pyseaflux

Release 2.2.2.dev0+g2cd577c.d20211008

Luke Gregor

Oct 08, 2021

CONTENTS

pySeaFlux is a Python 3.6+ package that can be used to calculate air-sea CO2 fluxes for the global ocean. The pySeaFlux package is a companion package to the pySeaFlux dataset that is under review in ESSD discussions (https://doi.org/10.5194/essd-2021-16).

pySeaFlux provides high level code to access the datasets presented in the ESSD manuscript. Further, the package provides the code to calculate these data sets.

CHAPTER

ONE

INSTALLATION

```
# for latest release
pip install pyseaflux

# for bleeding-edge up-to-date commit
pip install -e git+https://github.com/lukegre/pySeaFlux.git
```

1.1 The SeaFlux dataset

The SeaFlux data set allows for the homogenization of air-sea CO_2 flux calculations. To minimize the potential differences that may arise during calculation, we've made a product available at 1°1° by monthly resolution for 1988-2018. The product can be accessed and downloaded from https://doi.org/10.5281/zenodo.4133802.

Here, we'll show you how to download the data with Python and quickly calculate air-sea CO_2 fluxes when you only have pCO_2 based on the bulk flux formulation:

$$FCO_2 = K_0 \cdot k_w \cdot (pCO_2^{sea} - pCO_2^{atm}) \cdot ice^{free}$$

Where K_0 is the solubility of CO_2 in seawater, k_w is the gas transfer velocity using the formulation of Wanninkhof (1992) that has a square dependence on wind speed (second moment of the wind), pCO_2^{sea} is the oceanic partial pressure of carbon dioxide in the surface ocean, pCO_2^{atm} is the atmospheric partial pressure of carbon dioxide in the marine boundary layer, ice^{free} is the fraction of open ocean in a particular grid cell.

Here we tackle the differences that arise in each of these components. In this notebook, we'll detail how to implement these corrections using Python.

[2]:
```
%pylab inline

# for the rest of the document seaflux is referred to as sf
import pyseaflux as sf
import xarray as xr
```

```
Populating the interactive namespace from numpy and matplotlib
```

1.1.1 pCO2 area coverage

This correction applies to data-based surface pCO_2^{sea} products that provide pseudo-global coverage. However, due to various implementations and predictor variables used by each of these data-based products, there is a difference in the coverage of these products. This is true primarily for the coastal ocean and seasonally ice covered regions.

Recent work by Landschützer et al. (2020) provides a monthly pCO_2^{sea} climatology that is available for open ocean, coastal ocean, and seasonally ice-covered seas. By scaling this product, SeaFlux is able to provide a data product that can fill pCO_2^{sea} for the pseudo-global data-based products.

For details on this process, please see Fay et al. (2021).

Open example data set

I use the CSIR-ML6 product to demonstrate the data filling procedure. I have pre-downloaded the data into the folder `~/Downloads/`.

I use `xarray.mfdataset` to open the file. This allows for a preprocessor function to be passed. The SeaFlux package includes a `preprocess` function that tries to conform the data to the SeaFlux data.

[4]:
```
csir_fname = '~/Downloads/CSIRML6_CO2_19822019_figshare_v2020.6.nc'

preprocessor = sf.data.utils.preprocess()
csir = xr.open_mfdataset(csir_fname, preprocess=preprocessor)

csir_spco2 = csir.spco2
```

Download scaled climatology

The scaled climatology is downloadable directly at Zenodo (https://doi.org/10.5281/zenodo.4133802).
However, by using the SeaFlux tool to download the data you can be sure that you'll always have the latest version of the data.
The function downloads the data and returns it as an `xr.Dataset` which provides a useful interface to view the data.

Note that the default download location is set to `~/Downloads/SeaFlux_v2021.01`. This will create a folder in your downloads folder. This can be changed to any other folder. In addition to downloading the data, a README.txt file will be created in the given directory alongside a downloading log.

[5]:
```
sf_data = sf.get_seaflux_data(verbose=True)
```
```
2021-04-11 16:09:08 [DOWNLOAD] ␣
↪================================================================================


2021-04-11 16:09:08 [DOWNLOAD]  Start of logging session
2021-04-11 16:09:08 [DOWNLOAD]  ------------------------------------------------------------
↪----------------------
2021-04-11 16:09:08 [DOWNLOAD]    5 files at https://zenodo.org/record/4664653/files/
↪SeaFluxV2021.01_solWeis74.nc
2021-04-11 16:09:08 [DOWNLOAD]  Files will be saved to /Users/luke/Downloads/SeaFlux_
↪v2021.01
2021-04-11 16:09:08 [DOWNLOAD]  retrieving https://zenodo.org/record/4664653/files/
↪SeaFluxV2021.01_solWeis74.nc
```

(continues on next page)

(continued from previous page)

```
2021-04-11 16:09:08 [DOWNLOAD]  retrieving https://zenodo.org/record/4664653/files/
↪SeaFluxV2021.01_icefrac_1988-2018.nc
2021-04-11 16:09:08 [DOWNLOAD]  retrieving https://zenodo.org/record/4664653/files/
↪SeaFluxV2021.01_kwScaled16.5cmhr_1988-2018.nc
2021-04-11 16:09:08 [DOWNLOAD]  retrieving https://zenodo.org/record/4664653/files/
↪SeaFluxV2021.01_pCO2atm_NOAAmbl%2BERA5mslp_1988-2018.nc
2021-04-11 16:09:08 [DOWNLOAD]  retrieving https://zenodo.org/record/4664653/files/
↪SeaFluxV2021.01_spCO2filled_1988-2018.nc
2021-04-11 16:09:08 [DOWNLOAD]  SUMMARY: Retrieved=5, Failed=0 listing failed below:
```

[7]:
```
sf_data
```

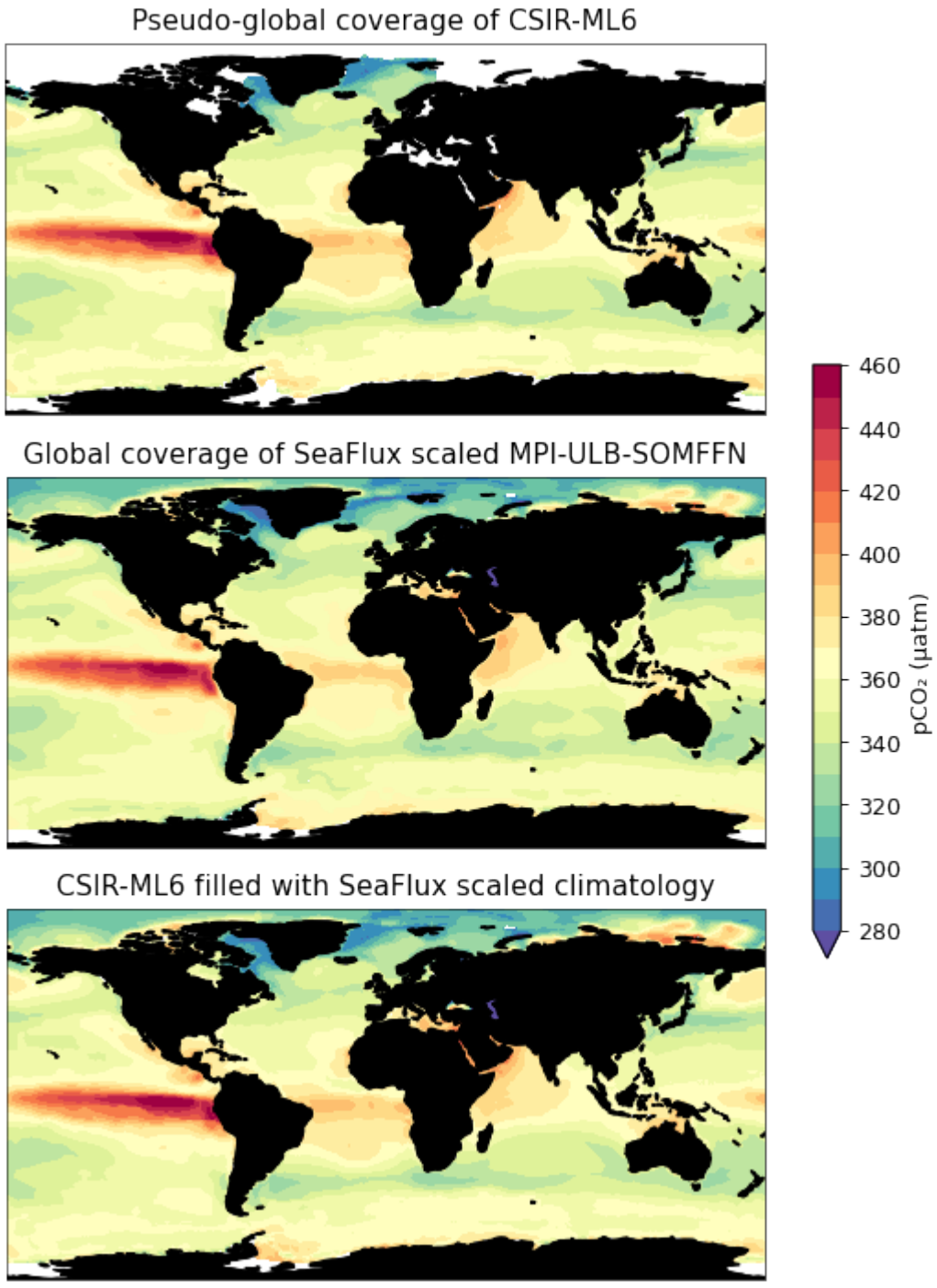
[7]:
```
<xarray.Dataset>
Dimensions:            (lat: 180, lon: 360, product: 6, time: 372, wind: 5)
Coordinates:
  * lat                (lat) float64 -89.5 -88.5 -87.5 -86.5 ... 87.5 88.5 89.5
  * lon                (lon) float64 -179.5 -178.5 -177.5 ... 177.5 178.5 179.5
  * time               (time) datetime64[ns] 1988-01-15 ... 2018-12-15
  * wind               (wind) object 'CCMP2' 'ERA5' 'JRA55' 'NCEP1' 'NCEP2'
  * product            (product) object 'MPI_SOMFFN' 'JMA_MLR' ... 'CSIR_ML6'
Data variables:
    ice                (time, lat, lon) float32 dask.array<chunksize=(372, 180, 360),␣
↪meta=np.ndarray>
    kw_scaled          (wind, time, lat, lon) float64 dask.array<chunksize=(5, 372, 180,␣
↪360), meta=np.ndarray>
    kw_a_scaling       (wind) float64 dask.array<chunksize=(5,), meta=np.ndarray>
    ice_frac           (time, lat, lon) float32 dask.array<chunksize=(372, 180, 360),␣
↪meta=np.ndarray>
    pCO2atm            (time, lat, lon) float64 dask.array<chunksize=(372, 180, 360),␣
↪meta=np.ndarray>
    spco2_filled       (product, time, lat, lon) float64 dask.array<chunksize=(6, 372,␣
↪180, 360), meta=np.ndarray>
    spco2_clim_scaled  (time, lat, lon) float64 dask.array<chunksize=(372, 180, 360),␣
↪meta=np.ndarray>
    scaling_factor     (time) float64 dask.array<chunksize=(372,), meta=np.ndarray>
    product_mask       (product, time, lat, lon) bool dask.array<chunksize=(6, 372, 180,␣
↪360), meta=np.ndarray>
    sol_Weiss74        (time, lat, lon) float64 dask.array<chunksize=(372, 180, 360),␣
↪meta=np.ndarray>
```

Filling the example data

Rather than creating a whole new way of doing things, SeaFlux leverages `xarray` functionality. The `fillna` method will fill missing data with the given input, which is exactly what we have to do! For this to work, the dimensions of both data arrays need to be the same, *i.e.*, the time, lat and lon have to have the same values.

[8]:
```
csir_filled = csir_spco2.fillna(sf_data.spco2_clim_scaled)
```

The figure below shows average pCO2 for the unfilled (CSIR), filler (MPI-ULB-SOMFFN scaled), and filled products (unfilled + filler).


Pseudo-global coverage of CSIR-ML6
Global coverage of SeaFlux scaled MPI-ULB-SOMFFN
CSIR-ML6 filled with SeaFlux scaled climatology
460
440
420
400
380
360
340
320
300
280
pCO₂ (µatm)

1.1.2 Fluxes

Gas transfer velocity: Here, for the sake of simplicity, we choose to use k_w that has been calculated with the ERA5 winds only. However, in the product, we have calculated k_w for the following wind products: CCMP, ERA5, JRA55, NCEP R1/2. The k_w for this dataset is scaled to a global value of 16.5 cm hr^{-1} over the period 1988-2017 (30 years). The period is chosen as all wind products overlap.

Solubility: Solubility is calculated using the Weiss (1980) formulation, with the coefficients from the table in Wanninkhof (2014). The other variables used are sea surface temperature and salinity.

Atmospheric pCO2: Atmospheric $p\mathrm{CO}_2$ for the marine boundary layer is calculated from the NOAAs marine boundary layer $x\mathrm{CO}_2$ product (https://www.esrl.noaa.gov/gmd/ccgg/trends/): $x\mathrm{CO}_2 * (P_{atm} - p\mathrm{H}_2\mathrm{O})$. Where P_{atm} is the ERA5 mean sea level pressure. $p\mathrm{H}_2\mathrm{O}$ is calculated using vapour pressure from Dickson et al. (2007).

Ice cover and sea surface temperature: We use the NOAA OI.v2 SST monthly fields. These are derived by a linear interpolation of the weekly optimum interpolation (OI) version 2 fields to daily fields then averaging the daily values over a month. The monthly fields are in the same format and spatial resolution as the weekly fields. The ice field shows the approximate monthly average of the ice concentration values input to the SST analysis. Ice concentration is stored as the percentage of area covered. For the ice fields, the land and coast grid cells have been set to the netCDF missing value. (ftp://ftp.cdc.noaa.gov/Datasets/noaa.oisst.v2/)

Salinity: We use the EN4.2.1 salinity for the shallowest level. Specifically, we use the objective analyses data with the Gouretski and Reseghetti (2010) corrections. (https://www.metoffice.gov.uk/hadobs/en4/download-en4-2-1.html)

[10]:
```
# download flux data - verbosity is False by default
sf_data = sf.data.get_seaflux_data()

ds = sf_data.sel(wind='ERA5').drop('wind')
# use the SeaFlux function to calculate area
ds['area'] = sf.get_area_from_dataset(ds)

# assigning variables and performing unit transformations
kw = ds.kw_scaled * (24/100)      # cm/hr --> m/d
K0 = ds.sol_Weiss74               # mol/m3/uatm
dpco2 = csir_filled - ds.pCO2atm  # uatm
ice_free = 1 - ds.ice.fillna(0)   # fill the open ocean with zeros
area = ds.area                    # m2
```

[11]:
```
# bulk flux calculation
flux_mol_m2_day = kw * K0 * dpco2 * ice_free
flux_avg_yr = flux_mol_m2_day.mean('time') * 365  # molC/m2/year
flux_integrated = (flux_mol_m2_day * area * 365 * 12.011).sum(dim=['lat', 'lon'])  # gC/
↪year
```

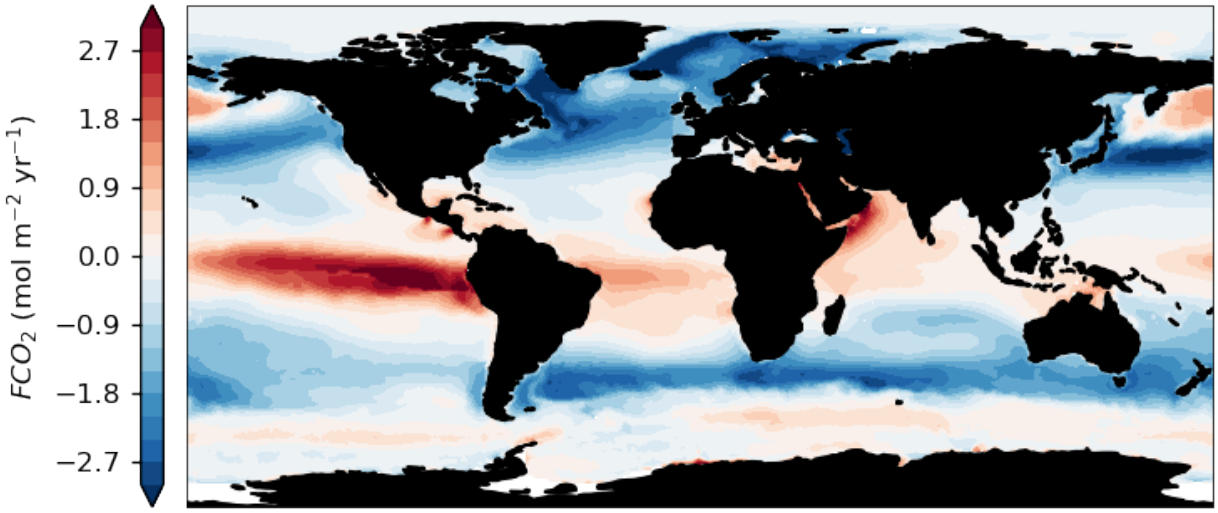
1.1.3 Unit analysis

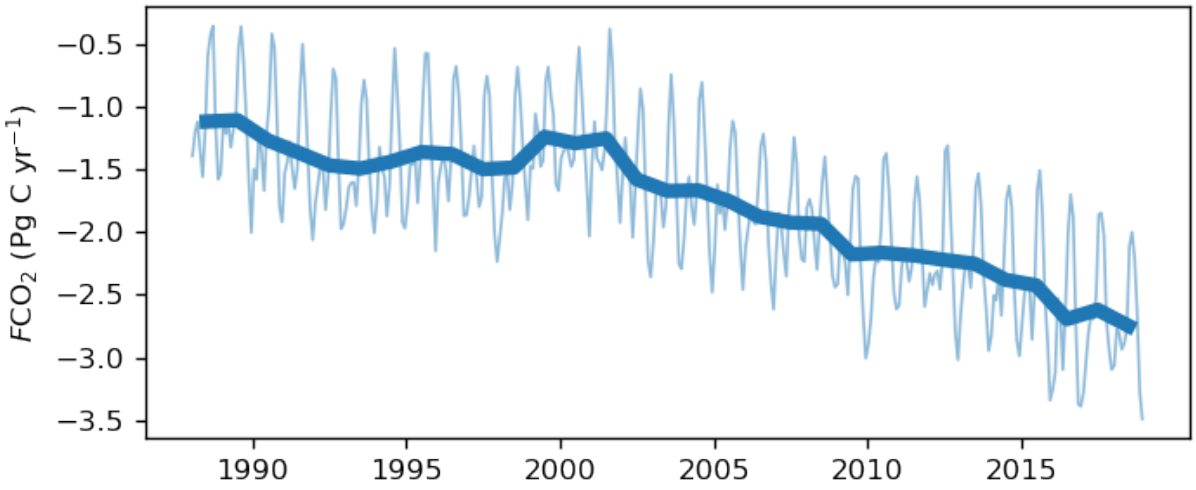
The SeaFlux product provides the remaining parameters to calculate fluxes. This means that the calculation can be a simple multiplication. However, units have to be taken care of. Below we show a table of the units for each of the SeaFlux variables used in the bulk flux calculation.

Variable	SeaFlux units	Transformation	Output units
ΔpCO_2	µatm		µatm
K_0	mol m^{-3} µatm^{-1}		mol m^{-3} µatm^{-1}
k_w	cm hr^{-1}	$\times \frac{24}{100}$	m day^{-1}
ice^{conc}	ice fraction	$1 - ice^{conc}$	ice free fraction
area	m^2		m^2
BULK FLUXES			
FCO_2		$K_0 \cdot k_w \cdot \Delta pCO_2 \cdot$ ice	molC m^{-2} d^{-1}
INTEGRATION			
FCO_2^{int}	molC m^{-2} d^{-1}	$\times$ (m$^2 \cdot$ 12.01 g mol^{-1} $\cdot$ 365 d yr^{-1})	gC yr^{-1}

The figure below shows the output for the fluxes calculated using the SeaFlux data. Notice that the ice covered regions, particularly the Arctic, have low air-sea CO_2 fluxes. The output has been multiplied by 365 days/year to convert the flux to $molC\ m^{-1}\ yr^{-2}$.

The bottom figure shows the globally integrated air-sea CO_2 fluxes. Here, the fluxes have been multiplied by the area (m^2) and converted to $gC\ yr^{-1}$ (using the conversion shown in the table above).





1.2 Autogenerated API

1.2.1 High level functions

pyseaflux.flux_calculations.**flux_bulk**(*temp_C*, *salt*, *pCO2_sea_uatm*, *pCO2_air_uatm*, *pres_hPa*, *kw_cmhr*)

Calculates bulk air-sea CO2 fluxes

$$FCO_2 = k_w \cdot K_0 \cdot \Delta pCO_2$$

Parameters

- **temp_C** (*array*) – temperature from OISST in degCelcius with an allowable range of [-2:45]
- **salt** (*array*) – salinity from EN4 in PSU. Allowable range [5 : 50]
- **pCO2_sea_uatm** (*array*) – partial pressure of CO2 in the sea in micro-atmospheres. Allowable range is [50 : 1000]
- **pCO2_air_uatm** (*array*) – partial pressure of CO2 in the air in micro-atmospheres. Allowable range is [50 : 1000].
- **press_hPa** (*array*) – atmospheric pressure in hecto-Pascals with allowable range [500:1500]
- **kw_cmhr** (*array*) – the gas transfer velocity in (cm/hr). Given the careful choices involved in estimating kw, we require the user to explicitly provide kw. kw can be calculated with pyseaflux.gas_transfer_velocity.<func>. Things to be aware of when calculating kw: wind product and scaling coeffient of gas transfer, resolution resampling, and the formulation (i.e. quadratic, cubic).

Returns Sea-air CO2 flux where positive is out of the ocean and negative is into the ocean. Units are gC.m-2.day-1 (grams Carbon per metre squared per day). If the input is an xarray.DataArray, then the output will be a data array with fluxes, globally integrated flux, and the area used to integrate the fluxes.

Return type array

1.2.2 Conversions of fCO2 - pCO2

pyseaflux.fco2_pco2_conversion.**fCO2_to_pCO2**(*fCO2SW_uatm*, *tempSW_C*, *pres_hPa=1013.25*, *tempEQ_C=None*, *checks=True*)

Convert fCO2 to pCO2 in sea water.

If equilibrator temperature is provided, we get a simple approximate for equilibrator xCO_2 that allows for the virial expansion to be calculated more accurately. If not, then a simple approximation is good enough. See the examples for the differences.

$$pCO_2^{sw} = fCO_2^{sw} \div virial(xCO_2^{eq})$$

where $xCO_2^{eq} = fCO_2^{sw} \times \Delta T^{(sw-eq)} \div P^{eq}$

Parameters

- **fCO2SW_uatm** (*array*) – seawater fugacity of CO2 in micro atmospheres
- **tempSW_C** (*array*) – sea water temperature in degrees C
- **pres_hPa** (*array, optional*) – equilibrator pressure in hecto Pascals. Defaults to 1013.25.

- **tempEQ_C** (*array, optional*) – equilibrator temperature in degrees C. Defaults to None.

Returns partial pressure of CO2 in seawater

Return type array

Note: In FluxEngine, they account fully solve for the original xCO2 that is used in the calculation of the virial expansion. I use the first estimate of xCO2 (based on fCO2 rather than pCO2). The difference between the two approaches is so small that it is not significant to be concerned. Their correction is more precise, but the difference between their iterative correction and our approximation is on the order of 1e-14 atm (1e-8 uatm).

Examples

```
>>> fCO2_to_pCO2(380, 8)
381.50806485658234
>>> fCO2_to_pCO2(380, 8, pres_hPa=985)
381.4659553134281
>>> fCO2_to_pCO2(380, 8, pres_hPa=985, tempEQ_C=14)
381.466027968504
```

pyseaflux.fco2_pco2_conversion.**pCO2_to_fCO2**(*pCO2SW_uatm*, *tempSW_C*, *pres_hPa=None*, *tempEQ_C=None*, *checks=False*)

Convert pCO2 to fCO2 in sea water to account for non-ideal behaviour of CO2

If equilibrator temperature is provided, we get a simple approximate for equilibrator xCO_2 that allows for the virial expansion to be calculated more accurately. If not, then a simple approximation is probably good enough. See the examples for the differences.

$$pCO_2^{sw} = fCO_2^{sw} \times virial(xCO_2^{eq})$$

where $xCO_2^{eq} = fCO_2^{sw} \times \Delta T^{(sw-eq)} \div P^{eq}$

Parameters

- **fCO2SW_uatm** (*array*) – seawater fugacity of CO2 in micro atmospheres
- **tempSW_C** (*array*) – sea water temperature in degrees C
- **pres_hPa** (*array, optional*) – equilibrator pressure in hecto Pascals. Defaults to 1013.25.
- **tempEQ_C** (*array, optional*) – equilibrator temperature in degrees C. Defaults to None.

Returns fugacity of CO2 in seawater

Return type array

Note: In FluxEngine, they account for the change in xCO2. This error is so small that it is not significant to be concerned about it. Their correction is more precise, but the difference between their iterative correction and our approximation is less than 1e-14 atm (or 1e-8 uatm).

Examples

```
>>> pCO2_to_fCO2(380, 8)
378.49789637942064
>>> pCO2_to_fCO2(380, 8, pres_hPa=985)
378.53967828231225
>>> pCO2_to_fCO2(380, 8, pres_hPa=985, tempEQ_C=14)
378.53960618459695
```

pyseaflux.fco2_pco2_conversion.**virial_coeff**(*temp_K*, *pres_atm*, *xCO2_mol=None*, *checks=False*)

Calculate the ideal gas correction factor for converting pCO2 to fCO2.

Based on the Lewis and Wallace 1998 Correction.

Parameters

- **temp_K** (*array*) – temperature in degrees Kelvin
- **pres_atm** (*array*) – atmospheric pressure in atmospheres
- **xCO2_mol** (*array, optional*) – mole fraction of CO2, can also be p/fCO2 if xCO2 not available. Can also be None which makes a small difference. See examples.

Returns

the factor to multiply/divide with pCO2/fCO2. Unitless

$$fCO_2 = pCO_2 \times \text{virial expansion}$$
$$pCO_2 = fCO_2 \div \text{virial expansion}$$

Return type array

Examples

From Dickson et al. (2007)

```
>>> 350 * virial_coeff(298.15, 1)  # CO2 [uatm] * correction factor
348.8836492182758
```

References

Weiss, R. (1974). Carbon dioxide in water and seawater: the solubility of a non-ideal gas. Marine Chemistry, 2(3), 203–215. https://doi.org/10.1016/0304-4203(74)90015-2

Compared with the Seacarb package in R

1.2.3 Gas Transfer Velocity

Modulates the magnitude of the flux between the atmosphere and the ocean.

pyseaflux.gas_transfer_velocity.**k_Ho06**(*wind_second_moment*, *temp_C*)
Calculates the gas transfer coeffcient for CO2 using the formulation of Ho et al. (2006)

The gas transfer velocity is for the QuickSCAT satellite wind product. Note that using this function for any other wind product is stricktly speaking not correct.

$$k_{600} = 0.266 \cdot U^2$$

The parameterization is based on the SOLAS Air-Sea Gas Exchange (SAGE) experiment.

Parameters

- **wind_ms** (*array*) – wind speed in m/s
- **temp_C** (*array*) – temperature in degrees C

Returns gas transfer velocity (k600) in cm/hr

Return type kw (array)

References

Ho, D. T., Law, C. S., Smith, M. J., Schlosser, P., Harvey, M., & Hill, P. (2006). Measurements of air-sea gas exchange at high wind speeds in the Southern Ocean: Implications for global parameterizations. Geophysical Research Letters, 33(16), 1–6. https://doi.org/10.1029/2006GL026817

pyseaflux.gas_transfer_velocity.**k_Li86**(*wind_ms*, *temp_C*)
Calculates the gas transfer coeffcient for CO2 using the formulation of Liss and Merlivat (1986)

Note: This is an old parameterization and we recommend using updated parameterisations that are calculated based on the wind product you choose to use. We include this parameterisation based purely for legacy purposes.

Parameters

- **wind_ms** (*array*) – wind speed in m/s
- **temp_C** (*array*) – temperature in degrees C

Returns gas transfer velocity (k600) in cm/hr

Return type kw (array)

References

Liss, P. S., & Merlivat, L. (1986). The Role of Air-Sea Exchange in Geochemical Cycling (Vol. 1983, Issue June 1983). D. Reidel Publishing Company.

pyseaflux.gas_transfer_velocity.**k_Mc01**(*wind_ms*, *temp_C*)
Calculates the gas transfer coeffcient for CO2 using the formulation of McGillis et al. (2001)

The gas transfer velocity has been scaled for in-situ short term wind products. Note that using this function for any other wind product is not correct.

$$k_{660} = 3.3 + 0.026 \cdot U^3$$

Parameters

- **wind_ms** (*array*) – wind speed in m/s
- **temp_C** (*array*) – temperature in degrees C

Returns gas transfer velocity (k660) in cm/hr

Return type kw (array)

References

McGillis, W. R., Edson, J. B., Ware, J. D., Dacey, J. W. H., Hare, J. E., Fairall, C. W., & Wanninkhof, R. H. (2001). Carbon dioxide flux techniques performed during GasEx-98. Marine Chemistry, 75(4), 267–280. https://doi.org/10.1016/S0304-4203(01)00042-1

pyseaflux.gas_transfer_velocity.**k_Ni00**(*wind_ms*, *temp_C*)

Calculates the gas transfer coeffcient for CO2 using the formulation of Nightingale et al (2000)

$$k_{600} = 0.333 \cdot U + 0.222 \cdot U^2$$

Parameters

- **wind_ms** (*array*) – wind speed in m/s
- **temp_C** (*array*) – temperature in degrees C

Returns gas transfer velocity (k600) in cm/hr

Return type kw (array)

References

Nightingale, P. D., Malin, G., Law, C. S., Watson, A. J., Liss, P. S., Liddicoat, M. I., Boutin, J., & Upstill-Goddard, R. C. (2000). In situ evaluation of air-sea gas exchange parameterizations using novel conservative and volatile tracers. In Global Biogeochemical Cycles (Vol. 14, Issue 1, p. 373). https://doi.org/10.1029/1999GB900091

pyseaflux.gas_transfer_velocity.**k_Sw07**(*wind_second_moment*, *temp_C*)

Calculates the gas transfer coeffcient for CO2 using the formulation Wanninkhof (1992) rescaled by Sweeny et al (2007)

The gas transfer velocity has been scaled for the NCEP/NCAR reanalysis 1 product. Note that using this function for any other wind product is not correct.

$$k_{660} = 0.27 \cdot U^2$$

Parameters

- **wind_second_moment** (*array*) – wind speed squared in m2/s2. Note that the second moment should be calculated at the native resolution of the wind to avoid losses of variability when taking the square product.
- **temp_C** (*array*) – temperature in degrees C

Returns gas transfer velocity (k660) in cm/hr

Return type kw (array)

References

Sweeney, C., Gloor, E., Jacobson, A. R., Key, R. M., McKinley, G. A., Sarmiento, J. L., & Wanninkhof, R. H. (2007). Constraining global air-sea gas exchange for CO2 with recent bomb 14C measurements. Global Biogeochemical Cycles, 21(2). https://doi.org/10.1029/2006GB002784

pyseaflux.gas_transfer_velocity.**k_Wa09**(*wind_ms*, *temp_C*)

Calculates the gas transfer coeffcient for CO2 using the formulation of Wanninkhof et al. (2009)

The gas transfer velocity has been scaled for the Cross-Calibrated Multi- Platform (CCMP) Winds product. Note that using this function for any other wind product is not correct.

$$k_{660} = 3.0 + 0.1 \cdot U + 0.064 \cdot U^2 + 0.011 \cdot U^3$$

Parameters

- **wind_ms** (*array*) – wind speed in m/s
- **temp_C** (*array*) – temperature in degrees C

Returns gas transfer velocity (k660) in cm/hr

Return type kw (array)

References

Wanninkhof, R. H., Asher, W. E., Ho, D. T., Sweeney, C., & McGillis, W. R. (2009). Advances in Quantifying Air-Sea Gas Exchange and Environmental Forcing*. Annual Review of Marine Science, 1(1), 213–244. https://doi.org/10.1146/annurev.marine.010908.163742

pyseaflux.gas_transfer_velocity.**k_Wa14**(*wind_second_moment*, *temp_C*)

Calculates the gas transfer coeffcient for CO2 using the formulation of Wanninkhof et al. (2014)

The gas transfer velocity has been scaled for the Cross-Calibrated Multi- Platform (CCMP) Winds product. Note that using this function for any other wind product is not correct.

$$k_{660} = 0.251 \cdot U^2$$

Parameters

- **wind_second_moment** (*array*) – wind speed squared in m2/s2. Note that the second moment should be calculated at the native resolution of the wind to avoid losses of variability when taking the square product.
- **temp_C** (*array*) – temperature in degrees C

Returns gas transfer velocity (k660) in cm/hr

Return type kw (array)

References

Wanninkhof, R. H. (2014). Relationship between wind speed and gas exchange over the ocean revisited. Limnology and Oceanography: Methods, 12(JUN), 351–362. https://doi.org/10.4319/lom.2014.12.351

pyseaflux.gas_transfer_velocity.**k_Wa92**(*wind_second_moment*, *temp_C*)

Calculates the gas transfer coeffcient for CO2 using the formulation of Wanninkhof (1992)

Note: This is an old parameterization and we recommend using updated parameterisations that are calculated based on the wind product you choose to use. We include this parameterisation based purely for legacy purposes.

The gas transfer velocity is scaled from instantaneous wind speeds. The study applies a correction to the scaling (0.39) based on instantaneous wind speeds to lower it to 0.31. This correction is based on the variability of wind.

$$k_{660} = 0.31 \cdot U^2$$

Parameters

- **wind_second_moment** (*array*) – wind speed squared in m2/s2. Note that the
- **the** (*second moment should be calculated at the native resolution of*) –
- **product.** (*wind to avoid losses of variability when taking the square*) –
- **temp_C** (*array*) – temperature in degrees C

Returns gas transfer velocity (k660) in cm/hr

Return type kw (array)

References

Wanninkhof, R. H. (1992). Relationship between wind speed and gas exchange over the ocean. Journal of Geophysical Research, 97(C5), 7373. https://doi.org/10.1029/92JC00188

pyseaflux.gas_transfer_velocity.**k_Wa99**(*wind_ms*, *temp_C*)

Calculates the gas transfer coeffcient for CO2 using the formulation of Wanninkhof and McGillis (1999)

The gas transfer velocity has been scaled for in-situ short term wind products. Note that using this function for any other wind product is not correct.

$$k_{600} = 0.0283 \cdot U^3$$

Parameters

- **wind_ms** (*array*) – wind speed in m/s
- **temp_C** (*array*) – temperature in degrees C

Returns gas transfer velocity (k600) in cm/hr

Return type kw (array)

References

Wanninkhof, R. H., & McGillis, W. R. (1999). A cubic relationship between air-sea CO2 exchange and wind speed. Geophysical Research Letters, 26(13), 1889–1892. https://doi.org/10.1029/1999GL900363

pyseaflux.gas_transfer_velocity.**schmidt_number**(*temp_C*)

Calculates the Schmidt number as defined by Jahne et al. (1987) and listed in Wanninkhof (2014) Table 1.

Parameters **temp_C** (*array*) – temperature in degrees C

Returns Schmidt number (dimensionless)

Return type array

Examples

```
>>> schmidt_number(20)  # from Wanninkhof (2014)
668.344
```

References

Jähne, B., Heinz, G., & Dietrich, W. (1987). Measurement of the diffusion coefficients of sparingly soluble gases in water. Journal of Geophysical Research: Oceans, 92(C10), 10767–10776. https://doi.org/10.1029/JC092iC10p10767

1.2.4 CO2 solubility in seawater

pyseaflux.solubility.**solubility_weiss1974**(*salt*, *temp_K*, *press_atm=1*, *checks=True*)

Calculates the solubility of CO2 in sea water

Used in the calculation of air-sea CO2 fluxes. We use the formulation by Weiss (1974) summarised in Wanninkhof (2014).

Parameters

- **salt** (*array*) – salinity in PSU
- **temp_K** (*array*) – temperature in deg Kelvin
- **press_atm** (*array*) – pressure in atmospheres. Used in the solubility correction for water vapour pressure. If not given, assumed that press_atm is 1atm

Returns solubility of CO2 in seawater (K_0) in mol/L/atm

Return type array

Examples

from Weiss (1974) Table 2 but with pH2O correction

```
>>> solubility_weiss1974(35, 299.15)
0.029285284543519093
```

1.2.5 Water vapour pressure

pyseaflux.vapour_pressure.**dickson2007**(*salt*, *temp_K*, *checks=False*)

Water vapour pressure of seawater after Dickson et al. (2007)

Calculates pH_2O at a given salinity and temperature using the methods defined in Dickson et al. (2007; CO2 manual)

Parameters

- **salt** (*np.array*) – salinity
- **temp_K** (*np.array*) – temperature in deg Kelvin

Returns **sea_vapress** – sea water vapour pressure in atm

Return type np.array

Examples

```
>>> vapress_dickson2007(35, 298.15)  # from Dickson et al. (2007) Ch 5.3.2
0.030698866245809465
```

pyseaflux.vapour_pressure.**weiss1980**(*salt*, *temp_K*, *checks=False*)

Water vapour pressure of seawater after Weiss and Price (1980)

For a given salinity and temperature using the methods defined in Weiss (1974)

Parameters

- **salt** (*array*) – salinity in PSU
- **temp_K** (*array*) – temperature in deg Kelvin

Returns sea water vapour pressure in atm (pH_2O)

Return type array

Examples

```
>>> vapress_weiss1980(35, 25+273.15)  # tempC + 273.15
0.03065529996317971
```

References

Weiss, R. (1974). Carbon dioxide in water and seawater: the solubility of a non-ideal gas. Marine Chemistry, 2(3), 203–215. https://doi.org/10.1016/0304-4203(74)90015-2

Weiss, R., & Price, B. a. (1980). Nitrous oxide solubility in water and seawater. Marine Chemistry, 8(4), 347–359. https://doi.org/10.1016/0304-4203(80)90024-9

1.2.6 Additional equations

Not necessarily linked to the marine carbonate system, but are useful.

pyseaflux.auxiliary_equations.**pressure_height_correction**(*pres_hPa*, *tempSW_C*, *sensor_height=10.0*, *checks=True*)

Returns exact sea level pressure if the sensor is measuring at height

Parameters

- **pres_hPa** (*array*) – Pressure in kiloPascal measured at height
- **tempSW_C** (*array*) – Temperature of the seawater in deg C
- **sensor_height** (*float, optional*) – the height of the sensor above sea level. Can be negative if you want to convert SLP to sensor height pressure. Defaults to 10.0.

Returns height corrected pressure

Return type array

pyseaflux.auxiliary_equations.**temperature_correction**(*temp_in*, *temp_out*)

pCO2 correction factor for temperature changes

Calculate a correction factor for the temperature difference between the intake and equilibrator. This is based on the empirical relationship used in Takahashi et al. 1993.

$$pCO_2^{Tout} = pCO_2^{Tin} * T^{factor}$$

Parameters

- **temp_in** (*array*) – temperature at which original pCO2 is measured (degK or degC)
- **temp_out** (*array*) – temperature for which pCO2 should be represented

Returns a correction factor to be multiplied to pCO2 (unitless)

Return type array

References

Takahashi, Taro et al. (1993). Seasonal variation of CO2 and nutrients in the high-latitude surface oceans: A comparative study. Global Biogeochemical Cycles, 7(4), 843–878. https://doi.org/10.1029/93GB02263

1.2.7 Area calculations

Calculates the area of pixels for a give grid input.

pyseaflux.area.**area_grid**(*lat*, *lon*, *return_dataarray=False*)
Calculate the area of each grid cell for given lats and lons

Parameters

- **lat** (*array*) – latitudes in decimal degrees of length N
- **lon** (*array*) – longitudes in decimal degrees of length M
- **return_dataarray** (*bool, optional*) – if True returns xr.DataArray, else array

Returns area of each grid cell in meters

Return type array, xr.DataArray

References

https://github.com/chadagreene/CDT/blob/master/cdt/cdtarea.m

pyseaflux.area.**earth_radius**(*lat*)
Calculate the radius of the earth for a given latitude

Parameters **lat** (*array, float*) – latitude value (-90 : 90)

Returns radius in metres

Return type array

pyseaflux.area.**get_area_from_dataset**(*dataarray*, *lat_name='lat'*, *lon_name='lon'*)
Calculate the grid cell area from a xr.Dataset or xr.DataArray.

This module is only intended to be by the SeaFlux authors to download the data required to create the SeaFlux ensemble. Has links to most data sources (ERA5 might not be included)

Hence, this module is not imported by default and submodules should be imported on demand.

1.3 Contribution Guide

Contributions are highly welcomed and appreciated. Every little help counts, so do not hesitate! You can make a high impact on `pyseaflux` just by using it and reporting issues.

The following sections cover some general guidelines regarding development in `pyseaflux` for maintainers and contributors.

Nothing here is set in stone and can't be changed. Feel free to suggest improvements or changes in the workflow.

Contribution links

1.3.1 Feature requests and feedback

We are eager to hear about your requests for new features and any suggestions about the API, infrastructure, and so on. Feel free to submit these as issues with the label "feature request."

Please make sure to explain in detail how the feature should work and keep the scope as narrow as possible. This will make it easier to implement in small PRs.

1.3.2 Report bugs

Report bugs for `seaflux` in the issue tracker with the label "bug".

If you can write a demonstration test that currently fails but should pass that is a very useful commit to make as well, even if you cannot fix the bug itself.

1.3.3 Fix bugs

Look through the GitHub issues for bugs.

Talk to developers to find out how you can fix specific bugs.

1.3.4 Preparing Pull Requests

1. Fork the seaflux GitHub repository. It's fine to use `pyseaflux` as your fork repository name because it will live under your username.
2. Clone your fork locally using git, connect your repository to the upstream (main project), and create a branch:

```
$ git clone git@github.com:YOUR_GITHUB_USERNAME/SeaFlux.git
$ cd SeaFlux
$ git remote add upstream git@github.com:lukegre/SeaFlux.git

# now, to fix a bug or add feature create your own branch off "master":

$ git checkout -b your-bugfix-feature-branch-name master
```

If you need some help with Git, follow this quick start guide: https://git.wiki.kernel.org/index.php/QuickStart

3. Set up a [conda](environment) with all necessary dependencies:

```
$ conda env create -f ci/environment-py3.8.yml
```

4. Activate your environment:

```
$ conda activate test_env_pyseaflux
```

5. Install the pySeaFlux package:

```
$ pip install -e . --no-deps
```

6. Before you modify anything, ensure that the setup works by executing all tests:

```
$ pytest
```

You want to see an output indicating no failures, like this:

```
$ ========================== n passed, j warnings in 17.07s␣
↪============================
```

7. Install pre-commit and its hook on the `seaflux` repo:

```
$ pip install --user pre-commit
$ pre-commit install
```

Afterwards `pre-commit` will run whenever you commit. If some errors are reported by pre-commit you should format the code by running:

```
$ pre-commit run --all-files
```

and then try to commit again.

https://pre-commit.com/ is a framework for managing and maintaining multi-language pre-commit hooks to ensure code-style and code formatting is consistent.

> You can now edit your local working copy and run/add tests as necessary. Please follow PEP-8 for naming. When committing, `pre-commit` will modify the files as needed, or will generally be quite clear about what you need to do to pass the commit test.

8. Break your edits up into reasonably sized commits:

```
$ git commit -a -m "<commit message>"
$ git push -u
```

Committing will run the pre-commit hooks (isort, black and flake8). Pushing will run the pre-push hooks (pytest and coverage)

We highly recommend using test driven development, but our coverage requirement is low at the moment due to lack of tests. If you are able to write tests, please stick to xarray's testing recommendations.

9. **Add yourself to the** Project Contributors list via `./docs/authors.md`.
10. Finally, submit a pull request through the GitHub website using this data:

```
head-fork: YOUR_GITHUB_USERNAME/pyseaflux
compare: your-branch-name

base-fork: lukegre/pySeaFlux
base: master
```

The merged pull request will undergo the same testing that your local branch had to pass when pushing.

1.4 Citing pySeaFlux

If you would like to cite or reference SeaFlux, please use one or both of the following if appropriate:

DATA: Fay, A. R., Gregor, L., Landschützer, P., McKinley, G. A., Gruber, N., Gehlen, M., Iida, Y., Laruelle, G. G., Rödenbeck, C., and Zeng, J.: Harmonization of global surface ocean pCO2 mapped products and their flux calculations; an improved estimate of the ocean carbon sink, Earth Syst. Sci. Data Discuss. [preprint], https://doi.org/10.5194/essd-2021-16, in review, 2021.

CODE: Luke Gregor, & Matthew Humphreys. (2021, April 1). lukegre/pySeaFlux: Updated continuous integration and docs (Version 2.0.0). Zenodo. http://doi.org/10.5281/zenodo.4659162

1.4.1 Code Contributors

- Luke Gregor - Environmental Physics, ETH Zuerich: Zurich, Switzerland. (ORCID: 0000-0001-6071-1857)
- Matthew Humphreys - Royal Netherlands Institute for Sea Research: Texel, The Netherlands. (ORCID: 0000-0002-9371-7128)

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

V

W