Review of

PyCO2SYS v1.7: marine carbonate system calculations in Python

submitted to *Geoscientific Model Development*
by Matthew P. Humphreys et al.

1 General comments

Matthew P. Humphreys and co-authors present PyCO2SYS, a Python version of the "industry-standard" carbonate chemistry calculation package CO2SYS, originally developed for DOS by Lewis and Wallace (1998) and over the years ported to MATLAB and Microsoft Excel. The port to Python presented here derives from the MATLAB version 2.0.5 of CO2SYS (Orr et al., 2018), but includes extensions from the v3 branch of CO2SYS-MATLAB up to version 3.2 (Sharp et al., 2021). In addition to these improvements, PyCO2SYS also includes new developments not found in the MATLAB sub-family, such as automatic differentiation. The amendments required to use input data pairs that could not be processed with CO2SYS-MATLAB v2.0.5 (e.g., $[HCO_3^-]$ or $[CO_3^{2-}]$ in combination with any other) are presented and a detailed assessment of the (generally negligible) differences between the results obtained with previous versions of CO2SYS is provided. What I find missing is an analysis of the computational performance and an exploration of the robustness of the adopted numerical procedures.

The paper fits well into the scope of *Geoscientific Model Development*. It is well written, goes to an informative level of detail and yet remains well readable. The material presentation is structured in an easy-to-follow way. There are a few minor imprecisions and inaccuracies here and there; a few things are missing in the analysis and discussion. These shortcomings are nevertheless relatively minor and can certainly be easily fixed. Until now, no similarly comprehensive carbonate chemistry calculation package has been available for Python. I am only aware of only two others, but that do not offer the same level of functionality: `mocsy 2.0` and `cbsyst 0.3.7`. Making CO2SYS available for the ever growing Python users community is a highly welcome move and will certainly contribute to further increase its popularity and usefulness.

2 Specific comment

Auto differentiation

The automatic differentiation, performed with the help of the Python package Autopar is highlighted as one of the distinguishing features of PyCO2SYS compared to its predecessors.

It is, however, not explained to which parts of the calculations this automatic differentiation is applied, and also not to what level: is it only used to differenti-

ate the expressions of the different terms of A_T, or also for the parametrisations of the chemical constants, and if so, with respect to which variables?

If it is only applied to the terms of A_T, automatic differentiation might actually make things unnecessarily inefficient. The derivatives of the rational function terms that make up A_T, which are all that would be required for the Newton-Raphson solver, are actually quite straightforward and could just as well implemented manually once and for all, instead of having to rely upon a package that is adding another (unnecessary) layer of complexity and that is not actively developed any longer.

Finally, where automatic differentiation could really become useful, i.e., when it comes to uncertainty propagation, automatic differentiation has to be abandoned because it is computationally inefficient.

3 Minor points and technical corrections

Abstract, Lines 15–16:

> "We discuss new insights that arose during the development process, for example that the marine carbonate system cannot be unambiguously solved from the total alkalinity and carbonate ion parameter pair."

The insights referred to in this sentence are actually not new, but simply little-known. The topology of the CO_3^{2-} concentration isolines in T_C-A_T space has been known for more than fifty years (Deffeyes, 1965). The existence of two pH roots for *most* pairs of A_T–$[CO_3^{2-}]$ values is an immediate consequence of that topology. Twenty years ago, Zeebe and Wolf-Gladrow (2001, pp. 276–277) also acknowledged the existence of two roots for this problem and recommend to use the larger one (in terms of $[H^+]$, i.e., the lower one in terms of pH). Please refer to Munhoven (2021) for a comprehensive discussion and a quantitative approach to characterise the A_C–CO_3^{2-} problem (number of roots, bracketing intervals for individual roots and individual starting values).

It might also be worth mentioning that the T_C–HCO_3^- problem is affected by similar ambiguities, acknowledged again shortly by Zeebe and Wolf-Gladrow (2001, pp. 276), and analysed and discussed in more detail by Munhoven (2021, Appendix A).

Page 5, Table 2: While I understand that salinity is zero for freshwater, I do not see why all the total concentrations from ammonia to sulfide have to be zero for freshwater. Had these values not better be left under the control of the user?

The table's footnote *b* states

> "In GEOSECS-Peng, phosphate is not included in the definition of total alkalinity."

Something must be wrong here. Since the very first version of CO2SYS, option 7 (originally "Peng", now "GEOSECS-Peng") included the contribution of the

phosphates to total alkalinity, although in a peculiar way based upon a charge weighted approach, instead of the proton donor/proton acceptor approach of Dickson (1981). Option 6 (originally "GEOSECS", now "GEOSECS-Takahashi"), on the other hand, used the A_{CB} approximation to A_T. Please clarify.

The information in the GEOSECS column is anyway not entirely clear: neither GEOSECS-Takahashi nor GEOSECS-Peng include contributions from ammonia or sulfides to A_T – the meaning of "User-defined" for these two at least is not obvious. GEOSECS-Takahashi only considers carbonate and borate alkalinity in their A_T definition: so what does PyCO2SYS do with the extra user-defined data?

This should be presented more precisely.

Page 11, lines 209: The implementation in PyCO2SYS actually follows the approach of Munhoven (2013a) more or less exactly. The subroutine `ahini_for_at` in `phsolvers.f90` from `mocsy` 2.0 was actually taken from SolveSAPHE 1.0.1 and the only adaptations made relate to the way the values for the chemical constants are passed (cf. subroutine `AHINI_FOR_AT` in `mod_phsolvers.f90` which is contained in the codes included in Munhoven (2013a, Supplement) or from Munhoven (2013b)).

Page 11, lines 225–226:

> "[...], so the approach of Munhoven (2013a) cannot be applied if A_T is indeed negative (e.g. after the alkalinity end-point in an acidimetric titration)."

This statement is in clear contradiction with the SolveSAPHE code provided by Munhoven (2013a, Supplement) which shows that if $A_T < 0$, h_0 is set to 10^{-3} mol/kg and if $A_T > 2T_C + T_B$, h_0 is set to 10^{-10} mol/kg (see next comment). The approach of Munhoven (2013a) thus obviously also considers the case $A_T < 0$ and addresses it exactly the same way as ... PyCO2SYS.

Please check out the original code and rewrite this sentence more accurately (see also previous comment).

Page 12, line 230: Although h_{min} indeed always has a real value (and even has a real *positive* value), that value does not always lead to a meaningful $h_0(s)$ as the resulting A_{CB} may possibly be greater than $2T_C + T_B$, which is not possible $h_0(s)$ thus actually has to fulfil some conditions to keep A_{CB} within bounds, as explained in the "Mathematical and Technical Details" memo in the Supplement to Munhoven (2021).

Starting a Newton-Raphson iteration with physically meaningless initial values may jeopardize convergence, which must of course be avoided.

Page 13, line 263: Is the reference to Cai et al. (2017) for the ammonia and sulfide extensions of CO2SYS correct? I would rather have expected Xu et al. (2017).

Page 14, line 287: It would be good to cite the paper by Hagens and Middelburg (2016) here, which also deals with various aspects of the calculation of generalised buffer factors.

Page 14, line 296–306: Do I understand this correctly: with the 'automatic' approach, all the buffer factors are calculated analytically, in a way that is fully consistent with the currently adopted A_T composition; with the 'explicit' approach, all the buffer factors are calculated from equations taken from the literature and that may rely on simplified A_T composition, except for the Revelle factor, for which only a finite difference approximation is calculated, for compatibility reasons with previous CO2SYS versions?

It is somehow unsatisfactory that no direct equation was implemented for the Revelle factor, which remains the best known of all buffer factors at the end of the day. Why not add a third approach ('legacy' or similar), which would only provide the finite difference approximation for the Revelle factor, and then include a direct equation for the Revelle factor with the 'explicit' approach and so bring it on par with the others.

Finally: please provide us with individual references for each buffer factor equation actually implemented.

Page 17, line 337–338:

> "We use finite differences rather than automatic differentiation here because the latter, while possible, is computationally inefficient to apply over the entire PyCO2SYS program."

Why is this so computationally inefficient? Is this due to the usage of the Autograd package? If only the derivatives of the alkalinity parts are required, why not implement them manually once and for all? These are comparatively straightforward. Now, the derivatives of the chemical constants might possibly be required as well—the text does unfortunately not include enough information about the level to which the differentiation is pushed. If so this approach would indeed be unrealistic. As requested in the specific comments above, additional information about which expressions exactly in the code automatic differentiation is applied to would be helpful.

Page 17, line 339: duplicate "an"

Page 18, line 385: duplicate "than"

Page 19, line 396: I suggest to rename section 4.2 to "Comparison with previous versions of CO2SYS" — "other software" is somewhat misleading as only CO2SYS variants are considered in this discussion.

Page 24, line 496: From experience, I think that "often" better had to read "most often" or "generally" in this context.

Page 25, line 504:

> "Which root the solver finds depends on the initial pH estimate [...]"

This is a rather unsatisfactory behaviour as it makes it impossible to foresee whether the pH calculation can terminate reliably. Such unpredictable (and therefore unwanted) behaviour can nevertheless be safely avoided by first proceeding to a root localisation, and then using a bracketing root-finding algorithm instead of a plain Newton-Raphson (see, e.g., Munhoven, 2021).

Page 24, line 552–554:

> "Now that the basis of PyCO2SYS is established, we would welcome more direct interaction with the groups developing these other tools, working towards a set of marine-carbonate-system-solving tools that return identical results regardless of the software platform."

This is an excellent idea. Such a joint effort could also define standard benchmark problems in order to assess the numerical and computational performances of the different tools.

Page 28, *Code availability* section: Does PyCO2SYS require any particular version of Python? If so, it would be good to state this here.

Page 29, Eq. (B2): The expression for A_w should actually read

$$A_\mathrm{w} = [\mathrm{OH}^-] - [\mathrm{H}^+]_\mathrm{free} = \frac{K^*_\mathrm{w}}{[\mathrm{H}^+]} - \frac{[\mathrm{H}^+]}{phcvt},$$

where *phcvt* is a factor to convert from the free to the working (total?) pH scale. This nevertheless appears to be correctly implemented in the code.

Page 32, Eq. (B20): This is not in line with the PyCO2SYS code. Since K^*_SO4 is on the free pH scale, $[\mathrm{H}^+]$ must actually be $[\mathrm{H}^+]_\mathrm{free}$ in this equation.

Page 32, Eq. (B22): This is not in line with the PyCO2SYS code. Since K^*_F is on the free pH scale, $[\mathrm{H}^+]$ must actually be $[\mathrm{H}^+]_\mathrm{free}$ in this equation

Page 32, Eq. (B24): This extension is not consistent with the definition of total alkalinity (Dickson, 1981), which states that

> "[t]he total alkalinity of a natural water is thus defined as the number of moles of hydrogen ion equivalent to the excess of proton acceptors (bases formed from weak acids with a dissociation constant $K \leq 10^{-4.5}$, at 25 °C and zero ionic strength) over proton donors (acids with $K > 10^{-4.5}$) in one kilogram of sample."

So, it is not K_α^* that has to be compared against this decisive threshold, but K_α, and not at any arbitrary combination of temperature, salinity and pressure, but at a clearly defined set. How reliable is the adopted approach, given that approximation?

Page 32, line 686: According to Wolf-Gladrow et al. (2007) the zero level of protons is a *species*, not a concentration or some other pZLP. Each acid-base system actually has a different zero level of protons. In order to define the zero levels of protons for a complex mixture of acids and bases in a consistent way, Wolf-Gladrow et al. (2007) call upon a threshold pK value (their Sect. 2.4.4). They denote that threshold pK value by pK_{zlp} (it would be recommendable to stick to that notation in order to avoid unnecessary confusion). The currently accepted definition of A_T by Dickson (1981) is thus based upon p$K_{\mathrm{zlp}} = 4.5$. However, pK_{zlp} is not the zero level of protons (there as many zero levels of protons as there are acid-base systems in the solution). Please rewrite this more precisely.

Page 34, Sect. C2.2: It might be worth mentioning that some pairs of A_T–pH data values lead to negative A_C (see, e.g., Munhoven, 2021, for illustrative examples), which is physically impossible. Does pyCO2SYS catch that kind of exception?

Page 34, Sect. C2.9: Similarly to the A_T–$\mathrm{CO_3^{2-}}$ problem, the T_C–$\mathrm{HCO_3^-}$ problem may have zero, one or two pH roots. There is actually an upper limit, which is strictly lower than 1, for the $[\mathrm{HCO_3^-}]/T_\mathrm{C}$ fraction above which there is no solution; at that exact limit, there is one, and below that limit there are two roots. Zeebe and Wolf-Gladrow (2001, p. 276) already mention the possibility of two roots and recommend to chose the low-$[\mathrm{H^+}]$ (high-pH) one, without any further explanation though. Munhoven (2021, Appendix A) provides a comprehensive analysis of this problem and a short discussion about the respective side effects of each one of the two roots, which may contribute to discriminate between the two and help to chose the relevant one.

Page 37, paragraph at lines 802–808: There are unfortunately several inaccurate statements in this paragraph. The described procedure indeed follows Orr and Epitalon (2015) who follow … Munhoven (2013a). The solver engines in mocsy 2.0 (Orr and Epitalon, 2015) stem from SolveSAPHE 1.0.1 (Munhoven, 2013a, Supplement) – this is clearly stated in the comments in the mocsy 2.0 code. The initialisation subroutine from SolveSAPHE — which was included in mocsy 2.0 modified only to transfer the chemical constants differently — furthermore uses a fall-back value $h_0(T_\mathrm{C}) = 10^{-7}\,\mathrm{mol/kg}$ in case $g_2^2 - 3g_1 \leq 0$, not mentioned in the text here, but nevertheless implemented exactly that same way in PyCO2SYS (according to `solve/initialise.py`).

Please have a look at the *original* code (also available from Munhoven, 2013b) and rewrite this paragraph more accurately.

Page 45, line 1039: The DOI provided for CO2SYS-MATLAB v1.1 has been dead for a long time. It does not resolve correctly since the CDIAC collections were moved to ESS-DIVE. Perhaps, you may get this problem fixed upstream, otherwise, it would be best to remove that DOI.

References

K. S. Deffeyes. Carbonate equilibria : A graphic and algebraic approach. *Limnol. Oceanogr.*, 10(3):412–426, 1965. doi: 10.4319/lo.1965.10.3.0412.

A. G. Dickson. An exact definition of total alkalinity and a procedure for the estimation of alkalinity and total inorganic carbon from titration data. *Deep-Sea Res. A*, 28(6):609–623, 1981. doi: 10.1016/0198-0149(81)90121-7.

M. Hagens and J. J. Middelburg. Generalised expressions for the response of pH to changes in ocean chemistry. *Geochim. Cosmochim. Ac.*, 187:334–349, 2016. doi: 10.1016/j.gca.2016.04.012.

E. Lewis and D. Wallace. Program developed for CO_2 system calculations. Technical Report 105, Carbon Dioxide Analysis Center, Oak Ridge National Laboratory, Oak Ridge (TN), 1998. URL `http://cdiac.ornl.gov/oceans/co2rprt.html`.

G. Munhoven. Mathematics of the total alkalinity-pH equation – pathway to robust and universal solution algorithms: the SolveSAPHE package v1.0.1. *Geosci. Model Dev.*, 6(4):1367–1388, 2013a. doi: 10.5194/gmd-6-1367-2013.

G. Munhoven. SolveSAPHE (Solver Suite for Alkalinity-PH Equations), v1.0.1, August 2013b.

G. Munhoven. SolveSAPHE-r2 (v2.0.1): revisiting and extending the Solver Suite for Alkalinity-PH Equations for usage with CO_2, HCO_3^- or CO_3^{2-} input data. *Geosci. Model Dev.*, 14(7):4225–4240, 2021. doi: 10.5194/gmd-14-4225-2021.

J. C. Orr and J.-M. Epitalon. Improved routines to model the ocean carbonate system : mocsy 2.0. *Geosci. Model Dev.*, 8(3):485–499, 2015. doi: 10.5194/gmd-8-485-2015.

J. C. Orr, J.-M. Epitalon, A. G. Dickson, and J.-P. Gattuso. Routine uncertainty propagation for the marine carbon dioxide system. *Mar. Chem.*, 207:84–107, 2018. doi: 10.1016/j.marchem.2018.10.006.

J. D. Sharp, D. Pierrot, M. P. Humphreys, J.-M. Epitalon, J. C. Orr, E. R. Lewis, and D. W. R. Wallace. CO2SYSv3 for MATLAB, v3.2, May 2021.

D. A. Wolf-Gladrow, R. E. Zeebe, C. Klaas, A. Körtzinger, and A. G. Dickson. Total alkalinity : The explicit conservative expression and its application to biogeochemical processes. *Mar. Chem.*, 106(1-2):287–300, 2007. doi: 10.1016/j.marchem.2007.01.006.

Y.-Y. Xu, D. Pierrot, and W.-J. Cai. Ocean carbonate system computation for anoxic waters using an updated CO2SYS program. *Mar. Chem.*, 195:90–93, 2017. doi: 10.1016/j.marchem.2017.07.002.

R. E. Zeebe and D. Wolf-Gladrow. *CO_2 in seawater : Equilibrium, kinetics, isotopes*, volume 65 of *Elsevier Oceanography Series*. Elsevier, Amsterdam (NL), 2001. ISBN 978-0-444-50579-8. URL `http://www.sciencedirect.com/science/bookseries/04229894/65`.

Liège, 8th July 2021

Guy Munhoven