# arc3o

Clara Burgard

Nov 18, 2020

# GETTING STARTED:

The Arctic Ocean Observation Operator (ARC3O) computes brightness temperatures at 6.9 GHz, vertical polarization, based on climate model output. More information about the motivation, structure and evaluation can be found in Burgard et al., 2020a and Burgard et al., 2020b.

Currently, it is customized for output of the Max Planck Institute Earth System Model but can be used for other models if the variable names are changed accordingly in the ARC3O functions.

CHAPTER

ONE

# DOCUMENTATION

## 1.1 About

### 1.1.1 The motivation behind ARC3O

The diversity in sea ice concentration observational estimates affects our understanding of past and future sea ice evolution as it inhibits reliable climate model evaluation [Notz et al., 2013] and initialization [Bunzel et al., 2016]. It also limits our ability to fully exploit relationships between the evolution of sea ice and other climate variables, such as global-mean surface temperature [Niederdrenk & Notz, 2018] and CO2 emissions [Notz & Stroeve, 2016].

To address these issues, we construct an observation operator for the Arctic Ocean at the frequency of 6.9 GHz. This operator provides an alternative approach for climate model evaluation and initialization with satellite observations.

The ARCtic Ocean Observation Operator (ARC3O) provides the possibility to simulate top-of-the-atmosphere brightness temperatures for the Arctic Ocean area at 6.9 GHz, vertical polarization, from climate model output. This simulated brightness temperature can be compared to brightness temperatures measured by satellites from space.

### 1.1.2 Authors

The ARC3O code is partly based on MATLAB code by C. Mätzler, A. Wiesmann, and R.T. Tonboe. Their contribution is acknowledged in the notes of the functions they developed.
This MATLAB code has been translated to Python and further ARC3O code has been developed by Clara Burgard - ClimateClara.

### 1.1.3 License

This project is licensed under the GPL3 License - see the license for details.

## 1.2 How to install

you can install it via pip

```
pip install arc3o
```

or via conda

```
conda install -c conda-forge arc3o
```

This package is programmed in python 3.6 and should be working with all *python versions > 3.6*. Additional requirements are numpy, xarray, pandas, tqdm and pathos.

We recommend to install the dependencies via

```
conda install -c conda-forge pandas tqdm pathos
```

as they might not work well using `pip`.

If you want to work on the code, please fork this github repository: https://github.com/ClimateClara/arc3o/

## 1.3 How to run

### 1.3.1 Input data

Currently, ARC3O is tailored for model output from ECHAM, the atmospheric module of the Max Planck Institute Earth System Model. However, it can be used for other models if the variable names are changed accordingly either in the ARC3O functions or the model's output.

The input data for ARC3O should be divided into monthly files, found in the folder `inputpath`. Also, to prepare the seasons and ice types mask, ARC3O needs one file in which all the data is merged, found in the folder `inputpath0`.

**These monthly and overview files should contain fields [time,lat,lon] of the following variables:**

- `snifrac`: snow fraction on ice [0-1]
- `siced`: sea-ice thickness [m]
- `sni`: snow water equivalent [m]
- `tsi`: surface temperature at surface of snow (if present) or of ice (if no snow) [K]
- `qvi`: columnar water vapor [mm]
- `wind10`: wind speed [m/s]
- `xlvi`: columnar liquid water [mm]
- `tsw`: sea-surface temperature [K]
- `seaice`: sea-ice concentration [0-1]
- `slm`: sea-land mask
- `ameltfrac`: melt-pond fraction [0-1]

### 1.3.2 Running ARC3O

You can run ARC3O with the function *`arc3o.core_functions.satsim_complete_parallel()`* as follows:

```
import xarray as xr
import arc3o as arc3o

# inputpath for the monthly chucked files of the climate model output
inputpath = 'pathtofolder1'
# inputpath for the merged file of the climate model output for the whole time series
inputpath0 = 'pathtofolder2'
# outputpath for output folders of your different experiments
```

(continues on next page)

(continued from previous page)

```
outputpath0 = 'pathtofolder3'

# outputpath for your experiment files
#'yes': create a new path for the output files (will create a new folder in␣
↪outputpath0, called
# yyyymmdd-hhmm)
#'no': keeps the path given in the third option
outputpath = arc3o.new_outputpath('no',outputpath0,'20190516-1047')


# read in the whole time period
orig_data = xr.open_dataset(inputpath0+'assim_SICCI2_50km_echam6_200211-200812_
↪selcode_Arctic.nc')
# transform the unusual MPI-ESM timestamp into a better readable one
orig_data = arc3o.prep_time(orig_data)

# give the first and last year of the time period of your data
start_year = 2003
end_year = 2008

# explain how the monthly chunked file names are built around yyyymm (the ones in␣
↪inputpath),
# example files are called 'assim_SICCI2_50km_echam6_yyyymm_selcode_Arctic.nc' where␣
↪yyyy is the year and mm the month
file_begin = 'assim_SICCI2_50km_echam6_'
file_end = '_selcode_Arctic.nc'

# frequency of interest in GHz (must fit one of the AMSR-E frequencies: 6.9, 10.7, 18.
↪7, 23.8, 36.5, 50.3, 52.8, 89.0)
freq_of_int = 6.9

# run ARC3O
arc3o.satsim_complete_parallel(orig_data,               # climate model output, whole␣
↪time series
                               freq_of_int,             # frequency of interest
                               start_year,end_year,     # first and last year of the␣
↪time period of interest
                               inputpath,               # where to find the monthly␣
↪chunked climate model output
                               outputpath,              # where to write out the␣
↪results
                               file_begin,file_end,     # file name as wrapped around␣
↪yyyymm for the monthly chunked files
                               timestep=6,              # timestep of climate model␣
↪data in hours
                               write_mask='yes',        # 'yes' if you want to␣
↪compute and write out the ice type and season mask, 'no' if you already have a file
↪'period_masks_assim.nc' in outputpath
                               write_profiles='yes',    # 'yes' if you want to␣
↪compute and write out the profiles, 'no' if you already have monthly chunked files
↪'profiles_for_memls_snowno_yyyymm.nc' and 'profiles_for_memls_snowyes yyyymm.nc' in␣
↪outputpath
                               compute_memls='yes',     # 'yes' if you want to␣
↪compute and write out the cold conditions ice surface brightness temperature, 'no'␣
↪if you already have monthly chunked files 'TB_assim_yyyymm_f.nc' in outputpath
                               e_bias_fyi=0.968,        # factor affecting the␣
↪temperature profiles to bias-correct the brightness temperature (for first-year ice)
```

(continues on next page)

(continued from previous page)

```
                            e_bias_myi=0.968,           # factor affecting the␣
↪temperature profiles to bias-correct the brightness temperature (for multiyear ice)
                            snow_emis=1,                # snow emissivity for periods␣
↪of melting snow
                            snow dens=300.,             # snow density in kg/m3
                            pool_nb=12)                 # amount of pool workers to␣
↪compute several months parallelly
```

**Note:** The process takes up a lot of memory so I recommend using a supercomputer if possible. I am working on a "lighter" version but not sure when it will be ready so you'll currently need to reduce the number of pool workers in the *Pool*. To do so, choose a `pool_nb` between 1 and 12. The lower the `pool_nb`, the less pool workers are activated at the moment (i.e. less memory usage but also longer time until results are ready).

If you only want to run one month instead of the whole time series, you can also run ARC3O with *arc3o.core_functions.satsim_complete_1month()*:

```
import xarray as xr
import arc3o as arc3o

# inputpath for the monthly chucked files of the climate model output
inputpath = 'pathtofolder1'
# inputpath for the merged file of the climate model output for the whole time series
inputpath0 = 'pathtofolder2'
# outputpath for output folders of your different experiments
outputpath0 = 'pathtofolder3'

# outputpath for your experiment files
#'yes': create a new path for the output files (will create a new folder in␣
↪outputpath0, called
# yyyymmdd-hhmm)
#'no': keeps the path given in the third option
outputpath = arc3o.new_outputpath('no',outputpath0,'20190516-1047')

# read in the whole time period
orig_data = xr.open_dataset(inputpath0+'assim_SICCI2_50km_echam6_200211-200812_
↪selcode_Arctic.nc')
# transform the unusual MPI-ESM timestamp into a better readable one
orig_data = arc3o.prep_time(orig_data)

# year and month of interest
yyyy = 2004
mm = 6

# explain how the monthly chunked file names are built around yyyymm (the ones in␣
↪inputpath),
# example files are called 'assim_SICCI2_50km_echam6_yyyymm_selcode_Arctic.nc' where␣
↪yyyy is the year and mm the month
file_begin = 'assim_SICCI2_50km_echam6_'
file_end = '_selcode_Arctic.nc'

### frequency of interest in GHz (must fit one of the AMSR-E frequencies)
freq_of_int = 6.9

### run the operator
```

(continues on next page)

(continued from previous page)

```
arc3o.satsim_complete_1month(orig_data,              # climate model output, whole time␣
↪series
                         freq_of_int,                # frequency of interest
                         yyyy,mm,                    # year and month of interest
                         inputpath,                  # where to find the monthly␣
↪chunked climate model output
                         outputpath,                 # where to write out the results
                         file_begin,file_end,        # file name as wrapped around␣
↪yyyymm for the monthly chunked files
                         timestep=6,                 # timestep of climate model data␣
↪in hours
                         write_mask='yes',           # 'yes' if you want to compute␣
↪and write out the ice type and season mask, 'no' if you already have a file 'period_
↪masks_assim.nc' in outputpath
                         write_profiles='yes',       # 'yes' if you want to compute␣
↪and write out the profiles, 'no' if you already have monthly chunked files
↪'profiles_for_memls_snowno_yyyymm.nc' and 'profiles_for_memls_snowyes yyyymm.nc' in␣
↪outputpath
                         compute_memls='yes',        # 'yes' if you want to compute␣
↪and write out the cold conditions ice surface brightness temperature, 'no' if you␣
↪already have monthly chunked files 'TB_assim_yyyymm_f.nc' in outputpath
                         e_bias_fyi=0.968,           # factor affecting the␣
↪temperature profiles to bias-correct the brightness temperature (for first-year ice)
                         e_bias_myi=0.968,           # factor affecting the␣
↪temperature profiles to bias-correct the brightness temperature (for multiyear ice)
                         snow_emis=1,                # snow emissivity for periods of␣
↪melting snow
                         snow dens=300.)             # snow density in kg/m3
```

### 1.3.3 Output

**The output of ARC3O is written into several netcdf files to `outputpath`:**

- `'period_masks_assim.nc'`: Masks for ice type and seasons.
- `'profiles_for_memls_snowno_yyyymm.nc'`: Snow-free profiles of ice and snow properties.
- `'profiles_for_memls_snowyes_yyyymm.nc'`: Snow-covered profiles of ice and snow properties.
- `'TB_assim_yyyymm_f.nc'`: Ice surface brightness temperatures (H and V polarization) for grid cells with ice in cold conditions.
- `'TBtot_assim_yyyymm_f.nc'`: Brightness temperatures (H and V polarization) at the top of atmosphere (incl. other seasons than cold conditions and ocean and atmosphere contribution) for all ocean grid cells.

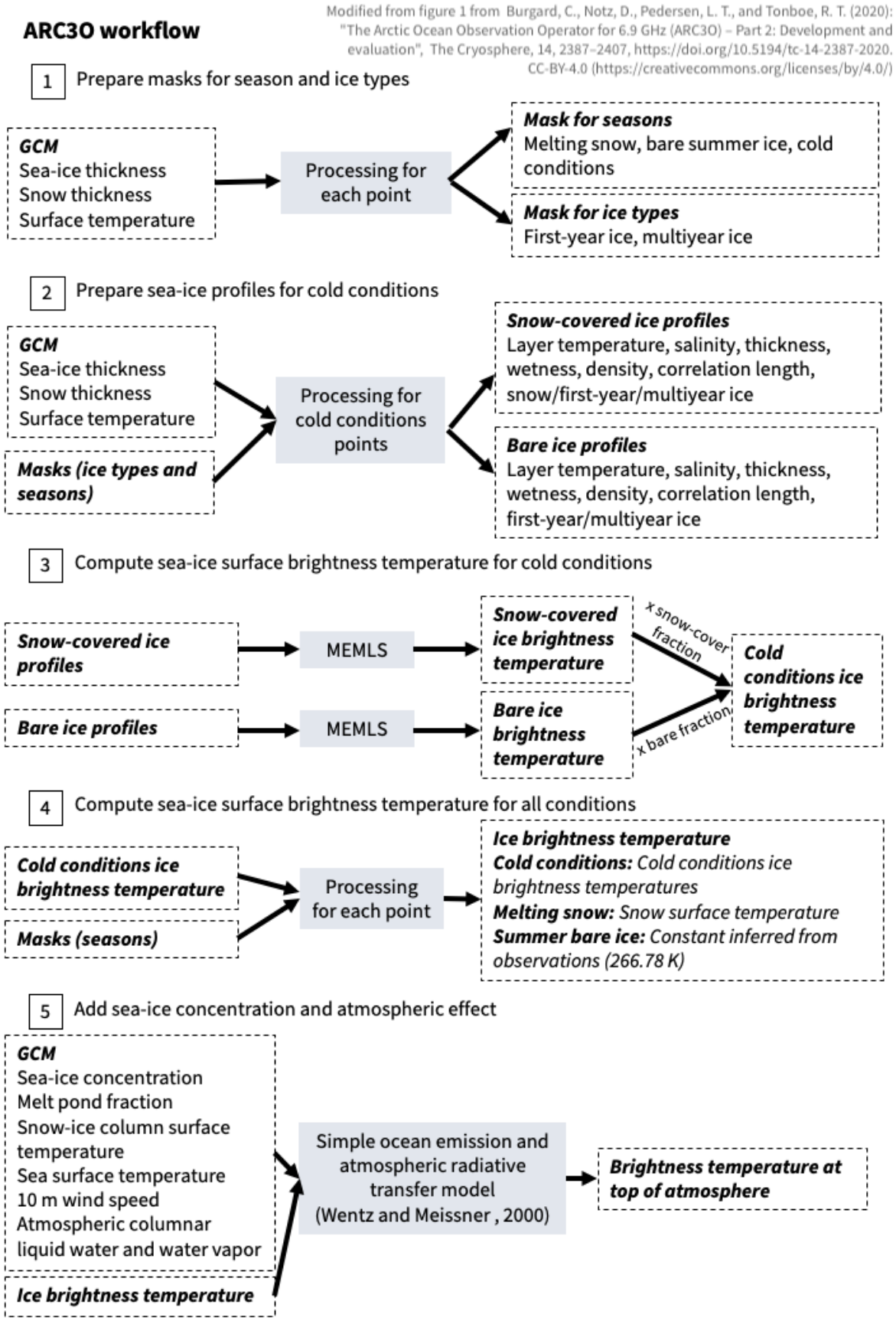
**Note:** Please remain aware that the assumptions used in ARC3O have only been evaluated for the frequency of 6.9 GHz, vertical polarization at the moment! The use for other frequencies and polarizations is at your own risk!

## 1.4 Workflow overview

### 1.4.1 General workflow

ARC3O simulates brightness temperatures from climate model output. It follows the following workflow.

**Note:** Please remain aware that the assumptions used in ARC3O have only been evaluated for the frequency of 6.9 GHz, vertical polarization at the moment! The use for other frequencies and polarizations is at your own risk!


ARC3O workflow
Modified from figure 1 from Burgard, C., Notz, D., Pedersen, L. T., and Tonboe, R. T. (2020): "The Arctic Ocean Observation Operator for 6.9 GHz (ARC3O) – Part 2: Development and evaluation", The Cryosphere, 14, 2387–2407, https://doi.org/10.5194/tc-14-2387-2020.
CC-BY-4.0 (https://creativecommons.org/licenses/by/4.0/)
1 Prepare masks for season and ice types
GCM
Sea-ice thickness
Snow thickness
Surface temperature
Processing for each point
Mask for seasons
Melting snow, bare summer ice, cold conditions
Mask for ice types
First-year ice, multiyear ice
2 Prepare sea-ice profiles for cold conditions
GCM
Sea-ice thickness
Snow thickness
Surface temperature
Masks (ice types and seasons)
Processing for cold conditions points
Snow-covered ice profiles
Layer temperature, salinity, thickness, wetness, density, correlation length, snow/first-year/multiyear ice
Bare ice profiles
Layer temperature, salinity, thickness, wetness, density, correlation length, first-year/multiyear ice
3 Compute sea-ice surface brightness temperature for cold conditions
Snow-covered ice profiles
MEMLS
Snow-covered ice brightness temperature
x snow-cover fraction
Bare ice profiles
MEMLS
Bare ice brightness temperature
x bare fraction
Cold conditions ice brightness temperature
4 Compute sea-ice surface brightness temperature for all conditions
Cold conditions ice brightness temperature
Masks (seasons)
Processing for each point
Ice brightness temperature
Cold conditions: Cold conditions ice brightness temperatures
Melting snow: Snow surface temperature
Summer bare ice: Constant inferred from observations (266.78 K)
5 Add sea-ice concentration and atmospheric effect
GCM
Sea-ice concentration
Melt pond fraction
Snow-ice column surface temperature
Sea surface temperature
10 m wind speed
Atmospheric columnar liquid water and water vapor
Ice brightness temperature
Simple ocean emission and atmospheric radiative transfer model (Wentz and Meissner , 2000)
Brightness temperature at top of atmosphere

### 1.4.2 Step 1: Prepare masks for season and ice types

In [Burgard et al., 2020a] and [Burgard et al., 2020b], we show that different approaches are needed to simulate the sea-ice surface brightness temperatures depending on the ice type (first-year or multiyea ice) and on the "season" (cold conditions, melting snow, bare ice in summer). This is why, in a first step, ARC3O creates masks to divide the grid cells into those criteria. This is done in the main function with *arc3o.core_functions.prep_mask()*.

To divide the grid cells into different ice types, the function *arc3o.mask_functions.ice_type_wholeArctic()* uses the time series of the sea-ice thickness to determine if there was ice in the grid cell in the year preceding each timestep, thus dividing into open water, first-year and multiyear ice.

To divide the grid cells into the different seasons, the function *arc3o.mask_functions.define_periods()* uses information about sea-ice thickness, snow thickness, and surface temperature to identify periods of melting snow and periods of bare ice in summer. All other grid cells are defined as "cold conditions".

The resulting masks are written to `period_masks_assim.nc` in `outputpath`.

### 1.4.3 Step 2: Prepare sea-ice profiles for cold conditions

In a second step, we prepare the climate model output data for the simulation of the ice surface brightness temperature for cold conditions through the Microwave Emission Model for Layered Snowpacks (MEMLS, [Wiesmann & Matzler, 1999] and [Tonboe et al., 2006]). This is done in the main function with *arc3o.core_functions.prep_prof()*.

Based on the climate model sea-ice thickness, snow thickness, and the snow/ice surface temperature, combined with the masks described in *Step 1: Prepare masks for season and ice types*, ARC3O adds the dimension *layer_number* to the lat-lon-time arrays. This way, it prepares two sets of profiles (ten ice layers and one snow layer), one set for snow-covered ice and one set for bare ice (see *arc3o.profile_functions.create_profiles()*). These profiles describe the layer:

- temperature *arc3o.profile_functions.build_temp_profile()*
- salinity *arc3o.profile_functions.build_salinity_profile()*
- thickness *arc3o.profile_functions.build_thickness_profile()*
- wetness *arc3o.profile_functions.build_wetness_profile()*
- density *arc3o.profile_functions.build_density_profile()*
- correlation length *arc3o.profile_functions.build_corrlen_profile()*
- type `arc3o.profile_functions.build_sisn_profile()`

The resulting profiles are written to `profiles_for_memls_snowno_yyyymm.nc` and `profiles_for_memls_snowyes_yyyymm.nc` in `outputpath`.

### 1.4.4 Step 3: Compute sea-ice surface brightness temperature for cold conditions

In a third step, the two sets of profiles prepared in *Step 2: Prepare sea-ice profiles for cold conditions* are used to simulate the brightness temperature at the surface of the snow and ice column, through the Microwave Emission Model for Layered Snowpacks (MEMLS, [Wiesmann & Matzler, 1999] and [Tonboe et al., 2006]). This results in two sets of brightness temperatures (one set for snow-covered ice and one set for bare ice). They are then combined, weighted by the bare-ice fraction given by the climate model output. This step is done in the main function with *arc3o.core_functions.memls_module_general()*.

The MEMLS simulation is conducted through calling the function *arc3o.core_functions.run_memls_2D()*. More details about the detailed steps of the brightness temperature simulation can be found in *arc3o.memls_functions_2D.memls_2D_1freq()* and its documentation of *memls_functions_2D*.

The resulting ice surface brightness temperatures for cold conditions are written to `TB_assim_yyyymm_f.nc` in `outputpath`, where $f$ is the rounded frequency of interest.

### 1.4.5 Step 4: Compute sea-ice surface brightness temperature for all conditions

In a fourth step, the ice surface brightness temperatures are extended to other seasons than "cold conditions", i.e. to the seasons "melting snow conditions" and "bare ice in summer conditions". This step is done in the main function with *arc3o.core_functions.compute_TBVice()* and *arc3o.core_functions.compute_emisV()*.

---

**Note:** Please remain aware that, currently, the brightness temperatures of "melting snow conditions" and "bare ice in summer conditions" are set to the values for 6.9 GHz, vertical polarization, **for all frequencies and polarizations**! This is because we concentrated our effort on 6.9 GHz, vertical polarization. You are welcome to try out new things though! :)

---

### 1.4.6 Step 5: Add sea-ice concentration and atmospheric effect

In a fifth step, the contribution of the ocean and melt ponds to the surface brightness temperature (through the sea-ice concentration the melt-pond fraction, the snow/ice surface temperature, the sea surface temperature and wind speed) is included. The resulting surface brightness temperature (combining ice and open water brightness temperatures) is used as an input as a surface brightness temperature for a simple atmospheric radiative transfer model. This radiative transfer model additionally requires the atmospheric columnar liquid water and water vapor. Both adding the oceanic contribution and adding the atmospheric contribution are done in the main function with *arc3o.core_functions.amsr()*, based on the geophysical model described in [Wentz & Meissner, 2000].

The resulting top-of-the-atmosphere brightness temperatures for all grid cells are written to `TBtot_assim_yyyymm_f.nc` in `outputpath`, where $f$ is the rounded frequency of interest.

## 1.5 API References

### 1.5.1 Package main functions

`arc3o.`**`new_outputpath`**(*log*, *outputpath*, *existing*)

Create new folders to organize experiments

This function helps to organize different experiments

**Parameters**

- **`log`** (*str*) – `'yes'` if you are starting a new experiment and want a new folder for it, `'no'` if you want to continue an experiment and want to work in an existing folder
- **`outputpath`** (*str*) – path where you want the folder to be/the folder is
- **`existing`** (*str*) – if your folder exists already, just write the name here, if not you can write `'default'`

**Returns** **new_dir** – prints the directory you are working in now, if new it will have created the directory

**Return type** str

`arc3o.`**`prep_time`**(*input_data*)

This function transforms the date format given by MPI-ESM into a proper date format for xarray

**Parameters** **input_data** (*xarray.Dataset*) – the MPI-ESM xarray.Dataset with the original date format

**Returns** **input_data** – the MPI-ESM xarray.Dataset with the new date format

**Return type** xarray.Dataset

arc3o.**satsim_complete_1month**(*orig_data*, *freq_of_int*, *yyyy*, *mm*, *inputpath*, *outputpath*, *file_begin*, *file_end*, *timestep=6*, *write_mask='yes'*, *write_profiles='yes'*, *compute_memls='yes'*, *e_bias_fyi=0.968*, *e_bias_myi=0.968*, *snow_emis=1*, *snow_dens=300.0*)

Compute top-of-atmosphere brightness temperature for one single month.

This function is a subset of the full observation operator and should be used as ARC3O main function if you only want to simulate brightness temperatures for one given month.

**Parameters**

- **orig_data** – MPI-ESM data all years merged together
- **freq_of_int** (*float*) – freguency in GHz
- **yyyy** (*int*) – year of interest (format yyyy)
- **mm** (*int*) – month of interest (format mm)
- **inputpath** (*str*) – path where to find the MPI-ESM data chunked in months
- **outputpath** (*str*) – path where files should be written
- **file_begin** (*str*) – how the MPI-ESM data filename starts (before date)
- **file_end** (*str*) – how the MPI-ESM data filename ends (after date)
- **timestep** (*int, optional*) – timestep of data in hours; *default is 6*
- **write_mask** (*str, optional*) – `'yes'` if you want to write to a file, `'no'` if you already have written it to a file before; *default is 'yes'*
- **write_profiles** (*str, optional*) – `'yes'` if you want to compute the property profiles and write them to files, `'no'` if you if you have already written them out in `outputpath` and nothing has changed; *default is 'yes'*
- **compute_memls** (*str, optional*) – `'yes'` if you want to feed profiles to MEMLS and write the result out, `'no'` if you have already written them out and nothing has changed; *default is 'yes'*
- **e_bias_fyi** (*float, optional*) – tuning parameter for the first-year ice temperature profile to bias-correct the MEMLS result; *default is 0.968*
- **e_bias_myi** (*float, optional*) – tuning parameter for the multiyear ice temperature profile to bias-correct the MEMLS result; *default is 0.968*
- **snow_emis** (*float, optional*) – assign the snow emissivity to `1` or `np.nan` for melting snow periods; *default is 1*
- **snow_dens** (*float, optional*) – constant snow density to use; *default is 300.*

**Returns**

- **'period_masks_assim.nc'** (*netcdf file*) – Masks for ice type and seasons. Written if *write_mask* is `'yes'`. Can be found in `outputpath`.
- **'profiles_for_memls_snowno_yyyymm.nc'** (*netcdf file*) – Snow-free profiles of ice and snow properties. Written if *write_profiles* is `'yes'`. Can be found in `outputpath`.

- **'profiles_for_memls_snowyes_yyyymm.nc'** (*netcdf file*) – Snow-covered profiles of ice and snow properties. Written if *write_profiles* is `'yes'`. Can be found in `outputpath`.
- **'TB_assim_yyyymm_f.nc'** (*netcdf file*) – Ice surface brightness temperatures for cold conditions. Written if *compute_memls* is `'yes'`. Can be found in `outputpath`.
- **'TBtot_assim_yyyymm_f.nc'** (*netcdf file*) – Brightness temperatures at the top of atmosphere. Can be found in `outputpath`.

### Notes

`f` in the filenames is the rounded `freq_of_int`.

---

**Note:** Please remain aware that the results from ARC3O have only been evaluated for the frequency of **6.9 GHz, vertical polarization** at the moment! The use for other frequencies and polarizations is at your own risk! Especially, this function will also give out results for H polarization. However, they have not been evaluated yet!

---

arc3o.**satsim_complete_parallel**(*orig_data*, *freq_of_int*, *start_year*, *end_year*, *inputpath*, *outputpath*, *file_begin*, *file_end*, *timestep=6*, *write_mask='yes'*, *write_profiles='yes'*, *compute_memls='yes'*, *e_bias_fyi=0.968*, *e_bias_myi=0.968*, *snow_emis=1*, *snow_dens=300.0*, *pool_nb=12*)

Compute top-of-atmosphere brightness temperature over a long time period.

This function is the full observation operator and should be used as ARC3O main function.

**Parameters**

- **orig_data** (*xarray.Dataset*) – MPI-ESM data all years merged together
- **freq_of_int** (*float*) – freguency in GHz
- **start_year** (*int*) – first year of interest (format yyyy)
- **end_year** (*int*) – last year of interest (format yyyy)
- **inputpath** (*str*) – path where to find the MPI-ESM data chunked in months
- **outputpath** (*str*) – path where files should be written
- **file_begin** (*str*) – how the MPI-ESM data filename starts (before date)
- **file_end** (*str*) – how the MPI-ESM data filename ends (after date)
- **timestep** (*int, optional*) – timestep of data in hours; *default is 6*
- **write_mask** (*str, optional*) – `'yes'` if you want to write to a file, `'no'` if you already have written it to a file before; *default is 'yes'*
- **write_profiles** (*str, optional*) – `'yes'` if you want to compute the property profiles and write them to files, `'no'` if you if you have already written them out in `outputpath` and nothing has changed; *default is 'yes'*
- **compute_memls** (*str, optional*) – `'yes'` if you want to feed profiles to MEMLS and write the result out, `'no'` if you have already written them out and nothing has changed; *default is 'yes'*
- **e_bias_fyi** (*float, optional*) – tuning parameter for the first-year ice temperature profile to bias-correct the MEMLS result; *default is 0.968*

- **e_bias_myi** (*float, optional*) – tuning parameter for the multiyear ice temperature profile to bias-correct the MEMLS result; *default is 0.968*
- **snow_emis** (*float, optional*) – assign the snow emissivity to `1` or `np.nan` for melting snow periods; *default is 1*
- **snow_dens** (*float, optional*) – constant snow density to use; *default is 300.*
- **pool_nb** (*int, optional*) – number of parallel pool workers to compute several months parallelly, *default is 12*

**Returns**

- **'period_masks_assim.nc'** (*netcdf file*) – Masks for ice type and seasons. Written if *write_mask* is `'yes'`. Can be found in `outputpath`.
- **'profiles_for_memls_snowno_yyyymm.nc'** (*netcdf files*) – Chunked by months. Snow-free profiles of ice and snow properties. Written if *write_profiles* is `'yes'`. Can be found in `outputpath`.
- **'profiles_for_memls_snowyes_yyyymm.nc'** (*netcdf files*) – Chunked by months. Snow-covered profiles of ice and snow properties. Written if *write_profiles* is `'yes'`. Can be found in `outputpath`.
- **'TB_assim_yyyymm_f.nc'** (*netcdf files*) – Chunked by months. Ice surface brightness temperatures for cold conditions. Written if *write_profiles* is `'yes'`. Can be found in `outputpath`.
- **'TBtot_assim_yyyymm_f.nc'** (*netcdf files*) – Chunked by months. Brightness temperatures at the top of atmosphere. Can be found in `outputpath`.

### Notes

`f` in the filenames is the rounded `freq_of_int`.

---

**Note:** Please remain aware that the results from ARC3O have only been evaluated for the frequency of **6.9 GHz, vertical polarization** at the moment! The use for other frequencies and polarizations is at your own risk! Especially, this function will also give out results for H polarization. However, they have not been evaluated yet!

---

## 1.5.2 core_functions

`arc3o.core_functions.`**`TB_tot`**(*input_data*, *info_ds*, *memls_output*, *freq*, *snow_emis*, *surf_temp*)

Compute the brightness temperature at top of the atmosphere

This function computes the brightness temperature at top of the atmosphere.

**Parameters**

- **input_data** (*xarray.Dataset*) – MPI-ESM data for period of interest (yyyymm)
- **info_ds** (*xarray.Dataset*) – dataset of mask for seasons and ice types for period of interest (yyyymm)
- **memls_output** (*xarray.Dataset*) – dataset with MEMLS output
- **freq** (*float*) – freguency in GHz

- **snow_emis** (*float*) – assign the snow emissivity to `1` or `np.nan` for melting snow periods
- **surf_temp** (*xarray.DataArray*) – snow surface temperature to use for the brightness temperature if snow emissivity is 1

**Returns** **ds** – dataset containing brightness temperatures at both polarizations at top of the atmosphere in K

**Return type** xarray.Dataset

### Notes

**Note:** Please remain aware that the results from ARC3O have only been evaluated for the frequency of **6.9 GHz, vertical polarization** at the moment! The use for other frequencies and polarizations is at your own risk! Especially, this function will also give out results for H polarization. However, they have not been evaluated yet!

arc3o.core_functions.**amsr**(*V*, *W*, *L*, *Ta*, *Ts*, *TBV_ice*, *TBH_ice*, *e_icev*, *e_iceh*, *c_ice*, *freq*, *slm*, *mpf*)

Add the atmospheric and oceanic contribution to the ice surface brightness temperature.

This function adds the atmospheric and oceanic contribution to the brightness temperature to the ice surface brightness temperature, resulting in a brightness temperature for the top of the atmosphere. This function is based on equations given in [Wentz & Meissner, 2000] and is tailored to AMSR2 frequencies. It was extended by C. Burgard to include melt ponds for the use in ARC3O.

**Parameters**

- **V** (*xarray.DataArray*) – columnar water vapor in mm
- **W** (*xarray.DataArray*) – windspeed over water in m/s
- **L** (*xarray.DataArray*) – columnar cloud liquid water in mm
- **Ta** (*xarray.DataArray*) – ice surface temperature in K
- **Ts** (*xarray.DataArray*) – sea surface temperature in K
- **TBV_ice** (*xarray.DataArray*) – brightness temperature ice surface vertical polarization in K
- **TBH_ice** (*xarray.DataArray*) – brightness temperature ice surface horizontal polarization in K
- **e_icev** (*xarray.DataArray*) – ice emissivity vertical polarization
- **e_iceh** (*xarray.DataArray*) – ice emissivity horizontal polarization
- **c_ice** (*xarray.DataArray*) – ice concentration between 0 and 1
- **freq** (*float*) – AMSR frequency of interest in GHz, one of the following: 6.9, 10.7, 18.7, 23.8, 36.5, 50.3, 52.8, 89.0
- **slm** (*xarray.DataArray*) – sea-land mask (0 for ocean, 1 for land)
- **mpf** (*xarray.DataArray*) – melt pond fraction between 0 and 1

**Returns**

- **TBH** (*xarray.DataArray*) – brightness temperature, horizontal polarization, at top of the atmosphere in K

- **TBV** (*xarray.DataArray*) – brightness temperature, vertical polarization, at top of the atmosphere in K

arc3o.core_functions.**comp_F**(*m1*, *m2*, *W*, *pol*)
Compute the empirical term for any residual non-linear wind variations

This function is necessary for the atmospheric and ocean contribution in *amsr()*. It computes the empirical term for any residual non-linear wind variations. It is based on Eq. 60 of [Wentz & Meissner, 2000]. Data for 6.9 GHz was not available so we use the same values as for 10.7 GHz

**Parameters**

- **m1** (*xarray.DataArray*) – coefficients given by [Wentz & Meissner, 2000]_
- **m2** (*xarray.DataArray*) – coefficients given by [Wentz & Meissner, 2000]_
- **W** (*xarray.DataArray*) – wind speed in m/s
- **pol** (*str*) – `'V'` for vertical polarization, `'H'` for horizontal polarization

**Returns** **F** – empirical term for any residual non-linear wind variations

**Return type** xarray.DataArray

arc3o.core_functions.**compute_TBVice**(*info_ds*, *TB*, *pol*, *snow_emis*, *surf_temp*)
Produce ice surface brightness temperature for all seasons.

This function combines ice surface brightness temperatures computed through MEMLS (i.e. for cold conditions) with ice surface brightness temperatures from other seasons.

**Parameters**

- **info_ds** (*xarray.Dataset*) – dataset containing masks about seasons and ice types
- **TB** (*xarray.DataArray*) – array of brightness temperatures in one polarization computed by MEMLS
- **pol** (*str*) – `'V'` for vertical polarization, `'H'` for horizontal polarization
- **snow_emis** (*float*) – assign the snow emissivity to `1` or `np.nan` for melting snow periods
- **surf_temp** (*xarray.DataArray*) – snow surface temperature to use for the brightness temperature if snow emissivity is 1

**Returns** **TBice** – array of ice surface brightness temperature in the given polarization for all seasons

**Return type** xarray.DataArray

### Notes

**Note:** Please remain aware that the results from ARC3O have only been evaluated for the frequency of **6.9 GHz, vertical polarization** at the moment! The use for other frequencies and polarizations is at your own risk! Especially, this function will also give out results for H polarization. However, they have not been evaluated yet!

arc3o.core_functions.**compute_emisV**(*info_ds*, *eice*, *pol*, *snow_emis*)
Produce emissivities for all seasons.

This function combines ice surface emissivities computed through MEMLS (i.e. for cold conditions) with ice surface emissivities from other seasons.

**Parameters**

- **info_ds** (*xarray.Dataset*) – dataset containing masks about seasons and ice types
- **eice** (*xarray.DataArray*) – array of emissivity in one polarization computed by MEMLS
- **pol** (*str*) – `'V'` for vertical polarization, `'H'` for horizontal polarization
- **snow_emis** (*float*) – assign the snow emissivity to `1` or `np.nan` for melting snow periods

**Returns** **e_ice** – array of ice emissivities in the given polarization

**Return type** xarray.DataArray

### Notes

---

**Note:** Please remain aware that the results from ARC3O have only been evaluated for the frequency of **6.9 GHz, vertical polarization** at the moment! The use for other frequencies and polarizations is at your own risk! Especially, this function will also give out results for H polarization. However, they have not been evaluated yet!

---

arc3o.core_functions.**compute_parallel**(*start_year*, *end_year*, *freq_of_int*, *e_bias_fyi*, *e_bias_myi*, *snow_emis*, *snow_dens*, *inputpath*, *outputpath*, *file_begin*, *file_end*, *info_ds*, *write_profiles*, *compute_memls*, *pool_nb*)

Parallelize the brightness temperature simulation.

This function parallelizes the brightness temperature simulation for the different months of a given year.

**Parameters**

- **start_year** (*int*) – first year of interest (format yyyy)
- **end_year** (*int*) – last year of interest (format yyyy)
- **freq_of_int** (*float*) – freguency in GHz
- **e_bias_fyi** (*float*) – tuning parameter for the first-year ice temperature profile to influence the MEMLS result
- **e_bias_myi** (*float*) – tuning parameter for the multiyear ice temperature profile to influence the MEMLS result
- **snow_emis** – assign the snow emissivity to `1` or `np.nan` for melting snow periods
- **snow_dens** (*float*) – constant snow density to use
- **inputpath** (*str*) – path where to find the MPI-ESM data chunked in month
- **outputpath** (*str*) – path where files should be written
- **file_begin** (*str*) – how the MPI-ESM data filename starts (before date)
- **file_end** (*str*) – how the MPI-ESM data filename ends (after date)
- **info_ds** (*xarray.Dataset*) – dataset of mask for seasons and ice types
- **write_profiles** (*str*) – `'yes'` if you want to compute the property profiles and write them to files, `'no'` if you if you have already written them out in `outputpath` and nothing has changed

- **compute_memls** (*str*) – `'yes'` if you want to feed profiles to MEMLS and write the result out, `'no'` if you have already written them out and nothing has changed
- **pool_nb** (*int, optional*) – number of parallel pool workers to compute several months parallelly

**Returns**

- **'profiles_for_memls_snowno_yyyymm.nc'** (*netcdf files*) – Chunked by months. Snow-free profiles of ice and snow properties. Written if *write_profiles* is `'yes'`. Can be found in `outputpath`.
- **'profiles_for_memls_snowyes_yyyymm.nc'** (*netcdf files*) – Chunked by months. Snow-covered profiles of ice and snow properties. Written if *write_profiles* is `'yes'`. Can be found in `outputpath`.
- **'TB_assim_yyyymm_f.nc'** (*netcdf files*) – Chunked by months. Ice surface brightness temperatures for cold conditions. Written if *write_profiles* is `'yes'`. Can be found in `outputpath`.
- **'TBtot_assim_yyyymm_f.nc'** (*netcdf files*) – Chunked by months. Brightness temperatures at the top of atmosphere. Can be found in `outputpath`.

### Notes

`f` in the filenames is the rounded `freq_of_int`.

---

**Note:** Please remain aware that the results from ARC3O have only been evaluated for the frequency of **6.9 GHz, vertical polarization** at the moment! The use for other frequencies and polarizations is at your own risk! Especially, this function will also give out results for H polarization. However, they have not been evaluated yet!

---

arc3o.core_functions.**memls_module_general**(*yy*, *freq_of_int*, *profiles_yes*, *profiles_no*, *snifrac*, *barefrac*, *outputpath*, *compute_memls*)

Combine the brightness temperatures of bare and snow-covered ice

This function wraps around MEMLS to combine the brightness temperatures of bare and snow-covered ice.

**Parameters**

- **yy** (*int*) – actually stands for yyyymm, defines the year and month we are looking at
- **freq_of_int** (*float*) – frequency in GHz
- **profiles_yes** (*xarray.Dataset*) – Dataset containing property profiles for snow-covered ice
- **profiles_no** (*xarray.Dataset*) – Dataset containing property profiles for bare ice
- **snifrac** (*xarray.DataArray*) – snow fraction given by MPI-ESM data, already month of interest selected
- **barefrac** (*xarray.DataArray*) – bare ice fraction given by MPI-ESM data, already month of interest selected
- **outputpath** (*str*) – path where you want the file to be written
- **compute_memls** (*str*) – `'yes'` if you want to feed profiles to MEMLS and write the result out, `'no'` if you do not want to run MEMLS again and use previous output

**Returns**

- **ds1** (*xarray.Dataset*) – Dataset containing the weighted ice brightness temperatures and emissivities (`TBH`, `TBV`, `eh`, `ev`).
- **'TB_assim_yyyymm_f.nc'** (*netcdf files*) – Chunked by months. Ice surface brightness temperatures for cold conditions. Written if *write_profiles* is `'yes'`. Can be found in `outputpath`.

### Notes

`f` in the filename is the rounded value of `freq_of_int`.

---

**Note:** Please remain aware that the results from ARC3O have only been evaluated for the frequency of **6.9 GHz, vertical polarization** at the moment! The use for other frequencies and polarizations is at your own risk! Especially, this function will also give out results for H polarization. However, they have not been evaluated yet!

---

`arc3o.core_functions.`**`new_outputpath`**(*log*, *outputpath*, *existing*)

Create new folders to organize experiments

This function helps to organize different experiments

**Parameters**

- **`log`** (*`str`*) – `'yes'` if you are starting a new experiment and want a new folder for it, `'no'` if you want to continue an experiment and want to work in an existing folder
- **`outputpath`** (*`str`*) – path where you want the folder to be/the folder is
- **`existing`** (*`str`*) – if your folder exists already, just write the name here, if not you can write `'default'`

**Returns** **new_dir** – prints the directory you are working in now, if new it will have created the directory

**Return type** str

`arc3o.core_functions.`**`prep_mask`**(*input_data*, *write_mask*, *outputpath*, *timestep*)

Prepare or read out the mask for the seasons and ice types This function prepares or reads out the mask for the seasons and ice types

**Parameters**

- **`input_data`** (*`xarray.Dataset`*) – the MPI-ESM data over the whole time period of interest
- **`write_mask`** (*`str`*) – `'yes'` if you want to write to a file, `'no'` if you already have written it to a file before
- **`outputpath`** (*`str`*) – path where you want the file to be written
- **`timestep`** (*`int`*) – timestep of data in hours

**Returns**

- **info_ds** (*xarray.Dataset*) – dataset with masks for seasons and ice types
- **'period_masks_assim.nc'** (*netcdf file*) – File containing `info_ds` in `outputpath`. Written if *write_mask* is `'yes'`. Can be found in `outputpath`.

arc3o.core_functions.**prep_prof**(*input_data*, *write_profiles*, *info_ds*, *outputpath*, *yy*, *e_bias_fyi*, *e_bias_myi*, *snow_dens*)

Prepare or read out the property profiles for use in MEMLS

This function prepares or reads out the property profiles for use in MEMLS.

**Parameters**

- **input_data** (*xarray.Dataset*) – the MPI-ESM data over the whole time period of interest
- **write_profiles** (*str*) – `'yes'` if you want to write to a file, `'no'` if you already have written it to a file in `outputpaht` before
- **info_ds** (*xarray.Dataset*) – dataset with masks for seasons and ice types over whole time period of interest
- **outputpath** (*str*) – path where you want the file to be written
- **yy** (*int*) – actually stands for yyyymm, defines the year and month we are looking at
- **e_bias_fyi** (*float*) – tuning parameters for the first-year ice temperature profile to influence the MEMLS result
- **e_bias_myi** (*float*) – tuning parameters for the multiyear ice temperature profile to influence the MEMLS result
- **snow_dens** (*float*) – snow density (constant)

**Returns**

- **profiles1** (*xarray.Dataset*) – profiles assuming snow-covered ice
- **profiles2** (*xarray.Dataset*) – profiles assuming bare ice
- **'profiles_for_memls_snowno_yyyymm.nc'** (*netcdf file*) – Snow-free profiles of ice and snow properties. Written if *write_profiles* is `'yes'`. Can be found in `outputpath`.
- **'profiles_for_memls_snowyes_yyyymm.nc'** (*netcdf file*) – Snow-covered profiles of ice and snow properties. Written if *write_profiles* is `'yes'`. Can be found in `outputpath`.

arc3o.core_functions.**prep_time**(*input_data*)

This function transforms the date format given by MPI-ESM into a proper date format for xarray

**Parameters** **input_data** (*xarray.Dataset*) – the MPI-ESM xarray.Dataset with the original date format

**Returns** **input_data** – the MPI-ESM xarray.Dataset with the new date format

**Return type** xarray.Dataset

arc3o.core_functions.**run_memls_2D**(*profiles*, *freq*)

Run MEMLS

This function calls MEMLS for multi-dimensional files.

**Parameters**

- **profiles** (*xarray.Dataset*) – either the snow covered or bare ice property profiles
- **freq** (*float*) – frequency in GHz

**Returns** **ds** – Dataset containing TBH, TBV, emissivity H, emissivity V

**Return type** xarray.Dataset

arc3o.core_functions.**satsim_complete_1month**(*orig_data*, *freq_of_int*, *yyyy*, *mm*, *inputpath*, *outputpath*, *file_begin*, *file_end*, *timestep=6*, *write_mask='yes'*, *write_profiles='yes'*, *compute_memls='yes'*, *e_bias_fyi=0.968*, *e_bias_myi=0.968*, *snow_emis=1*, *snow_dens=300.0*)

Compute top-of-atmosphere brightness temperature for one single month.

This function is a subset of the full observation operator and should be used as ARC3O main function if you only want to simulate brightness temperatures for one given month.

**Parameters**

- **orig_data** – MPI-ESM data all years merged together
- **freq_of_int** (*float*) – freguency in GHz
- **yyyy** (*int*) – year of interest (format yyyy)
- **mm** (*int*) – month of interest (format mm)
- **inputpath** (*str*) – path where to find the MPI-ESM data chunked in months
- **outputpath** (*str*) – path where files should be written
- **file_begin** (*str*) – how the MPI-ESM data filename starts (before date)
- **file_end** (*str*) – how the MPI-ESM data filename ends (after date)
- **timestep** (*int, optional*) – timestep of data in hours; *default is 6*
- **write_mask** (*str, optional*) – `'yes'` if you want to write to a file, `'no'` if you already have written it to a file before; *default is 'yes'*
- **write_profiles** (*str, optional*) – `'yes'` if you want to compute the property profiles and write them to files, `'no'` if you if you have already written them out in `outputpath` and nothing has changed; *default is 'yes'*
- **compute_memls** (*str, optional*) – `'yes'` if you want to feed profiles to MEMLS and write the result out, `'no'` if you have already written them out and nothing has changed; *default is 'yes'*
- **e_bias_fyi** (*float, optional*) – tuning parameter for the first-year ice temperature profile to bias-correct the MEMLS result; *default is 0.968*
- **e_bias_myi** (*float, optional*) – tuning parameter for the multiyear ice temperature profile to bias-correct the MEMLS result; *default is 0.968*
- **snow_emis** (*float, optional*) – assign the snow emissivity to `1` or `np.nan` for melting snow periods; *default is 1*
- **snow_dens** (*float, optional*) – constant snow density to use; *default is 300.*

**Returns**

- **'period_masks_assim.nc'** (*netcdf file*) – Masks for ice type and seasons. Written if *write_mask* is `'yes'`. Can be found in `outputpath`.
- **'profiles_for_memls_snowno_yyyymm.nc'** (*netcdf file*) – Snow-free profiles of ice and snow properties. Written if *write_profiles* is `'yes'`. Can be found in `outputpath`.
- **'profiles_for_memls_snowyes_yyyymm.nc'** (*netcdf file*) – Snow-covered profiles of ice and snow properties. Written if *write_profiles* is `'yes'`. Can be found in `outputpath`.
- **'TB_assim_yyyymm_f.nc'** (*netcdf file*) – Ice surface brightness temperatures for cold conditions. Written if *compute_memls* is `'yes'`. Can be found in `outputpath`.

- **'TBtot_assim_yyyymm_f.nc'** (*netcdf file*) – Brightness temperatures at the top of atmosphere. Can be found in `outputpath`.

### Notes

`f` in the filenames is the rounded `freq_of_int`.

---

**Note:** Please remain aware that the results from ARC3O have only been evaluated for the frequency of **6.9 GHz, vertical polarization** at the moment! The use for other frequencies and polarizations is at your own risk! Especially, this function will also give out results for H polarization. However, they have not been evaluated yet!

---

`arc3o.core_functions.`**`satsim_complete_parallel`**(*orig_data*, *freq_of_int*, *start_year*, *end_year*, *inputpath*, *outputpath*, *file_begin*, *file_end*, *timestep=6*, *write_mask='yes'*, *write_profiles='yes'*, *compute_memls='yes'*, *e_bias_fyi=0.968*, *e_bias_myi=0.968*, *snow_emis=1*, *snow_dens=300.0*, *pool_nb=12*)

Compute top-of-atmosphere brightness temperature over a long time period.

This function is the full observation operator and should be used as ARC3O main function.

**Parameters**

- **`orig_data`** (*`xarray.Dataset`*) – MPI-ESM data all years merged together
- **`freq_of_int`** (*`float`*) – freguency in GHz
- **`start_year`** (*`int`*) – first year of interest (format yyyy)
- **`end_year`** (*`int`*) – last year of interest (format yyyy)
- **`inputpath`** (*`str`*) – path where to find the MPI-ESM data chunked in months
- **`outputpath`** (*`str`*) – path where files should be written
- **`file_begin`** (*`str`*) – how the MPI-ESM data filename starts (before date)
- **`file_end`** (*`str`*) – how the MPI-ESM data filename ends (after date)
- **`timestep`** (*`int, optional`*) – timestep of data in hours; *default is 6*
- **`write_mask`** (*`str, optional`*) – `'yes'` if you want to write to a file, `'no'` if you already have written it to a file before; *default is 'yes'*
- **`write_profiles`** (*`str, optional`*) – `'yes'` if you want to compute the property profiles and write them to files, `'no'` if you if you have already written them out in `outputpath` and nothing has changed; *default is 'yes'*
- **`compute_memls`** (*`str, optional`*) – `'yes'` if you want to feed profiles to MEMLS and write the result out, `'no'` if you have already written them out and nothing has changed; *default is 'yes'*
- **`e_bias_fyi`** (*`float, optional`*) – tuning parameter for the first-year ice temperature profile to bias-correct the MEMLS result; *default is 0.968*
- **`e_bias_myi`** (*`float, optional`*) – tuning parameter for the multiyear ice temperature profile to bias-correct the MEMLS result; *default is 0.968*

- **snow_emis** (*float, optional*) – assign the snow emissivity to `1` or `np.nan` for melting snow periods; *default is 1*
- **snow_dens** (*float, optional*) – constant snow density to use; *default is 300.*
- **pool_nb** (*int, optional*) – number of parallel pool workers to compute several months parallelly, *default is 12*

**Returns**

- **'period_masks_assim.nc'** (*netcdf file*) – Masks for ice type and seasons. Written if *write_mask* is `'yes'`. Can be found in `outputpath`.
- **'profiles_for_memls_snowno_yyyymm.nc'** (*netcdf files*) – Chunked by months. Snow-free profiles of ice and snow properties. Written if *write_profiles* is `'yes'`. Can be found in `outputpath`.
- **'profiles_for_memls_snowyes_yyyymm.nc'** (*netcdf files*) – Chunked by months. Snow-covered profiles of ice and snow properties. Written if *write_profiles* is `'yes'`. Can be found in `outputpath`.
- **'TB_assim_yyyymm_f.nc'** (*netcdf files*) – Chunked by months. Ice surface brightness temperatures for cold conditions. Written if *write_profiles* is `'yes'`. Can be found in `outputpath`.
- **'TBtot_assim_yyyymm_f.nc'** (*netcdf files*) – Chunked by months. Brightness temperatures at the top of atmosphere. Can be found in `outputpath`.

### Notes

`f` in the filenames is the rounded `freq_of_int`.

---

**Note:** Please remain aware that the results from ARC3O have only been evaluated for the frequency of **6.9 GHz, vertical polarization** at the moment! The use for other frequencies and polarizations is at your own risk! Especially, this function will also give out results for H polarization. However, they have not been evaluated yet!

---

`arc3o.core_functions.`**satsim_loop**(*file*, *yy*, *mm*, *info_ds*, *freq_of_int*, *e_bias_fyi*, *e_bias_myi*, *outputpath*, *write_profiles*, *compute_memls*, *snow_emis*, *snow_dens*)

Simulate the brightness temperature of an Arctic Ocean surface.

This function combines all necessary functions for the brightness temperature simulation.

**Parameters**

- **file** (*str*) – path to file containing MPI-ESM data
- **yy** (*int*) – year (format yyyy)
- **mm** (*int*) – month (format mm)
- **info_ds** (*xarray.Dataset*) – dataset of mask for seasons and ice types
- **freq_of_int** (*float*) – freguency in GHz
- **e_bias_fyi** (*float*) – tuning parameter for the first-year ice temperature profile to influence the MEMLS result
- **e_bias_myi** (*float*) – tuning parameter for the multiyear ice temperature profile to influence the MEMLS result

- **outputpath** (*str*) – path where files should be written
- **write_profiles** (*str*) – `'yes'` if you want to write to a file, `'no'` if you already have written it to a file in `outputpath` before
- **compute_memls** (*str*) – `'yes'` if you want to feed profiles to MEMLS and write the result out, `'no'` if you do not want to run MEMLS again and use previous output
- **snow_emis** (*float*) – assign the snow emissivity to `1` or `np.nan` for melting snow periods
- **snow_dens** (*float*) – constant snow density to use

**Returns**

- **ds_TB** (*xarray.Dataset*) – brightness temperatures at both polarizations at top of the atmosphere in K
- **'profiles_for_memls_snowno_yyyymm.nc'** (*netcdf files*) – Chunked by months. Snow-free profiles of ice and snow properties. Written if *write_profiles* is `'yes'`. Can be found in `outputpath`.
- **'profiles_for_memls_snowyes_yyyymm.nc'** (*netcdf files*) – Chunked by months. Snow-covered profiles of ice and snow properties. Written if *write_profiles* is `'yes'`. Can be found in `outputpath`.
- **'TB_assim_yyyymm_f.nc'** (*netcdf files*) – Chunked by months. Ice surface brightness temperatures for cold conditions. Written if *write_profiles* is `'yes'`. Can be found in `outputpath`.
- **'TBtot_assim_yyyymm_f.nc'** (*netcdf files*) – Chunked by months. Brightness temperatures at the top of atmosphere. Can be found in `outputpath`.

### Notes

`f` in the filenames is the rounded `freq_of_int`.

---

**Note:** Please remain aware that the results from ARC3O have only been evaluated for the frequency of **6.9 GHz, vertical polarization** at the moment! The use for other frequencies and polarizations is at your own risk! Especially, this function will also give out results for H polarization. However, they have not been evaluated yet!

---

## 1.5.3 mask_functions

`arc3o.mask_functions.`**define_periods**(*sit*, *snow*, *tsi*, *timestep*)

Build masks for different seasons.

This function combines all season masking functions to have an overall season overview.

**Parameters**

- **sit** (*xarray.DataArray*) – sea-ice thickness in m
- **snow** (*xarray.DataArray*) – snow thickness in m
- **tsi** (*xarray.DataArray*) – sea-ice (or snow) surface temperature in K
- **timestep** (*int*) – timestep of your data in h

**Returns** **period_masks** – mask for all seasons, where 0 = open water, 1 = cold conditions, 2 = snow melt, 3 = bare ice in summer

**Return type** xarray.DataArray

arc3o.mask_functions.**ice_type_wholeArctic**(*sit*, *timestep*)
Prepare mask for ice types

This function defines the mask for ice types

**Parameters**

- **sit** (*xarray.DataArray*) – sea-ice thickness in m
- **timestep** (*integer*) – timestep of your data in h

**Returns** **ice_type** – mask defining the different ice types, where: 1 = open water OW, 2 = first-year ice FYI, 3 = multiyear ice MYI

**Return type** xarray.DataArray

arc3o.mask_functions.**is_summer**(*month*)
Filters warm conditions months

**Parameters** **month** (*int or np.array or xarray.DataArray*) – month of the year, January is 1, December is 12

**Returns** **same type as 'month'** – `True` if month is between April (4) and September (9), `False` if month is between October (10) and March (3)

**Return type** int or np.array or xarray.DataArray

arc3o.mask_functions.**snow_period_masks**(*tsi*, *snow*)
Identify snow growth and snow melt.

This function defines when there is snow growth and snow melt

**Parameters**

- **tsi** (*xarray.DataArray*) – sea-ice (or snow) surface temperature in K
- **snow** (*xarray.DataArray*) – snow thickness in m

**Returns**

- **snowdown_mask** (*xarray.DataArray*) – mask defining if the snow depth decreased since the last timestep
- **melt_mask** (*xarray.DataArray*) – mask defining grid cells where snow decreased and it was warm enough to be melt

arc3o.mask_functions.**summer_bareice_mask**(*sit*, *snow*, *timestep*)
Identify areas of bare ice in summer.

This function defines areas of bare ice in summer.

**Parameters**

- **sit** (*xarray.DataArray*) – sea-ice thickness in m
- **snow** (*xarray.DataArray*) – snow thickness in m
- **timestep** (*int*) – timestep of your data in h

**Returns** **bareice_summer_mask** – mask defining if the grid cells consist of bare ice in summer

**Return type** xarray.DataArray

### 1.5.4 memls_functions_2D

arc3o.memls_functions_2D.**Nsw**(*Ssw*)
Compute normality of sea water or brine

This function computes the normality of sea water or brine. It is Eq. 20 in [Ulaby et al., 1986].

**Parameters**

- **Ssw** (*np.array or xarray.DataArray*) – salinity of brine or sea water in g/kg
- **freq** (*float*) – frequency in GHz

**Returns** **N** – normality of sea water or brine

**Return type** np.array or xarray.DataArray

#### Notes

This function is part of the original MEMLS developed by the Institute of Applied Physics, University of Bern, Switzerland. A description of that model version can be found in [Wiesmann & Matzler, 1998] and [Wiesmann & Matzler, 1999]. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**abscoeff**(*epsi*, *epsii*, *Ti*, *freq*)
Compute absorption coefficient.

This function computes the absorption coefficient from the dielectric properties. Formulae from [Ulaby et al., 1986].

**Parameters**

- **epsi** (*np.array or xarray.DataArray*) – real part dielectric constant
- **epsii** (*np.array or xarray.DataArray*) – imaginary part dielectric constant
- **Ti** (*np.array or xarray.DataArray*) – ice temperature in K
- **freq** (*float*) – frequency in GHz

**Returns** **gai** – absorption coefficient

**Return type** np.array or xarray.DataArray

#### Notes

This function is part of the original MEMLS developed by the Institute of Applied Physics, University of Bern, Switzerland. A description of that model version can be found in [Wiesmann & Matzler, 1998] and [Wiesmann & Matzler, 1999]. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**absorp2f**(*gbih*, *gbiv*, *gs6*, *ga2i*, *epsi*, *epsii*, *roi*, *Ti*, *pci*, *freq*, *Wi*, *gai*)
Compute the absorption and scattering coefficient from structural parameters

This function computes the absorption and scattering coefficient from structural parameters.

**Parameters**

- **gbih** (*xarray.DataArray*) – 2-flux scattering coefficient at h pol
- **gbiv** (*xarray.DataArray*) – 2-flux scattering coefficient at v pol
- **gs6** (*xarray.DataArray*) – 6-flux scattering coefficient

- **ga2i** (*xarray.DataArray*) – 2-flux absorption coefficient
- **epsi** (*xarray.DataArray*) – permittivity
- **epsii** (*xarray.DataArray*) – loss
- **roi** (*xarray.DataArray*) – density in g/cm3
- **Ti** (*xarray.DataArray*) – temperature in K
- **pci** (*xarray.DataArray*) – correlation length in mm
- **freq** (*float*) – frequency in GHz
- **Wi** (*xarray.DataArray*) – wetness between 0 and 1
- **gai** (*xarray.DataArray*) – absorption coefficient

**Returns**

- **gbih** (*xarray.DataArray*) – 2-flux scattering coefficient at h pol
- **gbiv** (*xarray.DataArray*) – 2-flux scattering coefficient at v pol
- **gs6** (*xarray.DataArray*) – 6-flux scattering coefficient
- **ga2i** (*xarray.DataArray*) – 2-flux absorption coefficient

### Notes

This function was introduced into the MEMLS code by R.T. Tonboe. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**add_matrix_dim**(*A*, *name_new_dim*)

Add a dummy `matrix_dim` to enable matrix multiplication

This function is more technical, adds a dummy `matrix_dim` to enable matrix multiplication

**Parameters** **A** (*xarray.DataArray*) – matrix to be changed

**Returns** **res** – changed matrix

**Return type** xarray.DataArray

### Notes

This function was introduced by C. Burgard to adapt the python functions to `xarray.DataArray` parameters and output.

arc3o.memls_functions_2D.**append_laydim_begin**(*xrda*, *const*)

Append a constant at the beginning of the dimension `layer_nb`

This function has practical reasons, it appends a constant at the beginning of a dimension, here `layer_nb`.

**Parameters**

- **xrda** (*xarray.DataArray*) – variable where it has to be appended
- **const** (*float*) – constant to be appended at the beginning

**Returns** **xrda_app** – longer array

**Return type** xarray.DataArray

### Notes

This function was introduced by C. Burgard to adapt the python functions to `xarray.DataArray` parameters and output.

arc3o.memls_functions_2D.**append_laydim_end**(*xrda*, *const*)
Append a constant at the end of the dimension `layer_nb`

This function has practical reasons, it appends a constant at the end of a dimension, here `layer_nb`

**Parameters**

- **xrda** (*xarray.DataArray*) – variable where it has to be appended
- **const** (*float*) – constant to be appended at the end

**Returns** **xrda_app** – longer array

**Return type** xarray.DataArray

### Notes

This function was introduced by C. Burgard to adapt the python functions to `xarray.DataArray` parameters and output.

arc3o.memls_functions_2D.**build_xarray**(*data*, *temp*)
Transform `np.array` into `xarray.DataArray`

This function is more technical, transforms `np.array` into `xarray.DataArray`.

**Parameters**

- **data** (*np.array*) – data to be transformed
- **temp** (*xarray.DataArray*) – other xarray.DataArray that has the wished output dimensions

**Returns** **res** – resulting xarray.DataArray of same dimensions as *temp*

**Return type** xarray.DataArray

### Notes

This function was introduced by C. Burgard to adapt the python functions to `xarray.DataArray` parameters and output.

arc3o.memls_functions_2D.**build_xarray_matrix2D**(*data*, *temp*)
Transform a `np.array` into `xarray.DataArray` over the two dimensions `layer_nb` and `matrix_dim` for matrix operations

This function is more technical, transforms a `np.array` into `xarray.DataArray` over the two dimensions `layer_nb` and `matrix_dim` for matrix operations.

**Parameters**

- **data** (*np.array*) – data to be transformed
- **temp** (*xarray.DataArray*) – other xarray.DataArray that has the wished dimensions

**Returns** **test** – resulting xarray with two times dimension layer_nb for matrix operations

**Return type** xarray.DataArray

### Notes

This function was introduced by C. Burgard to adapt the python functions to `xarray.DataArray` parameters and output.

`arc3o.memls_functions_2D.`**`condbrine`**(*T*)
Compute the conductivity of brine

This function computes the conductivity of brine. It is Eq. 7 in [Stogryn & Desargant, 1985].

**Parameters** **T** (*np.array or xarray.DataArray*) – ice temperature in K or °C

**Returns** **condbrine** – conductivity of brine

**Return type** np.array or xarray.DataArray

### Notes

This function is part of the original MEMLS developed by the Institute of Applied Physics, University of Bern, Switzerland. A description of that model version can be found in [Wiesmann & Matzler, 1998] and [Wiesmann & Matzler, 1999]. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

`arc3o.memls_functions_2D.`**`ebrine`**(*T*, *freq*)
Compute brine permittivity

This function computes the brine permittivity, the fit is valid up to -25°C. It is Eq. 1 in [Stogryn & Desargant, 1985].

**Parameters**

- **T** (*np.array or xarray.DataArray*) – ice temperature in K or °C
- **freq** (*float*) – frequency in GHz

**Returns**

- **ebr.real** (*np.array or xarray.DataArray*) – real part of brine permittivity
- **ebr.imag** (*np.array or xarray.DataArray*) – imaginary part of brine permittivity

### Notes

This function was introduced into the MEMLS code by R.T. Tonboe. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

`arc3o.memls_functions_2D.`**`eice_s2p`**(*e1*, *e2*, *v*)
Compute effective dielectric constant of medium consisting of e1 and e2

This function computes the effective dielectric constant of medium consisting of e1 and e2. It is based on the improved born approximation by [Matzler, 1998a] and the Polder/VanSanten mixing formulae for spherical inclusions.

**Parameters**

- **e1** (*np.array or xarray.DataArray*) – dielectric constant of background
- **e2** (*np.array or xarray.DataArray*) – dielectric constant of sherical inclusions
- **v** (*np.array or xarray.DataArray*) – fraction of inclusions

**Returns** **eeff** – effective dielectric constant of medium consisting of e1 and e2

**Return type** np.array or xarray.DataArray

### Notes

This function was introduced into the MEMLS code by R.T. Tonboe. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**epice**(*T*, *freq*)
Compute the dielectric constant of pure ice.

This function computes the dielectric constant of pure ice, based on [Matzler, 1998b].

**Parameters**

- **T** (*np.array or xarray.DataArray*) – ice temperature in K or °C
- **freq** (*float*) – frequency in GHz

**Returns** **epui** – dielectric constant of pure ice

**Return type** np.array or xarray.DataArray

### Notes

This function is part of the original MEMLS developed by the Institute of Applied Physics, University of Bern, Switzerland. A description of that model version can be found in [Wiesmann & Matzler, 1998] and [Wiesmann & Matzler, 1999]. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**epsib0**(*T*)
Compute static dielectric constant of brine.

This function computes the static dielectric constant of brine. The fit is valid up to -25°C. It is Eq. 10 in [Stogryn & Desargant, 1985].

**Parameters** **T** (*np.array or xarray.DataArray*) – ice temperature in K or °C

**Returns** **epsib0** – static dielectric constant of brine

**Return type** np.array or xarray.DataArray

### Notes

This function was introduced into the MEMLS code by R.T. Tonboe. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**epsice**(*Ti*, *freq*)
Compute dielectric permittivity of ice

This function computes the dielectric permittivity of ice, after Hufford, Mitzima and Matzler.

**Parameters**

- **Ti** (*np.array or xarray.DataArray*) – ice temperature in K
- **freq** (*float*) – frequency in GHz

**Returns** **eice** – dielectric permittivity of ice

**Return type** np.array or xarray.DataArray

### Notes

This function is part of the original MEMLS developed by the Institute of Applied Physics, University of Bern, Switzerland. A description of that model version can be found in [Wiesmann & Matzler, 1998] and [Wiesmann & Matzler, 1999]. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**epsr**(*roi*)
Compute real part of dielectric permittivity for dry snow

This function computes the real part of dielectric permittivity for dry snow from density.

**Parameters** **roi** (*np.array or xarray.DataArray*) – snow density in g/cm3

**Returns** **epsi** – real part of dielectric permittivity of dry snow

**Return type** np.array or xarray.DataArray

### Notes

This function is part of the original MEMLS developed by the Institute of Applied Physics, University of Bern, Switzerland. A description of that model version can be found in [Wiesmann & Matzler, 1998] and [Wiesmann & Matzler, 1999]. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**fresnelc0**(*tei*, *epsi*)
Compute the fresnel reflection coefficients

This function computes the fresnel reflection coefficients (assuming eps" = 0), layer n+1 is the air above the snowpack

**Parameters**

- **tei** (*xarray.DataArray*) – local incidence angle
- **epsi** (*xarray.DataArray*) – real part of dielectric permittivity

**Returns**

- **sih** (*xarray.DataArray*) – interface reflectivity at h pol
- **siv** (*xarray.DataArray*) – interface reflectivity at v pol

### Notes

This function is part of the original MEMLS developed by the Institute of Applied Physics, University of Bern, Switzerland. A description of that model version can be found in [Wiesmann & Matzler, 1998] and [Wiesmann & Matzler, 1999]. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**iborn_s2p**(*e1*, *e2*, *eeff*, *v*, *k*, *pcc*)
Compute the scattering coefficient with improved born approximation

This function computes the scattering coefficient of a collection of spherical inclusions with correlation length pcc using improved born approximation (see [Matzler, 1998a]).

**Parameters**

- **e1** (*np.array or xarray.DataArray*) – dielectric constant of background
- **e2** (*np.array or xarray.DataArray*) – dielectric constant of sherical inclusions

- **eeff** (*np.array or xarray.DataArray*) – effective dielectric constant of medium consisting of e1 and e2
- **v** (*np.array or xarray.DataArray*) – brine volume fraction
- **k** (*float*) – given constant (f(freq))
- **pcc** (*np.array or xarray.DataArray*) – correlation length in mm

**Returns** **ss** – 6-flux scattering coefficient for ice

**Return type** np.array or xarray.DataArray

### Notes

This function was introduced into the MEMLS code by R.T. Tonboe. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**layer**(*ri*, *s_i*, *ti*, *Ti*, *Tgnd*, *Tsky*)
Compute the upwelling brightness temperatures

This function computes the upwelling brightness temperatures, see Eq. 14 in [Wiesmann & Matzler, 1998].

**Parameters**

- **ri** (*xarray.DataArray*) – reflectivity
- **s_i** (*xarray.DataArray*) – interface reflectivity
- **ti** (*xarray.DataArray*) – transmissivity
- **Ti** (*xarray.DataArray*) – temperature in K
- **Tgnd** (*float*) – brightness temperature of the ocean below the ice in K
- **Tsky** (*float*) – brightness temperature of the sky in K

**Returns** **D1** – brightness temperatures at each layer

**Return type** xarray.DataArray

### Notes

This function is part of the original MEMLS developed by the Institute of Applied Physics, University of Bern, Switzerland. A description of that model version can be found in [Wiesmann & Matzler, 1998] and [Wiesmann & Matzler, 1999]. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**memls_2D_1freq**(*freq*, *DI*, *TI*, *WI*, *ROI*, *PCI*, *SAL*, *SITYPE*)
Compute the brightness temperature of a lon-lat-time-depth field at a given frequency

This is the main MEMLS function. It computes the brightness temperature of a lon-lat-time-depth field at a given frequency.

**Parameters**

- **freq** (*float*) – frequency
- **DI** (*xarray.DataArray*) – ice thickness in m
- **TI** (*xarray.DataArray*) – temperature in K
- **WI** (*xarray.DataArray*) – wetness between 0 and 1
- **ROI** (*xarray.DataArray*) – density in kg/m3

- **PCI** (*xarray.DataArray*) – correlation length in mm
- **SAL** (*xarray.DataArray*) – salinity in g/kg
- **SITYPE** (*xarray.DataArray*) – snow 1 /first year ice 3 /multiyear ice 4

**Returns**

- **Tbh** (*xarray.DataArray*) – brightness temperature h-pol
- **Tbv** (*xarray.DataArray*) – brightness temperature v-pol
- **eh** (*xarray.DataArray*) – emissivity h-pol
- **ev** (*xarray.DataArray*) – emissivity v-pol

### Notes

This function is part of the original MEMLS developed by the Institute of Applied Physics, University of Bern, Switzerland. A description of that model version can be found in [Wiesmann & Matzler, 1998] and [Wiesmann & Matzler, 1999]. It was slightly modified to accomodate sea-ice layers by R.T. Tonboe. It was translated to Python, adapted for multi-dimensional input and further extended by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**meteo_sc**(*si*, *rroi*, *rTi*, *rpci*, *freq*, *rWi*, *rgai*, *gbih*, *gbiv*, *gs6*, *ga2i*)

Compute the scattering coefficient of fresh snow

This function computes the scattering coefficient of only partly recrystallized snow, linear combination of iborn, wahl==6 (see *sccoeff()*).

**Parameters**

- **si** (*xarray.DataArray*) – layer type ice/snow [1/0]
- **rroi** (*xarray.DataArray*) – density in g/cm3
- **rTi** (*xarray.DataArray*) – temperature in K or °C
- **rpci** (*xarray.DataArray*) – correlation length in mm
- **freq** (*float*) – frequency in GHz
- **rgai** (*xarray.DataArray*) – absorption coefficient
- **gbih** (*xarray.DataArray*) – 2-flux scattering coefficient at h pol
- **gbiv** (*xarray.DataArray*) – 2-flux scattering coefficient at v pol
- **gs6** (*xarray.DataArray*) – 6-flux scattering coefficient
- **ga2i** (*xarray.DataArray*) – 2-flux absorption coefficient

**Returns**

- **gbih** (*xarray.DataArray*) – 2-flux scattering coefficient at h pol
- **gbiv** (*xarray.DataArray*) – 2-flux scattering coefficient at v pol
- **gs6** (*xarray.DataArray*) – 6-flux scattering coefficient
- **ga2i** (*xarray.DataArray*) – 2-flux absorption coefficient

### Notes

This function was introduced into the MEMLS code by R.T. Tonboe. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**mixmod**(*f*, *Ti*, *Wi*, *epsi*, *epsii*)

Compute permittivity for snow wetness above 0

This function computes the permittivity for Wetness > 0 using the physical mixing model Weise 97, after Matzler 1987 (corrected). The water temperature is assumed constant at 273.15 K.

**Parameters**

- **f** (*float*) – frequency in GHz
- **Ti** (*np.array or xarray.DataArray*) – snow/ice temperature in K
- **Wi** (*np.array or xarray.DataArray*) – snow/ice wetness between 0 and 1
- **epsi** (*np.array or xarray.DataArray*) – real part of dielectric permittivity of dry snow
- **epsii** (*np.array or xarray.DataArray*) – imaginary part of dielectric permittivity

**Returns**

- **epsi** (*np.array or xarray.DataArray*) – real part of dielectric permittivity of wet snow
- **epsii** (*np.array or xarray.DataArray*) – imaginary part of dielectric permittivity of wet snow

### Notes

This function is part of the original MEMLS developed by the Institute of Applied Physics, University of Bern, Switzerland. A description of that model version can be found in [Wiesmann & Matzler, 1998] and [Wiesmann & Matzler, 1999]. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**mysie**(*si*, *rho*, *Ti*, *sal*, *freq*, *epsi*, *epsii*)

Compute dielectric constant of ice if it is a mutliyear ice layer

This function computes the dielectric constant of ice if it is a multiyear ice layer. Formulae from [Ulaby et al., 1986].

**Parameters**

- **si** (*np.array or xarray.DataArray*) – sea ice/snow layer 1 or 0
- **rho** (*np.array or xarray.DataArray*) – density of ice layer in g/cm3
- **Ti** (*np.array or xarray.DataArray*) – ice temperature in K
- **sal** (*np.array or xarray.DataArray*) – salinity in g/kg
- **freq** (*float*) – frequency in GHz
- **epsi** (*np.array or xarray.DataArray*) – initial permittivity (of snow)
- **epsii** (*np.array or xarray.DataArray*) – initial loss (of snow)

**Returns**

- **epsi** (*np.array or xarray.DataArray*) – permittivity of ice
- **epsii** (*np.array or xarray.DataArray*) – loss of ice

### Notes

This function was introduced into the MEMLS code by R.T. Tonboe. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**pfadc**(*teta*, *di*, *epsi*, *gs6*)
Compute the effective path length in a layer

This function computes the effective path length in a layer.

**Parameters**

- **teta** (*float*) – incidence angle at snow-air interface in degrees
- **di** (*xarray.DataArray*) – ice thickness in m
- **epsi** (*xarray.DataArray*) – permittivity
- **gs6** (*xarray.DataArray*) – 6-flux scattering coefficient

**Returns**

- **dei** (*xarray.DataArray*) – effective path length in m
- **tei** (*xarray.DataArray*) – local incidence angle
- **tscat** (*xarray.DataArray*) – tau scattering

### Notes

This function is part of the original MEMLS developed by the Institute of Applied Physics, University of Bern, Switzerland. A description of that model version can be found in [Wiesmann & Matzler, 1998] and [Wiesmann & Matzler, 1999]. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**pfadi**(*tei*, *di*)
Compute effective path length in a layer

This function computes the effective path length in a layer.

**Parameters**

- **tei** (*xarray.DataArray*) – local incidence angle
- **di** (*xarray.DataArray*) – ice thickness of layer in m

**Returns** **dei** – effective path length in m

**Return type** xarray.DataArray

### Notes

This function is part of the original MEMLS developed by the Institute of Applied Physics, University of Bern, Switzerland. A description of that model version can be found in [Wiesmann & Matzler, 1998] and [Wiesmann & Matzler, 1999]. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**polmix**(*tscat*, *sih*, *siv*)
Compute the polarization mixing of the interface reflectivities of each layer

This function computes the polarization mixing of the interface reflectivities of each layer (taking into account the first order scattering)

**Parameters**

- **tscat** (*xarray.DataArray*) – tau scattering
- **sih** (*xarray.DataArray*) – interface reflectivity at h-pol
- **siv** (*xarray.DataArray*) – interface reflectivity at v-pol

**Returns**

- **sih** (*xarray.DataArray*) – interface reflectivity at h-pol
- **siv** (*xarray.DataArray*) – interface reflectivity at v-pol

### Notes

This function is part of the original MEMLS developed by the Institute of Applied Physics, University of Bern, Switzerland. A description of that model version can be found in [Wiesmann & Matzler, 1998] and [Wiesmann & Matzler, 1999]. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**relaxt**(*T*)

Compute relaxation time

This function computes the relaxation time, the fit is valid up to -25°C. It is Eq. 12 in [Stogryn & Desargant, 1985].

**Parameters** **T** (*np.array or xarray.DataArray*) – ice temperature in K or °C

**Returns** **relax** – relaxation time in nanoseconds

**Return type** np.array or xarray.DataArray

### Notes

This function was introduced into the MEMLS code by R.T. Tonboe. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**ro2epsd**(*roi*, *Ti*, *freq*)

Compute real part of dielectric permittivity for dry snow

This function computes the dielectric permittivity for dry snow from density.

**Parameters**

- **roi** (*np.array or xarray.DataArray*) – snow density in g/cm3
- **Ti** (*np.array or xarray.DataArray*) – snow temperature in K
- **freq** (*float*) – frequency in GHz

**Returns**

- **epsi** (*np.array or xarray.DataArray*) – real part of dielectric permittivity of dry snow
- **epsii** (*np.array or xarray.DataArray*) – imaginary part of dielectric permittivity of dry snow

### Notes

This function is part of the original MEMLS developed by the Institute of Applied Physics, University of Bern, Switzerland. A description of that model version can be found in [Wiesmann & Matzler, 1998] and [Wiesmann & Matzler, 1999]. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**rt**(*gai*, *gbi*, *dei*)
Compute the layer reflectivity and transmissivity

This function computes the layer reflectivity and transmissivity.

**Parameters**

- **gai** (*xarray.DataArray*) – absorption coefficient
- **gbi** (*xarray.DataArray*) – scattering coefficient
- **dei** (*xarray.DataArray*) – effective path length in m

**Returns**

- **ri** (*xarray.DataArray*) – reflectivity
- **ti** (*xarray.DataArray*) – transmissivity

### Notes

This function is part of the original MEMLS developed by the Institute of Applied Physics, University of Bern, Switzerland. A description of that model version can be found in [Wiesmann & Matzler, 1998] and [Wiesmann & Matzler, 1999]. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**sccoeff**(*roi*, *Ti*, *pci*, *freq*, *Wi*, *gai*, *sccho*)
Compute the the scattering coefficient

This function computes the the scattering coefficient from structural parameters. Different algorithms can be chosen, by changing "sccho"

**Parameters**

- **roi** (*np.array or xarray.DataArray*) – density in g/cm3
- **Ti** (*np.array or xarray.DataArray*) – temperature in K
- **pci** (*np.array or xarray.DataArray*) – correlation length in mm
- **freq** (*float*) – frequency in GHz
- **Wi** (*np.array or xarray.DataArray*) – wetness between 0 and 1
- **gai** (*np.array or xarray.DataArray*) – absorption coefficient
- **sccho** (*int*) – scattering coefficient algorithm chosen

**Returns**

- **gbih** (*np.array or xarray.DataArray*) – 2-flux scattering coefficient at h pol
- **gbiv** (*np.array or xarray.DataArray*) – 2-flux scattering coefficient at v pol
- **gs6** (*np.array or xarray.DataArray*) – 6-flux scattering coefficient
- **ga2i** (*np.array or xarray.DataArray*) – 2-flux absorption coefficient

### Notes

This function is part of the original MEMLS developed by the Institute of Applied Physics, University of Bern, Switzerland. A description of that model version can be found in [Wiesmann & Matzler, 1998] and [Wiesmann & Matzler, 1999]. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**scice**(*si*, *gbih*, *gbiv*, *gs6*, *ga2i*, *Ti*, *sal*, *freq*, *pci*)
Compute the ice scattering coefficient from structural parameters

This function computes the scattering coefficient from structural parameters

**Parameters**

- **si** (*np.array or xarray.DataArray*) – layer type ice/snow [1/0]
- **gbih** (*np.array or xarray.DataArray*) – 2-flux scattering coefficient at h pol
- **gbiv** (*np.array or xarray.DataArray*) – 2-flux scattering coefficient at v pol
- **gs6** (*np.array or xarray.DataArray*) – 6-flux scattering coefficient
- **ga2i** (*np.array or xarray.DataArray*) – 2-flux absorption coefficient
- **Ti** (*xarray.DataArray*) – temperature in K
- **sal** (*np.array or xarray.DataArray*) – salinity in g/kg
- **freq** (*float*) – frequency in GHz
- **pci** (*np.array or xarray.DataArray*) – correlation length in mm

**Returns**

- **gbih** (*np.array or xarray.DataArray*) – 2-flux scattering coefficient at h pol
- **gbiv** (*np.array or xarray.DataArray*) – 2-flux scattering coefficient at v pol
- **gs6** (*np.array or xarray.DataArray*) – 6-flux scattering coefficient
- **ga2i** (*np.array or xarray.DataArray*) – 2-flux absorption coefficient

### Notes

This function was introduced into the MEMLS code by R.T. Tonboe. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**scice_my**(*si*, *gbih*, *gbiv*, *gs6*, *ga2i*, *Ti*, *dens*, *freq*, *pci*, *sal*)
Compute the scattering coefficient of multiyear ice from structural parameters

This function computes the scattering coefficient of multiyear ice from structural parameters.

**Parameters**

- **si** (*xarray.DataArray*) – layer type ice/snow [1/0]
- **gbih** (*xarray.DataArray*) – 2-flux scattering coefficient at h pol
- **gbiv** (*xarray.DataArray*) – 2-flux scattering coefficient at v pol
- **gs6** (*xarray.DataArray*) – 6-flux scattering coefficient
- **ga2i** (*xarray.DataArray*) – 2-flux absorption coefficient
- **Ti** (*xarray.DataArray*) – temperature in K
- **dens** (*xarray.DataArray*) – density in g/cm3

- **freq** (*float*) – frequency in GHz
- **pci** (*xarray.DataArray*) – correlation length in mm
- **sal** (*xarray.DataArray*) – salinity in g/kg

**Returns**

- **gbih** (*xarray.DataArray*) – 2-flux scattering coefficient at h pol
- **gbiv** (*xarray.DataArray*) – 2-flux scattering coefficient at v pol
- **gs6** (*xarray.DataArray*) – 6-flux scattering coefficient
- **ga2i** (*xarray.DataArray*) – 2-flux absorption coefficient

### Notes

This function was introduced into the MEMLS code by R.T. Tonboe. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**sie**(*si*, *sal*, *Ti*, *freq*, *epsi*, *epsii*)
Compute dielectric constant of ice if it is an ice layer

This function computes the dielectric constant of ice if it is a sea-ice layer. Formulae from [Ulaby et al., 1986].

**Parameters**

- **si** (*np.array or xarray.DataArray*) – sea ice/snow layer 1 or 0
- **sal** (*np.array or xarray.DataArray*) – salinity in g/kg
- **Ti** (*np.array or xarray.DataArray*) – ice temperature in K
- **freq** (*float*) – frequency in GHz
- **epsi** (*np.array or xarray.DataArray*) – initial permittivity (of snow)
- **epsii** (*np.array or xarray.DataArray*) – initial loss (of snow)

**Returns**

- **epsi** (*np.array or xarray.DataArray*) – permittivity of ice
- **epsii** (*np.array or xarray.DataArray*) – loss of ice

### Notes

This function was introduced into the MEMLS code by R.T. Tonboe. It was translated to Python and adapted for multi-dimensional input by C. Burgard to be used in ARC3O.

arc3o.memls_functions_2D.**tei_ndim**(*teta*, *ns*)
Append `teta` at the end of the dimension `layer_nb`

This function has practical reasons, it appends `teta` at the end of `ns` over the dimension `layer_nb`

**Parameters**

- **teta** (*float*) – incidence angle in degrees
- **ns** (*xarray.DataArray*) – real part of the refractive index of the slab

**Returns** **tei_ndim** – local incidence angle

**Return type** xarray.DataArray

### Notes

This function was introduced by C. Burgard to adapt the python functions to `xarray.DataArray` parameters and output.

`arc3o.memls_functions_2D.`**`xr_diag`**(*v*, *k=0*)
Create diagonal matrix with values of v on the diagonal

This function is more technical, creates diagonal matrix with values of v on the diagonal

**Parameters**

- **v** (*xarray.DataArray*) – xarray with the wished dimensions
- **k** (*int*) – coefficient if diagonal is shifted from middle

**Returns** **res** – resulting diagonal matrix

**Return type** xarray.DataArray

### Notes

This function was introduced by C. Burgard to adapt the python functions to `xarray.DataArray` parameters and output.

`arc3o.memls_functions_2D.`**`xr_eye`**(*v*, *k=0*)
Create diagonal matrix with ones on the diagonal

This function is more technical, creates diagonal matrix with ones on the diagonal, gives out an `xarray.DataArray`.

**Parameters**

- **v** (*xarray.DataArray*) – xarray.DataArray with the wished dimensions
- **k** (*int*) – coefficient if you want to shift the ones

**Returns** **test** – resulting xarray with two times dimension layer_nb for matrix operations

**Return type** xarray.DataArray

### Notes

This function was introduced by C. Burgard to adapt the python functions to `xarray.DataArray` parameters and output.

`arc3o.memls_functions_2D.`**`xr_linalg_inv`**(*A*)
Compute the (multiplicative) inverse of a matrix.

This function is more technical, equivalent of `numpy.linalg.inv()` for `xarray.DataArray`

**Parameters** **A** (*xarray.DataArray*) – matrix to be inverted

**Returns** **res** – inverted matrix

**Return type** xarray.DataArray

#### Notes

This function was introduced by C. Burgard to adapt the python functions to `xarray.DataArray` parameters and output.

`arc3o.memls_functions_2D.`**`xr_matmul`**(*A*, *B*, *input_dims*, *output_dims*)
Compute matrix multiplication

This function is more technical, matrix multiplication for `xarray.DataArray`, `xarray.dot()` gave weird results

**Parameters**

- **A** (*xarray.DataArray*) – matrix 1
- **B** (*xarray.DataArray*) – matrix 2

**Returns** **asol** – result of matrix multiplications

**Return type** xarray.DataArray

#### Notes

This function was introduced by C. Burgard to adapt the python functions to `xarray.DataArray` parameters and output.

### 1.5.5 profile_functions

`arc3o.profile_functions.`**`Sb`**(*T*)
Compute brine salinity.

This function computes the brine salinity. It is based on Eq. 39 in [Vant et al., 1978] and Eq. 3.4 and 3.5 in [Notz, 2005].

**Parameters** **T** (*xarray.DataArray*) – temperature in K or °C

**Returns** **tot_Sb** – Brine Salinity in g/kg

**Return type** xarray.DataArray

`arc3o.profile_functions.`**`Vb`**(*S*, *Sbr*)
Compute brine volume fraction

This function computes the brine volume fraction. It is based on Eq. 1.5 in [Notz, 2005].

**Parameters**

- **S** (*xarray.DataArray*) – bulk salinity in g/kg
- **Sbr** (*xarray.DataArray*) – brine salinity in g/kg

**Returns** **bvf** – brine volume fraction (between 0 and 1)

**Return type** xarray.DataArray

`arc3o.profile_functions.`**`build_corrlen_profile`**(*empty_corrlen_prof*, *prof_thick*, *fyi_mask*, *winter_mask*, *layer_amount*)
Build correlation length profiles.

This function builds correlation length profiles. Based on experiments by R.T. Tonboe, the correlation lengths are defined as follows:

- First-year ice upper 20 cm: 0.25 mm

- First-year ice lower 20 cm: 0.35 mm
- Multiyear ice: 1.5 mm

**Parameters**

- **empty_corrlen_prof** (*xarray.DataArray*) – empty xarray with the expected dimensions of the output
- **prof_thick** (*xarray.DataArray*) – thickness profile in m
- **fyi_mask** (*xarray.DataArray*) – mask telling where there is first-year ice
- **winter_mask** (*xarray.DataArray*) – mask telling us which points are in "winter"
- **layer_amount** (*int*) – number of layers the profile should have

**Returns** **corrlen** – correlation length over layer_amount in mm

**Return type** xarray.DataArray

arc3o.profile_functions.**build_density_profile**(*empty_dens_prof*, *temperature*, *salinity*, *winter_mask*, *layer_amount*, *snow_dens*)

Build density profiles.

This function builds sea-ice and snow density profiles

**Parameters**

- **empty_dens_prof** (*xarray.DataArray*) – empty xarray with the expected dimensions of the output
- **temperature** (*xarray.DataArray*) – temperature profile in K
- **salinity** (*xarray.DaraArray*) – salinity profile in g/kg
- **winter_mask** (*xarray.DataArray*) – mask telling us which points are in "winter"
- **layer_amount** (*int*) – number of layers the profile should have
- **snow_dens** (*float*) – constant snow density we want to assign to the snow (usually 300 or 330 kg/m3)

**Returns** **dens** – density profiles over *layer_amount* taking into account ice and snow

**Return type** xarray.DataArray

arc3o.profile_functions.**build_salinity_profile**(*empty_sal_prof*, *fyi_mask*, *myi_mask*, *winter_mask*, *layer_amount*)

Combine FYI and MYI salinity profiles

This function builds salinity profiles according to the ice type, using *sal_approx_fy()* and *sal_approx_my()*.

**Parameters**

- **empty_sal_prof** (*xarray.DataArray*) – empty xarray with the expected dimensions of the output
- **fyi_mask** (*xarray.DataArray*) – mask telling where there is first-year ice
- **myi_mask** (*xarray.DataArray*) – mask telling where there is multiyear ice
- **winter_mask** (*xarray.DataArray*) – mask telling us which points are in "winter"
- **layer_amount** (*int*) – number of layers the profile should have

**Returns** **tot_sal** – salinity profiles over layer_amount and for different ice types in g/kg

**Return type** xarray.DataArray

arc3o.profile_functions.**build_sisn_prof**(*empty_def_prof*, *myi_mask*, *fyi_mask*, *winter_mask*, *layer_amount*)

Build snow/FYI/MYI information profiles

This function builds layer type profiles (1 = snow, 3 = first-year ice, 4 = multiyear ice)

**Parameters**

- **empty_def_prof** (*xarray.DataArray*) – empty xarray with the expected dimensions of the output
- **myi_mask** (*xarray.DataArray*) – mask telling where there is multiyear ice
- **fyi_mask** (*xarray.DataArray*) – mask telling where there is first-year ice
- **winter_mask** (*xarray.DataArray*) – mask telling us which points are in "winter"
- **layer_amount** (*int*) – number of layers the profile should have

**Returns** **sisn** – profiles defining layer type, where 1 = snow, 3 = first-year ice, 4 = multiyear ice

**Return type** xarray.DataArray

arc3o.profile_functions.**build_temp_profile**(*surf_temp_ice*, *empty_temp_prof*, *winter_mask*, *fyi_mask*, *myi_mask*, *layer_amount*, *snow_opt*, *snd*, *sit*, *e_bias_fyi*, *e_bias_myi*)

Build the temperature profile.

This function builds linear temperature profiles over depth between the snow surface temperature and ice surface temperature and between ice surface temperature and bottom freezing temperature (-1.8°C).

**Parameters**

- **surf_temp_ice** (*xarray.DataArray*) – sea-ice (or snow) surface temperature in K
- **empty_temp_prof** (*xarray.DataArray*) – empty xarray with the expected dimensions of the output
- **winter_mask** (*xarray.DataArray*) – mask telling us which points are in "winter"
- **layer_amount** (*int*) – number of layers the profile should have
- **snow_opt** (*str*) – `'yes'` if there is snow on top, `'no'` if there is no snow on top
- **snd** (*xarray.DataArray*) – snow thickness in m
- **sit** (*xarray.DataArray*) – sea-ice thickness in m
- **e_bias_fyi** (*xarray.DataArray*) – tuning parameter to correct for MEMLS bias in emissivity for first-year ice
- **e_bias_myi** (*xarray.DataArray*) – tuning parameter to correct for MEMLS bias in emissivity for multiyear ice

**Returns** **shorter_prof** – tuned temperature profile over layer_amount in K

**Return type** xarray.DataArray

### Notes

All input data arrays must be broadcastable towards each other!

arc3o.profile_functions.**build_thickness_profile**(*sit*, *snd*, *empty_thick_prof*, *winter_mask*, *layer_amount*)

Build the thickness profile.

This function builds thickness profiles with equidistant layers over total ice thickness.

**Parameters**

- **sit** (*xarray.DataArray*) – sea-ice thickness in m
- **snd** (*xarray.DataArray*) – snow thickness in m
- **empty_thick_prof** (*xarray.DataArray*) – empty xarray with the expected dimensions of the output
- **winter_mask** (*xarray.DataArray*) – mask telling us which points are in "winter"
- **layer_amount** (*int*) – number of layers the profile should have

**Returns** **empty_thick_prof** – thickness profile over layer_amount in m

**Return type** xarray.DataArray

arc3o.profile_functions.**build_wetness_profile**(*empty_wet_prof*, *winter_mask*)

Build wetness profiles

This function builds wetness profiles currently everything is set to 0 as the ice brine volume fraction is computed in *Vb()*.

**Parameters**

- **empty_wet_prof** (*xarray.DataArray*) – empty xarray with the expected dimensions of the output
- **winter_mask** (*xarray.DataArray*) – mask telling us which points are in "winter"

**Returns** **wet** – wetness profiles (everything set to zero)

**Return type** xarray.DataArray

arc3o.profile_functions.**compute_ice_snow_int_temp**(*sit*, *snd*, *tsi*)

Compute the snow/ice interface temperature.

This function computes the temperature at the snow-ice interface, inspired from [Semtner, 1976].

**Parameters**

- **sit** (*xarray.DataArray*) – sea-ice thickness in m
- **snd** (*xarray.DataArray*) – snow thickness in m
- **tsi** (*xarray.DataArray*) – sea-ice (or snow) surface temperature in K

**Returns** **T_i** – temperature at snow-ice interface in K

**Return type** xarray.DataArray

arc3o.profile_functions.**create_profiles**(*surf_temp_ice*, *sit*, *snow*, *layer_amount*, *info_ds*, *e_bias_fyi*, *e_bias_myi*, *snow_dens*)

Combine all profile informations.

This is the main profile function. It summarizes all profiles to write them out, for both snow-covered and snow-free profiles.

**Parameters**

- **surf_temp_ice** (*xarray.DataArray*) – sea ice (or snow) surface temperature
- **sit** (*xarray.DataArray*) – sea-ice thickness in m
- **snow** (*xarray.DataArray*) – snow thickness in m
- **layer_amount** (*int*) – number of layers the profile should have
- **info_ds** (*xarray.Dataset*) – dataset containing the mask information
- **e_bias_fyi** (*float*) – tuning coefficient for first-year ice
- **e_bias_myi** (*float*) – tuning coefficient for multiyear ice
- **snow_dens** (*float*) – snow density (constant)

**Returns**

- **profiles1** (*xarray.Dataset*) – profiles of all properties if covered by snow
- **profiles2** (*xarray.Dataset*) – profiles of all properties if bare ice

arc3o.profile_functions.**icerho**(*T*, *S*)

Compute sea-ice density.

This function computes the sea-ice density. It is based on [Pounder, 1965] and Eq. 3.8 in [Notz, 2005].

**Parameters**

- **T** (*xarray.DataArray*) – temperature in K
- **S** (*xarray.DataArray*) – bulk salinity in g/kg

**Returns** **rho_tot** – sea-ice density in kg/m3

**Return type** xarray.DataArray

arc3o.profile_functions.**sal_approx_fy**(*norm_z*)

Build salinity profile f(depth) for first-year ice.

This function builds a salinity profile as a function of depth for first-year ice. It is based on Eq. 17 in [Griewank & Notz, 2015].

**Parameters** **norm_z** (*xarray.DataArray*) – normalized depth, 1 is bottom and 0 is top

**Returns** **sal_fy** – salinity profile for first-year ice in g/kg

**Return type** xarray.DataArray

arc3o.profile_functions.**sal_approx_my**(*norm_z*)

Build salinity profile f(depth) for multiyear ice.

This function builds a salinity profile as a function of depth for multiyear ice. It is based on Eq. 18 in [Griewank & Notz, 2015].

**Parameters** **norm_z** (*xarray.DataArray*) – normalized depth, 1 is bottom and 0 is top

**Returns** **sal_my** – salinity profile for multiyear ice in g/kg

**Return type** xarray.DataArray

## 1.6 References

## 1.7 Publications

Burgard, C., Notz, D., Pedersen, L. T., and Tonboe, R. T. (2020): "The Arctic Ocean Observation Operator for 6.9GHz (ARC3O) – Part 1: How to obtain sea-ice brightness temperatures at 6.9 GHz from climate model output", *The Cryosphere*, 14, 2369–2386, https://doi.org/10.5194/tc-14-2369-2020.

Burgard, C., Notz, D., Pedersen, L. T., and Tonboe, R. T. (2020): "The Arctic Ocean Observation Operator for 6.9GHz (ARC3O) – Part 2: Development and evaluation", *The Cryosphere*, 14, 2387–2407, https://doi.org/10.5194/tc-14-2387-2020.

CHAPTER

TWO

# HOW TO CITE ARC3O

The detailed description and evaluation of ARC3O is found in Burgard et al., 2020b and should therefore, when used, be cited as follows:

Burgard, C., Notz, D., Pedersen, L. T., and Tonboe, R. T. (2020): "The Arctic Ocean Observation Operator for 6.9GHz (ARC3O) – Part 2: Development and evaluation", *The Cryosphere*, 14, 2387–2407, https://doi.org/10.5194/tc-14-2387-2020.

CHAPTER

THREE

# INDICES AND TABLES

- genindex
- modindex
- search

# BIBLIOGRAPHY


[Bunzel et al., 2016] Bunzel, F., Notz, D., Baehr, J., Müller, W.A., & Fröhlich, K. (2016). Seasonal climate forecasts significantly affected by observational uncertainty of arctic sea ice concentration. *Geophys. Res. Lett.*, *43*(2), 852–859. doi:10.1002/2015GL066928

[Burgard et al., 2020a] Burgard, C., Notz, D., Pedersen, L.T., & Tonboe, R.T. (2020). The arctic ocean observation operator for 6.9 ghz (arc3o) - part 1: how to obtain sea-ice brightness temperatures at 6.9 ghz from climate model output. *The Cryosphere*, *14*, 2369 - 2386. doi:10.5194/tc-14-2369-2020

[Burgard et al., 2020b] Burgard, C., Notz, D., Pedersen, L.T., & Tonboe, R.T. (2020). The arctic ocean observation operator for 6.9 ghz (arc3o) - part 2: development and evaluation. *The Cryosphere*, *14*, 2387 - 2407. doi:10.5194/tc-14-2387-2020

[Griewank & Notz, 2015] Griewank, P.J., & Notz, D. (2015). A 1-d modelling study of arctic sea-ice salinity. *Cryosphere*, *9*, 305-329. doi:10.5194/tc-9-305-2015

[Matzler, 1998a] Mätzler, C. (1998). Improved born approximation for scattering of radiation in a granular medium. *Journal of Applied Physics*, *83*(11), 6111-6117. doi:10.1063/1.367496

[Matzler, 1998b] Mätzler, C. (1998). Microwave properties of ice and snow. In B. Schmitt, C. De Bergh, & M. Festou (Eds.), *Solar System Ices: Based on Reviews Presented at the International Symposium "Solar System Ices" held in Toulouse, France, on March 27–30, 1995* (pp. 241–257). Dordrecht: Springer Netherlands.

[Niederdrenk & Notz, 2018] Niederdrenk, A.L., & Notz, D. (2018). Arctic sea ice in a 1.5°c warmer world. *Geophys. Res. Lett.*, *45*(4), 1963–1971. doi:10.1002/2017GL076159

[Notz, 2005] Notz, D. (2005). *Thermodynamic and Fluid-Dynamical Processes in Sea Ice* (Doctoral dissertation). University of Cambridge.

[Notz et al., 2013] Notz, D., Haumann, A.F., Haak, H., & Marotzke, J. (2013). Arctic sea-ice evolution as modeled by max planck institute for meteorology's earth system model. *J. Adv. Model Earth Sy.*, *5*, 173-194. doi:10.1002/jame.20016

[Notz & Stroeve, 2016] Notz, D., & Stroeve, J. (2016). Observed arctic sea-ice loss directly follows anthropogenic co$_2$ emission. *Science*, *364*, 747-750. doi:10.1126/science.aag2345

[Pounder, 1965] Pounder, E.R. (1965). Jacobs, J.A., & Wilson, J.T. (Eds.). *The Physics of Ice*. 1st ed ed. Elsevier.

[Semtner, 1976] Semtner, A. J. (1976). A model for the thermodynamic growth of sea ice in numerical investigations of climate. *Journal of Physical Oceanography*, *6*(3), 379-389. doi:10.1175/1520-0485(1976)006<0379:AMFTTG>2.0.CO;2

[Stogryn & Desargant, 1985] Stogryn, A., & Desargant, G. (1985). The dielectric properties of brine in sea ice at microwave frequencies. *IEEE Transactions on Antennas and Propagation*, *33*, 523-532. doi:10.1109/TAP.1985.1143610

[Tonboe et al., 2006] Tonboe, R., Andersen, S., Toudal, L., & Heygster, G. (2006). Mätzler, C., Rosenkranz, P.W., Battaglia, A., & Wigneron, J.P. (Eds.). Sea ice emission modelling. *Thermal Microwave Radiation - Applications for Remote Sensing* (pp. 382–400). IET Electromagnetic Waves Series 52.

[Ulaby et al., 1986] Ulaby, F.T., Moore, R.K., & Fung, A.K. (1986). Passive microwave sensing of the ocean. *Microwave Remote Sensing, Active and Passive Volume III, From Theory to Applications* (pp. 1412–1521). Artech House, Inc.

[Vant et al., 1978] Vant, M.R., Ramseier, R.O., & Makios, V. (1978). The complex dielectric constant of sea ice at frequencies in the range 0.1-40 ghz. *Journal of Applied Physics*, *49*, 1264-1280. doi:10.1063/1.325018

[Wentz & Meissner, 2000] Wentz, F.J., & Meissner, T. (2000). *Algorithm theoretical basis document (atbd), version 2*. Remote Sensing Systems, Santa Rosa, CA.

[Wiesmann & Matzler, 1998] Wiesmann, A., & Mätzler, C. (1998). *Documentation for MEMLS 98.1*. Institute of Applied Physics, Department of Microwave Physics, University of Bern.

[Wiesmann & Matzler, 1999] Wiesmann, A., & Mätzler, C. (1999). Microwave emission model of layered snowpacks. *Remote Sens. Environ.*, *70*, 307-316.

# PYTHON MODULE INDEX

## a

# INDEX

## M



## N



## P



## R



## S



## T



## V



## X