo_data::format_t2m_top
```

4.1.2.48 format_thick

`character*12000 mo_data::format_thick`

4.1.2.49 freeboard

`real(wp) mo_data::freeboard`

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

4.1.2.50 freeboard_snow_flag

`integer mo_data::freeboard_snow_flag`

Niels, 2017 add: 0: respect the mass of snow in the freeboard calculation, 1: don't.

4.1.2.51 freshwater

`real(wp) mo_data::freshwater`

Meters of freshwater stored in column [m].

4.1.2.52 grav_drain

`real(wp) mo_data::grav_drain`

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

4.1.2.53 grav_flag

`integer mo_data::grav_flag`

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

4.1.2.54 grav_heat_flag

`integer mo_data::grav_heat_flag`

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

4.1.2.55 grav_salt

`real(wp) mo_data::grav_salt`

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

4.1.2.56 grav_temp

`real(wp) mo_data::grav_temp`

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

4.1.2.57 h

`real(wp), dimension(:), allocatable mo_data::h`

Enthalpy [J].

4.1.2.58 h_abs

`real(wp), dimension(:), allocatable mo_data::h_abs`

specific Enthalpy [J/kg]

4.1.2.59 h_abs_snow

`real(wp) mo_data::h_abs_snow`

Absolute enthalpy of snow layer [J].

4.1.2.60 harmonic_flag

```
integer mo_data::harmonic_flag
```

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

4.1.2.61 i

```
integer mo_data::i
```

Index, normally used for time.

4.1.2.62 i_time

```
integer mo_data::i_time
```

Number of timesteps.

4.1.2.63 i_time_out

```
integer mo_data::i_time_out
```

Number of timesteps between each output.

4.1.2.64 k

```
integer mo_data::k
```

Index, normally used for layer.

4.1.2.65 lab_snow_flag

```
integer mo_data::lab_snow_flag
```

Niels, 2017 add: 0: lab setup without snow covers, 1: lab setup include snow influence on heat fluxes.

4.1.2.66 length_input

`integer mo_data::length_input`

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

4.1.2.67 length_input_lab

`integer mo_data::length_input_lab`

Niels, 2017 add: used to allocate lab testcase input arrays in mo_init, set value in testcases.

4.1.2.68 liquid_precip

`real(wp) mo_data::liquid_precip`

Liquid precip, [meter of water/s].

4.1.2.69 m

`real(wp), dimension(:), allocatable mo_data::m`

Mass [kg].

4.1.2.70 m_snow

`real(wp) mo_data::m_snow`

Mass of snow layer [kg].

4.1.2.71 m_total

`real(wp) mo_data::m_total`

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

4.1.2.72 melt_err

```
real(wp) mo_data::melt_err =0._wp
```

Niels, 2017 add: used to check how much meltwater vanishes in flushing routine.

4.1.2.73 melt_thick

```
real(wp) mo_data::melt_thick
```

thickness of fully liquid part of top layer [m]

4.1.2.74 melt_thick_output

```
real(wp), dimension(3) mo_data::melt_thick_output
```

Niels, 2017 add: output field of surface liquid meltwater sizes.

4.1.2.75 melt_thick_snow

```
real(wp) mo_data::melt_thick_snow
```

4.1.2.76 melt_thick_snow_old

```
real(wp) mo_data::melt_thick_snow_old
```

Niels(2017) add: thickness of excess fully liquid part from snow_melt_processes [m].

4.1.2.77 n_active

```
integer mo_data::n_active
```

Number of Layers active in the present.

4.1.2.78 n_bgc

`integer mo_data::n_bgc`

Number of chemicals.

4.1.2.79 n_bottom

`integer mo_data::n_bottom`

Number of bottom layers.

4.1.2.80 n_middle

`integer mo_data::n_middle`

Number of middle layers.

4.1.2.81 n_time_out

`integer mo_data::n_time_out`

Counts number of timesteps between output.

4.1.2.82 n_top

`integer mo_data::n_top`

Number of top layers.

4.1.2.83 nlayer

`integer mo_data::nlayer`

Number of layers.

4.1.2.84 ocean_flux_input

```
real(wp), dimension(:), allocatable mo_data::ocean_flux_input
```

Niels, 2017 add: used to read in oceanic heat flux for field experiment tests, dimension needs to be set in the code.

4.1.2.85 ocean_t_input

```
real(wp), dimension(:), allocatable mo_data::ocean_t_input
```

Niels, 2017 add: used to read in ocean temperature for field experiment tests, dimension needs to be set in the code.

4.1.2.86 perm

```
real(wp), dimension(:), allocatable mo_data::perm
```

4.1.2.87 phi

```
real(wp), dimension(:), allocatable mo_data::phi
```

Solid mass fraction.

4.1.2.88 phi_s

```
real(wp) mo_data::phi_s
```

Solid mass fraction of snow layer.

4.1.2.89 precip_flag

```
integer mo_data::precip_flag
```

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

4.1.2.90 precip_input

```
real(wp), dimension(:), allocatable mo_data::precip_input
```

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

4.1.2.91 precipinput

```
real(wp), dimension(:), allocatable mo_data::precipinput
```

Niels, 2017 add: used to read in precipation for field experiment tests, dimension needs to be set in the code.

4.1.2.92 prescribe_flag

```
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!)

4.1.2.93 psi_g

```
real(wp), dimension(:), allocatable mo_data::psi_g
```

Gas volume fraction.

4.1.2.94 psi_g_snow

```
real(wp) mo_data::psi_g_snow
```

Gas volume fraction of snow layer.

4.1.2.95 psi_l

```
real(wp), dimension(:), allocatable mo_data::psi_l
```

Liquid volume fraction.

4.1.2.96 psi_l_snow

`real(wp) mo_data::psi_l_snow`

Liquid volume fraction of snow layer.

4.1.2.97 psi_s

`real(wp), dimension(:), allocatable mo_data::psi_s`

Solid volume fraction.

4.1.2.98 psi_s_snow

`real(wp) mo_data::psi_s_snow`

Solid volume fraction of snow layer.

4.1.2.99 q

`real(wp), dimension(:), allocatable mo_data::q`

Heat in layer [J].

4.1.2.100 ray

`real(wp), dimension(:), allocatable mo_data::ray`

Rayleigh number of each layer.

4.1.2.101 s_abs

`real(wp), dimension(:), allocatable mo_data::s_abs`

Absolute Salinity [g].

4.1.2.102 s_abs_snow

`real(wp) mo_data::s_abs_snow`

Absolute salinity of snow layer [g].

4.1.2.103 s_br

`real(wp), dimension(:), allocatable mo_data::s_br`

Brine salinity [g/kg].

4.1.2.104 s_bu

`real(wp), dimension(:), allocatable mo_data::s_bu`

Bulk Salinity [g/kg].

4.1.2.105 s_bu_bottom

`real(wp) mo_data::s_bu_bottom`

Salinity beneath the ice [g/kg].

4.1.2.106 s_total

`real(wp) mo_data::s_total`

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

4.1.2.107 salt_flag

`integer mo_data::salt_flag`

1: Sea salt, 2: NaCL

4.1.2.108 snow_flush_flag

```
integer mo_data::snow_flush_flag
```

Niels, 2017 add: 0: all meltwater from snow forms slush, 1: meltwater partly leads to flushing, ratio defined by "k_snow_flush".

4.1.2.109 snow_precip_flag

```
integer mo_data::snow_precip_flag
```

Niels, 2017 add: 0: all precipitation is set to zero, 1: physical behaviour.

4.1.2.110 solid_precip

```
real(wp) mo_data::solid_precip
```

Solid precip, [meter of water /s].

4.1.2.111 styropor_flag

```
integer mo_data::styropor_flag
```

4.1.2.112 styropor_input

```
real(wp), dimension(:), allocatable mo_data::styropor_input
```

Niels, 2017 add: if styropor is used in the lab on top of the ice to simulate snow heat fluxes.

4.1.2.113 surface_water

```
real(wp) mo_data::surface_water
```

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

4.1.2.114 t

```
real(wp), dimension(:), allocatable mo_data::t
```

Temperature [C].

4.1.2.115 t2m

```
real(wp) mo_data::t2m
```

Two meter Temperature [C].

4.1.2.116 t2m_input

```
real(wp), dimension(:), allocatable mo_data::t2m_input
```

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

4.1.2.117 t_bottom

```
real(wp) mo_data::t_bottom
```

Temperature of water beneath the ice [C].

4.1.2.118 t_freeze

```
real(wp) mo_data::t_freeze
```

Freezing temperature [C].

4.1.2.119 t_snow

```
real(wp) mo_data::t_snow
```

Temperature of snow layer [C].

4.1.2.120 t_test

```
real(wp), save mo_data::t_test
```

First guess for getT subroutine.

4.1.2.121 t_top

```
real(wp) mo_data::t_top
```

Temperature at the surface [C].

4.1.2.122 tank_depth

```
real(wp) mo_data::tank_depth
```

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

4.1.2.123 tank_flag

```
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

4.1.2.124 test

```
real(wp) mo_data::test
```

Thickness of snow layer [m].

4.1.2.125 thick

```
real(wp), dimension(:), allocatable mo_data::thick
```

Layer thickness [m].

4.1.2.126 thick_0

`real(wp) mo_data::thick_0`

Initial layer thickness [m].

4.1.2.127 thick_min

`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.

4.1.2.128 thick_snow

`real(wp) mo_data::thick_snow`

4.1.2.129 thickness

`real(wp) mo_data::thickness`

Meters of ice [m].

4.1.2.130 time

`real(wp) mo_data::time`

Time [s].

4.1.2.131 time_counter

`integer mo_data::time_counter`

Keeps track of input data.

4.1.2.132 time_input

```
real(wp), dimension(:), allocatable mo_data::time_input
```

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

4.1.2.133 time_out

```
real(wp) mo_data::time_out
```

Time between outputs [s].

4.1.2.134 time_total

```
real(wp) mo_data::time_total
```

Time of simulation [s].

4.1.2.135 tinput

```
real(wp), dimension(:), allocatable mo_data::tinput
```

Niels, 2017 add: used to read in top temperature for field experiment tests, dimension needs to be set in the code.

4.1.2.136 total_resist

```
real(wp) mo_data::total_resist
```

Thermal resistance of the whole column [].

4.1.2.137 ttop_input

```
real(wp), dimension(:), allocatable mo_data::ttop_input
```

Niels, 2017 add: used for testcase 111, comparison with greenland harp data, uppermost harp temperature is seen as Ttop.

4.1.2.138 turb_flag

```
integer mo_data::turb_flag
```

1: No bottom turbulence, 2: Bottom mixing

4.1.2.139 v_ex

```
real(wp), dimension(:), allocatable mo_data::v_ex
```

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

4.1.2.140 v_g

```
real(wp), dimension(:), allocatable mo_data::v_g
```

Volume [m^3] of gas.

4.1.2.141 v_l

```
real(wp), dimension(:), allocatable mo_data::v_l
```

Volume [m^3] of liquid.

4.1.2.142 v_s

```
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, public 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, public 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 flood()

```
subroutine, public 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 flood_simple()

```
subroutine, public 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, public 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, melt_err, perm, flush_v, flush_h, psi_g, thick_snow, rho_l, snow_flush_flag, fl_brine_bgc)

    Subroutine for complex flushing.
- subroutine, public 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

4.3.2 Function/Subroutine Documentation

4.3.2.1 flush3()

```
subroutine, public mo_flush::flush3 (
            real(wp), intent(in) freeboard,
            real(wp), dimension(nlayer), intent(inout) 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), intent(inout) melt_err,
            real(wp), dimension(nlayer), intent(out) perm,
            real(wp), dimension(n_active), intent(inout) flush_v,
```

```
            real(wp), dimension(n_active), intent(inout) flush_h,
            real(wp), dimension(nlayer), intent(inout) psi_g,
            real(wp), intent(in) thick_snow,
            real(wp), intent(in) rho_l,
            integer, intent(in) snow_flush_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)
Changed: Permeability calculation (only for snow_flush_flag==1), hydraulic head and output data by Niels Fuchs, MPIMET (2017-03-01)

Parameters

in	*snow_flush_flag*	Niels, 2017 add: snow_flush_flag
in	*t*	Niels, 2017 add: moved psi_l -> INTENT(inout)
in,out	*psi_g*	Niels, 2017 add: psi_l, psi_g
in,out	*flush_v*	mass of vertically flushed brine of each layer [kg] !< Niels, 2017 add: inout
in,out	*flush_h*	mass of brine which leaves the ice of each layer [kg] !< Niels, 2017 add: inout
out	*perm*	Niels, 2017 add: out
in,out	*fl_brine_bgc*	Niels, 2017 add: if loop, enhanced the permeability, revise
in,out	*fl_brine_bgc*	Niels, 2017 add: psi_g to permeability calculation, improved the results but must be checked
in,out	*fl_brine_bgc*	Niels, 2017 add: melt thich is on top of the ice and therefore also part of the hydraulic head
in,out	*fl_brine_bgc*	Niels, 2017 add: melt_err, check how much meltwater vanishes in the line above

4.3.2.2 flush4()

```
subroutine, public 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) function func_density (T, S)

  Calculates the physical density for given S and T.
- real(wp) function func_freeboard (N_active, Nlayer, psi_s, psi_g, m, thick, m_snow, freeboard_snow_flag)

  Calculates the freeboard of the 1d ice column.
- real(wp) function func_albedo (thick_snow, T_snow, psi_l, thick_min, albedo_flag)

  Calculates the albedo.
- real(wp) function func_sat_o2 (T, S_bu)

  Calculates the oxygen saturation as a function of salinity and temperature.
- real(wp) function 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.

Author

Philipp Griewank

4.4.2 Function/Subroutine Documentation

4.4.2.1 func_albedo()

```
real(wp) function 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 func_density()

```
real(wp) function 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 func_freeboard()

```
real(wp) function 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,
            integer, intent(in) freeboard_snow_flag )
```

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)
Add freeboard_snow_flag calculation of snow mass, check the code for further explanations by Niels Fuchs, MPIMET (2017-03-91)

4.4.2.4 func_sat_o2()

```
real(wp) function 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 func_t_freeze()

```
real(wp) function 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 sub_input()

```
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 sub_melt_snow()

```
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 sub_melt_thick()

```
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 sub_notzflux()

```
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 sub_turb_flux()

```
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, public 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, public 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

4.5.2 Function/Subroutine Documentation

4.5.2.1 fl_grav_drain()

```
subroutine, public 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, 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

out	*ray*	Rayleigh number

4.5.2.2 fl_grav_drain_simple()

```
subroutine, public 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

out	*ray*	Rayleigh number

4.6 mo_grotz Module Reference

The most important module of SAMSIM.

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.

4.6.1 Detailed Description

The most important module of SAMSIM.

The module mo_grotz contains the most important subroutine grotz (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. Subroutine grotz contains the time loop, as well as the initialization, and calls all other branches of the 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 most subroutines, modules, functions, parameters, and global variables have doxygen compatible descriptions. In addition to the doxygen generated description, some python plotscripts are available to plot model output.

Author

Philipp Griewank

4.6.2 Function/Subroutine Documentation

4.6.2.1 grotz()

```
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.

The basic rundown of the time loop is:

1. Calculate the current ice/snow state and forcing, as well as gravity drainage and flooding
2. Apply all the fluxes, recalculate ice state
3. Flushing and layer dynamics

Here is the full rundown of what happens in mo_grotz:

- Initialization: all fields are initialized for the given testcase, and the output is formatted
- Input and Forcing read in: Only if needed by the chosen testcase TIME LOOP BEGINS:
  - Calculate the total ice properties, total freshwater, thermal resistivity, energy, bulk salinity
  - Determine snow and rain rates
  - Calculate snow thermodynamics
  - Calculate inner ice thermodynamic fluxes
  - Calculate brine flux from expulsion
  - Raw output written out if debug_flag is set to 2
  - Standard output written
  - Flooding parametrized
  - Lowest layer mixing with underlying water
  - Gravity drainage parametrized
  - Various testcase specifics
  - Calcuating and applying the heat fluxes
  - After heatfluxes are applied new liquidus thermal equilibrium is calculated
  - Flushing is parametrized
  - Chemistry advection calculated
  - Layer Dynamics TIME LOOP ENDS -Final output, files closed, and fields deallocated

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)
Add some more outputs, changed routine names and arguments with respect to newly introduces flags by Niels Fuchs, MPIMET (2017-03-01)
Added a bit of description with the run down of what happends by Philipp Griewank, Uni K (2018-08-08)

Parameters

in	*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 related to heat fluxes happens in sub_heat_fluxes, which is why it is a very crucial part of SAMSIM.

Author

Philipp Griewank

4.7.2 Function/Subroutine Documentation

4.7.2.1 sub_heat_fluxes()

```
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)
Second version by Niels Fuchs (2017-02-02)

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, length_input_lab)

    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

COPYRIGHT

4.8.2 Function/Subroutine Documentation

4.8.2.1 init()

```
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 sub_allocate()

```
subroutine mo_init::sub_allocate (
            integer, intent(in) Nlayer,
            integer, intent(in), optional length_input_lab )
```

Allocates Arrays.

For a given number of layers Nlayers all arrays are allocated

Parameters

in	*nlayer*	number of layers
in	*length_input_lab*	Niels, 2017 add: dimension of input arrays
in	*length_input_lab*	Niels, 2017
in	*length_input_lab*	Niels, 2017
in	*length_input_lab*	Niels, 2017
in	*length_input_lab*	Niels, 2017
in	*length_input_lab*	Niels, 2017
in	*length_input_lab*	Niels, 2017
in	*length_input_lab*	Niels, 2017
in	*length_input_lab*	Niels, 2017
in	*length_input_lab*	Niels, 2017
in	*length_input_lab*	Niels, 2017
in	*length_input_lab*	Niels, 2017
in	*length_input_lab*	Niels, 2017

4.8.2.3 sub_allocate_bgc()

```
subroutine mo_init::sub_allocate_bgc (
            integer, intent(in) Nlayer,
            integer, intent(in) N_bgc )
```

Allocates BGC Arrays.

4.8.2.4 sub_deallocate()

```
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, public 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, melt_thick_output, N_bgc, bgc_abs, bgc_bottom)

    Organizes the Semi-Adaptive grid SAMSIM uses.
- subroutine, public top_melt (Nlayer, N_active, N_bottom, N_middle, N_top, thick_0, m, S_abs, H_abs, thick, N_bgc, bgc_abs)
- subroutine, public 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

4.9.2 Function/Subroutine Documentation

4.9.2.1 layer_dynamics()

```
subroutine, public 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,
              real(wp), intent(inout) melt_thick_output,
              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)

Parameters

`in,out`	*melt_thick_output*	Niels, 2017 add: melt_thick_output !OBS: only 3rd element in standard melt_thick_output vector!
`in`	*bgc_bottom*	Niels, 2017 add: subtract top growth from melt thick output

4.9.2.2 top_grow()

```
subroutine, public 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 top_melt()

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

  Generates the fluxes caused by expulsion.
- subroutine, public 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

4.10.2 Function/Subroutine Documentation

4.10.2.1 bgc_advection()

```
subroutine, public 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 expulsion_flux()

```
subroutine, public 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 mass_transfer()

```
subroutine, public 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, public 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, k_snow_flush)

  Settings output.
- subroutine, public 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, perm, format_perm, flush_v, flush_h, psi_g, melt_thick_output, format_melt)

Standard output.

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

  Standard bgc output.
- subroutine, public output_raw (Nlayer, N_active, time, T, thick, S_bu, psi_s, psi_l, psi_g)

  Output for debugging purposes.
- subroutine, public 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, public output_raw_lay (Nlayer, N_active, H_abs, m, S_abs, thick, string)

  Output for debugging layer dynamics..
- subroutine, public output_begin (Nlayer, debug_flag, format_T, format_psi, format_thick, format_snow, format_T2m_top, format_perm, format_melt)

  Output files are opened and format strings are created.
- subroutine, public 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

4.11.2 Function/Subroutine Documentation

4.11.2.1 output()

```
subroutine, public 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,
              real(wp), dimension(nlayer), intent(in) perm,
              character*12000, intent(in) format_perm,
              real(wp), dimension(nlayer), intent(in) flush_v,
              real(wp), dimension(nlayer), intent(in) flush_h,
              real(wp), dimension(nlayer), intent(in) psi_g,
              real(wp), dimension(3), intent(in) melt_thick_output,
              character*12000, intent(in) format_melt )
```

Standard output.

For time=n*time_out data is exported.

Revision History

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

Parameters

in	*melt_thick_output*	Niels, 2017: 1: accumulated melt_thick, 2: accumulated melt_thick_snow, 3: accumulated top ice thickness variations (recheck 3: in mo_layer_dynamics)
in	*format_melt*	Niels, 2017 add: output permeability
in	*format_melt*	Niels, 2017 add: output vertical flushing
in	*format_melt*	Niels, 2017 add: output horizontal flushing
in	*format_melt*	Niels, 2017 add: output gas fraction !OBS: not simulated physically in SAMSIM
in	*format_melt*	Niels, 2017

4.11.2.2 output_begin()

```
subroutine, public 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,
              character*12000, intent(out) format_perm,
              character*12000, intent(out) format_melt )
```

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 output_begin_bgc()

```
subroutine, public 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. Added ADJUSTL to the output strings because they got wierd

Revision History

created by Dr. Philipp Griewank, MPI (2014-02-07) fix by Dr. Philipp Griewank, UniK (2018-05-18)

4.11.2.4 output_bgc()

```
subroutine, public 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 output_raw()

```
subroutine, public 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 output_raw_lay()

```
subroutine, public 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 output_raw_snow()

```
subroutine, public 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 output_settings()

```
subroutine, public 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,
            real(wp), intent(in) k_snow_flush )
```

Settings output.

Writes important values to latter identify run.

Revision History

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

Parameters

in	*description*	Niels, 2017

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 = 1.0_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], !< Niels, 2017 add: can be adjusted to lab values if they are measured
- 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 = 10000.0

  Maximal heating rate of a heating plate, set so high so that it doesn't interfere with testcase 1.
- real(wp) k_snow_flush = 0.75_wp

  Niels, 2017 add: Percentage of excess liquid water content in the snow that is used for flushing instead of forming slush.
- real(wp) k_styropor = 0.8_wp

  Niels, 2017 add: heat conduction of styropor (empirical value to fit measurement data)

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.

Author

Philipp Griewank

COPYRIGHT

Revision History

Started by Philipp Griewank 2010-07-08
add parameters: heat conductivity of styropor under special sea ice lab conditions and ratio of penetrating melt water by Niels Fuchs, MPIMET (2017-03-01)

4.12.2 Variable Documentation

4.12.2.1 bbeta

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

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

4.12.2.2 c_l

```
real(wp), parameter mo_parameters::c_l = 3400._wp
```

liquid heat capacity [J/ kg K]

4.12.2.3 c_s

```
real(wp), parameter mo_parameters::c_s = 2020.0_wp
```

solid heat capacity [J/ kg K]

4.12.2.4 c_s_beta

```
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 emissivity_ice

```
real(wp), parameter mo_parameters::emissivity_ice = 0.95_wp
```

Emissivity of water and ice.

4.12.2.6 emissivity_snow

```
real(wp), parameter mo_parameters::emissivity_snow = 1.00_wp
```

Emissivity of Snow.

4.12.2.7 extinc

```
real(wp), parameter mo_parameters::extinc = 2.00_wp
```

Extinction coefficient of ice.

4.12.2.8 gas_snow_ice

```
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 gas_snow_ice2

```
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 grav

```
real, parameter mo_parameters::grav = 9.8061_wp
```

gravitational constant [m/s^2]

4.12.2.11 k_l

```
real(wp), parameter mo_parameters::k_l = 0.523_wp
```

liquid heat conductivity [J / m s K] 0.523

4.12.2.12 k_s

```
real(wp), parameter mo_parameters::k_s = 2.2_wp
```

solid heat conductivity [J / m s K] 2.2

4.12.2.13 k_snow_flush

```
real(wp) mo_parameters::k_snow_flush = 0.75_wp
```

Niels, 2017 add: Percentage of excess liquid water content in the snow that is used for flushing instead of forming slush.

4.12.2.14 k_styropor

```
real(wp) mo_parameters::k_styropor = 0.8_wp
```

Niels, 2017 add: heat conduction of styropor (empirical value to fit measurement data)

4.12.2.15 kappa_l

```
real(wp), parameter mo_parameters::kappa_l = k_l/rho_l/c_l
```

heat diffusivity of water

4.12.2.16 latent_heat

```
real(wp), parameter mo_parameters::latent_heat = 333500._wp
```

latent heat release [J/kg]

4.12.2.17 max_flux_plate

```
real(wp) mo_parameters::max_flux_plate = 10000.0
```

Maximal heating rate of a heating plate, set so high so that it doesn't interfere with testcase 1.

4.12.2.18 mu

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

dynamic viscosity [kg /m s]

4.12.2.19 neg_free

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

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

4.12.2.20 para_flush_gamma

```
real(wp), parameter mo_parameters::para_flush_gamma = 0.9_wp
```

Strength of desalination per timestep (guess)

4.12.2.21 para_flush_horiz

```
real(wp), parameter mo_parameters::para_flush_horiz = 1.0_wp
```

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

4.12.2.22 penetr

```
real(wp), parameter mo_parameters::penetr = 0.30_wp
```

Amount of penetrating sw radiation.

4.12.2.23 pi

```
real, parameter mo_parameters::pi = 3.1415_wp
```

4.12.2.24 psi_s_min

```
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.

4.12.2.25 psi_s_top_min

```
real(wp), parameter mo_parameters::psi_s_top_min = 0.40_wp
```

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

4.12.2.26 ratio_flood

```
real(wp), parameter mo_parameters::ratio_flood = 1.50_wp
```

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

4.12.2.27 ray_crit

```
real(wp), parameter mo_parameters::ray_crit = 4.89_wp
```

4.12.2.28 ref_salinity

```
real(wp), parameter mo_parameters::ref_salinity = 34._wp
```

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

4.12.2.29 rho_l

```
real(wp), parameter mo_parameters::rho_l = 1028.0_wp
```

density of liquid [kg / m^3]

4.12.2.30 rho_s

```
real(wp), parameter mo_parameters::rho_s = 920._wp
```

density of solid [kg / m^3]

4.12.2.31 rho_snow

```
real(wp), parameter mo_parameters::rho_snow = 330._wp
```

density of new snow [kg/m**3], !< Niels, 2017 add: can be adjusted to lab values if they are measured

4.12.2.32 sigma

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

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

4.12.2.33 turb_a

```
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.

4.12.2.34 turb_b

```
real(wp), parameter mo_parameters::turb_b = 0.05_wp
```

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

4.12.2.35 wp

```
integer, parameter mo_parameters::wp = SELECTED_REAL_KIND(12, 307)
```

set working precision _wp

4.12.2.36 x_grav

```
real(wp), parameter mo_parameters::x_grav = 0.000584_wp
```

4.12.2.37 zerok

```
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, public 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, public 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, public 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, public 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, public snow_thermo_meltwater (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, melt_thick_snow)

Subroutine for calculating snow thermodynamics.

- subroutine, public 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, public 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, public 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) function, public 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.

Author

Philipp Griewank

4.13.2 Function/Subroutine Documentation

4.13.2.1 func_k_snow()

```
real(wp) function, public 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 snow_coupling()

```
subroutine, public 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 snow_precip()

```
subroutine, public 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.

The two meter temperature (T2m) is used to determine the thermal energy of the snow/rain. Snow is assumed to never be higher than -1 Celsius.

Revision History

Sired by Philipp Griewank, IMPRS (2010-12-14) Fixed T bug by Philipp Griewank, UzK (2020-08-13)

4.13.2.4 snow_precip_0()

```
subroutine, public 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 snow_thermo()

```
subroutine, public 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

in,out	*h_abs*	Top ice layer variables

4.13.2.6 snow_thermo_meltwater()

```
subroutine, public mo_snow::snow_thermo_meltwater (
            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,
            real(wp), intent(inout) melt_thick_snow )
```

Subroutine for calculating snow thermodynamics.

most of the physics are taken from snow_thermo() based on lab observations: parts of the snow meltwater percolate directly into the ice

Revision History

introduced by Niels Fuchs (2016-10-13)

Parameters

in,out	*h_abs*	Top ice layer variables

4.13.2.7 sub_fl_q_0_snow()

```
subroutine, public 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

in	*t_bound*	T_bound temperature of boundary layer

4.13.2.8 sub_fl_q_0_snow_thin()

```
subroutine, public 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)

4.13.2.9 sub_fl_q_snow()

```
subroutine, public 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, public sub_test1 (time, T_top)
    Subroutine for changing T_top for testcase 1.
- subroutine, public sub_test2 (time, T2m)
    Subroutine for changing T_top for testcase 2.
- subroutine, public sub_test9 (time, T2m)
    Subroutine for changing T2m for testcase 9.
- subroutine, public sub_test34 (time, T2m)
    Subroutine for changing T2m for testcase 34.
- subroutine, public sub_test3 (time, liquid_precip, solid_precip)
    Subroutine for setting snow for testcase 3.
- subroutine, public sub_test4 (time, fl_q_bottom)
    Subroutine for setting snow for testcase 4.
- subroutine, public 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

Revision History

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

4.14.2 Function/Subroutine Documentation

4.14.2.1 sub_test1()

```
subroutine, public 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 sub_test2()

```
subroutine, public 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 sub_test3()

```
subroutine, public 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 sub_test34()

```
subroutine, public mo_testcase_specifics::sub_test34 (
            real(wp), intent(in) time,
            real(wp), intent(inout) T2m )
```

Subroutine for changing T2m for testcase 34.

T2m is adjusted over time.

Revision History

adjusted by Niels Fuchs, MPI (2016-01-18)

4.14.2.5 sub_test4()

```
subroutine, public 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)

4.14.2.6 sub_test6()

```
subroutine, public 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)

4.14.2.7 sub_test9()

```
subroutine, public mo_testcase_specifics::sub_test9 (
            real(wp), intent(in) time,
            real(wp), intent(inout) T2m )
```

Subroutine for changing T2m for testcase 9.

T2m is adjusted over time.

Revision History

Formed by Niels Fuchs, MPI (2016-01-18)

4.15 mo_thermo_functions Module Reference

Contains subroutines and functions related to multi-phase thermodynamics.

Functions/Subroutines

- subroutine, public gett (H, S_bu, T_in, T, phi, k)
  Determines equilibrium Temperature of a layer for given S_bu and H as well as solid fraction.
- subroutine, public expulsion (phi, thick, m, psi_s, psi_l, psi_g, V_ex)
  Determines Brine flux expelled from out of a layer due to freezing.
- subroutine, public sub_fl_q (psi_s_1, psi_l_1, psi_g_1, thick_1, T_1, psi_s_2, psi_l_2, psi_g_2, thick_2, T_2, fl_Q)
  Determines conductive heat flux between two layers.
- subroutine, public sub_fl_q_0 (psi_s, psi_l, psi_g, thick, T, T_bound, direct_flag, fl_Q)
  Determines conductive Heat flux between layer and boundary temperatures.
- subroutine, public sub_fl_q_styropor (k_styropor, fl_Q)
  Niels, 2017 add: Determines conductive Heat flux below styropor cover.
- real(wp) function, public func_s_br (T, S_bu)
  Computes salinity of brine pockets for given temperature in Celsius of mushy layer.
- real(wp) function, public func_ddt_s_br (T)
  Computes temperature derivative of brine pocket salinity for given temperature in Celsius of mushy layer.

4.15.1 Detailed Description

Contains subroutines and functions related to multi-phase thermodynamics.

See the subroutine and function descriptions for details.

Author

Philipp Griewank

Revision History

Started by Philipp Griewank 2010-07-08
Add function for styropor cover by Niels Fuchs, MPIMET (2017-01-03)
Modified salinity functions by Philipp Griewank, Uni K (2018-08-01)

4.15.2 Function/Subroutine Documentation

4.15.2.1 expulsion()

```
subroutine, public mo_thermo_functions::expulsion (
            real(wp), intent(in) phi,
            real(wp), intent(in) thick,
            real(wp), intent(inout) m,
            real(wp), intent(out) psi_s,
            real(wp), intent(out) psi_l,
            real(wp), intent(out) psi_g,
            real(wp), intent(out) V_ex )
```

Determines Brine flux expelled from out of a layer due to freezing.

If the volume of ice and brine exceed the Volume of the layer brine is expelled. The volume of the ejected brine is calculated and exported. The volume fractions are also calculated.

Revision History

first version by Philipp Griewank, (2010-07-19)
changes to mass, Enthalpy and Salinity are now computed in subroutine mass_transfer by Philipp Griewank, (2010-08-24)

4.15.2.2 func_ddt_s_br()

```
real(wp) function, public mo_thermo_functions::func_ddt_s_br (
            real(wp), intent(in) T )
```

Computes temperature derivative of brine pocket salinity for given temperature in Celsius of mushy layer.

Subroutine computes one ddT_S_br for one given T NaCl solutions and seawater produce slight variations. Which solution is used is specified by salt_flag. Based on Notz 2005 p. 36 ddT_S_br = c2+2∗c3∗T+3∗c4∗T^2

For T below -20 we simply use a linear extension based on "Composition of sea ice and its tensile strength". The actual salinity function is much more complicated and depends on the salt composition, but the linear fit is far better than using the polynomial fit.

The NaCL precipitates at -22, leading to ice/salt kristall mix below -22. Ideally the whole code would be modified to take the non-continous transition at -22 but given that there is currently little interest I can't be bothered to put in the effor.

Revision History

First version by Philipp Griewank (2010-07-13)
Added linear bit by Philipp Griewank (2018-07-22)

Parameters

in	*t*	Temperature in Celsius

Returns

derivative of Brine salinity

4.15.2.3 func_s_br()

```
real(wp) function, public mo_thermo_functions::func_s_br (
            real(wp), intent(in) T,
            real(wp), intent(in), optional S_bu )
```

Computes salinity of brine pockets for given temperature in Celsius of mushy layer.

Subroutine computes one S_br for one given T in Celsius by third-order polynomial. NaCl solutions and seawater produce slight variations. Which solution is used is determined by salt_flag. S_br = c1+c2∗T+c3∗T^2+c4∗T^3 Was originally based on that of Notz 2005 p. 36, but was switched to the POLY3 of Vancoppenolle 2019 et al, Thermodynamics of Sea ICe Phase compositon Revisited

For T below -20 we simply use a linear extension based on "Composition of sea ice and its tensile strength". The actual salinity function is much more complicated and depends on the salt composition, but the linear fit is far better than using the polynomial fit.

The NaCL precipitates at -22, leading to ice/salt kristall mix below -22. Ideally the whole code would be modified to take the non-continous transition at -22 but given that there is currently little interest I can't be bothered to put in the effor.

Revision History

First version by Philipp Griewank (2010-07-12)
Changed to go through 0 by Philipp Griewank (2014-05-07)
Added linear bit by Philipp Griewank (2018-07-22)
Changed constants to those of POLY3 Vancoppenolle 2019 by Philipp Griewank (2019-02-15)

Parameters

in	*t*	Temperature in Celsius

Returns

Brine salinity

4.15.2.4 gett()

```
subroutine, public mo_thermo_functions::gett (
            real(wp), intent(in) H,
            real(wp), intent(in) S_bu,
            real(wp), intent(in) T_in,
            real(wp), intent(out) T,
            real(wp), intent(out) phi,
            integer, intent(in), optional k )
```

Determines equilibrium Temperature of a layer for given S_bu and H as well as solid fraction.

The temperature of a fully liquid layer is used to see if the resulting brine salinity is lower than the bulk salinity. After checking if the layer is a fluid or a mushy layer the temperature is calculated by solving f(T) = 0 using the Newton method. f(T) = -latent_heat-H+latent_heat*S_bu/S_br(T) + c_s*T+c_s_beta*T^2/2 f'(T) = c_s+c_s_beta*T-latent_heat*S_bu*S_br'(T)/S_br^2 Described in Notz2005, subsubsection 5.6.1. See func_S_br(T) and func_ddT_S_br(T). First guess T_0 must be given, low first guess lead to overshooting which would lead to very high Temperatures. To avoid this, an if loop sets T to freezing T when T>0. Freezing T is also calculated at the beginning using the Newton-Method. If S_bu<0.001 then it is treated as pure ice.

Revision History

first version by Philipp Griewank (2010-07-13)
Freezing temperature is calculated and introduced if T goes above 0 by Philipp Griewank (2010-07-13)
Added if loops to deal with saltless ice by Philipp Griewank (2010-11-27)

Parameters

in	*h*	Enthalpy [J/kg]
in	*s_bu*	Bulk Salinity [g/kg]
in	*t_in*	input Temperature for T_0 [C]
out	*t*	Temperature [C]
out	*phi*	solid fraction

4.15.2.5 sub_fl_q()

```
subroutine, public mo_thermo_functions::sub_fl_q (
            real(wp), intent(in) psi_s_1,
            real(wp), intent(in) psi_l_1,
            real(wp), intent(in) psi_g_1,
            real(wp), intent(in) thick_1,
            real(wp), intent(in) T_1,
            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 two layers.

Details can be found in Notz 2005, especially equation 5.7. The gas volume is assumed to have no thermal properties at all. First the thermal resistance R is calculated using the approximated thermal conductivity of the mushy layer (see Notz 2005 eq. 3.41.). Then the heat flux Q is simply (T_1-T_2)/R "_1" denotes the upper layer and "_2" the lower layer. A positive heat flux is from lower to upper layer.

Revision History

First version by Philipp Griewank (2010-07-21)

4.15.2.6 sub_fl_q_0()

```
subroutine, public mo_thermo_functions::sub_fl_q_0 (
            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,
            real(wp), intent(in) T_bound,
            integer direct_flag,
            real(wp), intent(out) fl_Q )
```

Determines conductive Heat flux between layer and boundary temperatures.

Details can be found in Notz 2005, especially equation 5.10 and 5.11. The gas volume is assumed to have no thermal properties. direct_flag denotes if the boundary layer is above or below the layer. 1 : = layer above boundary -1: = layer below boundary

Revision History

first version by Philipp Griewank (2010-07-21)

Parameters

in	*t_bound*	T_bound temperature of boundary layer

4.15.2.7 sub_fl_q_styropor()

```
subroutine, public mo_thermo_functions::sub_fl_q_styropor (
            real(wp), intent(in) k_styropor,
            real(wp), intent(inout) fl_Q )
```

Niels, 2017 add: Determines conductive Heat flux below styropor cover.

Standard approach.

Revision History

first version by Niels Fuchs, MPIMET (2017-01-03)

Chapter 5

File Documentation

5.1 mo_data.f90 File Reference

Modules

- module mo_data

  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
- real(wp), dimension(:), allocatable mo_data::flush_v
- real(wp), dimension(:), allocatable mo_data::flush_h
- real(wp), dimension(:), allocatable mo_data::flush_v_old
- real(wp), dimension(:), allocatable mo_data::flush_h_old

  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.
- integer mo_data::styropor_flag
- 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
- character ∗12000 mo_data::format_melt

  Format strings for output. Niels(2017) add: melt output.
- character ∗12000 mo_data::format_perm

  Niels(2017) add: permeability 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
- real(wp) mo_data::test

  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::melt_thick_snow
- real(wp) mo_data::melt_thick_snow_old

  Niels(2017) add: thickness of excess fully liquid part from snow_melt_processes [m].
- real(wp), dimension(3) mo_data::melt_thick_output

  Niels, 2017 add: output field of surface liquid meltwater sizes.
- 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::lab_snow_flag

  Niels, 2017 add: 0: lab setup without snow covers, 1: lab setup include snow influence on heat fluxes.
- integer mo_data::freeboard_snow_flag

  Niels, 2017 add: 0: respect the mass of snow in the freeboard calculation, 1: don't.
- integer mo_data::snow_flush_flag

  Niels, 2017 add: 0: all meltwater from snow forms slush, 1: meltwater partly leads to flushing, ratio defined by "k_snow_flush".
- integer mo_data::snow_precip_flag

  Niels, 2017 add: 0: all precipitation is set to zero, 1: physical behaviour.
- integer mo_data::length_input

  Sets the input length for atmoflux_flag==2, common value of 13169.
- real(wp), dimension(:), allocatable mo_data::tinput

  Niels, 2017 add: used to read in top temperature for field experiment tests, dimension needs to be set in the code.
- real(wp), dimension(:), allocatable mo_data::precipinput

  Niels, 2017 add: used to read in precipation for field experiment tests, dimension needs to be set in the code.
- real(wp), dimension(:), allocatable mo_data::ocean_t_input

  Niels, 2017 add: used to read in ocean temperature for field experiment tests, dimension needs to be set in the code.
- real(wp), dimension(:), allocatable mo_data::ocean_flux_input

  Niels, 2017 add: used to read in oceanic heat flux for field experiment tests, dimension needs to be set in the code.
- real(wp), dimension(:), allocatable mo_data::styropor_input

Niels, 2017 add: if styropor is used in the lab on top of the ice to simulate snow heat fluxes.
- real(wp), dimension(:), allocatable mo_data::ttop_input

  Niels, 2017 add: used for testcase 111, comparison with greenland harp data, uppermost harp temperature is seen as Ttop.
- 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
- character ∗3 mo_data::flush_question ='No!'

  Niels, 2017 add: used to indicate in stdout wether flushing occurs at this moment or not.
- real(wp) mo_data::melt_err =0._wp

  Niels, 2017 add: used to check how much meltwater vanishes in flushing routine.
- integer mo_data::length_input_lab

  Niels, 2017 add: used to allocate lab testcase input arrays in mo_init, set value in testcases.

5.2 mo_flood.f90 File Reference

Modules

- module mo_flood

  Computes the fluxes caused by liquid flooding the snow layer.

Functions/Subroutines

- subroutine, public 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, public 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.

5.3 mo_flush.f90 File Reference

Modules

- module mo_flush

  Contains various subroutines for flushing.

Functions/Subroutines

- subroutine, public 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, melt_err, perm, flush_v, flush_h, psi_g, thick_snow, rho_l, snow_flush_flag, fl_brine_bgc)

  Subroutine for complex flushing.
- subroutine, public 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.

5.4 mo_functions.f90 File Reference

Modules

- module mo_functions

  Module houses functions which have no home :(.

Functions/Subroutines

- real(wp) function mo_functions::func_density (T, S)

  Calculates the physical density for given S and T.
- real(wp) function mo_functions::func_freeboard (N_active, Nlayer, psi_s, psi_g, m, thick, m_snow, freeboard_snow_flag)

  Calculates the freeboard of the 1d ice column.
- real(wp) function mo_functions::func_albedo (thick_snow, T_snow, psi_l, thick_min, albedo_flag)

  Calculates the albedo.
- real(wp) function mo_functions::func_sat_o2 (T, S_bu)

  Calculates the oxygen saturation as a function of salinity and temperature.
- real(wp) function 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.

5.5 mo_grav_drain.f90 File Reference

Modules

- module mo_grav_drain

  Computes the Salt fluxes caused by gravity drainage.

Functions/Subroutines

- subroutine, public 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, public 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.

5.6 mo_grotz.f90 File Reference

Modules

- module mo_grotz

  The most important module of SAMSIM.

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.

5.7 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.

5.8 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, length_input_lab)
  Allocates Arrays.
- subroutine mo_init::sub_allocate_bgc (Nlayer, N_bgc)
  Allocates BGC Arrays.
- subroutine mo_init::sub_deallocate
  Deallocates Arrays.

5.9 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, public 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, melt_thick_output, N_bgc, bgc_abs, bgc_bottom)
  Organizes the Semi-Adaptive grid SAMSIM uses.
- subroutine, public 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, public 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.

5.10 mo_mass.f90 File Reference

Modules

- module mo_mass

  Regulates mass transfers and their results.

Functions/Subroutines

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

  Generates the fluxes caused by expulsion.
- subroutine, public 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.

5.11 mo_output.f90 File Reference

Modules

- module mo_output

  All things output.

Functions/Subroutines

- subroutine, public 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, k_snow_flush)

  Settings output.
- subroutine, public 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, perm, format_perm, flush↩ _v, flush_h, psi_g, melt_thick_output, format_melt)

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

  Standard bgc output.
- subroutine, public mo_output::output_raw (Nlayer, N_active, time, T, thick, S_bu, psi_s, psi_l, psi_g)

  Output for debugging purposes.
- subroutine, public 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, public mo_output::output_raw_lay (Nlayer, N_active, H_abs, m, S_abs, thick, string)

  Output for debugging layer dynamics..
- subroutine, public mo_output::output_begin (Nlayer, debug_flag, format_T, format_psi, format_thick, format↩ _snow, format_T2m_top, format_perm, format_melt)

  Output files are opened and format strings are created.
- subroutine, public mo_output::output_begin_bgc (Nlayer, N_bgc, format_bgc)

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

5.12 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 = 1.0_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], !< Niels, 2017 add: can be adjusted to lab values if they are measured*
- 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 = 10000.0

    Maximal heating rate of a heating plate, set so high so that it doesn't interfere with testcase 1.
- real(wp) mo_parameters::k_snow_flush = 0.75_wp

    Niels, 2017 add: Percentage of excess liquid water content in the snow that is used for flushing instead of forming slush.
- real(wp) mo_parameters::k_styropor = 0.8_wp

    Niels, 2017 add: heat conduction of styropor (empirical value to fit measurement data)

5.13 mo_snow.f90 File Reference

Modules

- module mo_snow

    Module contains all things directly related to snow.

Functions/Subroutines

- subroutine, public 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, public 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, public 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, public 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, public mo_snow::snow_thermo_meltwater (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, melt_thick_snow)

  Subroutine for calculating snow thermodynamics.
- subroutine, public 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, public 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, public 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) function, public mo_snow::func_k_snow (m_snow, thick_snow)

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

5.14 mo_testcase_specifics.f90 File Reference

Modules

- module mo_testcase_specifics

  Module contains changes specific testcases require during the main timeloop.

Functions/Subroutines

- subroutine, public mo_testcase_specifics::sub_test1 (time, T_top)

  Subroutine for changing T_top for testcase 1.
- subroutine, public mo_testcase_specifics::sub_test2 (time, T2m)

  Subroutine for changing T_top for testcase 2.
- subroutine, public mo_testcase_specifics::sub_test9 (time, T2m)

  Subroutine for changing T2m for testcase 9.
- subroutine, public mo_testcase_specifics::sub_test34 (time, T2m)

  Subroutine for changing T2m for testcase 34.
- subroutine, public mo_testcase_specifics::sub_test3 (time, liquid_precip, solid_precip)

  Subroutine for setting snow for testcase 3.
- subroutine, public mo_testcase_specifics::sub_test4 (time, fl_q_bottom)

  Subroutine for setting snow for testcase 4.
- subroutine, public mo_testcase_specifics::sub_test6 (time, T2m)

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

5.15 mo_thermo_functions.f90 File Reference

Modules

- module mo_thermo_functions

  Contains subroutines and functions related to multi-phase thermodynamics.

Functions/Subroutines

- subroutine, public mo_thermo_functions::gett (H, S_bu, T_in, T, phi, k)

    Determines equilibrium Temperature of a layer for given S_bu and H as well as solid fraction.
- subroutine, public mo_thermo_functions::expulsion (phi, thick, m, psi_s, psi_l, psi_g, V_ex)

    Determines Brine flux expelled from out of a layer due to freezing.
- subroutine, public mo_thermo_functions::sub_fl_q (psi_s_1, psi_l_1, psi_g_1, thick_1, T_1, psi_s_2, psi_l_2, psi_g_2, thick_2, T_2, fl_Q)

    Determines conductive heat flux between two layers.
- subroutine, public mo_thermo_functions::sub_fl_q_0 (psi_s, psi_l, psi_g, thick, T, T_bound, direct_flag, fl_Q)

    Determines conductive Heat flux between layer and boundary temperatures.
- subroutine, public mo_thermo_functions::sub_fl_q_styropor (k_styropor, fl_Q)

    Niels, 2017 add: Determines conductive Heat flux below styropor cover.
- real(wp) function, public mo_thermo_functions::func_s_br (T, S_bu)

    Computes salinity of brine pockets for given temperature in Celsius of mushy layer.
- real(wp) function, public mo_thermo_functions::func_ddt_s_br (T)

    Computes temperature derivative of brine pocket salinity for given temperature in Celsius of mushy layer.

5.16 SAMSIM.f90 File Reference

Functions/Subroutines

- program samsim

5.16.1 Function/Subroutine Documentation

5.16.1.1 samsim()

```
program samsim ( )
```

Index