ICLASS 1.0 user manual

Peter Bosman

# Running ICLASS

The actual file to be executed in Python is optimisation.py. This is also the file which needs some user input for the specific optimisation to be performed. The required user input is described in section 2. The files ‘forwardmodel.py’ and ‘inverse\_modelling.py’ need to be located in the same folder as ‘optimisation.py’ to run ICLASS.

# User input paragraphs

This section shortly describes the different blocks within the optimisation file that need to filled in/edited by the user. The beginning and end of each block is indicated in the optimisation file. Be careful to always respect the indentation present in the file!

## settings

General settings for the optimisation, such as whether to use a numerical or analytical derivative etc. The individual settings are explained with comments in the optimisation file. We generally recommend to set ana\_deriv = True , use\_backgr\_in\_cost = True, optim\_method = 'tnc', maxnr\_of\_restarts = 1, use\_ensemble = True, est\_post\_pdf\_covmatr = True, imposeparambounds = True, paramboundspenalty = False, discard\_nan\_minims = False, write\_to\_f = True and wr\_obj\_to\_pickle\_files = True.

Many of the settings are easy to understand by the comments in the code. Some of the settings are explained in a bit more detail here:

Setting the variable ‘imposeparambounds’ to True will force the items in the state vector to stay within specified bounds. This holds both during an optimisation whereby the state vector is updated iteratively, as well as for the ensemble of optimisations. For the ensemble, prior state vectors are sampled from a distribution based on prior information provided by the user. When a sampled state vector falls outside of the specified bounds, the sample will be discarded and a new will be drawn.

Alternatively, the variable ‘paramboundspenalty’ can also be set to True. In that case, a penalty will be added to the cost function. This penalty is calculated according to the following equations. The equation when the parameter x is below the lower bound:

Where is a positive number to be chosen by the user and is the specified lower bound for variable x. Similarly, when the the parameter x is above the upper bound:

Where is the upper bound. There is a variable ‘setNanCostfOutBoundsTo0’ that can also be set to True. In this case, if the cost function was nan before adding the penalty, the cost function will be set to zero before adding the penalty. This is implemented because in Python, adding a number to a nan-variable will result in nan.

Variable ‘wr\_obj\_to\_pickle\_files’ is a switch that if set to True, will result in a set of variables that will be stored as .pkl files. These variables are stored near the end of the optimisation and can be read in again in the ‘postprocessing.py’ files. This way postprocessing can be done after the optimisations without having to redo the entire optimisation if variables are no longer in memory.

**We strongly recommend using ‘imposeparambounds’ instead of ‘paramboundspenalty’, unless there are good reasons to do otherwise. Using ‘paramboundspenalty’ is tested less and will likely lead to a worse convergence of the solution. We furthermore recommend to set optim\_method = 'tnc' instead of ‘bfgs’ to get the best convergence.**

## load obs

In this section observation files can be read in or artificial observations can be manually defined. These observations can afterwards be used in section ‘observation information’. More information on the observations in the last mentioned section.

## prior model param

Here all the input for the CLASS model should be given, e.g. the initial potential temperature. This also includes prior estimates of the parameters that are in the state vector. Everything should be given in the format priormodinput.parameter = value, e.g. priormodinput.theta = 290.

## state, list of used obs and non-model priorinput

Here the variables in the state have to be specified as a list. Also an ‘obsvarlist’ variable has to be defined which should be a list of CLASS variables for which we have observations. Optionally, non-model related prior information can be given here, e.g. priorinput.obs\_sca\_cf\_LE = 1.5.

## prior variance/covar

This section only needs to be filled in when use\_backgr\_in\_cost or use\_ensemble is set to True. In this section, the prior variances and covariances of the state parameters should be specified. This should be in the following format:

e.g. for the variance of ‘z0m’: priorvar['z0m'] = 0.1\*\*2

e.g. for the covariance of ‘z0m’ and ‘z0h’: priorcovar['z0m,z0h'] = 0.5

When no covariances are specified, they are assumed 0. Providing variances is obligatory, ICLASS will raise an exception if they are not specified.

## parameter bounds

This section only needs to be filled in when imposeparambounds or paramboundspenalty is set to True (see section 1.1). In this section bounds can be specified within which each parameter in the state should remain. It should be in the following format:

boundedvars['alfa\_sto'] = [0.1,5] if you wish variable 'alfa\_sto' to remain in the interval [0.1,5].

It is allowed to only specify bounds for one or more specific parameters, there is no need to provide bounds for every parameter in the state.

## observation information

The obligatory information to be specified here is the measurement error for each observation, the observation variables to be used and the times in the model those observations correspond to.

An example of what the user should specify in this section, for observations of variable ‘qmh’:

if item == ‘qmh’:

optim.\_\_dict\_\_['obs\_'+item] = q200\_selected

measurement\_error[item] = [0.00008 for j in range(len(optim.\_\_dict\_\_['obs\_'+item]))]

obs\_times[item] = obstimes\_qmh

Where in this case the variables ‘q200\_selected’ and ‘obstimes\_qmh’ can result from reading in the data in section 1.2. We use optim.obs\_qmh here as example, the observation variables that can be readily assigned can be any variable that occurs in the python class ‘model\_output’ in forwardmodel.py, i.e. any variable for which model output is stored in the store function of CLASS. Other examples would be optim.obs\_theta, optim.obs\_H, optim.obs\_rs, …

Additional variables that can be specified here are weights for the observations, e.g.:

obs\_weights[item] = [2 for j in range(len(optim.\_\_dict\_\_['obs\_'+item]))]

Or the units to be displayed in the observation fit plots created by default:

disp\_units[item] = 'g kg$^{-1}$'

Note that the displayed units only influence the label that will be displayed on the axes, it does not change the actual values of the variable. Therefore the actual units of the CLASS variables should be specified here. There is however one exception, if for variables ‘q’, ‘qmh’,’qmh2’,… the units 'g/kg' or 'g kg$^{-1}$' are specified (the units of q in CLASS are kg/kg), the variable will be multiplied with a factor 10^3 so that the values of the variable in the plots actually change. This is done for nicer plotting purposes.

The name of the variable that will be displayed in plots can also be adapted, e.g.:

display\_names[item] = 'q\_200'

Note that the observation, the observation times, measurement error and the weights have to be provided either as a list or as a numpy.ndarray. Observations which have the value numpy.nan (a representation of ‘not a number’ data), will be automatically discarded.

## units of parameters for pdf figures

This section only needs attention when use\_ensemble and est\_post\_pdf\_covmatr are set to True. The units to be displayed for variables in probability density functions can be specified here, e.g.:

disp\_units\_par['theta'] = 'K'

Note that those units should be the actual units of the CLASS variables, and are only relevant for the label that is displayed on the x-axis.

## energy balance information

This section is only relevant if 'EnBalDiffObsHFrac' is included in the state. In this section the user should specify the energy balance gap at the times that we have observations of sensible and or latent heat fluxes. As an example:

optim.EnBalDiffObs\_atHtimes = np.array((SWD\_obs + LWD\_obs - SWU\_obs - LWU\_obs) - (H\_obs + LE\_obs + G\_obs))

optim.EnBalDiffObs\_atLEtimes = np.array((SWD\_obs + LWD\_obs - SWU\_obs - LWU\_obs) - (H\_obs + LE\_obs + G\_obs))

Where SWD is the incoming shortwave radiation, LWU the outgoing longwave radiation etc., H is the sensible heat flux, LE is the latent heat flux and G the soil heat flux (all resulting from section 1.2 or 1.7). The resulting array should be of type numpy.ndarray or list. In case only LE observations are used, only one variable (optim.EnBalDiffObs\_atLEtimes) has to be specified (only H observations is also possible in a similar way).

## model and representation error

In this section, the representation error and model error can be specified. For the model error there are two options, it can either be estimated by ICLASS (see reference paper) or explicitly specified. In the first case, ICLASS constructs an ensemble of model runs to estimate the model error from. In this case the user should specify the distributions from which random numbers will be sampled to add to the default (prior) model parameters for creating an ensemble, and has the choice between a ’normal’, ’bounded normal’, ’uniform’ or ’triangular’ distribution. This should be done in a parameter dictionary called ‘me\_paramdict’. As an example, if the user wants to perturb 'cveg', 'Lambda', ‘z0m’ and ‘w2’ in the model error ensemble:

me\_paramdict['cveg'] = {'distr':'uniform','leftbound': 0.1,'rightbound': 1.0}

me\_paramdict['Lambda'] = {'distr':'normal','scale': 0.3}

me\_paramdict['z0m'] = {'distr':' triangular ','leftbound': 0.1,’mode’: 0.5,'rightbound': 1.0}

me\_paramdict['w2'] = {'distr':' bounded normal','scale': 0.3,'leftbound': 0.1,'rightbound': 0.7}

For parameter cveg, a uniform distribution is specified in the example, in the interval [0.1,1.0]. For parameter Lambda, a normal distribution is specified with a standard deviation of 0.3. For z0m, a triangular distribution between [0.1,1.0] with the peak (mode) at 0.5 is specified. For w2 a bounded normal distribution is given in the interval [0.1,0.7]. The standard deviation (0.3) specified in the example is not the standard deviation of the bounded normal distribution, but the standard deviation of the distribution if no bounds would be present, i.e. those of the ‘normal’ normal distribution.

Those statements (those in case of ICLASS estimating the model error) have to be placed before the ‘else:’ statement in the input block. In case the user does not want ICLASS to estimate the model error, but prefers to specify the errors himself, this can be done as follows:

for j in range(len(measurement\_error['qmh'])):

mod\_error['qmh'][j] = 0.00015

Note that those statements should be placed after the ‘else’ statement in the input block.

In case no model error is specified or estimated for an observation variable, it is set to zero. If no representation error specified it is also set to zero. Similar to the model error, an observation error can be specified as:

repr\_error['qmh'] = [0.3 for j in range(len(measurement\_error['qmh']))]

## non-state parameters to perturb in ensemble

This section is only relevant if use\_ensemble and pert\_non\_state\_param are set to True. The user can specify here which parameters to perturb in the ensemble of optimisations (not the model error ensemble!). As an example:

non\_state\_paramdict['cveg'] = {'distr':'uniform','leftbound': 0.1,'rightbound': 1.0}

In this example, parameter cveg has a random number added sampled from a uniform distribution in the interval [0.1,1.0]. The user has the choice between between a ’normal’, ’bounded normal’, ’uniform’ or ’triangular’ distribution for each parameter (see section 1.10 to see how to specify the normal, triangular or bounded normal distributions).

# List of parameters that can be optimised

Initial values of:

* theta
* q
* CO2
* COS
* u
* v
* wtheta
* wq
* wCO2
* wCOS
* deltatheta
* deltaq
* deltaCO2
* deltaCOS
* deltau
* deltav
* gammatheta
* gammatheta2
* gammaq
* gammaCO2
* gammaCOS
* gammau
* gammav
* advtheta
* advq
* advCO2
* advCOS
* advu
* advv
* Cs
* ustar
* h
* wg
* Tsoil
* Ts
* Wl
* dz\_h

initial mixed-layer potential temperature [K]

initial mixed-layer specific humidity [kg kg-1]

initial mixed-layer CO2 [ppm]

initial mixed-layer COS [ppb]

initial mixed-layer u-wind speed [m s-1]

initial mixed-layer v-wind speed [m s-1]

(initial) surface kinematic heat flux [K m s-1]

(initial) surface kinematic moisture flux [kg kg-1 m s-1]

(initial) surface total CO2 flux [mgCO2 m-2 s-1]

(initial) surface kinematic COS flux [ppb m s-1]

initial temperature jump at h [K]

initial specific humidity jump at h [kg kg-1]

initial CO2 jump at h [ppm]

initial COS jump at h [ppb]

initial u-wind jump at h [m s-1]

initial v-wind jump at h [m s-1]

free atmosphere potential temperature lapse rate [K m-1]

free atmosphere potential temperature lapse rate 2 [K m-1]

free atmosphere specific humidity lapse rate [kg kg-1 m-1]

free atmosphere CO2 lapse rate [ppm m-1]

free atmosphere COS lapse rate [ppb m-1]

free atmosphere u-wind speed lapse rate [s-1]

free atmosphere v-wind speed lapse rate [s-1]

advection of heat [K s-1]

advection of moisture [kg kg-1 s-1]

advection of CO2 [ppm s-1]

advection of COS [ppb s-1]

advection of u-wind [m s-2]

advection of v-wind [m s-2]

drag coefficient for scalars [-]

surface friction velocity [m s-1]

initial ABL height [m]

volumetric water content top soil layer [m3 m-3]

temperature top soil layer [K]

surface temperature [K]

equivalent water layer depth for wet vegetation [m]

Transition layer thickness [m]

Constants that can be optimised:

* gciCOS
* COSmeasuring\_height
* divU
* w2
* wfc
* wwilt
* wsat
* beta
* z0m
* z0h
* lat
* lon
* doy
* cc
* alpha
* fc
* dFz
* cveg
* LAI
* T2
* a
* b
* p
* CGsat
* C1sat
* C2ref
* gD
* rsmin
* rssoilmin
* Wmax
* Lambda
* PARfract
* R10
* E0
* alfa\_sto

Internal conductance for COS [m s-1]

height 1 COS mixing ratio measurements [m]

horizontal large-scale divergence of wind [s-1]

volumetric water content deeper soil layer [m3 m-3]

volumetric water content field capacity [-]

volumetric water content wilting point [-]

saturated volumetric water content ECMWF config [-]

entrainment ratio for virtual heat [-]

roughness length for momentum [m]

roughness length for scalars [m]

latitude [deg]

longitude [deg]

day of the year [-]

cloud cover fraction [-]

surface albedo [-]

coriolis parameter [s-1]

cloud top radiative divergence [W m-2]

vegetation fraction [-]

leaf area index [-]

temperature deeper soil layer [K]

Clapp and Hornberger retention curve parameter a [-]

Clapp and Hornberger retention curve parameter b [-]

Clapp and Hornberger retention curve parameter p [-]

saturated soil conductivity for heat

Coefficient force term moisture [-] (p131 Vilà-Guerau De Arellano et al 2015)

Coefficient restore term moisture [-]

correction factor transpiration for VPD [-]

minimum resistance transpiration [s m-1]

minimun resistance soil evaporation [s m-1]

thickness of water layer on wet vegetation [m]

thermal diffusivity skin layer [-]

fraction of incoming shortwave radiation that is PAR[-]

respiration at 10 °C [mg CO2 m-2 s-1]

activation energy (for respiration calc) [kJ kmol-1]

Scaling factor for stomatal conductance [-]

Non-model parameters that can be optimised:

* EnBalDiffObsHfrac
* obs\_sca\_cf\_x (replace x by an observation variable, e.g. H)
* Fraction of energy balance gap partitioned to H [-]
* Scale for the observations of x in the cost function [-]

# Technical notes output of ICLASS

ICLASS produces several output files when switch write\_to\_f is set to True, those files are described in the reference paper. Some technical notes on the output here:

* In the automatically generated observation fit plots, the observations that are shown are the observations of ensemble member 0.
* In Modelerrorfile.txt, ‘nr of non-nan members in model err ensemble’ means the number of members that do not have any ‘not-a-number’ value at the times for which the model error is calculated.
* In Optstatsfile.txt, the cost functions (or cost function parts) calculated are based on the observations given to each ensemble member. e.g. if the optimisation of member 5 gave the lowest posterior cost function, it does not automatically mean that the cost function will still be the lowest if the original observations (in case obs are perturbed) would be used to calculate the cost function.
* In Optstatsfile.txt, the mean bias error and ratio of model and obs variance uses the observations scaled with observation scales (if used). In case the 'EnBalDiffObsHFrac' parameter is used, the energy balance corrected observations will be used (see also reference paper). Only model output at the times of observations is used in these calculations. The mean bias error and the ratio of model and obs variance calculations use the parameters of the best state , which is defined here as the posterior state of the ensemble member (if ensemble used, otherwise there is only 1 optimisation) that resulted in the lowest posterior cost function.

# General notes

* When the cost function varies less or equal than 0.1% over a certain number of optimisations, the optimisation will be ended with the message 'too slow progress in costf'. The optimisation will then be restarted, if the cost function is not yet lower than a specified criterion and the maximum number of restarts is not yet reached. (details in code, see min\_func in inverse\_modelling.py).

# Changes to the forward model code with respect to CLASS version October 1th 2019.

Below the changes to the CLASS code as it was on October 1th 2019 on GitHub (<https://github.com/classmodel/modelpy>) are shown . For more details on CLASS itself see Vilà-Guerau De Arellano et al. (2015).

## List of changes

* COS (carbonyl sulphide) has been added. See the reference paper for more details. Note that as a consequence, the user should specify a number of input parameters to the model. Those parameters are COS (initial mixed-layer COS [ppb]), gciCOS (COS canopy scale internal conductance [m/s], only needed if using ags), deltaCOS (initial COS jump at h [ppb]), gammaCOS (free atmosphere COS lapse rate [ppb m-1]), advCOS (advection of COS [ppb s-1]), wCOS (surface kinematic COS flux [ppb m s-1], only the initial flux if wCOS is calculated by the model). Note that COS purely acts as a tracer in the model, it will not change the model results for other quantities.
* The names of the variables dtheta, dq, dthetav, dCO2, du, dv, dthetatend, dqtend, dCO2tend, dutend, dvtend, dtheta0, dq0, dCO20, du0, dv0 have been changed into deltatheta, deltaq, deltathetav, deltaCO2, deltau, deltav, deltathetatend, deltaqtend, deltaCO2tend, deltautend, deltavtend, deltatheta0, deltaq0, deltaCO20, deltau0, deltav0 respectively
* Some additional switches are added: ‘sw\_dynamicsl\_border’, ‘sw\_use\_ribtol’, ‘sw\_advfp’, ‘sw\_printwarnings’, ‘sw\_dyn\_beta’, ‘sw\_useWilson’ and ‘sw\_model\_stable\_con’. Those are explained in section …
* Some additional variables (besides COS-related variables) are added, these are ‘thetamh’,’ thetamh2’,’ thetamh3’,’ thetamh4’,’ thetamh5’,’ thetamh6’,’ thetamh7’,’ Tmh’,’ Tmh2’,’ Tmh3’,’ Tmh4’,’ Tmh5’,’ Tmh6’,’Tmh7’,’qmh’,’qmh2’ ,’qmh3’ ,’qmh4’ ,’qmh5’ ,’qmh6’ ,’qmh7’ ,’ CO2mh’ ,’ CO2mh2’ ,’ CO2mh3’ ,’ CO2mh4’. Those are potential temperature, temperature, specific humidity and CO2 mixing ratio respectively, at different heights. Furthermore there are the variables Tmeasuring\_height (also Tmeasuring\_height2 … Tmeasuring\_height7), qmeasuring\_height (also qmeasuring\_height2 … qmeasuring\_height7) and CO2measuring\_height (also CO2measuring\_height2 … CO2measuring\_height4). These represent the heights at which variables should be calculated, e.g. Tmeasuring\_height7 is the height at which Tmh7 and thetamh7 are calculated. Additionally, the variables CO2surf, esurf and Tsurf are added.
* Some variables that were fixed in the original model, can now be specified as model input. Those are the variables ‘CO2comp298’,’ Q10CO2’,’ gm298’,’ Ammax298’,’ Q10gm’,’ T1gm’,’ T2gm’,’ Q10Am’,’ T1Am’,’ T2Am’,’ f0’,’ ad’,’ alpha0’,’ Kx’,’ gmin’, ‘E0’ and ’R10’. If they are not specified as model input, their default values will be used.
* The variable name ‘zsl’ from the ‘run\_surface\_layer’ module is changed into ‘zelf.zsl’
* When the run function is called with argument ‘checkpoint=True’, the values of many variables are saved, so called checkpointing. This does not influence the working of the actual model, but some extra model code is added for this
* In the constructor of the ‘model’ class (\_\_init\_\_), there is an additional variable defined: *self.nr\_of\_surf\_lay\_its = 10*

This variable determines how many times the ‘run\_surface\_layer’ module is called within the ‘init’ function. The value of 10 is the same as in the original model.

* Additional arguments can be passed to the ‘run’ fnction: ‘checkpoint’, ‘updatevals\_surf\_lay’, ‘delete\_at\_end’ and ‘save\_vars\_indict’. ‘checkpoint’ defaults to False and is already explained above. ‘updatevals\_surf\_lay’ defaults to True and determines whether the values of self.Cs and self.ustar are updated after a call to the ‘run\_surface\_layer’ module. When set to True, the behaviour is as in the original model. ‘delete\_at\_end’ defaults to True determines whether the function ‘exitmodel’ is called after running the model. When this switch is set to True, the behaviour is as in the original model. ‘save\_vars\_indict’ defaults to False and only serves for gradient test purposes. When set to False the behaviour is as in the original model.
* In the model code there are some lines such as ‘elif(self.ls\_type == 'canopy\_model'):’ and ‘elif(self.ls\_type == 'sib4'):’. The switch ls\_type should only be set to the default CLASS options ‘js’ or ‘ags’ and should not be set to ‘sib4’ or 'canopy\_model', as those are not (yet) implemented. The presence of these statements does not change the output of the model though compared to the original model.
* Some variables have been split or renamed to make adjoint coding easier, e.g. the variable ‘sinlea’ from the ‘run\_radiation’ module has been split into ‘part1\_sinlea’ and ‘part2\_sinlea’. This has no influence on the model results.
* An additional input variable ‘PARfract’can be specified, which defines the fraction of PAR that gets absorbed by the leaves. The default value is 0.5, if this value is chosen the behaviour of the model is the same as in the original version.
* Additional input variables ‘htrans’ and ‘gammatheta2’ can be specified, when the boundary layer height exceeds ‘htrans’, gammatheta2 will be used instead of ‘gammatheta’.
* The initial value of the variable ‘Cs’ can now be given as model input, if nothing specified the default value of the original model will be used.
* An additional input variable ‘ags\_C\_mode’ can be specified as either ‘MXL’ or ‘surf’, this determines wether in ags the surface or the mixed layer mixing ratio of CO2 and COS will be used (variable CO2surf vs CO2). The default is MXL (MXL will be used if the user does not specify anything for ags\_C\_mode), this will use the same mixing ratio as in the original model. When set to ‘surf’, make sure that sw\_sl is set to True.
* An additional input variable ‘alfa\_sto’ can be specified, this scales the stomatal conductance with a specified factor. If no value is given, a default value of 1.0 will be used, which gives the same conductance as in the original model.
* In the surface layer part, The following three statements:

qsatsurf = qsat(self.thetasurf, self.Ps)

cq = (1. + self.Cs \* ueff \* self.rs) \*\* -1.

self.qsurf = (1. - cq) \* self.q + cq \* qsatsurf

Have been replaced by the following statement:

self.qsurf = self.q + self.wq / (self.Cs \* ueff)

* The model now allows to prescribe varying surface fluxes during the day (variables wtheta, wq , wCO2 and wCOS). For this, the switch sw\_ls should be set to False.
* The variable ‘itmax’ in the ‘statistics’ module has been increased from 30 to 50
* In the ‘run\_cumulus’ module, the following has been added:

If the variable self.q2\_h is equal or smaller than 0, it is set to a value of 1.e-200. The same holds for the variables ‘self.CO22\_h’ and ‘self.COS2\_h’.

* soilCOSmodel….
* Minor changes to the Cm variable, in the ‘exitmodel’ function ‘self.Cm’ is only deleted if sw\_sl is set to True, the statement self.Cm = 1e12 was removed from the ‘init’ function (The value would always be overwritten when Cm is actually used in the model), and the statement self.Cm = None was removed from the class ‘model\_input’. Also, as a minor change Cm will only be stored in the model output if sw\_sl is set to True.
* Variable sinlea is changed into self.sinlea, is stored as model output, initialised as None and added to the exitmodel function
* In a-gs, in the calculation of variable Ag, a minus sign has been added in front of alphac
* In a-gs, the calculation of fmin is corrected such that it matches eq A9 from Ronda et al. 2001.

## Newly added switches

‘**sw\_dynamicsl\_border**’ is a switch that determines whether the dynamic height of the surface layer is taken into account when calculating scalars at different heights. As an example, let us look at variable ‘Tmh’, which is the variable containing the temperature at height ‘Tmeasuring\_height’, e.g. temperature at 150m. If the surface layer height variable ‘zsl’ is larger than ‘Tmeasuring\_height’ (i.e. 150 m in this case), the model will always use surface layer theory to calculate Tmh, independent of the switch. However, if zsl < 150 m, the model behaviour depends on the switch sw\_dynamicsl\_border. If sw\_dynamicsl\_border = True, the model will use the mixed layer temperature value, if False it will use surface layer theory, even though the height for which we calculate the temperature is located above the surface layer. If the user does not set this switch in the model input, it is set to True.

‘**sw\_use\_ribtol**’ is a switch that determines whether to use the default calculation in CLASS for obtaining the Obukhov length or a more simple one. The default implementation involves an iterative calculation via the bulk Richardson number. The more simple implementation uses the following equation:

where is virtual potential temperature, is friction velocity, k is the Von Kármán constant (0.4), g is the gravitational acceleration, and is the surface kinematic virtual heat flux.

Setting sw\_use\_ribtol to True gives the original CLASS calculation, setting it to False gives the more simple calculation. If the user does not set this switch in the model input, it is set to True

‘**sw\_advfp**‘ is a switch that determines how advection is implemented, if set to True, the same amount of advection takes place both in the mixed layer and in the free troposphere, if set to False advection only takes place in the mixed layer. Setting this switch to False results in the original CLASS behaviour. If the user does not set this switch in the model input, it is set to False.

‘**sw\_printwarnings**’ is a switch that determines whether to print warnings if something notable happens in the forwardmodel, e.g. ‘LCL calculation not converged’. Some warnings have also been added compared to the original CLASS model, e.g. when the solar angle is very low. If the user does not set this switch in the model input, it is set to True.

‘**sw\_useWilson**’ is a switch that determines, for the surface layer calculations, whether the Businger-Dyer equations (Paulson 1970) or the formulation from Wilson (2001) are used in the psim and psih functions (for unstable conditions). Setting this switch to False gives the default CLASS behaviour. If the user does not set this switch in the model input, it is set to False.

Finally, ‘**sw\_model\_stable\_con**’ determines whether surface layer calculations also take place in unstable conditions, i.e. z/L (‘zeta’ in the model code) > 0. If set to False, the psih and psim function will return ‘nan’ (not a number) if z/L is larger than zero. Setting this switch to True results in the original CLASS behaviour. If the user does not set this switch in the model input, it is set to True.

# References

Paulson, C. A.: The Mathematical Representation of Wind Speed and Temperature Profiles in the Unstable Atmospheric Surface Layer, Journal of Applied Meteorology, 9(6), 1962-1982, <https://www.jstor.org/stable/26174934>, 1970

Ronda, R. J., de Bruin H. A. R. and Holtslag A. A. M.: Representation of the Canopy Conductance in Modeling the Surface Energy Budget for Low Vegetation, American Meteorological Society, 40(8), 1431-1444, http://www.jstor.org/stable/26184869, 2001

Vilà-Guerau De Arellano, J., Van Heerwaarden, C. C., Van Stratum, B. J., and Van Den Dries, K.: Atmospheric boundary layer: Integrating air chemistry and land interactions, Cambridge University Press, 2015

Wilson, D. K.: An alternative function for the wind and temperature gradients in unstable surface layers, *Boundary-Layer Meteorology*, *99*(1), 151–158, <https://doi.org/10.1023/A:1018718707419>, 2001