SOLVEr Suite for Alkalinity-PH Equations
SOLVESAPHE, Version 2.0

—

User Manual

G. Munhoven

Dépt. d'Astrophysique, Géophysique et Océanographie,
Université de Liège,
B–4000 Liège, Belgium,
eMail: Guy.Munhoven@uliege.be

Manual Version 2.0 (31st December 2020)

Abstract

This manual describes the usage and the main technical aspects of the SOLVEr *Suite for Alkalinity-PH Equations* (SOLVESAPHE), Version 2.0). The codes provide a self-contained Fortan 90 implementation of the universal and robust algorithms for solving the total alkalinity-*p*H equation variants presented in Munhoven (2021), along with new implementations of a few other, previously published solvers, as well as some auxiliary functions and subroutines. Also included are the original driver programs that were used to produce the results reported in Munhoven (2021). The codes of SOLVESAPHE are free software and they are released under the GNU Lesser General Public Licence Version 3 or later. Bug reports, reports about successful builds on other platforms or with other compilers than those listed below, as well as contributions are welcome!

All publicly available SOLVESAPHE code versions are archived on ZENODO under the DOI 10.5281/zenodo.3752250

Contents

1 Requirements

1.1 Compiler and Preprocessor

A standard compliant Fortran 90 compiler and a C-preprocessor or a compatible preprocessor (such as, e.g., fpp) are required to compile the codes. Preprocessor directives are included for enabling or disabling specific parts (debugging messages, optional code parts and variants, ...) in the solver modules. In the main driver programs, they are used to select among the cases treated in the paper.

After pre-processing, the modules' source files are strictly standard conforming Fortran 90. The codes take advantage of the configurable precision facilities offered by Fortran 90. A single change in the module `mod_precision.f90` is thus sufficient to consistently use the codes in single or double precision, as required by the user.

SOLVESAPHE is self-contained and does not require any external libraries.

The codes were tested with

- GNU Fortran (GCC) 5.0.4 on Ubuntu 16.04 (x86_64)

1.2 Contents and summary description of `solvesaphe2.tar.gz`

Core modules

`mod_phsolvers.F90`
: module containing the solvers for the total alkalinity-*p*H equations; also contains the functions and subroutines for the equation and its derivative

`mod_chemconst.f90`
: module with the parametrizations for the stoichiometric constants; also holds the products of the constants (the Π_j factors in the main paper) for the considered acid systems and the *p*H scale conversion factor (denoted s in the main paper) alkalinity-*p*H equation

`mod_phsolvers_logging.F90`
: extended version of `mod_phsolvers.F90` that does extra bookkeeping regarding the number of iterations, types of limiting events, etc.

Configuration Modules

`mod_precision.f90`
: module to select the working precision (REAL data type) to be used in all the source codes

Drivers and main programs

`main_check.f90`
: program to carry out the constants' value checks provided by the subroutine `checkconstants` in `mod_chemconsts.f90`

`driver_at_logging.F90`
: driver for determining the numbers of iterations and other internal information for test cases SW1, SW2, SW3 and SW4; uses `mod_phsolvers_logging`

`driver_at_general2.F90`
: driver for running the test cases SW1, SW2, SW3 and SW4; uses `mod_phsolvers`

`driver_at_carbonate.F90`
: driver for calculating the complete carbonate system speciation for the test cases SW1, SW2, SW3 and SW4 (and SW5, a reduced variant of SW3) with the Alk_T-C_T pair; uses `mod_phsolvers` and `mod_chemspeciation.f90` and is compatible with SOLVESAPHE v. 1

Other files

`mod_chemspeciation.f90`
: module providing a collection of subroutines to calculate the speciation of the different acid systems considered, as a function of *p*H

`makefile`
: makefile for building the test case programs and the stoichiometric constants' checking utility

`COPYING.LESSER`
: text of the GNU Lesser General Public Licence version 3

`COPYING`
: text of the GNU General Public Licence version 3 (underlying the GNU Lesser General Public Licence version 3)

`manual.pdf`
: this manual.

2 Building the test cases

After uncompressing and extracting the contents of the archive, e.g., with

```
$ tar xvfz solvesaphe2.tar.gz
```

the various provided programs can be build with

```
$ make target
```

where *target* may be one of

`checkconsts` — for compiling `main_check.f90` and its dependencies;

`at_general2` — for compiling `driver_at_general2.F90` and its dependencies;

`at_logging` — for compiling `driver_at_logging.F90` and its dependencies.

`at_carbonate` — for compiling `driver_at_carbonate.F90` and its dependencies.

The generated executables have the same name as the target in each instance. Please notice that it may be necessary to proceed to a clean build (i.e., first `make clean`) to consistently rebuild a program after changes in a module. This is due to complications arising from the compilation of Fortran 90 modules, during which two files (the `*.o` and the `*.mod` files are generated, whereas make can only control one of them (here the `*.o`).

`checkconsts` writes its output to the file named `checkconsts.log`

Specific test cases and other boundary conditions (temperature, pressure, salinity) and variants (type of initialisation, etc.) are selected by adapting the precompiler directives that can be found right after the copyright and licencing statements at the beginning of each one of the driver files.

The optionally generated result files (when `#define CREATEFILES` is used, these are created) are Fortran unformatted (binary) files which can be directly read in by some post-processing applications (e.g., IDL).

3 Description and usage of the core modules

All of the solvers are implemented as `FUNCTION` sub-programs. The arguments to provide include the relevant alkalinity and total dissolved acid concentrations, and optionally, one argument (`p_hini`) to provide an initial value to start the iterations and a second one (`p_val`) to retrieve the equation residual if wanted. For floating-point calculations, all of the codes consistently use a configurable `REAL` data type that must be selected in `MOD_PRECISION` (source file `mod_precision.f90`). The identifier of that data type is stored in the `INTEGER` parameter `wp`. The default type is set to `DOUBLE PRECISION` (`wp=KIND(1D0)`). The solver `FUNCTION` sub-programs are accordingly declared to be of type `REAL(KIND=wp)` throughout.

If the optional initialisation argument is left out, the solver calls the cubic polynomial initialisation schemes described in Munhoven (2021) and its technical supplement. Each module contains a version of that initialisation routine suitable for the respective solvers. Upon completion, the solver functions either return the calculated $[H^+]$ value(s), or the `NaN` value for $[H^+]$ defined by `pp_hnan` (set to -1 by default) in the respective solver module if no such root could be found or if there is no root. In the latter case, the corresponding `p_val` is set to `HUGE(1._wp)`.

All of the solvers use the thermodynamic constants' products (Π_j's) stored in the module `MOD_CHEMCONSTS` at the time of the call. These have to be initialised before calling the solver.

The convergence criterion used with all of the solvers requires that the relative extent of the bracketing interval, which is continuously updated as iterations proceed, compared to its midpoint falls below a given threshold. The threshold value is set by the parameter `pp_rdel_ah_target` that can be found in each module. It is set to 10^{-8} by default.

3.1 Module `MOD_CHEMCONSTS`

The source code of this module is in `mod_chemconsts.f90`.

It provides a basic, but comprehensive set of `FUNCTION` sub-programs (type `REAL(KIND=wp)`) to calculate the stoechiometric constants for

- the self-ionization of water (id.: `wat`)
- the dissociation series of carbonic acid (id.: `dic`)
- the dissociation of boric acid (id.: `bor`)
- the dissociation of silicic acid (id.: `sil`)
- the dissociation series of phosphoric acid (id.: `po4`)
- the dissociation of ammonium (id.: `nh4`)
- the dissociation of hydrogen sulphide (id.: `h2s`)
- the dissociation of bisulphate (id.: `so4`)
- the dissociation of hydrogen fluoride (id.: `flu`)

as a function of temperature (in kelvin), salinity (no units) and applied pressure (in bar). All of the concentrations are supposed to be expressed in mol/kg-solution. For some acids, several parametrizations may be given. Calculations in the respective `FUNCTION` subprograms use the same *p*H scale as originally published. Auxiliary functions to convert between *p*H scales (free, total and seawater scales) are provided.

The solver modules interact with `MOD_CHEMCCONSTS` only via the `api1_aaa`, `api2_aaa`, ... and the `aphscale` variables that hold the stoichiometric constants' products, i.e., the Π_j factors in the main paper. The *aaa* part in the names identify the respective acid systems on the basis of the three-letter codes given in brackets in the list above. `aphscale` holds the *p*H scale conversion factor s (Munhoven, 2013, eqn. (20)).

The FUNCTION sub-programs for calculating the individual constants are kept PRIVATE in the module. This helps to avoid potential misuse. A specific SUBROUTINE should be used to initialize the relevant api1_*aaa*, api2_*aaa*, ... and aphscale variables. Special attention must be paid to consistently use a common *p*H scale for the set of constants, and convert where necessary. Two sample subroutines, SETUP_API4PHTOT and SETUP_API4PHSWS respectively based upon the total and seawater scales are provided in mod_chemconsts.f90. For the exact names and detailed characteristics (references, *p*H scale, units, etc.), please refer to the source code file and the comments included.

mod_chemconsts.f90 further contains FUNCTION sub-programs to calculate

- the solubility of CO_2 gas
- the solubility product of calcite
- the solubility product of aragonite
- the concentrations of some conservative seawater solutes as a function of salinity (calcium, boron, fluoride, sulphate)
- the density of seawater as a function of temperature, salinity and pressure.

For further information, please refer to the detailed comments in the source code.

3.2 Module MOD_PHSOLVERS

The source code of this module is in mod_phsolvers.F90. It provides the following solvers, suitable for solving the different variants of the alkalinity-*p*H equation obtained for the C_T, CO_2, HCO_3^- and CO_3^{2-} as the characteristic variable for quantifying the carbonate system (eqn.(2) in combination with either one of eqns. (3)–(6) in Munhoven (2021)):

SOLVE_AT_GENERAL2
: the algorithm from Munhoven (2021) based upon Newton-Raphson, regula falsi and bisection iterations;

SOLVE_AT_GENERAL
: a wrapper around SOLVE_AT_GENERAL2 to replicate the API of SOLVESAPHE v. 1 – uses the Alk_T-C_T pair;

SOLVE_AT_GENERAL2_SEC
: the variant of SOLVE_AT_GENERAL2 that uses secant instead of Newton-Raphson iterations;

SOLVE_AT_GENERAL_SEC
: a wrapper around SOLVE_AT_GENERAL2_SEC to replicate the API of SOLVESAPHE v. 1 – uses the Alk_T-C_T pair;

The module furthermore contains the following auxiliary sub-programs:

ANW
: a function sub-program to calculate the total alkalinity component not related to water self-ionization and optionally also its derivative;

ANW_INFSUP
: a subroutine to calculate the infimum and the supremum of $Alk_{nW}([H^+])$, i.e., of the total alkalinity component not related to water self-ionization — these are required to calculate the safe bounds;

EQUATION_AT
: a function sub-program to evaluate the rational function form of the total alkalinity-*p*H equation and optionally also its derivative – requires ANW;

HINFSUPINI
: a subroutine that returns the number of roots of the problem being solved, the upper and lower bounds for each root, if any, and an estimate for the initial value for the iterative solver. This subroutine actually only selects the specific subroutine for the type of input data being used:

  - HINFSUPINI_DIC for AlkT & C_T — further uses HINI_ACB_DIC to determine the initial value;
  - HINFSUPINI_CO2 for AlkT & CO_2 — further uses HINI_ACB_CO2 to determine the initial value;
  - HINFSUPINI_HCO3 for AlkT & HCO_3^- — further uses HINI_ACBW_HCO3 to determine the initial value;
  - HINFSUPINI_CO3 for AlkT & CO_3^{2-} — further uses HINI_ACBW_CO3 to determine the initial value;

The module offers a few customization options:

- the pre-compiler directive SAFEGEOMEAN_INIT allows to bypass the initialisation of the iterations based upon the cubic polynomial approximations: if it is define'd, the geometric mean of the upper and lower root bounds is used instead. By default, it is not define'd and th standard initialisation is used. Please notice that if initial values are provided when calling a solver, these are unconditionally used, and only brought within brackets.
- the maximum number of iterations allowed for each method is controlled by the parameters jp_maxniter_*idmethod*, where the method identificator *idmethod* should be substituted by
  - atgen for SOLVE_AT_GENERAL
  - atsec for SOLVE_AT_GENERAL_SEC

  All of these are set to 100 by default.

After the call, the number of iterations actually performed to determine each of the roots can be retrieved from the niter_*idmethod* variable (a DIMENSION(2) array) related to the used solver, and that is provided in the module (with *idmethod* as above). For other details, such as the number, order, type and shape of the arguments in the solver function sub-programs, please refer to the comments in the source code.

3.3 Practical usage: typical sequence

After the user has chosen a suitable solver from mod_phsolvers.F90, the steps required to include it into their own program are as follows.

1. Select an adequate precision in the MOD_PRECISION module (mod_precision.f90).
2. Make your choice for the stoichiometric constants (i.e., chose *p*H scale, etc.): either use one of the SETUP_API_... routines that are provided in mod_chemconsts.f90, adapt one of them or create a new one – make sure it initializes all of the required api_... variables in mod_chemconsts.f90 and, if necessary, also the aphscale variable for the *p*H scale conversion.
3. In the scoping unit of your code that requires *p*H calculation, include the Fortran directives

   ```
   USE MOD_PRECISION
   USE MOD_CHEMCONSTS
   USE MOD_PHSOLVERS
   ```

   If the exact speciation of the acid systems are also required, one may furthermore include

   ```
   USE MOD_CHEMSPECIATION
   ```

4. Before each call of the solver, make sure that the set of chemical constants (for the desired temperature, salinity and pressure) is correctly initialized by calling the adequate `SETUP_API...` subroutine.

5. Call the chosen solver function.

6. If required, call the relevant `SPECIATION_aaa(...)` subroutines from `MOD_CHEMSPECIATION` to calculate the actual speciations (where the *aaa* parts in the subroutine names identify the respective acid systems on the basis of the three-letter codes given in brackets in section 3.1 above). Notice that the `SPECIATION_aaa(...)` subroutines rely upon the `api1_aaa`, `api2_aaa`, ...values that were used to calculate *p*H. Their values must therefore not be changed before the speciation calculations for the sake of consistency.

3.4 Streamlining, extensions, ...

Recommended streamlining

In case a large number of *p*H calculations for many different temperature, salinity and pressure conditions are required, it is recommended to restrict the `SETUP_API_...` routine to the strict minimal set of stoichiometric constants required to solve the problem. Because of the exponential or power function evaluations required for the calculation of each single of the chemical constants, calculating even the bare minimum set of constants may take a significant fraction of the total time needed to complete one *p*H determination.

How to extend it

SOLVESAPHE is obviously extensible: users may add extra acid systems if needed. This task is not complicated, but requires changes at different places.

1. First, decide to which extent the dissociation of considered acid is going to be taken into account (define the number of dissociations n and the integer m that sets the zero proton level of the system — see Munhoven (2013, 2021) for more details).

2. Add function subprograms in `mod_chemconsts.f90` to make available the required chemical constants. All of the routines in SOLVESAPHE expect concentrations to be expressed in mol/kg. The arguments of the subroutine functions should adhere to the common structure `t_k, s, p_bar` (in this order), where `t_k` is the temperature in kelvin, `s` is salinity and `p_bar` the applied pressure in bar. It is recommended to keep the actual functions that evaluate the parametrizations private in the module.

3. Add the `api1_aaa`, `api2_aaa`, ... variables to `mod_chemconsts.f90`, chosing a unique identificator *aaa* for the new contributing acid system.

4. Add a new speciation subroutine `SPECIATION_aaa(...)` to `mod_chemspeciation.f90` if required, distinguished by the unique identificator *aaa* for the new contributing acid system.

5. Prepare a `SETUP_API_...` routine to initialize all of the `api*` variables of all the acid systems considered (incl water self-ionization and the *p*H scale conversion factor `aphscale`.

6. Amend `mod_at_phsolvers.F90` to take the effect of the additional acid system into account by amending `ANW_INFSUP` and `ANW` (add the dummy `p_aaatot` to the argument lists and adapt the code), then include `p_aaatot` in the argument list of the solver function subprogram and adapt the calls to `ANW_INFSUP` and each of the four `HINFSUPINI_ccc` and in `HINFSUPINI`, as well as in the two solver routines.

7. Change the calls to the new `SETUP_API_...` (only if its name has changed, the argument list should have remained the same) and adapt the list of arguments wherever the solver is called in the main program to make it conforming with the changes made in the module.

History of the SOLVESAPHE v. 2 codes

31st December 2020
: initial release

History of this document

31st December 2020 (version 2.0 of the manual)
: initial release

References

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

Munhoven, G.: SolveSAPHE-r2: revisiting and extending the Solver Suite for Alkalinity-PH Equations for usage with CO_2, bicarbonate or carbonate concentrations instead of C_T, Geosci. Model Dev. Discuss., 2021.