ICLASS 1.1 user manual

Peter Bosman

(peter.bosman.publicaddress@gmail.com)

Contents

1 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.

Note that the optimisation.py file is by default filled in to perform an optimisation with 14 state-vector elements using data from Cabauw. This merely serves as example, the user should adapt the input paragraphs as desired.

Similarly to optimisation.py, there is a file 'optimisation_OSSE.py' that is meant for observation system simulation experiments (also here, the user should adapt the input paragraphs as desired).

2 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!

2.1 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, pert_Hx_min_sy_ens = True, pert_obs_ens = False, print_status_dur_ens = False, imposeparambounds = True, paramboundspenalty = False, discard_nan_minims = False, abort_slow_minims = True, write_to_f = True, plot_errbars = 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, (part of) the sample will be discarded and (part of) 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:

$$penalty = \frac{1}{penalty\_exp}(2 - [x - bound\_l])^{penalty\_exp}$$

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

$$penalty = \frac{1}{penalty\_exp}(2 + [x - bound\_u])^{penalty\_exp}$$

Where $bound\_u$ 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 (not-a-number) 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.

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.

When the switch abort_slow_minims is set to True: 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 or equal than a specified criterion and the maximum number of restarts is not yet reached. (details in code, see min_func in inverse_modelling.py).

If ‘estimate_model_err’ is set to True, the variable ‘nr_of_members_moderr’ has to be set to an integer >= 2 (More information on model error in section 2.10).

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 can be read in again in the ‘postprocessing.py’ file. This way postprocessing can be done after the optimisations without having to redo the entire optimisation if variables are no longer in memory.

The user can choose wether the framework uses a truncated Newton (tnc; The SciPy community; Nash (2000)) method or the Broyden–Fletcher–Goldfarb–Shanno (BFGS, Nocedal(1999)) algorithm for the optimisations.

We strongly recommend to set optim_method = 'tnc' instead of ‘bfgs’ to get the best convergence. Setting optim_method = 'bfgs' does not allow for setting hard bounds on the state parameter values that will be used in the optimisation.

Variable ‘remove_prev’ is a switch that if set to True, will remove certain files from the directory. It will remove the following files if they exist:

- Every file with a filename starting with *Optimfile*
- Every file with a filename starting with *Gradfile*
- *Optstatsfile.txt*
- *Modelerrorfile.txt*
- Every file with a filename starting with *pdf_posterior*
- Every file with a filename starting with *pdf_nonstate_*
- Every file with a filename starting with *fig_fit*
- Every file with a filename starting with *fig_obs*
- Every file stored within the folder designated for Pickle objects (called *pickle_objects*)
- Every file with a filename starting with *pp_* (since the postprocessing.py script produces output starting with *pp_*)

2.2 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.

2.3 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.

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

Here the variables in the state have to be specified as a list. The possible variables are given in section 3. Also an ‘obsvarlist’ variable has to be defined which should be a list of CLASS variables for which we have observations. The possible variables for this are given in section 4. Optionally, non-model related prior information can be given here, i.e prior scaling factors or the prior value for the FracH parameter related to the energy balance closure. As an example, priorinput.obs_sca_cf_LE = 1.5, to scale the LE observations, with a prior value of the scaling factor equal to 1.5. Or priorinput.FracH = 0.6, if you want to use the FracH parameter and set its prior value to 0.6.

2.5 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 optionally 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 variance of ‘obs_sca_cf_LE’: priorvar[' obs_sca_cf_LE '] = 0.15**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.

2.6 parameter bounds

This section only needs to be filled in when imposeparambounds or paramboundspenalty is set to True (see section 2.1), in this section bounds for state parameters can be specified. It should be done in the following format:

boundedvars['sca_sto'] = [0.1,5] if imposeparambounds is True, this implies you wish variable 'sca_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.

You can also set a state parameter to have only a left bound or only a right bound. As an example: boundedvars['sca_sto'] = [0.1,None] leads to only a left bound for parameter 'sca_sto'.

2.7 observation information

The obligatory information to be specified here is the measurement error standard deviation for each observation, the observation variables to be used (which determines the model output variables to link to the observations one wants to use) 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 2.2. We use optim.obs_qmh here as example of an assigned observation variable. The observation variables that can be assigned should start with ‘optim.obs_’, and be combined with the name of any model variable for which model output is stored in the store function of CLASS (except ‘sinlea’ or ‘t’). In this example we chose qmh, resulting in the variable optim.obs_qmh. Other examples would be optim.obs_theta, optim.obs_H, optim.obs_rs, … (see section 4 for all possibilities, make sure to add ‘optim.obs_’ to the variable name).

In order for an observation stream to be eligible for inclusion in the observation vector, there are two requirements. Firstly, the observation stream should correspond to a variable in the stored output of CLASS (except ‘sinlea’ or ‘t’). Secondly, there should be model output available at the times of the observations.

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 in the observation fit and perturbed observations plots. This is done for the purpose of nicer plotting.
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 observations, the observation times, measurement error standard deviations and the weights have to be provided either as a list or as a numpy.ndarray. When within an observation stream, some observations have the value numpy.nan (a representation of 'not a number' data), those data points will be automatically discarded (as well as the corresponding entries in the measurement error standard deviation array etc, see code for details).

Note that in the cost function algorithm, observation and model times (converted to the unit of seconds) are rounded to 8 decimal places. This should be kept in mind when providing extremely precise observation times or when using very precise model output times.

2.8 units and displayed names 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 (for state variables and perturbed non-state variables if applicable) 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. The displayed names for the parameters can also be entered here, e.g.:

disp_nms_par['theta'] = r'θ'

2.9 energy balance information

This section is only relevant if 'FracH' 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 2.2 or 2.7). 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).

2.10 model and representation error

In this section, the representation error standard deviations and model error standard deviations can be specified. For the model error standard deviations there are two options, they 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 standard deviations 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': -0.6,'rightbound': 1.0}
me_paramdict['Lambda'] = {'distr':'normal','mean': 0.0,'scale': 0.3}
me_paramdict['z0m'] = {'distr':' triangular ','leftbound': -0.6,'mode': 0.5,'rightbound': 1.0}
me_paramdict['w2'] = {'distr':' bounded normal','mean': 0.0,'scale': 0.3,'leftbound': -0.4,'rightbound': 0.7}
```

For parameter cveg, a uniform distribution is specified in the example, in the half-open interval [-0.6,1.0) (includes -0.6, but excludes 1.0). For parameter Lambda, a normal distribution is specified with a mean of zero and a standard deviation of 0.3. For z0m, a triangular distribution between [-0.6,1.0] with the peak (mode) at 0.5 is specified. For w2 a bounded normal distribution is given in the interval [-0.4,0.7] (including both -0.4 and 0.7), with a mean of 0. The mean (0.0) and standard deviation (0.3) specified in the example are not the mean and standard deviation of the bounded normal distribution, but the mean and standard deviation of the distribution if no bounds would be present, i.e. those of the 'normal' normal distribution. Note that the user should specify the distributions of the random numbers to add to the parameters, not the distributions of the parameters themselves!

Those statements (those in case of ICLASS estimating the model error) have to be placed before the 'else:' statement in the input block. Note that the parameters to perturb in the model ensemble should not be part of the state (they will be deleted from me_paramdict in that case). In case the user does not want ICLASS to estimate the model error standard deviations, but prefers to specify the model error standard deviations himself, this can be done as follows (example for qmh):

```
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 standard deviations are specified or estimated for an observation variable, the model errors for this variable are set to zero. If no representation error standard deviations are specified for an observation variable, the representation errors for this variable will be set to zero. Similar to the model error, representation error standard deviations can be specified as (example for qmh):

```
repr_error['qmh'] = [0.3 for j in range(len(measurement_error['qmh']))]
```

2.11 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 non-state parameters to perturb in the ensemble of optimisations (not the model error ensemble!) and specify the distributions from which random numbers will be sampled to add to the parameters. As an example:

non_state_paramdict['cveg'] = {'distr':'uniform','leftbound': -0.6,'rightbound': 1.0}

In this example, parameter cveg has a random number added sampled from a uniform distribution in the half-open interval [-0.6,1.0) (includes -0.6, but excludes 1.0). The user has the choice between between a 'normal', 'bounded normal', 'uniform' or 'triangular' distribution for each parameter (see section 2.10 to see how to specify the normal, triangular or bounded normal distributions). Note that the user should specify the distributions of the random numbers to add to the parameters, not the distributions of the parameters themselves!

Note that when a parameter is specified in non_state_paramdict, and this parameter is already part of the state, the parameter will automatically be removed from non_state_paramdict.

2.12 additional plotting

This section is found at the end of the optimisation file, here the user can plot additional figures etc. However, note that there is also a file 'postprocessing.py' available for post-processing output data, that can be run after the optimisation has finished. This script should be adapted by the user to the optimisation performed and the output desired.

3 List of parameters that can be optimised

Model parameters that can be optimised:

- theta — initial mixed-layer potential temperature [K]
- q — initial mixed-layer specific humidity [kg kg-1]
- CO2 — initial mixed-layer CO2 [ppm]
- COS — initial mixed-layer COS [ppb]
- u — initial mixed-layer u-wind speed [m s-1]
- v — initial mixed-layer v-wind speed [m s-1]
- wtheta — (initial) surface kinematic heat flux [K m s-1]
- wq — (initial) surface kinematic moisture flux [kg kg-1 m s-1]
- wCO2 — (initial) surface total CO2 flux [mgCO2 m-2 s-1]
- wCOS — (initial) surface kinematic COS flux [ppb m s-1]
- deltatheta — initial temperature jump at h [K]
- deltaq — initial specific humidity jump at h [kg kg-1]
- deltaCO2 — initial CO2 jump at h [ppm]
- deltaCOS — initial COS jump at h [ppb]
- deltau — initial u-wind jump at h [m s-1]
- deltav — initial v-wind jump at h [m s-1]
- gammatheta — free atmosphere potential temperature lapse rate [K m-1]
- gammatheta2 — free atmosphere potential temperature lapse rate 2 [K m-1]
- gammaq — free atmosphere specific humidity lapse rate [kg kg-1 m-1]
- gammaCO2 — free atmosphere CO2 lapse rate [ppm m-1]
- gammaCOS — free atmosphere COS lapse rate [ppb m-1]
- gammau — free atmosphere u-wind speed lapse rate [s-1]
- gammav — free atmosphere v-wind speed lapse rate [s-1]
- Cs — drag coefficient for scalars [-]
- ustar — surface friction velocity [m s-1]
- h — initial ABL height [m]
- wg — volumetric water content top soil layer [m3 m-3]
- Tsoil — temperature top soil layer [K]
- Ts — surface temperature [K]
- Wl — equivalent water layer depth for wet vegetation [m]
- dz_h — Transition layer thickness [m]
- gciCOS — Internal conductance for COS [m s-1]
- COSmeasuring_height — height 1 COS mixing ratio measurements [m]
- divU — horizontal large-scale divergence of wind [s-1]
- w2 — volumetric water content deeper soil layer [m3 m-3]
- wfc — volumetric water content field capacity [-]
- wwilt — volumetric water content wilting point [-]
- wsat — saturated volumetric water content ECMWF config [-]
- beta — entrainment ratio for virtual heat [-]
- z0m — roughness length for momentum [m]
- z0h — roughness length for scalars [m]

Continued next page

- lat — latitude [deg]
- lon — longitude [deg]
- doy — day of the year [-]
- cc — cloud cover fraction [-]
- alpha — surface albedo [-]
- fc — coriolis parameter [s-1]
- dFz — cloud top radiative divergence [W m-2]
- cveg — vegetation fraction [-]
- LAI — leaf area index [-]
- T2 — temperature deeper soil layer [K]
- a — Clapp and Hornberger retention curve parameter a [-]
- b — Clapp and Hornberger retention curve parameter b [-]
- p — Clapp and Hornberger retention curve parameter p [-]
- CGsat — saturated soil conductivity for heat
- C1sat — Coefficient force term moisture [-] (p131 Vilà-Guerau De Arellano et al 2015)
- C2ref — Coefficient restore term moisture [-]
- gD — correction factor transpiration for VPD [-]
- rsmin — minimum resistance transpiration [s m-1]
- rssoilmin — minimum resistance soil evaporation [s m-1]
- Wmax — thickness of water layer on wet vegetation [m]
- Lambda — thermal diffusivity skin layer [-]
- PARfract — fraction of incoming shortwave radiation that is PAR[-]
- R10 — respiration at 10 °C [mg CO2 m-2 s-1]
- E0 — activation energy (for respiration calc) [kJ kmol-1]
- sca_sto — Scaling factor for stomatal conductance [-]
- Q — net radiation [W m-2] (not relevant if sw_rad==True)
- advtheta — advection of heat [K s-1]
- advq — advection of moisture [kg kg-1 s-1]
- advCO2 — advection of CO2 [ppm s-1]
- advCOS — advection of COS [ppb s-1]
- advu — advection of u-wind [m s-2]
- advv — advection of v-wind [m s-2]

Non-model parameters that can be optimised (parameters from bias correction scheme):

- FracH — Fraction of energy balance gap partitioned to H obs [-]
- obs_sca_cf_x (replace x by an observation variable, e.g. H) — Scaling factor for the observations of x in the cost function [-]

4 Observation variables that can be used

The following list contains the variables that can in principle be used to assign observations to. Note that the availability of these variables also depends on the configuration of the model.

h	ABL height [m]
theta	mixed-layer potential temperature [K]
thetav	mixed-layer virtual potential temperature [K]
deltatheta	potential temperature jump at h [K]
deltathetav	virtual potential temperature jump at h [K]
wtheta	surface kinematic heat flux [K m s-1]
wthetav	surface kinematic virtual heat flux [K m s-1]
wthetae	entrainment kinematic heat flux [K m s-1]
wthetave	entrainment kinematic virtual heat flux [K m s-1]
q	mixed-layer specific humidity [kg kg-1]
deltaq	initial specific humidity jump at h [kg kg-1]
wq	surface kinematic moisture flux [kg kg-1 m s-1]
wqe	entrainment kinematic moisture flux [kg kg-1 m s-1]
wqM	cumulus mass-flux kinematic moisture flux [kg kg-1 m s-1]
qsatvar	mixed-layer saturated specific humidity [kg kg-1]
e	mixed-layer vapor pressure [Pa]
esatvar	mixed-layer saturated vapor pressure [Pa]
CO2	mixed-layer CO2 [ppm]
deltaCO2	CO2 jump at h [ppm]
wCO2	surface total CO2 flux [mgCO2 m-2 s-1]
wCO2A	surface assimilation CO2 flux [mgCO2 m-2 s-1]
wCO2R	surface respiration CO2 flux [mgCO2 m-2 s-1]
wCO2e	entrainment CO2 flux [mgCO2 m-2 s-1]
wCO2M	CO2 mass flux [mgCO2 m-2 s-1]
COS	mixed-layer COS [ppb]
deltaCOS	mixed-layer COS jump at h [ppb]
wCOS	COS surface flux [ppb m s-1]
u	mixed-layer u-wind speed [m s-1]
deltau	u-wind jump at h [m s-1]
uw	surface momentum flux u [m2 s-2]
v	mixed-layer v-wind speed [m s-1]
deltav	v-wind jump at h [m s-1]
vw	surface momentum flux v [m2 s-2]
T2m	2m temperature [K]
q2m	2m specific humidity [kg kg-1]
u2m	2m u-wind [m s-1]
v2m	2m v-wind [m s-1]
e2m	2m vapor pressure [Pa]
esat2m	2m saturated vapor pressure [Pa]
thetamh	pot temperature at measuring height 1 [K]
thetamh2	pot temperature at measuring height2 [K]
thetamh3	pot temperature at measuring height3 [K]
thetamh4	pot temperature at measuring height4 [K]

thetamh5	pot temperature at measuring height5 [K]
thetamh6	pot temperature at measuring height6 [K]
thetamh7	pot temperature at measuring height7 [K]
Tmh	temperature at measuring height 1 [K]
Tmh2	temperature at measuring height2 [K]
Tmh3	temperature at measuring height3 [K]
Tmh4	temperature at measuring height4 [K]
Tmh5	temperature at measuring height5 [K]
Tmh6	temperature at measuring height6 [K]
Tmh7	temperature at measuring height7 [K]
COSmh	COS at measuring height 1 [ppb]
COSmh2	COS at measuring height 2 [ppb]
COSmh3	COS at measuring height 3 [ppb]
COSmh4	COS at measuring height 4 [ppb]
CO2mh	CO2 at measuring height 1 [ppm]
CO2mh2	CO2 at measuring height 2 [ppm]
CO2mh3	CO2 at measuring height 3 [ppm]
CO2mh4	CO2 at measuring height 4 [ppm]
COS2m	COS at 2m height [ppb]
CO22m	CO2 at 2m height [ppm]
COSsurf	COS at the surface [ppb]
CO2surf	CO2 at the surface [ppm]
Tsurf	Temp at the surface [K]
qmh	specific humidity at measuring height 1[kg kg-1]
qmh2	specific humidity at measuring height 2[kg kg-1]
qmh3	specific humidity at measuring height 3[kg kg-1]
qmh4	specific humidity at measuring height 4[kg kg-1]
qmh5	specific humidity at measuring height 5[kg kg-1]
qmh6	specific humidity at measuring height 6[kg kg-1]
qmh7	specific humidity at measuring height 7[kg kg-1]
Cm	drag coefficient for momentum [-]
thetasurf	surface potential temperature [K]
thetavsurf	surface virtual potential temperature [K]
qsurf	surface specific humidity [kg kg-1]
ustar	surface friction velocity [m s-1]
Cs	drag coefficient for scalars [-]
L	Obukhov length [m]
Rib	bulk Richardson number [-]
Swin	incoming short wave radiation [W m-2]
Swout	outgoing short wave radiation [W m-2]
Lwin	incoming long wave radiation [W m-2]
Lwout	outgoing long wave radiation [W m-2]
Q	net radiation [W m-2]
ra	aerodynamic resistance [s m-1]
rs	surface resistance [s m-1]
H	sensible heat flux [W m-2]
LE	evapotranspiration [W m-2]
LEliq	open water evaporation [W m-2]
LEveg	transpiration [W m-2]
LEsoil	soil evaporation [W m-2]
LEpot	potential evaporation [W m-2]

LEref	reference evaporation at rs = rsmin / LAI [W m-2]
G	ground heat flux [W m-2]
Ts	Skin temperature [K]
zlcl	lifting condensation level [m]
RH_h	mixed-layer top relative humidity [-]
ac	cloud core fraction [-]
M	cloud core mass flux [m s-1]
dz	transition layer thickness [m]

5 Technical notes output of ICLASS

ICLASS produces several output files when switch write_to_f is set to True. Some technical notes on the output here:

- The Numpy functions cov and corrcoef are used for the construction of the estimated posterior error covariance and correlation matrices repectively. For the construction of the covariance matrix, we specify one degree of freedom for the normalisation (ddof=1).
- The File 'Optstatsfile.txt' contains information about the obtained solution. Several statistics can be found here (chi squared, root mean squared error,…), see also the reference paper.
- The file 'Optimfile.txt' contains, for every model simulation in the iterative optimisation process, the parameter values used in this simulation. It also lists the value of the cost function, split up into a data and a background part. When ICLASS is run in ensemble mode, a separate file is stored for every ensemble member, with the member number in the file name, starting at index 0 (the member with unperturbed prior).
- Similarly to the previous, the file 'Gradfile.txt' contains for the cost function gradient calculations the parameter values used as well as the derivatives of the cost function with respect to every state parameter. The derivatives of the background part are also provided separately. As for the previous, every ensemble member has its own file.
- When the model error standard deviations are estimated by ICLASS, there is a file containing statistics on the estimated model error standard deviations, namely time-mean, median, min and max, as well as the number of (non-nan) ensemble members used in the model-error ensemble.
- By default, ICLASS will produce figures that show the fit of both the prior and posterior runs to the observations. If an ensemble is used, figures with both the prior and posterior probability density functions of state parameters (and prior pdfs of perturbed non-state params, if applicable) can also automatically be produced by ICLASS.
- In the automatically generated observation fit plots, the observations that are labelled as 'obs' are the observations of ensemble member 0.
- When 'FracH' is included in the state, separate file(s) will be written showing the fit of both the prior and posterior runs to the H and/or LE observations corrected using the optimised 'FracH' parameter (of the ensemble member with the lowest posterior cost function). The other automatically generated observation fit plots of H and/or LE do not account for the FracH parameter.
- The perturbed observations of each ensemble member will be plotted when plot_perturbed_obs is set to True. Those plotted observations are not corrected using observation scales (see reference paper) or using the FracH parameter.
- 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 standard deviations are 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, switch pert_obs_ens) would be used to calculate the cost function. Similarly, if not the observations themselves but the model-data mismatch is perturbed (switch pert_Hx_min_sy_ens), these perturbations differ along ensemble members and thus influence the cost function. If non-

state parameters are perturbed, than keep in mind that those parameters also differ along members and influence the cost function.

- In Optstatsfile.txt, ‘best state’ means the posterior state of the ensemble member with the lowest posterior cost function (if ensemble used, otherwise there is only 1 optimisation). Sometimes we also use ‘optimal state’ for this.
- In Optstatsfile.txt, the mean bias error, root mean squared error and ratio of model and obs variance use the observations scaled with observation scales (if used). In case the 'FracH' parameter is used, the energy balance corrected observations will be used (see also reference paper, and the remark below). Only model output at the times of observations is used in these calculations.
- Note that, when using an ensemble, there are multiple prior and posterior states. The posterior mean bias error, posterior root mean squared error and the posterior ratio of model and obs variance calculations use the parameters of the best state (no perturbations in non-state params, obs or scaled obs with respect to member 0), which is defined here as the posterior state of the ensemble member that resulted in the lowest posterior cost function (if ensemble used, otherwise there is only 1 optimisation). For the prior mean bias error etc. however, we use the unperturbed prior state, i.e. the prior state of member 0. This also means e.g. that the observations used for the prior mean bias error, are scaled with the prior observation scales of member 0 (if applicable). Also for the prior mbe, rmse and ratio of model and obs variance calculations, we do not apply perturbations in non-state params, obs or scaled obs with respect to member 0.
- In Optstatsfile.txt, if an ensemble is used and the member with the lowest posterior cost function is NOT member 0, there will be an additional calculation of cost-function parts, using the best state (see earlier), but with everything else from member 0, i.e. no perturbations in the data part of the cost function, no perturbations in the obs with respect to member 0, and no perturbations in non-state parameters. The prior state used in the background part of the cost function (if applicable) will be the prior state from member 0. This calculation of cost-function parts gets the header ‘costf parts best state with obs, prior and non-state parameters of member 0:’. This extra output is written besides the normal output with the cost-function parts of the best state using the observations, perturbations in the data part of the cost function (if applicable), prior and non-state parameters of the member with the lowest posterior cost function.
- In the calculation of ensemble-based statistics, i.e. posterior covariance and correlation matrices (with and without perturbed non-state parameters), post/prior variance ratio in the ensemble and the mean posterior state, member 0 (the member with an unperturbed prior) is excluded from the calculations. Also, these calculations are based only on successful ensemble members (see reference paper for criterion of successful).
- The correlation and covariance matrices in the output are marginal correlation and covariance matrices respectively, i.e. a correlation between two posterior parameters might involve a third parameter that correlates with both (in contrast to partial correlations).

6 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 1 October 2019 on GitHub (https://github.com/classmodel/modelpy, commit 2bdb7ea) are shown. This list might not be fully complete. For more details on CLASS itself, see Vilà-Guerau De Arellano et al. (2015).

6.1 List of changes

- COS (carbonyl sulphide; Whelan et al., 2018) has been added. See section 6.3 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_useWilson' and 'sw_model_stable_con'. Those are explained in section 6.2.
- Warning statements have been added for the cases that variables ueff < 0.01 or sinlea < 0.0001 (only active if sw_printwarnings set to True).
- 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.
- If one of the measurement heights (e.g. Tmeasuring_height, CO2measuring_height4) is smaller than z0h, an exception will be raised.
- Some model parameters that could not be set via model input in the original model, can now be set via 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 (for use in the analytical derivative). 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' function: '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, and 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.
- The constructor of the class 'model_output' takes an additional argument '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. Also, there is some model code involving a soil COS model. As this soil COS model is not implemented in this version of ICLASS, in the model input no value should be given for the variable 'soilCOSmodeltype', or the value should be set to None.
- 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.
- The variable f2 has been renamed to f2js in the jarvis_stewart module
- The variable *Ts* is now stored as model output
- 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, or no value is specified as model input, 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 as free atmospheric potential temperature lapse rate instead of 'gammatheta'.
- The initial value of the variable 'Cs' can now be set via 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 ‘surf’ or ‘MXL’, this determines whether in ags the surface or the mixed-layer mixing ratio of CO_2 and COS will be used (variable CO2surf vs CO2 and COSsurf vs COS). 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 ‘sca_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, and a time series of fluxes should be provided as model input. The variables to specify as model input are *wtheta_input* (in case you want to prescribe potential temperature kinematic surface flux), *wq_input* (in case you want to prescribe specific humidity kinematic surface flux), *wCO2_input* (in case you want to prescribe CO2 surface flux) and *wCOS_input* (in case you want to prescribe COS surface flux).
- 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’.
- 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.
- A statement self.sinlea = sinlea is added, and the self.sinlea variable is stored as model output, initialised as None and added to the exitmodel function
- In a-gs, in the calculation of *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 (difference is that -fmin0 is now also divided by 2*gm).
- In integrate_land_surface, there is a slight change in the order of statements, which does not matter for the model output
- The statement *self.out.wCO2M[t] = self.wCO2M ** fac is added to the store module
- The statement self.L = None is removed from class ‘model_input’.
- In the ribtol function, a variable *it* is added that keeps track of the number of iterations in the while loop

- os.path imported near the beginning of the file
- Added two statements that define the molecular weights of water and carbonyl sulphide
- Variables qsat and esat renamed into qsatvar and esatvar respectively
- Some changes to comments, order of statements etc., that do not influence model output.

6.2 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. The variables 'T2m', 'q2m', 'COS2m', 'CO22m', 'u2m', 'v2m', 'esat2m' and 'e2m' will always be calculated using surface layer theory, independent of the switch 'sw_dynamicsl_border'.

'**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:

$$L = \frac{-1 * \theta_v * {u_{star}}^3}{k * g * \overline{w'\theta_v'}}$$

where $\theta_v$is mixed-layer virtual potential temperature [K], u_{star} is friction velocity [m s^{-1}], k is the Von Kármán constant (0.4), g is the gravitational acceleration [m s^{-2}], and $\overline{w'\theta_v}$ is the surface kinematic virtual heat flux [K m s^{-1}].

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 stable 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.

6.3 Simple COS implementation

The implementation of COS in CLASS is similar to the implementation of CO_2 (which was already in CLASS), with the exception of exchange with the land surface. For this exchange, a canopy-scale conductance for COS [m s^{-1}] is calculated as

$$gCOS_t = \frac{1}{\frac{1}{\mathrm{gCOS}_i} + \frac{1.21}{\mathrm{gCO2}_s}}$$

Where gCOS_i [m s^{-1}] is the internal conductance for COS and $\mathrm{gCO2}_s$ [m s^{-1}] is the stomatal conductance for CO2. The factor 1.21 is taken from Seibt et al. (2010). The flux of COS into the vegetation [ppb m s^{-1}] is than calculated as:

$$FCOS_p = \frac{-1}{\frac{1}{\mathrm{gCOS}_t} + ra} * [COS]$$

Where ra is the aerodynamic resistance [s m^{-1}], which is also used in the calculation of the CO_2 flux into the vegetation. [COS] is either the mixing ratio of COS in the mixed layer or at the surface (in ppb), depending on switch 'ags_C_mode' (see section 6.1). The soil flux of COS is taken as zero at the moment. The current implementation of COS is thus basic, a future paper might follow in which ICLASS will be used for studying (part of) the coupled budgets of COS and CO_2, with an improved COS representation.

7 References

Nash, S. G.: A survey of truncated-Newton methods, Journal of Computational and Applied Mathematics, 124, 45–59, https://doi.org/10.1016/S0377-0427(00)00426-X, 2000.

Nocedal, J. and Wright, S. J.: Numerical Optimization, Springer-Verlag, New York, 1999.

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.

Seibt, U., Kesselmeier, J., Sandoval-Soto, L., Kuhn, U., and Berry, J. A.: A kinetic analysis of leaf uptake of COS and its relation to transpiration, photosynthesis and carbon isotope fractionation, Biogeosciences, 7, 333–341, https://doi.org/10.5194/bg-7-333-2010, 2010.

The SciPy community: scipy.optimize.fmin_tnc, https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin_tnc.html.

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.

Whelan, M. E., Lennartz, S. T., Gimeno, T. E., Wehr, R., Wohlfahrt, G., Wang, Y., Kooijmans, L. M., Hilton, T. W., Belviso, S., Peylin, P., Commane, R., Sun,W., Chen, H., Kuai, L., Mammarella, I., Maseyk, K., Berkelhammer, M., Li, K. F., Yakir, D., Zumkehr, A., Katayama, Y., Oge, J., Spielmann, F. M., Kitz, F., Rastogi, B., Kesselmeier, J., Marshall, J., Erkkila, K. M., Wingate, L., Meredith, L. K., He, W., Bunk, R., Launois, T., Vesala, T., Schmidt, J. A., Fichot, C. G., Seibt, U., Saleska, S., Saltzman, E. S., Montzka, S. A., Berry, J. A., and Elliott Campbell, J.: Reviews and syntheses: Carbonyl sulfide as a multi-scale tracer for carbon and water cycles, Biogeosciences, 15, 3625–3657, https://doi.org/10.5194/bg-15-3625-2018, 2018.

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.