NOTES: Community Firn Model Notes

Thea Quistgaard

Master Thesis 2020/2021

1 diffusivity_vas.py

File containing a number of classes, to compute and calculate different diffusion parameters.

- **class** `Porosity()`
  Computes porosity, closed and open, for a given rho-array. Includes constants $\rho_{\text{cl}} = 830$ and $\rho_{\text{ice}} = 917$.
  Contains four methods and an **__init__**:
  - **def** `s(self, rho)`
    Computes porosity as
    $$s = 1 - \frac{\rho}{\rho_{\text{ice}}} \tag{1}$$
  - **def** `s_cl(self, rho)`
    Computes the closed porosity (not close-off), but porosity at which fluid flow is no longer possible.
    $$s_{\text{closed}} = \rho\, e^{75\cdot\left(\frac{\rho}{\rho_{\text{cl}}}-1\right)} \tag{2}$$
    (Schwander 1989)
  - **def** `s_op(self, rho)`
    Calculates the open porosity, i.e. the difference between s and s_{cl}.
    $$s_{\text{op}} = s - s_{\text{cl}} \tag{3}$$
  - **def** `show_all(self, rho)`
    Function that plots all porosities versus density
- **class** `TortuosityInv()` Computes the inverted tortuosity $\frac{1}{\tau}$ as an array on the basis of porosity and density, through different methods, presented in various articles.
  Resulting array is clipped (values higher than $a = 0$ is set equal to a) Defines closed density $\rho_{\text{cl}} = 830$ and ice density $\rho_{\text{ice}} = 917$. Contains four methods and an **__init__**:
  - **def** `schwander1989(self, rho)`
    Computes inverse of tortuosity(J. Schwander 1989, "*The transformation of snow to ice and the occlusion of gases*", equation 5) as:
    $$\frac{1}{\tau} = 1.7 \cdot s_{\text{op}} - 0.2 \tag{4}$$
  - **def** `johnsen2000(self, rho, rho_co = 804.3)`
    Computes inverse of tortuosity(S. J. Johnsen, 2000, "*Diffusion of stable isotopes in polar firnand ice: The isotope effect in firn diffusion*", equation 18) as:
    $$\frac{1}{tau} = 1 - b \cdot \left(\frac{\rho}{\rho_{\text{ice}}}\right)^2 \tag{5}$$
    where $b = \left(\frac{\rho_{\text{ice}}}{\rho_{\text{co}}}\right)^2$.

– **def** witrant2012(**self**, rho, temp, p) Computes inverse of tortuosity(E. Witrant, 2012, "*A new multi-gas constrained model of trace gas non-homogeneous transport in firn: evaluation and behaviorat eleven polar sites*", equation 20) as:

$$\frac{1}{\tau} \approx (2.5 \cdot s_{\text{op}} - 0.31) \cdot \left(\frac{T}{T_0}\right)^{1.8} \cdot \frac{P_0}{P_{\text{atm}}} \tag{6}$$

where $T_0 = 273.15$ K and $P_atm = 101325$.

– **def** show_all(**self**, rho)
Plot all inverse tortuosities versus porosity, separate plots.

- **class** FractionationFactor()
Class containing different methods to compute fractionation factor for deuterium, δ^{18}O and δ^{17}O, given different methods presented in various articles. Contains an __init__ and five methods(D, δ^{18}O, δ^{17}O and liquid fractionation for D and δ^{18}O), each containing a number of functions.

  – **def** deuterium(**self**)
  Computes fractionation factor for deuterium between water and ice given a temperature for

    * L. Merlivat, 1967, "*Fractionnement isotopique lors des changements d'état solide-vapeur et liquide-vapeur de l'eau à des températures inférieures à* $0°C$":

$$\alpha_{D\_Merlivat} = e^{-4.10 \cdot 10^{-2}} \cdot e^{\frac{16288}{T^2}} \tag{7}$$

    * M. D. Ellehøj, 2011, "*Ice-vapor equilibrium fractionation factor*, PhD thesis":

$$\alpha_{D\_Ellehoj} = e^{0.2133 - \frac{203.10}{T} + \frac{48888}{T^2}} \tag{8}$$

    * K. D. Lamb, 2017, "*Laboratory measurements of HDO/H2O isotopic fractionation during ice deposition in simulated cirrus clouds*":

$$\alpha_{D\_Lamb} = e^{\frac{12525}{T^2} - 5.59 \cdot 10^{-2}} \tag{9}$$

  – **def** o18(**self**)
  Computes fractionation factor for δ^{18}O between water and ice given a temperature as:

    * M. Majoube, 1970, "*Fractionation factor of 18o between water vapour and ice*":

$$\alpha_{O18\_Majoube} = 0.9722 \cdot e^{\frac{11.839}{T}} \tag{10}$$

    * M. D. Ellehøj, 2011, "*Ice-vapor equilibrium fractionation factor*, PhD thesis":

$$\alpha_{O18\_Ellehoj} = e^{0.0831 - \frac{49.192}{T} + \frac{8312.5}{T^2}} \tag{11}$$

  – **def** o17(**self**)
  Computes fractionation factor for δ^{17}O in ice given a temperature. Both are based on E. Barkan, 2005, "*High precision measurements of 17o/16o and 18o/16o ratios in h2o*".

    * M. Majoube, 1970, "*Fractionation factor of 18o between water vapour and ice*":

$$\alpha_{O17\_Majoube} = (\alpha_{O18\_Majoube})^{0.529} \tag{12}$$

    * M. D. Ellehøj, 2011, "*Ice-vapor equilibrium fractionation factor*, PhD thesis":

$$\alpha_{O17\_Ellehoj} = (\alpha_{O18\_Ellehoj})^{0.529} \tag{13}$$

- **def** deuterium_liquid(**self**)
  Computes fractionation factor for deuterium between water and steam given a temperature. M. Majoube, 1971, "*Oxygen-18 and deuterium fractionation between water and steam*".

$$\alpha_{D\_liq} = e^{\frac{52.612 - 76.248 \cdot \frac{1000}{T} + 24.844 \cdot \frac{10^6}{T^2}}{1000}} \tag{14}$$

  - **def** o18_liquid(**self**)
    Computes fractionation factor for δ^{18}O between water and steam given a temperature. M. Majoube, 1971, "*Oxygen-18 and deuterium fractionation between water and steam*".

$$\alpha_{o18\_liq} = e^{\frac{-2.0667 - 0.4156 \cdot \frac{1000}{T} + 1.137 \cdot \frac{10^6}{T^2}}{1000}} \tag{15}$$

- **class** P_Ice()
  Various evaluations for saturation vapor pressure over ice. D. M. Murphy and T. Koop, 2005, "*Review of the vapour pressures of ice and supercooled water for atmospheric applications*". Contains four methods and an __**init**__. Temperature in K and pressure in Pa.

  - **def** clausius_clapeyron_simple(**self**, T)
    Simple evaluation using Clausius Clapeyron equation(used to characterize a discontinuous phase transition) with constant latent heat of sublimation.

$$P_{\text{ice}} = e^{28.9074 - \frac{6143.7}{T}} \tag{16}$$

  - **def** clausius_clapeyron_Lt(**self**, T)
    Evaluation using temperature dependence of latent heat plus a numerical fit to experimental data:

$$P_{\text{ice}} = e^{9.550426 - \frac{5723.265}{T} + 3.53068 \cdot \ln(T) - 0.00728332 \cdot T} \tag{17}$$

  - **def** sigfus_2000(**self**, T)
    Expression used in S. J. Johnsen, 2000, "*Diffusion of stable isotopes in polar firn and ice: the isotope effect in firn diffusion*".

$$P_{\text{ice}} = 3.454 \cdot 10^{12} 1 \cdot e^{\frac{-6133}{T}} \tag{18}$$

  - **def** p_ice_dict(**self**, T)
    Dictionary containing all calculated values of pressure over ice.

- P_Water()
  Various evaluations for saturation vapour pressure over water. Contains only one method and an __**init**__.

  - **def** goff(**self**, T)
    Goff-Gratch equation, 1946, "*Low-pressure properties of water from -160 to 212 °F*", pressure in Pa, temperature in Kelvin.

$$T_{steam} = 373.15 \tag{19}$$

$$p_{steam} = 101325 \tag{20}$$

$$p_{water} = -7.90298 \cdot \left(\frac{T_{steam}}{T} - 1\right) + 5.02808 \cdot \log_{10}\left(\frac{T_{steam}}{T}\right) - 1.3816 \cdot 10^{-7} \tag{21}$$

$$\cdot \left(10^{11.344 * (1 - \frac{T}{T_steam})} - 1\right) + 8.1328 \cdot 10^{-3} \cdot (10^{(-3.49149 * (T_s t/T - 1))} - 1) \tag{22}$$

$$+ \log_{10}(p_{steam}) \tag{23}$$

- **class** `FirnDiffusivity()`
  Contains three methods - firn diffusivity for deuterium, δ^{18}O and δ^{17}O - and an `__init__`.
  - **def** `__init__(self, rho, rho_co = 804.3, T = 218.5, P = 1, p_ice_version = "sigfus_2000", tortuosity_version = "johnsen2000")`
    Sets the given method to compute the vapor pressure over ice and the inverse tortuosity. Computes fractionation factors and air diffusivity with parameters given. Here, ρ is an array, and the only thing that needs to be feeded to the class.
  - **def** `deuterium(self, f_factor_version = "Merlivat")`
    Computes the firn diffusivity for deuterium in $[\frac{m^2}{s}]$. Defines a number of constants, $m = 18{\cdot}10^{-3}$[kg], $R = 8.314$, $\rho_{\text{ice}} = 917[\frac{\text{kg}}{\text{m}^3}]$, $D_{\text{air, HDO}}$(computed via `AirDiffusivity(T,P).deuterium(`
    and the fractionation factor computed via `FractionationFactor(T).deuterium()['Merlivat']`.
    The diffusivity for HDO is then finally computed as:

$$D_{\text{firn, HDO}} = \frac{m \cdot P_{\text{ice}} \cdot D_{\text{air, HDO}} \cdot \frac{1}{\tau}}{RT \cdot \alpha_D \cdot (\frac{1}{\rho} - \frac{1}{\rho_{\text{ice}}})} \tag{24}$$

  - **def** `o18(self, f_factor_version = "Majoube")`
    Return Diffusivity in firn for $H_2^{18}O$ $[\frac{m^2}{s}]$. Computed as for deuterium, so:

$$D_{\text{firn, O18}} = \frac{m \cdot P_{\text{ice}} \cdot D_{\text{air, O18}} \cdot \frac{1}{\tau}}{RT \cdot \alpha_{\text{O18}} \cdot (\frac{1}{\rho} - \frac{1}{\rho_{\text{ice}}})} \tag{25}$$

  - **def** `o18(self, f_factor_version = "Majoube")`
    Return Diffusivity in firn for $H_2^{17}O$ $[\frac{m^2}{s}]$. Computed as for other two isotopes:

$$D_{\text{firn, O17}} = \frac{m \cdot P_{\text{ice}} \cdot D_{\text{air, O17}} \cdot \frac{1}{\tau}}{RT \cdot \alpha_{\text{O17}} \cdot (\frac{1}{\rho} - \frac{1}{\rho_{\text{ice}}})} \tag{26}$$

- **class** `FirnDiffusivityFast()`
  Computes diffusivity of firn, but faster?? Contains three methods, diffusivity of deuterium, δ^{18}O and δ^{17}O and an `__init__`.
  `__init__` contains definitions of $\rho_{\text{co}} = 804.3$, $T = 218.5$, $P = 1$(not dependent on `p_ice_version`), fractionation factor computed from `FractionationFactor(T)`, air diffusivity computed from `AirDiffusivity(T,P)`, saturated vapor pressure $3.454 \cdot 10^{12} \cdot e^{\frac{-6133}{T}}$ and air diffusivity computed as $D_{\text{air}} = 10^{-4} \cdot 0.211 \cdot \left(\frac{T}{273.15}\right)^{1.94} \cdot \left(\frac{1}{P}\right)$
  - **def** `deuterium(self, f_factor_version = "Merlivat")`
    Returns diffusivity in firn for deuterium in $[\frac{m^2}{s}]$. Computes as above, except the `f_factor_version` is taken out of the picture. The diffusivity is computed through:

$$D_{\text{air, HDO}} = D_{\text{air}} \cdot 0.9755 \tag{27}$$

    α_D is not dependent on `f_factor_version` and is computed as:

$$\alpha_D = 0.9098 \cdot e^{\frac{16288}{T^2}} \tag{28}$$

    $1/\tau$ is not dependent on `toruosity` and is computed as:

$$\tau = \frac{1}{(1 - 1.3 \cdot \left(\frac{\rho}{\rho_i ce}\right)^2)} \tag{29}$$

    and finally $D_{\text{firn, HDO}}$ is computed as in `deuterium()` in **class** `FirnDiffusivity()`.

– **def** o18(**self**, f_factor_version = "Majoube")
Same as for deuterium with adjusted calculations:

$$D_{\text{air, O18}} = D_{\text{air}} \cdot 0.9723 \tag{30}$$

$$\alpha_D = 0.9722 \cdot e^{\frac{11.839}{T^2}} \tag{31}$$

$$\tau = \frac{1}{(1 - 1.3 \cdot \left(\frac{\rho}{\rho_i ce}\right)^2)} \tag{32}$$

and finally $D_{\text{firn, O18}}$ is computed as in o18() in **class** FirnDiffusivity().

– **def** o18(**self**, f_factor_version = "Majoube")
Same as for O18 with adjusted calculations:

$$D_{\text{air, O18}} = D_{\text{air}} \cdot 0.98555 \tag{33}$$

$$\alpha_D = 0.9722 \cdot \left(e^{\frac{11.839}{T^2}}\right)^{0.529} \tag{34}$$

$$\tau = \frac{1}{(1 - 1.3 \cdot \left(\frac{\rho}{\rho_i ce}\right)^2)} \tag{35}$$

and finally $D_{\text{firn, O17}}$ is computed as in o17() in **class** FirnDiffusivity().

- **class** IceDiffusivity()
Computes the ice diffusivity through five different methods, given in five different articles. Contains five methods and one __**init**__ initializing the temperature as $T = 218.15$.

– **def** sigfus(**self**)
Uses parametrization from S. J. Johnsen, 2000, "*Diffusion of stable isotopes in polar firn and ice: the isotope effect in firn diffusion*":

$$D_{\text{ice}} = 1.255 \cdot 10^{-3} \cdot e^{\frac{-7273}{T}} \tag{36}$$

– **def** ramseier(**self**)
Uses parameterization from R. O. Ramseier, 1967, "*Self-diffusion of tritium in natural and synthetic ice monocrystals*":

$$D_{\text{ice}} = 9.2 \cdot 10^{-4} \cdot e^{\frac{-7186}{T}} \tag{37}$$

– **def** blicks(**self**)
Uses parameterization from H. Blicks, 1966, "*Diffusion von Protonen (Tritonen) in reinen und dotierten Eis-Einkristallen*":

$$D_{\text{ice}} = 2.5 \cdot 10^{-3} \cdot e^{\frac{-7302}{T}} \tag{38}$$

– **def** delibaltas(**self**)
Uses parameterization from P. Delibaltas, 1966, "*Diffusion von 18O in Eis-Einkristallen*":

$$D_{\text{ice}} = 0.0264 \cdot e^{\frac{-7881}{T}} \tag{39}$$

– **def** itagaki100(**self**)
Uses parameterization from K. Itagaki, 1964, "*Self-Diffusion in Single Crystals of Ice*":

$$D_{\text{ice}} = 0.014 \cdot e^{\frac{-7650}{T}} \tag{40}$$

- **class** `AirDiffusivity()`
  Calculation of air diffusivity for deuterium, O^{18} and O^{17}. Contains four methods and an `__init__` which initializes the temperature to $T = 218.5$, the pressure to $P = 1$ and the air diffusivity for $H_2^{16}O$ from H. Pruppacher and W.D. Hall, 1976, "*The survival if ice particles falling from cirrus clouds in subsaturated air*" asz

$$D_{\text{air}} = 10^{-4} \cdot 0.211 \cdot \left(\frac{T}{273.15}\right)^{1.94} \cdot \left(\frac{1}{P}\right) \tag{41}$$

  The following calculations are based on L. Merlivat, 1978, "*The dependence of bulk evaporation coefficients on air-water interfacial conditions as determined by the isotopic method*".

  - **def** `deuterium(self)`
    Returns diffusivity in air for HDO:

$$D_{\text{air, HDO}} = D_{\text{air}} \cdot 0.9755 \tag{42}$$

  - **def** `o18(self)`
    Returns diffusivity in air for $H_2^{18}O$:

$$D_{\text{air, O18}} = D_{\text{air}} \cdot 0.9723 \tag{43}$$

  - **def** `o17(self)`
    Returns diffusivity in air for $H_2^{17}O$:

$$D_{\text{air, O17}} = D_{\text{air}} \cdot 0.98555 \tag{44}$$

2 solver.py

Need to understand and describe the algorithm. Essential for the CFM.
Function to compute transient one-dimensional diffusion through the finite volume method. Contains two functions, one that solves the 1D diffusion problem and one that solves the inhibiting matrix problem.

- **def** `transient_solve_TR(z_edges_vec, z_P_vec, nt, dt, Gamma_P, phi_0, nz_P, nz_fv, phi_s)`
  The actual 1D diffusion solver. Takes a number of parameters as input:
  - `z_edges_vec`[array of floats]: uniform edge spacing of volume elements
  - `z_P_vec`[array of floats]: depth profile (edge locations of boxes)
  - `nt`[float]: Number of time steps
  - `dt`[float]: size of time step
  - `Gamma_P`[array of floats]: diffusivity profile, for heat diffusion: $K_{firn}/(c_{firn} * \rho)$. Different for iso diffusion.
  - `phi_0`[array of floats]:Initial profile of conserved quantity to be diffused (i.e. temperature or isotope values)
  - `nz_P`[int]: number of nodes in depth profile z
  - `nz_fv`[int]: number of finite volumes in model z
  - `phi_s`[float]: value of conserved quantity (temp/iso) at surface

  and returns a single ouptut `phi_t`[array of floats], the diffused and final distribution of the 1D conserved quantity.
  The algorithm builds on a finite volume method for solving the discrete diffusion equation.

- **def** `solver(a_U, a_D, a_P, b)`
  Function to solve the matrix problem created in the finite volume method above. Uses a sparse diagonal matrix to solve the linear system.