AirSeaFluxCode

Release 1.2.0

Stavroula Biri

Apr 22, 2024

CONTENTS:

GETTING STARTED

AirSeaFluxCode.py is a Python 3.6+ module designed to process data (input as numpy ndarray float number type) to calculate surface turbulent fluxes, flux product estimates and to provide height adjusted values for wind speed, air temperature and specific humidity of air at a user defined reference height from a minimum number of meteorological parameters (wind speed, air temperature, and sea surface temperature) and for a variety of different bulk algorithms (at the time of the release amount to ten).

Several optional parameters can be input such as: an estimate of humidity (relative humidity, specific humidity or dew point temperature) is required in the case an output of latent heat flux is requested; atmospheric pressure. If cool skin/warm layer adjustments are switched on then shortwave/longwave radiations should be provided as input. Other options the user can define on input are the height on to which the output parameters would be adjusted, the function of the cool skin adjustment provided that the option for applying the adjustment is switched on, the option to consider the effect of convective gustiness. The user can: choose from a wide variety of saturation vapour pressure function in order to compute specific humidity from relative humidity or dew point temperature, provide user defined tolerance limits, user define the maximum number of iterations.

For recommendations or bug reports, please visit https://github.com/NOCSurfaceProcesses/AirSeaFluxCode

1.1 Description of test data

A suite of data is provided for testing, containing values for air temperature, sea surface temperature, wind speed, air pressure, relative humidity, shortwave radiation, longitude and latitude.

The first test data set (data_all.csv) is developed as daily averages from minute data provided by the Shipboard Automated Meteorological and Oceanographic System SAMOS ([Smith2019] , [Smith2018]); it contains a synthesis of various conditions from meteorological and surface oceanographic data from research vessels and three that increase the accuracy of the flux estimate (atmospheric pressure, relative humidity, shortwave radiation). We use quality control level three (research level quality), and we only keep variables flagged as Z (good data) (for details on flag definitions see [Smith2018]). The input sensors' heights vary by ship and sometimes by cruise. The data contain wind speeds ranging between 0.015 and 18.5ms[-1], air temperatures ranging from -3 to 9.7C and air-sea temperature differences (T-T_0, hereafter ΔT) from around -3 to 3C. A sample output file is given (data_all_out.csv and its statistics in data_all_stats.txt) run with default options (see data_all_stats.txt for the input summary); note that deviations from the output values might occur due to floating point errors.

The second test data set contained in era5_r360x180.nc contains ERA5 ([Hersbach2020], [ECMWF2019]) hourly data for one sample day (15/07/2019) remapped to 1x1regular grid resolution using cdo ([Schulzweida2022]). In this case all essential and optional input SSVs are available. For the calculation of TSFs we only consider values over the ice-free ocean by applying the available land mask and sea-ice concentration (equal to zero) and setting values over land or ice to missing (flag="m"). The data contain wind speeds ranging from 0.01 to 24.9 ms[-1], air temperatures ranging from -17.2 to 35.4C and ΔT from around -16.2 to 8C.

1.2 Description of sample code

In the AirSeaFluxCode repository we provide two types of sample routines to aid the user running the code. The first is the routine toy_ASFC.py which is an example of running AirSeaFluxCode either with one-dimensional data sets (like a subset of R/V data) loading the necessary parameters from the test data (data_all.csv) or gridded 3D data sampled in era5_r360x180.nc.

The routine first loads the data in the appropriate format (numpy.ndarray, type float), then calls AirSeaFluxCode loads the data as input, and finally saves the output as text or as a NetCDF file and at the same time generates a table of statistics for all the output parameters and figures of the mean values of the turbulent surface fluxes.

Second a jupyter notebook (ASFC_notebook.ipynb) is provided as a step by step guide on how to run AirSeaFluxCode, starting from the libraries the user would need to import. It also provides an example on how to run AirSeaFluxCode with the research vessel data as input and generate basic plots of momentum and (sensible and latent) heat fluxes. The user can launch the Jupyter Notebook App by clicking on *Jupyter Notebook* icon in Anaconda start menu, this will launch a new browser window in your browser of choice (more details can be found here).

USERS GUIDE

2.1 Introduction

The flux calculation code was implemented in order to provide a useful, easy to use and straightforward "roadmap" of when and why to use different bulk formulae for the calculation of surface turbulent fluxes.

Differences in the calculations between different methods can be found in:

- the way they compute specific humidity from relative humidity, temperature and pressure
- the way they parameterise the exchange coefficients
- the inclusion of heat and moisture roughness lengths
- the inclusion of cool skin/warm layer correction instead of the bulk sea surface temperature
- the inclusion of gustiness in the wind speed, and
- the momentum, heat and moisture stability functions definitions

The available parameterizations in AirSeaFluxCode provided in order to calculate the momentum, sensible heat and latent heat fluxes are implemented following:

- [Smith1980] as S80: the surface drag coefficient is related to 10m wind speed (u_{10}), surface heat and moisture exchange coefficients are constant. The stability parameterizations are based on the Monin-Obukhov similarity theory for stable and unstable condition which modify the wind, temperature and humidity profiles and derives surface turbulent fluxes in open ocean conditions (valid for wind speeds from 6 to 22 ms^{-1}).

- [Smith1988] as S88: is an improvement of the S80 parameterization in the sense that it provides the surface drag coefficient in relation to surface roughness over smooth and viscous surface and otherwise derives surface turbulent fluxes in open ocean conditions as described for S80.

- [LargePond1981], [LargePond1982] as LP82: the surface drag coefficient is computed in relation to u_{10} and has different parameterization for different ranges of wind speed. The heat and moisture exchange coefficients are constant for wind speeds<11ms^{-1} and a function of u_{10} for wind speeds between 11 and 25ms^{-1}. The stability parameterizations are based on the Monin-Obukhov similarity theory for stable and unstable condition.

- [YellandTaylor1996], [Yelland1998] as YT96: the surface drag coefficient is a function of u_*. The heat and moisture exchange coefficients are considered constant as in the cases of S80 and S88.

- [Zeng1998] as UA: the drag coefficient is given as a function of roughness length over smooth and viscous surface. The parameterization includes the effect of gustiness. The heat and moisture exchange coefficients are a function of heat and moisture roughness lengths and are valid in the range of 0.5 and 18 ms^{-1}.

- [LargeYeager2004], [LargeYeager2009] as NCAR: the surface drag coefficient is computed in relation to wind speed for u_{10} >0.5 ms^{-1}. The heat exchange coefficient is given as a function of the drag coefficient (one for stable and one for unstable conditions) and the moisture exchange coefficient is also a function of the drag coefficient.

- [Fairall1996], [Fairall2003], [Edson2013] as C30, and C35: is based on data collected from four expeditions in order to improve the drag and exchange coefficients parameterizations relative to surface roughness. It includes the effects of "cool skin", and gustiness. The effects of waves and sea state are neglected in order to keep the software as simple as possible, without compromising the integrity of the outputs though.

- [ECMWF2019] as ecmwf: the drag, heat and moisture coefficients parameterizations are computed relative to surface roughness estimates. It includes gustiness in the computation of wind speed.

- [Beljaars1995a], [Beljaars1995b], [ZengBeljaars2005] as Beljaars: the drag, heat and moisture coefficients parameterizations are computed relative to surface roughness estimates. It includes gustiness in the computation of wind speed.

2.2 Description of AirSeaFluxCode

In AirSeaFluxCode we use a consistent calculation approach across all algorithms; where this requires changes from published descriptions the effect of those changes are quantified and shown to be small compared to the significance levels we set in Table 1. The AirSeaFluxCode software calculates air-sea flux of momentum, sensible heat and latent heat fluxes from bulk meteorological variables (wind speed (spd), air temperature (T), and relative humidity (RH)) provided at a certain height (hin) above the surface and sea surface temperature (SST) and height adjusted values for wind speed, air temperature and specific humidity of air at a user specified reference height (default is 10 m).

Additionally, non essential parameters can be given as inputs, such as: downward long/shortwave radiation (Rl, Rs), latitude (lat), reference output height (hout), cool skin (cskin), cool skin correction method (skin, following either [Fairall1996b] (default for C30, and C35), [ZengBeljaars2005] (default for Beljaars), [ECMWF2019] (default for ecmwf)), warm layer correction (wl), gustiness (gust) and boundary layer height (zi), choice of bulk algorithm method (meth), the choice of saturation vapour pressure function (qmeth), tolerance limits (tol), choice of Monin-Obukhov length function (L), and the maximum number of iterations (maxiter). Note that all input variables need to be loaded as numpy.ndarray.

The air and sea surface specific humidity are calculated using the functions qsat_air(T, P, RH, qmeth) and qsat_sea(SST, P, qmeth) , which call functions contained in VaporPressure.py to calculate saturation vapour pressure following a chosen method (default is [Buck2012]).

- The air temperature is converted to air temperature for adiabatic expansion following: $Ta = T + 273.16 + \Gamma \cdot hin$

- The density of air is defined as $\rho = (0.34838 \cdot P)/T_{v10n}$

- The specific heat at constant pressure is defined as $c_p = 1004.67(1 + 0.00084 \cdot q_{sea})$

- The latent heat of vapourization is defined as $L_v = (2.501 - 0.00237 \cdot SST)10^6$ (SST in C)

Initial values for the exchange coefficients and friction velocity are calculated assuming neutral stability. The program iterates to calculate the temperature and humidity fluxes and the virtual temperature as $T_v = T_a(1 + 0.61q_{air})$, then the stability parameter z/L either as,

$$\frac{z}{L} = \frac{z(g \cdot k \cdot T_{*v})}{T_{v10n} \cdot u_*^2} \tag{2.2.1}$$

or as a function of the Richardson number as described by [ECMWF2019] [their equations 3.23–3.25]; hence a new value for u_{10n}, hence new transfer coefficients, hence new flux values until convergence is obtained (Table 1). At every iteration step if there are points where the neutral 10 m wind speed (u_{10n}) becomes negative the wind speed value at these points is set to NaN. The values for air density, specific heat at constant volume, and the latent heat of vaporisation are used in converting the scaled fluxes u_*, T_*, and q_* (Eq. 2.2.2, for UA we retain their equations 7-14) to flux values

in Nm^{-2} and Wm^{-2}, respectively.

$$u_* = \frac{k \cdot u_z}{\log(\frac{z}{z_{om}}) - \Psi_m(\frac{z}{L}) + \Psi_m(\frac{z_{om}}{L})}$$
$$t_* = \frac{k \cdot (T - SST)}{\log(\frac{z}{z_{oh}}) - \Psi_h(\frac{z}{L}) + \Psi_h(\frac{z_{oh}}{L})} \quad (2.2.2)$$
$$q_* = \frac{k \cdot (q_{air} - q_{sea})}{\log(\frac{z}{z_{oq}}) - \Psi_q(\frac{z}{L}) + \Psi_q(\frac{z_{oq}}{L})}$$

AirSeaFluxCode is set up to test for convergence between the i$^{\text{th}}$ and (i-1)$^{\text{th}}$ iteration according to the tolerance limits shown in Table 1 for six variables in total, of which three are relative to the height adjustment (u$_{10}$, t$_{10}$, q$_{10}$) and three to the flux calculation (τ, shf, lhf) respectively. The tolerance limits are set according to the maximum accuracy that can be feasible for each variable. The user can choose to allow for convergence either only for the fluxes (default), or only for height adjustment or for both (all six variables). Values that have not converged are by default set to missing, but the number of iterations until convergence is provided as an output (this number is set to -1 for non convergent points). A set of flags are provided as an output that signify: "m" where input values are missing; "o" where the wind speed for this point is outside the nominal range for the used parameterization; "u" or "q" for points that produce unphysical values for u$_{10n}$ or q$_{10n}$ respectively during the iteration loop; "r" where relative humidity is greater than 100%; "l" where the bulk Richardson number is below -0.5 or above 0.2 or z/L is greater than 1000; "i" where the value failed to converge after n number of iterations, if the points converged normally they are flagged with "n". The user should expect NaN values if out is set to zero (namely output only values that have converged) for values that have not converged after the set number of iterations (default is ten) or if they produced unphysical values for u$_{10n}$ or q$_{10n}$.

Table 2.2.1: Table 1: Tolerance and significance limits

Variable	Tolerance	Significance
u$_{10n}$ [ms^{-1}]	0.01	0.1
T$_{10n}$ [K]	0.01	0.1
q$_{10n}$ [g/kg]	10^{-2}	10^{-1}
τ [Nm^{-2}]	10^{-3}	10^{-2}
shf [Wm^{-2}]	0.1	2
lhf [Wm^{-2}]	0.1	2

2.3 AirSeaFluxCode module

AirSeaFluxCode.**AirSeaFluxCode**(*spd, T, SST, SST_fl, meth, lat=None, hum=None, P=None, hin=18, hout=10, Rl=None, Rs=None, cskin=0, skin=None, wl=0, gust=None, qmeth='Buck2', tol=None, maxiter=30, out=0, out_var=None, L=None*)

Calculate turbulent surface fluxes using different parameterizations.

Calculate height adjusted values for spd, T, q

> **Parameters**
>
> - **spd** (*float*) – relative wind speed in [m/s] (is assumed as magnitude difference between wind and surface current vectors)
>
> - **T** (*float*) – air temperature [K] (will convert if < 200)
>
> - **SST** (*float*) – sea surface temperature [K] (will convert if < 200)
>
> - **SST_fl** (*str*) – provides information on the type of the input SST; "bulk" or "skin"
>
> - **meth** (*str*) – "S80", "S88", "LP82", "YT96", "UA", "NCAR", "C30", "C35", "ecmwf", "Beljaars"
>
> - **lat** (*float*) – latitude [deg], default 45deg

- **hum** (*float*) – humidity input switch 2x1 [x, values] default is relative humidity x='rh' : relative humidity [%] x='q' : specific humidity [g/kg] x='Td' : dew point temperature [K]

- **P** (*float*) – air pressure [hPa], default 1013hPa

- **hin** (*float*) – sensor heights [m] (array 3x1 or 3xn), default 18m

- **hout** (*float*) – output height [m], default is 10m

- **Rl** (*float*) – downward longwave radiation [W/m^2]

- **Rs** (*float*) – downward shortwave radiation [W/m^2]

- **cskin** (*int*) – 0 switch cool skin adjustment off, else 1 default is 0

- **skin** (*str*) – cool skin method option "C35", "ecmwf" or "Beljaars"

- **wl** (*int*) – warm layer correction default is 0, to switch on set to 1

- **gust** (*int*) – 4x1 [x, beta, zi, ustb] x=0 gustiness is OFF, x=1-5 gustiness is ON and use gustiness factor: 1. Fairall et al. 2003, 2. GF is removed from TSFs u10n, uref, 3. GF=1, 4. following ECMWF, 4. following Zeng et al. 1998, 6. following C35 matlab code; beta gustiness parameter, default is 1.2, zi PBL height [m] default is 600, min is the value for gust speed in stable conditions [m/s], default is 0.01 m/s

- **qmeth** (*str*) –

    is the saturation evaporation method to use amongst [
    "HylandWexler", "Hardy", "Preining", "Wexler", "GoffGratch", "WMO", "MagnusTetens", "Buck", "Buck2", "WMO2018", "Sonntag", "Bolton", "IAPWS", "MurphyKoop"]

    default is Buck2

- **tol** (*float*) – 4x1 or 7x1 [option, lim1-3 or lim1-6] option : 'flux' to set tolerance limits for fluxes only lim1-3 option : 'ref' to set tolerance limits for height adjustment lim-1-3 option : 'all' to set tolerance limits for both fluxes and height

    adjustment lim1-6

    default is tol=['all', 0.01, 0.01, 1e-2, 1e-3, 0.1, 0.1]

- **maxiter** (*int*) – number of iterations (default = 10)

- **out** (*int*) –

    set 0 to set points that have not converged, negative values of
    u10n, q10n or T10n out of limits to missing (default)

    set 1 to keep points

- **out_var** (*str*) – optional. user can define pandas array of variables to be output. the default full pandas array, with cskin=0 gust=0, is :

    out_var = ("tau", "sensible", "latent", "monob", "cd", "cd10n",
    "ct", "ct10n", "cq", "cq10n", "tsrv", "tsr", "qsr", "usr", "psim", "psit", "psiq", "psim_ref", "psit_ref", "psiq_ref", "u10n", "t10n", "q10n", "zo", "zot", "zoq", "uref", "tref", "qref", "qair", "qsea", "Rb", "rh", "rho", "cp", "lv", "theta", "itera")

    the "limited" pandas array is:
    out_var = ("tau", "sensible", "latent", "uref", "tref", "qref")

    the user can define a custom pandas array of variables to output.

- L (`str`) – Monin-Obukhov length definition options "tsrv" : default "Rb" : following ecmwf (IFS Documentation cy46r1)

Returns

- **res**

    [array that contains]

    1. momentum flux [N/m^2]

    2. sensible heat [W/m^2]

    3. latent heat [W/m^2]

    4. Monin-Obhukov length [m]

    5. drag coefficient (cd)

    6. neutral drag coefficient (cd10n)

    7. heat exchange coefficient (ct)

    8. neutral heat exchange coefficient (ct10n)

    9. moisture exhange coefficient (cq)

    10. neutral moisture exchange coefficient (cq10n)

    11. star virtual temperatcure (tsrv)

    12. star temperature (tsr) [K]

    13. star specific humidity (qsr) [g/kg]

    14. star wind speed (usr) [m/s]

    15. momentum stability function (psim)

    16. heat stability function (psit)

    17. moisture stability function (psiq)

    18. momentum stability function at hout (psim_ref)

    19. heat stability function at hout (psit_ref)

    20. moisture stability function at hout (psiq_ref)

    21. 10m neutral wind speed (u10n) [m/s]

    22. 10m neutral temperature (t10n) [K]

    23. 10m neutral specific humidity (q10n) [g/kg]

    24. surface roughness length (zo) [m]

    25. heat roughness length (zot) [m]

    26. moisture roughness length (zoq) [m]

    27. wind speed at reference height (uref) [m/s]

    28. temperature at reference height (tref) [K]

    29. specific humidity at reference height (qref) [g/kg]

    30. cool-skin temperature depression (dter) [K]

    31. cool-skin humidity depression (dqer) [g/kg]

    32. warm layer correction (dtwl)

33. thickness of the viscous layer (delta)

34. specific humidity of air (qair) [g/kg]

35. specific humidity at sea surface (qsea) [g/kg]

36. downward longwave radiation (Rl)

37. downward shortwave radiation (Rs)

38. downward net longwave radiation (Rnl)

39. gust wind speed (ug) [m/s]

40. star wind speed with gust (usr_gust) [m/s]

41. Gustiness Factor (GustFact)

42. Bulk Richardson number (Rb)

43. relative humidity (rh) [%]

44. air density (rho)

45. specific heat of moist air (cp)

46. lv latent heat of vaporization (Jkg1)

47. potential temperature (theta)

48. number of iterations until convergence

49. **flag ("n": normal, "o": out of nominal range,**
    "u": u10n<0, "q":q10n<0 or q>40 "m": missing, "l": Rib<-0.5 or Rib>0.2 or
    z/L>1000, "r" : rh>100%, "t" : t10n<173K or t10n>373K "i": convergence fail at n)

- *2021 / Author S. Biri*

- *2021 / Restructured by R. Cornes*

- *2021 / Simplified by E. Kent*

- *2024 / Units corrected by J. Siddons*

2.4 Description of Sub-Routines

This section provides a description of the constants and sub-routines that are called in AirSeaFluxCode.

2.4.1 Drag Coefficient Functions

AirSeaFluxCode.flux_subs.**cd_calc**(*cdn*, *hin*, *hout*, *psim*)

Calculate drag coefficient at reference height.

Parameters

- **cdn** (*float*) – neutral drag coefficient

- **hin** (*float*) – wind speed height [m]

- **hout** (*float*) – reference height [m]

- **psim** (*float*) – momentum stability function

Returns
cd

Return type
float

AirSeaFluxCode.flux_subs.**cdn_calc**(*u10n*, *usr*, *Ta*, *grav*, *meth*)

Calculate neutral drag coefficient.

Parameters

- **u10n** (*float*) – neutral 10m wind speed [m/s]

- **usr** (*float*) – friction velocity [m/s]

- **Ta** (*float*) – air temperature [K]

- **grav** (*float*) – gravity [m/s^2]

- **meth** (*str*) –

Returns

- **cdn** (*float*)

- **zo** (*float*)

AirSeaFluxCode.flux_subs.**cdn_from_roughness**(*u10n*, *usr*, *Ta*, *grav*, *meth*)

Calculate neutral drag coefficient from roughness length.

Parameters

- **u10n** (*float*) – neutral 10m wind speed [m/s]

- **usr** (*float*) – friction velocity [m/s]

- **Ta** (*float*) – air temperature [K]

- **grav** (*float [m/s]*) – gravity

- **meth** (*str*) –

Returns
cdn

Return type
float

2.4.2 Heat and Moisture Exchange Coefficient Functions

2.4.3 Stratification Functions

The stratification functions Ψ_i are integrals of the dimensionless profiles Φ_i, which are determined experimentally, and are applied as stablility corrections to the wind speed, temperature and humidity profiles. They are a function of the stability parameter z/L where L is the Monin-Obukhov length.

2.4.4 Other Flux Functions

2.4.5 Cool-skin/Warm-layer Functions

AirSeaFluxCode.cs_wl_subs.**cs**(*sst*, *d*, *rho*, *Rs*, *Rnl*, *cp*, *lv*, *usr*, *tsr*, *qsr*, *grav*, *opt*)

> Compute cool skin.
>
> Based on COARE3.5 (Fairall et al. 1996, Edson et al. 2013)
>
> > **Parameters**
> >
> > > - **sst** (*float*) – sea surface temperature [K]
> > > - **d** (*float*) – cool skin thickness [m]
> > > - **rho** (*float*) – density of air [kg/m^3]
> > > - **Rs** (*float*) – downward shortwave radiation [Wm-2]
> > > - **Rnl** (*float*) – net upwelling IR radiation [Wm-2]
> > > - **cp** (*float*) – specific heat of air at constant pressure [J/K/kg]
> > > - **lv** (*float*) – latent heat of vaporization [J/kg]
> > > - **usr** (*float*) – friction velocity [ms^-1]
> > > - **tsr** (*float*) – star temperature [K]
> > > - **qsr** (*float*) – star humidity [g/kg]
> > > - **grav** (*float*) – gravity [ms^-2]
> > > - **opt** (*str*) – method to follow
> >
> > **Returns**
> >
> > > - **dter** (*float*) – cool skin correction [K]
> > > - **delta** (*float*) – cool skin thickness [m]

AirSeaFluxCode.cs_wl_subs.**cs_Beljaars**(*rho*, *Rs*, *Rnl*, *cp*, *lv*, *usr*, *tsr*, *qsr*, *grav*, *Qs*)

> cool skin adjustment based on Beljaars (1997) air-sea interaction in the ECMWF model
>
> > **Parameters**
> >
> > > - **rho** (*float*) – density of air [kg/m^3]
> > > - **Rs** (*float*) – downward solar radiation [Wm-2]
> > > - **Rnl** (*float*) – net thermal radiaion [Wm-2]
> > > - **cp** (*float*) – specific heat of air at constant pressure [J/K/kg]
> > > - **lv** (*float*) – latent heat of vaporization [J/kg]
> > > - **usr** (*float*) – friction velocity [m/s]
> > > - **tsr** (*float*) – star temperature [K]
> > > - **qsr** (*float*) – star humidity [g/kg]
> > > - **grav** (*float*) – gravity [ms^-2]
> > > - **Qs** (*float*) – radiation balance
> >
> > **Returns**
> >
> > > - **Qs** (*float*) – radiation balance

- **dtc** (*float*) – cool skin temperature correction [K]

AirSeaFluxCode.cs_wl_subs.**cs_C35**(*sst*, *rho*, *Rs*, *Rnl*, *cp*, *lv*, *delta*, *usr*, *tsr*, *qsr*, *grav*)

    Compute cool skin.

    Based on COARE3.5 (Fairall et al. 1996, Edson et al. 2013)

        Parameters

- **sst** (`float`) – sea surface temperature [K]
- **rho** (`float`) – density of air [kg/m^3]
- **Rs** (`float`) – downward shortwave radiation [Wm-2]
- **Rnl** (`float`) – net upwelling IR radiation [Wm-2]
- **cp** (`float`) – specific heat of air at constant pressure [J/K/kg]
- **lv** (`float`) – latent heat of vaporization [J/kg]
- **delta** (`float`) – cool skin thickness [m]
- **usr** (`float`) – friction velocity [m/s]
- **tsr** (`float`) – star temperature [K]
- **qsr** (`float`) – star humidity [g/kg]
- **grav** (`float`) – gravity [ms^-2]

        Returns

- **dter** (*float*) – cool skin correction [K]
- **dqer** (*float*) – humidity corrction [g/kg]
- **delta** (*float*) – cool skin thickness [m]

AirSeaFluxCode.cs_wl_subs.**cs_ecmwf**(*rho*, *Rs*, *Rnl*, *cp*, *lv*, *usr*, *tsr*, *qsr*, *sst*, *grav*)

    cool skin adjustment based on IFS Documentation cy46r1

        Parameters

- **rho** (`float`) – density of air [kg/m^3]
- **Rs** (`float`) – downward solar radiation [Wm-2]
- **Rnl** (`float`) – net thermal radiation [Wm-2]
- **cp** (`float`) – specific heat of air at constant pressure [J/K/kg]
- **lv** (`float`) – latent heat of vaporization [J/kg]
- **usr** (`float`) – friction velocity [m/s]
- **tsr** (`float`) – star temperature [K]
- **qsr** (`float`) – star humidity [g/kg]
- **sst** (`float`) – sea surface temperature [K]
- **grav** (`float`) – gravity [ms^-2]

        Returns
            dtc – cool skin temperature correction [K]

        Return type
            float

AirSeaFluxCode.cs_wl_subs.**delta**(*aw*, *Q*, *usr*, *grav*)

    Compute the thickness (m) of the viscous skin layer.

    Based on Fairall et al., 1996 and cited in IFS Documentation Cy46r1 eq. 8.155 p. 164

        Parameters

- **aw** (*float*) – thermal expansion coefficient of sea-water [1/K]
- **Q** (*float*) – part of the net heat flux actually absorbed in the warm layer [W/m^2]
- **usr** (*float*) – friction velocity in the air (u*) [m/s]
- **grav** (*float*) – gravity [ms^-2]

        Returns

            delta – the thickness (m) of the viscous skin layer

        Return type

            float

AirSeaFluxCode.cs_wl_subs.**get_dqer**(*dter*, *sst*, *qsea*, *lv*)

    Calculate humidity correction.

        Parameters

- **dter** (*float*) – cool skin correction [K]
- **sst** (*float*) – sea surface temperature [K]
- **qsea** (*float*) – specific humidity over sea [g/kg]
- **lv** (*float*) – latent heat of vaporization [J/kg]

        Returns

            dqer – humidity correction [g/kg]

        Return type

            float

AirSeaFluxCode.cs_wl_subs.**wl_ecmwf**(*rho*, *Rs*, *Rnl*, *cp*, *lv*, *usr*, *tsr*, *qsr*, *sst*, *skt*, *dtc*, *grav*)

    Calculate warm layer correction following IFS Documentation cy46r1. and aerobulk (Brodeau et al., 2016)

        Parameters

- **rho** (*float*) – density of air [kg/m^3]
- **Rs** (*float*) – downward solar radiation [Wm-2]
- **Rnl** (*float*) – net thermal radiation [Wm-2]
- **cp** (*float*) – specific heat of air at constant pressure [J/K/kg]
- **lv** (*float*) – latent heat of vaporization [J/kg]
- **usr** (*float*) – friction velocity [m/s]
- **tsr** (*float*) – star temperature [K]
- **qsr** (*float*) – star humidity [g/kg]
- **sst** (*float*) – bulk sst [K]
- **skt** (*float*) – skin sst from previous step [K]
- **dtc** (*float*) – cool skin correction [K]
- **grav** (*float*) – gravity [ms^-2]

Returns
    dtwl – warm layer correction [K]

Return type
    float

2.4.6 Humidity Functions

AirSeaFluxCode.hum_subs.**VaporPressure**(*temp*, *P*, *phase*, *meth*)

    Calculate the saturation vapor pressure.

    For temperatures above 0 deg C the vapor pressure over liquid water is calculated.

    The optional parameter 'liquid' changes the calculation to vapor pressure over liquid water over the entire temperature range.

    The current default fomulas are Hyland and Wexler for liquid and Goff Gratch for ice.

    Ported to Python and modified by S. Biri from Holger Voemel's original

    Parameters

- **temp** (`float`) – Temperature [C]
- **phase** (`str`) – 'liquid' : Calculate vapor pressure over liquid water or 'ice' : Calculate vapor pressure over ice
- **meth** (`str`) – formula to be used Hardy : vaporpressure formula from Hardy (1998) MagnusTetens : vaporpressure formula from Magnus Tetens GoffGratch : vaporpressure formula from Goff Gratch Buck : vaporpressure formula from Buck (1981) Buck2 : vaporpressure formula from the Buck (2012) WMO : vaporpressure formula from WMO (1988) WMO2018 : vaporpressure formula from WMO (2018) Wexler : vaporpressure formula from Wexler (1976) Sonntag : vaporpressure formula from Sonntag (1994) Bolton : vaporpressure formula from Bolton (1980) HylandWexler : vaporpressure formula from Hyland and Wexler (1983) IAPWS : vaporpressure formula from IAPWS (2002) Preining : vaporpressure formula from Preining (2002) MurphyKoop : vaporpressure formula from Murphy and Koop (2005)

    Returns
        P – Saturation vapor pressure [hPa]

    Return type
        float

AirSeaFluxCode.hum_subs.**gamma**(*opt*, *sst*, *t*, *q*, *cp*)

    Compute the adiabatic lapse-rate.

    Parameters

- **opt** (`str`) – type of adiabatic lapse rate dry or "moist" dry has options to be constant "dry_c", for dry air "dry", or for unsaturated air with water vapor "dry_v"
- **sst** (`float`) – sea surface temperature [K]
- **t** (`float`) – air temperature [K]
- **q** (`float`) – specific humidity of air [g/kg]
- **cp** (`float`) – specific capacity of air at constant Pressure

    Returns
        gamma – lapse rate [K/m]

> > > **Return type**
> > > > float

AirSeaFluxCode.hum_subs.**get_hum**(*hum*, *T*, *sst*, *P*, *qmeth*)

> Get specific humidity output.

> > **Parameters**

> > > - **hum** (`array`) –

> > >   **humidity input switch 2x1 [x, values] default is relative humidity**
> > >   > x='rh' : relative humidity [%] x='q' : specific humidity [g/kg] x='Td' : dew point temperature [K]

> > > - **T** (`float`) – air temperature [K]

> > > - **sst** (`float`) – sea surface temperature [K]

> > > - **P** (`float`) – air pressure at sea level [hPa]

> > > - **qmeth** (`str`) – method to calculate specific humidity from vapor pressure

> > **Returns**

> > > - **qair** (*float*) – specific humidity of air [g/kg]

> > > - **qsea** (*float*) – specific humidity over sea surface [g/kg]

AirSeaFluxCode.hum_subs.**qsat_air**(*T*, *P*, *rh*, *qmeth*)

> Compute saturation specific humidity [g/kg].

> > **Parameters**

> > > - **T** (`float`) – temperature [$^\circ$,C]

> > > - **P** (`float`) – pressure [mb]

> > > - **rh** (`float`) – relative humidity [%]

> > > - **qmeth** (`str`) – method to calculate vapor pressure

> > **Returns**
> > > **q** – specific humidity [g/kg]

> > **Return type**
> > > float

AirSeaFluxCode.hum_subs.**qsat_sea**(*T*, *P*, *qmeth*)

> Compute surface saturation specific humidity [g/kg].

> > **Parameters**

> > > - **T** (`float`) – temperature [$^\circ$,C]

> > > - **P** (`float`) – pressure [mb]

> > > - **qmeth** (`str`) – method to calculate vapor pressure

> > **Returns**
> > > **qs** – surface saturation specific humidity [g/kg]

> > **Return type**
> > > float

2.4.7 Utility Functions

AirSeaFluxCode.util_subs.**CtoK = 273.16**

> Conversion factor for $^\circ$C to K

AirSeaFluxCode.util_subs.**gc**(*lat*, *lon=None*)

> Computes gravity relative to latitude
>
> > **Parameters**
> >
> > - **lat** (*float*) – latitude [$^\circ$]
> >
> > - **lon** (*float*) – longitude [$^\circ$, optional]
> >
> > **Returns**
> > **gc** – gravity constant [m/s^2]
> >
> > **Return type**
> > float

AirSeaFluxCode.util_subs.**get_heights**(*h*, *dim_len*)

> Reads input heights for velocity, temperature and humidity
>
> > **Parameters**
> >
> > - **h** (*float*) – input heights [m]
> >
> > - **dim_len** (*int*) – length dimension
> >
> > **Returns**
> > **hh**
> >
> > **Return type**
> > array

AirSeaFluxCode.util_subs.**kappa = 0.4**

> von Karman's constant

AirSeaFluxCode.util_subs.**rho_air**(*T*, *qair*, *p*)

> Compute density of (moist) air using the eq. of state of the atmosphere.
>
> as in aerobulk (https://github.com/brodeau/aerobulk/) Brodeau et al. (2016)
>
> > **Parameters**
> >
> > - **T** (*float*) – absolute air temperature [K]
> >
> > - **qair** (*float*) – air specific humidity [g/kg]
> >
> > - **p** (*float*) – pressure in [Pa]
> >
> > **Returns**
> > **rho_air** – density of moist air [kg/m^3]
> >
> > **Return type**
> > TYPE

AirSeaFluxCode.util_subs.**set_flag**(*miss*, *rh*, *u10n*, *q10n*, *t10n*, *Rb*, *hin*, *monob*, *itera*, *out=0*)

> Set general flags.
>
> > **Parameters**
> >
> > - **miss** (*int*) – mask of missing input points
> >
> > - **rh** (*float*) – relative humidity [%]

- **u10n** (*float*) – 10m neutral wind speed [ms^{-1}]

- **q10n** (*float*) – 10m neutral specific humidity [g/kg]

- **t10n** (*float*) – 10m neutral air temperature [K]

- **Rb** (*float*) – bulk Richardson number

- **hin** (*float*) – measurement heights [m]

- **monob** (*float*) – Monin-Obukhov length [m]

- **itera** (*int*) – number of iteration

- **out** (*int, optional*) – output option for non converged points. The default is 0.

> **Returns**
>> **flag**

> **Return type**
>> str

AirSeaFluxCode.util_subs.**visc_air**(*T*)

> Computes the kinematic viscosity of dry air as a function of air temp. following Andreas (1989), CRREL Report 89-11.

> **Parameters**
>> **Ta** (*float*) – air temperature [$^\circ$,C]

> **Returns**
>> **visa** – kinematic viscosity [m^2/s]

> **Return type**
>> float

[Beljaars1995a] Beljaars, A. C. M. (1995a). The impact of some aspects of the boundary layer scheme in the ecmwf model. Proc. Seminar on Parameterization of Sub-Grid Scale Physical Processes, Reading, United Kingdom, ECMWF.

[Beljaars1995b] Beljaars, A. C. M. (1995b). The parameterization of surface fluxes in large scale models under free convection. Quart. J. Roy. Meteor. Soc., 121:255–270.

[Buck2012] Buck, A. L. (2012). Buck research instruments, LLC, chapter Appendix I, pages 20–21. unknown, Boulder, CO 80308.

[ECMWF2019] ECMWF, 2019. "Part IV: Physical processes," in Turbulent transport and interactions with the surface. IFS documentation CY46R1 (Reading, RG2 9AX, England: ECMWF), 33–58. Available at: https://www.ecmwf.int/node/19308.

[Edson2013] Edson, J. B., Jampana, V., Weller, R. A., Bigorre, S. P., Plueddemann, A. J., Fairall, C. W., Miller, S. D., Mahrt, L., Vickers, D., and Hersbach, H. (2013). On the exchange of momentum over the open ocean. Journal of Physical Oceanography, 43.

[Fairall1996] Fairall, C. W., Bradley, E. F., Godfrey, J. S., Wick, G. A., Edson, J. B., and Young, G. S. (1996). Cool-skin and warm-layer effects on sea surface temperature. Journal of Geophysical Research, 101(C1):1295–1308.

[Fairall1996b] Fairall, C. W., Bradley, E. F., Rogers, D. P., Edson, J. B., and Young, G. S. (1996b). Bulk parameterization of air-sea fluxes for tropical ocean global atmosphere coupled-ocean atmosphere response experiment. Journal of Geophysical Research, 101(C2):3747–3764.

[Fairall2003] Fairall, C. W., Bradley, E. F., Hare, J. E., Grachev, A. A., and Edson, J. B. (2003). Bulk parameterization of air-sea fluxes: updates and verification for the coare algorithm. Journal of Climate, 16:571–591.

[Hersbach2020] Hersbach, H., Bell, B., Berrisford, P., Hirahara, S., Horanyi, A., Muñoz-Sabater, J., et al. (2020). The ERA5 global reanalysis. Q. J. R. Meteorological Soc. 146, 1999–2049. doi: 10.1002/qj.3803

[LargePond1981] Large, W. G. and Pond, S. (1981). Open ocean momentum flux measurements in moderate to strong winds. Journal of Physical Oceanography, 11(324–336).

[LargePond1982] Large, W. G. and Pond, S. (1982). Sensible and latent heat flux measurements over the ocean. Journal of Physical Oceanography, 12:464–482.

[LargeYeager2004] Large, W. G. and Yeager, S. (2004). Diurnal to decadal global forcing for ocean and sea-ice models: The data sets and flux climatologies. University Corporation for Atmospheric Research.

[LargeYeager2009] Large, W. G. and Yeager, S. (2009). The global climatology of an interannually varying air–sea flux data set. Climate Dyn., 33:341–364.

[Schulzweida2022] Schulzweida, Uwe. (2022). CDO User Guide (2.1.0). Zenodo. https://doi.org/10.5281/zenodo.7112925

[Smith1980] Smith, S. D. (1980). Wind stress and heat flux over the ocean in gale force winds. Journal of Physical Oceanography, 10:709–726.

[Smith1988] Smith, S. D. (1988). Coefficients for sea surface wind stress, heat flux, and wind profiles as a function of wind speed and temperature. Journal of Geophysical Research, 93(C12):15467–15472.

[Smith2018] Smith, S. R., Briggs, K., Bourassa, M. A., Elya, J., and Paver, C. R. (2018). Shipboard automated meteorological and oceanographic system data archive: 2005–2017. Geoscience Data Journal, 5:73–86.

[Smith2019] Smith, S. R., Rolph, J. J., Briggs, K., and Bourassa, M. A. (2019). Quality Controlled Shipboard Automated Meteo- rological and Oceanographic System (SAMOS) data. Center for Ocean-Atmospheric Prediction Studies, pages The Florida State University, Tallahassee, FL, USA, http://samos.coaps.fsu.edu.

[YellandTaylor1996] Yelland, M. and Taylor, P. K. (1996). Wind stress measurements from the open ocean. Journal of Physical Oceanography, 26:541–558.

[Yelland1998] Yelland, M., Moat, B. I., Taylor, P. K., Pascal, R. W., Hutchings, J., and Cornell, V. C. (1998). Wind stress measurements from the open ocean corrected for airflow distortion by the ship. Journal of Physical Oceanography, 28:1511–1526.

[ZengBeljaars2005] Zeng, X. and Beljaars, A. (2005). A prognostic scheme of sea surface skin temperature for modeling and data assimilation. Geophys. Res. Lett., 32(L14605).

[Zeng1998] Zeng, X., Zhao, M., and Dickinson, R. (1998). Intercomparison of bulk aerodynamic algorithms for the computation of sea surface fluxes using toga coare and tao data. J. Climate, 11:2628–2644.

PYTHON MODULE INDEX

a

A

C

D

G

K

M

Q

R

S

V

W