HGC

Release 0.1

Jun 22, 2023

Contents

HydroGeoChemistry - A python package for correction, validation and analysis of ground water quality samples

CHAPTER 1

Getting Started

HGC (HydroGeoChemistry) is a Python package for correction, validation and analysis of ground water quality samples.

Main features:

- Handle common erros and peculiarities in hydrochemical data, such as negative concentrations, detection limits and non-numerical placeholders
- Calculate common ratios, such as for example MONC, SUVA and HCO3 to the sum of all anions
- Classify groundwater samples according to the Stuyfzand water types
- Calculate missing concentrations through the ion balance, with PhreeqPython

1.1 Install

HGC requires Python 3.6 or later.

To get HGC, use the following command:

```
pip install hgc
```

1.2 Philosophy

HGC is an extension of the Pandas DataFrame, giving your DataFrame hydrochemistry superpowers. You can thus mix HGC with your regular Pandas/Numpy workflows.

A DataFrame does not need to conform to a specific format to work with HGC, however it is required that:

- Each row in the DataFrame represents a groundwater quality sample
- Each column represents a groundwater quality parameter

HGC checks if column names in the DataFrame match with chemical parameters that it recognizes. Such columns should be in the units that HGC expects. In addition to 'HGC-enabled' columns, the DataFrame can contain an arbitrary number of non-hydrochemistry columns (such as XY-locations, comments, or other measured quantities), HGC simply ignores those columns.

1.3 Conventions

The naming conventions of the column names is that they are all in lower case with an underscore between separate words. E.g. the EC measured in the lab is indicated with *ec_lab*. The only exception to this is the notation of chemical structures and atoms; there standard capitalization is used. E.g. the column name for total total nitrogen is *N_total* and for ortho-phosphate *PO4_ortho*.

CHAPTER 2

Tutorial

Practical examples

We always start by importing HGC:

```
In [1]: import pandas as pd

In [2]: import hgc
```

2.1 Creating a HGC-enabled DataFrame

A hydrochemical groundwater analysis typically starts with a 'normal' Pandas DataFrame, in which each row contains a groundwater quality sample, and each column represents a water quality parameter. The DataFrame may contain concentrations of different chemical compounds, possibly exceeding the detection limit (denoted with a '<' or '>' prefix). There may also be errors in the data, such as negative concentrations or text placeholders.

Note: Please refer to the excellent WaDI package to get your excel or csv file with measurements in a format that HGC understands. In this tutorial, we create our own *DataFrame* for clarity.

```
In [3]: testdata = {'alkalinity': [0.0], 'Al': [2600], 'Ba': [44.0],
   ...:             'Br': [0.0], 'Ca': [2.0], 'Cl': [19.0],
   ...:             'Co': [1.2], 'Cu': [4.0], 'doc': [4.4],
   ...:             'F': [0.08], 'Fe': [0.29], 'K': [1.1],
   ...:             'Li': [5.0], 'Mg': [1.6], 'Mn': ['< 0.05'],
   ...:             'Na': [15.0], 'Ni': [7.0], 'NH4': ['< 0.05'],
   ...:             'NO2': [0.0], 'NO3': [22.6], 'Pb': [2.7],
   ...:             'PO4': ['0.04'], 'ph': [4.3], 'SO4': [16.0],
   ...:             'Sr': [50], 'Zn': [60.0] }
   ...:

In [4]: df = pd.DataFrame.from_dict(testdata)
```

(continues on next page)

(continued from previous page)

```
In [5]: df
Out[5]:
   alkalinity    Al    Ba   Br   Ca    Cl  ...   Pb   PO4   ph   SO4  Sr    Zn
0         0.0  2600  44.0  0.0  2.0  19.0  ...  2.7  0.04  4.3  16.0  50  60.0

[1 rows x 26 columns]
```

Since the data in this DataFrame is messy, we cannot use it yet for hydrochemical calculations. HGC can check if the data contains obvious errors:

```
In [6]: is_valid = df.hgc.is_valid

In [7]: is_valid
Out[7]: False
```

The DataFrame may contain any kind of columns and column names. However, HGC will only recognize a specific set of columns with names of hydrochemical parameters.

```
In [8]: from hgc.constants import constants

In [9]: print([*constants.atoms])
['H', 'He', 'Li', 'Be', 'B', 'C', 'N', 'O', 'F', 'Ne', 'Na', 'Mg', 'Al', 'Si', 'P', 'S
↪', 'Cl', 'Ar', 'K', 'Ca', 'Sc', 'Ti', 'V', 'Cr', 'Mn', 'Fe', 'Co', 'Ni', 'Cu', 'Zn',
↪ 'Ga', 'Ge', 'As', 'Se', 'Br', 'Kr', 'Rb', 'Sr', 'Y', 'Zr', 'Nb', 'Mo', 'Tc', 'Ru',
↪'Rh', 'Pd', 'Ag', 'Cd', 'In', 'Sn', 'Sb', 'Te', 'I', 'Xe', 'Cs', 'Ba', 'La', 'Ce',
↪'Pr', 'Nd', 'Pm', 'Sm', 'Eu', 'Gd', 'Tb', 'Dy', 'Ho', 'Er', 'Tm', 'Yb', 'Lu', 'Hf',
↪'Ta', 'W', 'Re', 'Os', 'Ir', 'Pt', 'Au', 'Hg', 'Tl', 'Pb', 'Bi', 'Po', 'At', 'Rn',
↪'Fr', 'Ra', 'Ac', 'Th', 'Pa', 'U', 'Np', 'Pu', 'Am', 'Cm', 'Bk', 'Cf', 'Es', 'Fm',
↪'Md', 'No', 'Lr', 'Rf', 'Db', 'Sg', 'Bh', 'Hs', 'Mt', 'Ds', 'Rg', 'Cn', 'Nh', 'Fl',
↪'Mc', 'Lv', 'Ts', 'Og']

In [10]: print([*constants.ions])
['CH4', 'H2S', 'S', 'CO2', 'alkalinity', 'O2_field', 'O2_lab', 'O2', 'KMnO4', 'NH4',
↪'NO2', 'NO3', 'N_kj', 'N', 'PO4', 'PO4_ortho', 'P', 'SiO2', 'SO4_ic', 'SO4', 'doc',
↪'toc', 'cod']

In [11]: print([*constants.properties])
['ec_field', 'ec_lab', 'ec', 'ph_field', 'ph_lab', 'ph', 'temp_field', 'temp_lab',
↪'temp', 'eh_field', 'turb', 'uva254']
```

You can also retreive the details of each compound, such as the expected units, full name or molar weight:

```
In [12]: constants.atoms['H']
Out[12]: Atom(feature='H', name='Hydrogen', unit='mg/L', mw=1.00794, oxidized=1.0,␣
↪reduced=0.0, SMOW=0.0)

In [13]: constants.properties['ec']
Out[13]: Properties(feature='ec', name='EC converted to 20°C', example='read', unit=
↪'µS/cm', phreeq_name=None)
```

For your convenience, all units for all allowed (columns with) atoms, ions and properties are enlisted here *here*.

Since in this case our DataFrame contains negative concentrations, detection limits (rows with '<' or '>') and incorrect data types (e.g. string columns that are supposed to be numeric), HGC will initially report that the DataFrame is invalid. HGC can automatically solve inconsistencies with the 'make_valid' method. As a result, negative concentrations are replaced by 0; concentrations below detection limit are replaced by half the limit; concentrations above the upper

detection limit are replaced by 1.5 times that limit.

```
In [14]: df.hgc.make_valid()

In [15]: is_valid = df.hgc.is_valid

In [16]: is_valid
Out[16]: True

In [17]: df
Out[17]:
   alkalinity    Al    Ba   Br   Ca    Cl  ...   Pb   PO4   ph   SO4  Sr    Zn
0         0.0  2600  44.0  0.0  2.0  19.0  ...  2.7  0.04  4.3  16.0  50  60.0

[1 rows x 26 columns]

# Recognized HGC columns
In [18]: hgc_cols = df.hgc.hgc_cols

In [19]: print(hgc_cols)
['Li', 'F', 'Na', 'Mg', 'Al', 'Cl', 'K', 'Ca', 'Mn', 'Fe', 'Co', 'Ni', 'Cu', 'Zn', 'Br
↪', 'Sr', 'Ba', 'Pb', 'alkalinity', 'NH4', 'NO2', 'NO3', 'PO4', 'SO4', 'doc', 'ph']
```

2.2 Calculations

Now that our DataFrame is valid, we can use all HGC methods, such as calculating the Base Exchange Index of each row; this is added as column to *df*:

```
In [20]: df.hgc.get_bex()

In [21]: df.bex
Out[21]:
0    0.238022
Name: bex, dtype: float64
```

We can also classify each sample into the Stuyfzand water type:

```
In [22]: df.hgc.get_stuyfzand_water_type()

In [23]: df.water_type
Out[23]:
0    g*NaNO3o
Name: water_type, dtype: object
```

Or get the sum of all anions (using the Stuyfzand method):

```
In [24]: df.hgc.get_sum_anions()

In [25]: df.sum_anions
Out[25]:
0    1.282127
Name: sum_anions, dtype: float64
```

It is also possible to compute common hydrochemical ratios between different compounds. HGC calculates ratios for all columns that are available and ignores any missing columns.

```
In [26]: df.hgc.get_ratios()

In [27]: df.cl_to_na
Out[27]:
0    0.82138
Name: cl_to_na, dtype: float64
```

For all these above mentioned *get* functions, the columns are added to the dataframe. Most of the times this is convenient, but there are also cases where you don't want to add them to the DataFrame but only want to return the result. In that case, one could use the *inplace* argument; this works the same as native pandas methods that have this argument With *inplace=True* (the default), the columns are added to the DataFrame (as shown in the examples above). With *inplace=False* the columns are not added to the database but returned as a pandas *Series* or *DataFrame*. E.g., for the Stuyfzand water type (a *Series*) or *ratios* (a *DataFrame*):

```
In [28]: water_type = df.hgc.get_stuyfzand_water_type(inplace=False)

In [29]: water_type
Out[29]:
0    g*NaNO3o
Name: swt, dtype: object

In [30]: ratios = df.hgc.get_ratios(inplace=False)

In [31]: ratios
Out[31]:
   cl_to_br  cl_to_na  ...  hco3_to_sum_anions  hco3_to_ca_and_mg
0       inf   0.82138  ...                 0.0                0.0

[1 rows x 8 columns]
```

2.2.1 Consolidation

A common situation is that one single parameter of a sample is measured with several methods or in different places. Parameters such as EC and pH are frequently measured both in the lab and field, and SO4 and PO4 are frequently measured both by IC and ICP-OES. Normally we prefer the field data for EC and pH, but ill calibrated sensors or tough field circumstances may prevent these readings to be superior to the lab measurement. In such cases we want select from multiple columns the one to use for subsequent calculations, by consolidating into one single column containing the best measurements, possibly filling gaps with measurements from the inferior method. Let's consider this example:

```
In [32]: testdata = {
   ....:     'ph_lab': [4.3, 6.3, 5.4], 'ph_field': [4.4, 6.1, 5.7],
   ....:     'ec_lab': [304, 401, 340], 'ec_field': [290, 'error', 334.6],
   ....: }
   ....:

In [33]: df = pd.DataFrame.from_dict(testdata)

In [34]: df
Out[34]:
   ph_lab  ph_field  ec_lab ec_field
0     4.3       4.4     304      290
1     6.3       6.1     401    error
2     5.4       5.7     340    334.6
```

(continues on next page)

(continued from previous page)

```
In [35]: df.hgc.make_valid()

In [36]: df
Out[36]:
   ph_lab  ph_field  ec_lab  ec_field
0     4.3       4.4     304     290.0
1     6.3       6.1     401       NaN
2     5.4       5.7     340     334.6

In [37]: df.hgc.consolidate(use_ph='field', use_ec='lab', use_temp=None,
   ....:                    use_so4=None, use_o2=None)
   ....:

In [38]: df
Out[38]:
    ph     ec
0  4.4  304.0
1  6.1  401.0
2  5.7  340.0
```

Warning: Note that omitting `use_so4=None` in the function call, would let the function fall back to the default which is `ic`. Because the column `so4_ic` is not in the dataframe this will return an error. The same holds for `use_temp` and `use_o2`.

```
In [39]: df.hgc.consolidate(use_ph='field', use_ec='lab', use_temp=None,)
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-39-2cbb7cd1e5e6> in <module>
----> 1 df.hgc.consolidate(use_ph='field', use_ec='lab', use_temp=None,)

~/checkouts/readthedocs.org/user_builds/hgc/checkouts/latest/hgc/samples_frame.py in␣
↪consolidate(self, use_ph, use_ec, use_so4, use_o2, use_temp, use_alkalinity, merge_
↪on_na, inplace)
    346                 raise ValueError(f"Column {source} not present in DataFrame.␣
↪Use " +
    347                                  f"use_{param.lower()}=None to explicitly␣
↪ignore consolidating " +
--> 348                                  f"this column.")
    349
    350

ValueError: Column SO4_ic not present in DataFrame. Use use_so4=None to explicitly␣
↪ignore consolidating this column.
```

2.3 Visualizing and exporting

The great thing about HGC is that your DataFrame gets hydrochemical superpowers, yet all functions that you expect from a regular Pandas DataFrame are still available, allowing you to easily import/export and visualize data.

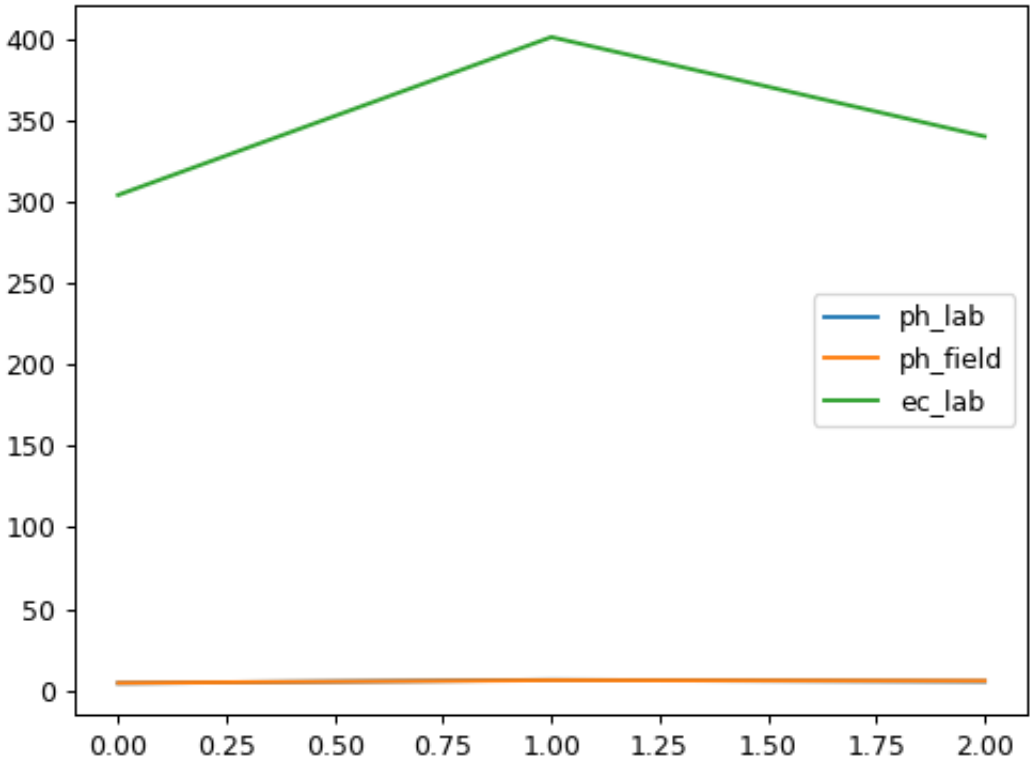
```
In [40]: df.std()
Out[40]:
```

(continues on next page)

(continued from previous page)

```
ph      0.888819
ec     49.034002
dtype: float64

In [41]: df.plot()
Out[41]: <AxesSubplot:>
```



2.4 Coupling to PHREEQC

Another great superpower of HGC is that it allows easy geochemistry *directly on your dataframe*! It currently has coupling with the popular geochemistry software PHREEQC via its python wrappers as implemented by the phreeqpython package.

Let's extend the above DataFrame a little to make it more meaningful in the context of this coupling:

```
In [42]: testdata = {
   ....:     'ph_lab': [4.5, 5.5, 7.6], 'ph_field': [4.4, 6.1, 7.7],
   ....:     'ec_lab': [304, 401, 340], 'ec_field': [290, 'error', 334.6],
   ....:     'temp': [10, 10, 10],
   ....:     'alkalinity':  [0, 7, 121],
   ....:     'O2':  [11, 0, 0],
```

(continues on next page)

(continued from previous page)

```
   ....:     'Na': [9,20,31], 'K':[0.4, 2.1, 2.0],
   ....:     'Ca':[1,3,47],
   ....:     'Fe': [0.10, 2.33, 0.4],
   ....:     'Mn': [0.02, 0.06, 0.13],
   ....:     'NH4': [1.29, 0.08, 0.34],
   ....:     'SiO2': [0.2, 15.4, 13.3],
   ....:     'SO4': [7,19,35],
   ....:     'NO3': [3.4,0.1,0],
   ....: }
   ....:

In [43]: df = pd.DataFrame.from_dict(testdata)

In [44]: df.hgc.make_valid()

In [45]: df.hgc.consolidate(use_ph='lab', use_ec='lab', use_temp=None,
   ....:                    use_so4=None, use_o2=None)
   ....:
```

With this DataFrame, we can do some PHREEQC calculations. For example, we can calculate the saturation index of different minerals like Calcite:

```
In [46]: df.hgc.get_saturation_index('Calcite')

In [47]: df['si_calcite'] # or df.si_calcite
Out[47]:
0   -999.000000
1     -4.722956
2     -0.288641
Name: si_calcite, dtype: float64
```

The mineral name will be added as a column named *si_<mineral_name>* where *<mineral_name>* is the name of the mineral as given to PHREEQC but all letters in *lower case* (and don't forget the underscore). The saturation index (SI) of a mineral can only be retrieved if they are defined in the phreeqc database used by phreeqpython. If the mineral is not defined, always an SI of -999 will be returned.

This also works for the partial pressure of gasses (because in PhreeqC both minerals and gasses are defined as *PHASES*; see below for explanation of the coupling to PhreeqC). But it looks better if one uses the alias *partial_pressure* which returns the same values but with a different name of the column (prepending pp instead of si, since it is the partial pressure and not the saturation index).

```
In [48]: df.hgc.get_saturation_index('CO2(g)')

In [49]: df['si_co2(g)']
Out[49]:
0   -999.000000
1     -1.715479
2     -2.609626
Name: si_co2(g), dtype: float64

In [50]: df.hgc.get_partial_pressure('CO2(g)')

In [51]: df['pp_co2(g)']
Out[51]:
0   -999.000000
1     -1.715479
```

(continues on next page)

(continued from previous page)

```
2     -2.609626
Name: pp_co2(g), dtype: float64
```

Similar to the SI, the specific conductance (SC), also known as electric conductance (EC) or EGV, is simply retrieved by calling:

```
In [52]: df.hgc.get_specific_conductance()

In [53]: df.sc
Out[53]:
0     37.193926
1     64.734047
2    218.187466
Name: sc, dtype: float64
```

Internally, these methods call the method *get_phreeqpython_solutions* to retrieve instances of the PhreeqPython *Solution* class. PhreeqPython is a Python package that allows the use of the Geochemical modeling package PhreeqC from within Python. HGC leverages this package to have a PhreeqC solution (or actually a PhreeqPython solution) for every row of the *SamplesFrame*. These are available to the user by calling

```
In [54]: df.hgc.get_phreeqpython_solutions()

In [55]: df.pp_solutions
Out[55]: ---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~/checkouts/readthedocs.org/user_builds/hgc/envs/latest/lib/python3.7/site-packages/
↪IPython/core/formatters.py in __call__(self, obj)
    700                 type_pprinters=self.type_printers,
    701                 deferred_pprinters=self.deferred_printers)
--> 702             printer.pretty(obj)
    703             printer.flush()
    704             return stream.getvalue()

~/checkouts/readthedocs.org/user_builds/hgc/envs/latest/lib/python3.7/site-packages/
↪IPython/lib/pretty.py in pretty(self, obj)
    392                         if cls is not object \
    393                                 and callable(cls.__dict__.get('__repr__')):
--> 394                             return _repr_pprint(obj, self, cycle)
    395
    396             return _default_pprint(obj, self, cycle)

~/checkouts/readthedocs.org/user_builds/hgc/envs/latest/lib/python3.7/site-packages/
↪IPython/lib/pretty.py in _repr_pprint(obj, p, cycle)
    698     """A pprint that just redirects to the normal repr function."""
    699     # Find newlines and replace them with p.break_()
--> 700     output = repr(obj)
    701     lines = output.splitlines()
    702     with p.group():

~/checkouts/readthedocs.org/user_builds/hgc/envs/latest/lib/python3.7/site-packages/
↪pandas/core/series.py in __repr__(self)
   1469             min_rows=min_rows,
   1470             max_rows=max_rows,
-> 1471             length=show_dimensions,
   1472         )
   1473         return buf.getvalue()
```

(continues on next page)

(continued from previous page)

```
~/checkouts/readthedocs.org/user_builds/hgc/envs/latest/lib/python3.7/site-packages/
↪pandas/core/series.py in to_string(self, buf, na_rep, float_format, header, index,␣
↪length, dtype, name, max_rows, min_rows)
   1532             max_rows=max_rows,
   1533         )
-> 1534         result = formatter.to_string()
   1535
   1536         # catch contract violations

~/checkouts/readthedocs.org/user_builds/hgc/envs/latest/lib/python3.7/site-packages/
↪pandas/io/formats/format.py in to_string(self)
    389
    390         fmt_index, have_header = self._get_formatted_index()
--> 391         fmt_values = self._get_formatted_values()
    392
    393         if self.is_truncated_vertically:

~/checkouts/readthedocs.org/user_builds/hgc/envs/latest/lib/python3.7/site-packages/
↪pandas/io/formats/format.py in _get_formatted_values(self)
    378             float_format=self.float_format,
    379             na_rep=self.na_rep,
--> 380             leading_space=self.index,
    381         )
    382

~/checkouts/readthedocs.org/user_builds/hgc/envs/latest/lib/python3.7/site-packages/
↪pandas/io/formats/format.py in format_array(values, formatter, float_format, na_rep,
↪ digits, space, justify, decimal, leading_space, quoting)
   1238     )
   1239
-> 1240     return fmt_obj.get_result()
   1241
   1242

~/checkouts/readthedocs.org/user_builds/hgc/envs/latest/lib/python3.7/site-packages/
↪pandas/io/formats/format.py in get_result(self)
   1269
   1270     def get_result(self) -> list[str]:
-> 1271         fmt_values = self._format_strings()
   1272         return _make_fixed_width(fmt_values, self.justify)
   1273

~/checkouts/readthedocs.org/user_builds/hgc/envs/latest/lib/python3.7/site-packages/
↪pandas/io/formats/format.py in _format_strings(self)
   1332         for i, v in enumerate(vals):
   1333             if not is_float_type[i] and leading_space:
-> 1334                 fmt_values.append(f" {_format(v)}")
   1335             elif is_float_type[i]:
   1336                 fmt_values.append(float_format(v))

~/checkouts/readthedocs.org/user_builds/hgc/envs/latest/lib/python3.7/site-packages/
↪pandas/io/formats/format.py in _format(x)
   1312             else:
   1313                 # object dtype
-> 1314                 return str(formatter(x))
   1315
```

(continues on next page)

(continued from previous page)

```
   1316         vals = extract_array(self.values, extract_numpy=True)

~/checkouts/readthedocs.org/user_builds/hgc/envs/latest/lib/python3.7/site-packages/
↪pandas/io/formats/printing.py in pprint_thing(thing, _nest_lvl, escape_chars,␣
↪default_escapes, quote_strings, max_seq_items)
    231         result = f"'{as_escaped_string(thing)}'"
    232     else:
--> 233         result = as_escaped_string(thing)
    234
    235     return result

~/checkouts/readthedocs.org/user_builds/hgc/envs/latest/lib/python3.7/site-packages/
↪pandas/io/formats/printing.py in as_escaped_string(thing, escape_chars)
    207             escape_chars = escape_chars or ()
    208
--> 209         result = str(thing)
    210         for c in escape_chars:
    211             result = result.replace(c, translate[c])

~/checkouts/readthedocs.org/user_builds/hgc/envs/latest/lib/python3.7/site-packages/
↪phreeqpython/solution.py in __str__(self)
    239     # pretty printing
    240     def __str__(self):
--> 241         return "<PhreeqPython.Solution."+self.__class__.__name__ + " with␣
↪number '" + self.number + "'>"

TypeError: can only concatenate str (not "int") to str
```

Because all elements in this column are PhreeqPython *Solution*'s, PhreeqC can be used to calculate all kind of properties of each water sample of each row in the *SamplesFrame*. In the documentation of PhreeqPython all these are described. For example, one can derive the specific conductance or pH from the first sample:

```
In [56]: df.pp_solutions[0].sc
Out[56]: 37.19392619168676

In [57]: df.pp_solutions[0].pH
Out[57]: 4.5
```

or for all the samples:

```
In [58]: [s.sc for s in df.pp_solutions]
Out[58]: [37.19392619168676, 64.73404717149515, 218.18746625402372]
```

Note that these are the exact same results as above when *df.hgc.get_specific_conductance()* was called.

But also more involved operations are possible, for example, inspecting the speciation of the first sample in the original *SamplesFrame df*:

```
In [59]: df.pp_solutions[0].species
Out[59]:
{'Amm': 4.107427346238537e-10,
 'AmmH+': 7.145332058236736e-05,
 'AmmHSO4-': 6.038481976861183e-08,
 'Ca+2': 2.4719556914911714e-05,
 'CaHSO4+': 4.107750059503443e-11,
 'CaOH+': 1.206793597413229e-13,
 'CaSO4': 2.3258121349793236e-07,
```

(continues on next page)

(continued from previous page)

```
 'Fe(OH)2': 3.377294151086896e-19,
 'Fe(OH)2+': 3.0790467275285987e-13,
 'Fe(OH)3': 6.15008335198951e-16,
 'Fe(OH)3-': 3.481413027742945e-25,
 'Fe(OH)4-': 9.631965209582988e-21,
 'Fe(SO4)2-': 4.549314587256489e-19,
 'Fe+2': 1.7766173469538303e-06,
 'Fe+3': 8.017708354854963e-16,
 'Fe2(OH)2+4': 2.084732141128855e-25,
 'Fe3(OH)4+5': 7.00155604671684e-35,
 'FeHSO4+': 2.9536979682997056e-12,
 'FeHSO4+2': 3.2081256063657047e-20,
 'FeOH+': 5.078860843975143e-12,
 'FeOH+2': 5.756579318382429e-14,
 'FeSO4': 1.4104744825322718e-08,
 'FeSO4+': 3.3701491576993005e-16,
 'H+': 3.237684629013282e-05,
 'H2': 8.284273096100859e-21,
 'H2O': 55.50929780738274,
 'H2SiO4-2': 7.095512381986744e-21,
 'H3SiO4-': 8.93438650865223e-12,
 'H4SiO4': 3.3288309795364082e-06,
 'HSO4-': 1.5224767993236832e-07,
 'K+': 1.022746595129961e-05,
 'KSO4-': 3.578529154040609e-09,
 'Mn(NO3)2': 3.873920247618094e-15,
 'Mn(OH)3-': 1.6845954285204035e-28,
 'Mn+2': 3.612266976206157e-07,
 'Mn+3': 1.2482278567749458e-29,
 'MnOH+': 7.539734502001295e-14,
 'MnSO4': 2.8321415294557053e-09,
 'NO3-': 5.483700575514323e-05,
 'Na+': 0.0003913712170485659,
 'NaOH': 3.552855616723773e-24,
 'NaSO4-': 1.1642908789175805e-07,
 'O2': 0.00034378298237867114,
 'OH-': 9.532511716174231e-11,
 'SO4-2': 7.229439162542111e-05}
```

Note that units of these speciation calculations are in mmol/L.

One could even manipulate the solution by letting for example calcite precipitate and see how this changes pH

```
In [60]: desaturated_solutions = [s.desaturate('Calcite') for s in df.pp_solutions]

In [61]: pd.DataFrame(dict(
   ....:     original=df.ph,
   ....:     desaturated=[s.pH for s in desaturated_solutions],)
   ....: ).round(2)
   ....:
Out[61]:
   original  desaturated
0       4.5         4.48
1       5.5         5.47
2       7.6         7.60
```

For more examples, please visit the examples on the Github page of PhreeqPython.

CHAPTER 3

FAQ

Frequently Asked Questions

3.1 Where are total P and total N?

These are stored as the columns *P* and *N* respectively.

3.2 Which redox state is used for the phreeqc simulations?

It uses the default values as used by phreeqc itself, that is, *Fe(2)*, *As(3)* and *Mn(2)*.

3.3 Why does ammonium not contribute in the redox-equilibrium?

This is by design as its kinetics are generally too slow. It is added as a separate species in phreeqc (*Amm* instead of NH_4^+).

3.4 How do I report alkalinity and/or bicarbonate (HCO_3^-)?

It is assumed everywhere that the HCO_3^- concentration equals the alkalinity.

3.5 Why is my pH, temperature or other column not added to the SamplesFrame and/or recognized by HGC?

A common mistake is that the temperature is added with

```
df.temp = 10
```

But this is an invalid way of adding columns to a DataFrame and therefore, it is not recognized as a column Instead, use

```
df['temp'] = 10
```

CHAPTER 4

Glossary

Key definitions and abbreviations

Base Exchange Index (BEX) Can be negative (salinized water), zero (no base exchange) or positive (freshened water).

Chemical Oxygen Demand (COD) Chemical Oxygen Demand.

Detection limit Sensor accuracy limit, denoted by '<' or '>' in the data.

Electrical Conductivity (EC) Electrical Conductivity.

Mean Oxidation Number of Carbon (MONC) Mean Oxidation Number of Carbon

Stuyfzand Water Type Classification.

Total Dissolved Solids (TDS) Total Dissolved Solids.

Total Inorganic Carbon (TIC) Total Inorganic Carbon.

CHAPTER 5

Units

An overview of the used units of all columns in a SamplesFrame.

5.1 Atoms

```
In [1]: import pandas as pd

In [2]: from hgc.constants import constants

In [3]: {a: constants.units(a) for a in constants.atoms}
Out[3]:
{'H': 'mg/L',
 'He': 'mg/L',
 'Li': 'µg/L',
 'Be': 'µg/L',
 'B': 'µg/L',
 'C': 'mg/L',
 'N': 'mg/L',
 'O': 'mg/L',
 'F': 'mg/L',
 'Ne': 'mg/L',
 'Na': 'mg/L',
 'Mg': 'mg/L',
 'Al': 'µg/L',
 'Si': 'mg/L',
 'P': 'mg/L',
 'S': 'mg/L',
 'Cl': 'mg/L',
 'Ar': 'mg/L',
 'K': 'mg/L',
 'Ca': 'mg/L',
 'Sc': 'µg/L',
 'Ti': 'µg/L',
```

(continues on next page)

(continued from previous page)

```
'V': 'μg/L',
'Cr': 'μg/L',
'Mn': 'mg/L',
'Fe': 'mg/L',
'Co': 'μg/L',
'Ni': 'μg/L',
'Cu': 'μg/L',
'Zn': 'μg/L',
'Ga': 'μg/L',
'Ge': 'μg/L',
'As': 'μg/L',
'Se': 'μg/L',
'Br': 'μg/L',
'Kr': 'mg/L',
'Rb': 'μg/L',
'Sr': 'μg/L',
'Y': 'μg/L',
'Zr': 'μg/L',
'Nb': 'μg/L',
'Mo': 'μg/L',
'Tc': 'mg/L',
'Ru': 'μg/L',
'Rh': 'μg/L',
'Pd': 'mg/L',
'Ag': 'μg/L',
'Cd': 'μg/L',
'In': 'μg/L',
'Sn': 'μg/L',
'Sb': 'μg/L',
'Te': 'μg/L',
'I': 'μg/L',
'Xe': 'mg/L',
'Cs': 'μg/L',
'Ba': 'μg/L',
'La': 'μg/L',
'Ce': 'μg/L',
'Pr': 'μg/L',
'Nd': 'μg/L',
'Pm': 'mg/L',
'Sm': 'μg/L',
'Eu': 'μg/L',
'Gd': 'μg/L',
'Tb': 'μg/L',
'Dy': 'μg/L',
'Ho': 'μg/L',
'Er': 'μg/L',
'Tm': 'μg/L',
'Yb': 'μg/L',
'Lu': 'μg/L',
'Hf': 'μg/L',
'Ta': 'μg/L',
'W': 'μg/L',
'Re': 'μg/L',
'Os': 'μg/L',
'Ir': 'μg/L',
'Pt': 'μg/L',
'Au': 'μg/L',
```

(continues on next page)

(continued from previous page)

```
 'Hg': 'μg/L',
 'Tl': 'μg/L',
 'Pb': 'μg/L',
 'Bi': 'μg/L',
 'Po': 'mg/L',
 'At': 'mg/L',
 'Rn': 'mg/L',
 'Fr': 'mg/L',
 'Ra': 'mg/L',
 'Ac': 'mg/L',
 'Th': 'μg/L',
 'Pa': 'mg/L',
 'U': 'μg/L',
 'Np': 'mg/L',
 'Pu': 'mg/L',
 'Am': 'mg/L',
 'Cm': 'mg/L',
 'Bk': 'mg/L',
 'Cf': 'mg/L',
 'Es': 'mg/L',
 'Fm': 'mg/L',
 'Md': 'mg/L',
 'No': 'mg/L',
 'Lr': 'mg/L',
 'Rf': 'mg/L',
 'Db': 'mg/L',
 'Sg': 'mg/L',
 'Bh': 'mg/L',
 'Hs': 'mg/L',
 'Mt': 'mg/L',
 'Ds': 'mg/L',
 'Rg': 'mg/L',
 'Cn': 'mg/L',
 'Nh': 'mg/L',
 'Fl': 'mg/L',
 'Mc': 'mg/L',
 'Lv': 'mg/L',
 'Ts': 'mg/L',
 'Og': 'mg/L'}
```

5.2 Ions

```
In [4]: import pandas as pd

In [5]: from hgc.constants import constants

In [6]: {a: constants.units(a) for a in constants.ions}
Out[6]:
{'CH4': 'mg/L as CH4',
 'H2S': 'mg/L as S',
 'S': 'mg/L',
 'CO2': 'mg/L as CO2',
 'alkalinity': 'mg/L as HCO3',
 'O2_field': 'mg/L',
```

(continues on next page)

(continued from previous page)

```
 'O2_lab': 'mg/L',
 'O2': 'mg/L',
 'KMnO4': ' mg/L  as KMnO4',
 'NH4': 'mg/L',
 'NO2': 'mg/L as NO2',
 'NO3': 'mg/L as NO3',
 'N_kj': 'mg/L',
 'N': 'mg/L',
 'PO4': 'mg/L as PO4',
 'PO4_ortho': 'mg/L',
 'P': 'mg/L',
 'SiO2': 'mg/L as SiO2',
 'SO4_ic': 'mg/L as SO4',
 'SO4': 'mg/L as SO4',
 'doc': 'mg/L',
 'toc': 'mg/L',
 'cod': 'mg/L'}
```

5.3 Other properties

```
In [7]: import pandas as pd

In [8]: from hgc.constants import constants

In [9]: {a: constants.units(a) for a in constants.properties}
Out[9]:
{'ec_field': 'µS/cm',
 'ec_lab': 'µS/cm',
 'ec': 'µS/cm',
 'ph_field': '-',
 'ph_lab': '-',
 'ph': '-',
 'temp_field': '°C',
 'temp_lab': '°C',
 'temp': '°C',
 'eh_field': 'mV',
 'turb': 'FTU',
 'uva254': 'E/m'}
```

CHAPTER 6

hgc.samples_frame module

The *SamplesFrame* class is an extended Pandas DataFrame, offering additional methods for validation of hydrochemical data, calculation of relevant ratios and classifications.

class hgc.samples_frame.**SamplesFrame**(*pandas_obj*)

Bases: object

DataFrame with additional hydrochemistry-specific methods. All HGC methods and attributes defined in this class are available in the namespace 'hgc' of the Dataframe.

Examples

To use HGC methods, we always start from a Pandas DataFrame:

```
import pandas as pd
import hgc

# We start off with an ordinary DataFrame
df = pd.DataFrame({'Cl': [1,2,3], 'Mg': [11,12,13]})

# Since we imported hgc, the HGC-methods become available
# on the DataFrame. This allows for instance to use HGC's
# validation function
df.hgc.is_valid
False
df.hgc.make_valid()
```

allowed_hgc_columns

Returns allowed columns of the hgc *SamplesFrame*

consolidate(*use_ph='field'*, *use_ec='lab'*, *use_so4='ic'*, *use_o2='field'*, *use_temp='field'*, *use_alkalinity='alkalinity'*, *merge_on_na=False*, *inplace=True*)

Consolidate parameters measured with different methods to one single parameter.

Parameters such as EC and pH are frequently measured both in the lab and field, and SO4 and PO4 are frequently measured both by IC and ICP-OES. Normally we prefer the field data for EC and pH, but

ill calibrated sensors or tough field circumstances may prevent these readings to be superior to the lab measurement. This method allows for quick selection of the preferred measurement method for each parameter and select that for further analysis.

For each consolidated parameter HGC adds a new column that is either filled with the lab measurements or the field measurements. It is also possible to fill it with the preferred method, and fill remaining NaN's with measurements gathered with the other possible method.

Parameters

- **use_ph** (*{'lab', 'field', None}, default 'field'*) – Which pH to use? Ignored if None.
- **use_ec** (*{'lab', 'field', None}, default 'lab'*) – Which EC to use?
- **use_so4** (*{'ic', 'field', None}, default 'ic'*) – Which SO4 to use?
- **use_o2** (*{'lab', 'field', None}, default 'field'*) – Which O2 to use?
- **use_alkalinity** (*str, default 'alkalinity'*) – name of the column to use for alkalinity
- **merge_on_na** (*bool, default False*) – Fill NaN's from one measurement method with measurements from other method.
- **inplace** (*bool, default True*) – Modify *SamplesFrame* in place. inplace=False is not implemented (yet)

Raises *ValueError: if one of the 'use_' parameters is set to a column that is not in the dataframe – or* if one of the default parameters is not in the dataframe while it is not set to None.

fillna_concentrations(*how='phreeqc'*)

Calculate missing concentrations based on the charge balance.

Parameters **how** (*{'phreeqc', 'analytic'}, default 'phreeqc'*) – Method to compute missing concentrations.

fillna_ec(*use_phreeqc=True*)

Calculate missing Electrical Conductivity measurements using known anions and cations.

get_bex(*watertype='G'*, *inplace=True*)

Get Base Exchange Index (meq/L). By default this is the BEX without dolomite.

Parameters

- **watertype** (*{'G', 'P'}, default 'G'*) – Watertype (Groundwater or Precipitation)
- **inplace** (*bool, optional, default True*) – whether the saturation index should be added to the *pd.DataFrame* (inplace=True) as column *si_<mineral_name>* or returned as a *pd.Series* (inplace=False).

Returns Returns None if *inplace=True* or *pd.Series* with base exchange index for each row in SamplesFrame if *inplace=False*.

Return type pandas.Series or None

get_dominant_anions(*inplace=True*)

calculates the dominant anion of each row in the SamplesFrame as used by the Stuyfzand water type classification (See: http://www.hydrology-amsterdam.nl/valorisation/HGCmanual_v2_1.pdf chapter 5 for the definitions.)

Parameters **inplace** (*bool, optional, default True*) – whether the dominant anion should be added to the *pd.DataFrame* as column *dominant_anion* (inplace=True) or returned as a *pd.Series* (inplace=False).

Returns Returns None if *inplace=True* or *pd.Series* with dominant anion for each row in *SamplesFrame* if *inplace=False*.

Return type pandas.Series or None

get_dominant_cations(**args*, ***kwargs*)

get_ion_balance(*inplace=True*)

Calculate the balance between anion and cations and add it as a percentage [%] to the column 'ion_balance' to the SamplesFrame

Parameters **inplace** (*bool, optional, default True*) – whether the ion balance should be added to the *SamplesFrame* (inplace=True) as column *ion_balance* or returned as a *pd.Series* (inplace=False).

Returns Returns None if *inplace=True* or *pd.Series* with ion balance for each row in *SamplesFrame* if *inplace=False*.

Return type pandas.Series or None

get_partial_pressure(*gas*, *use_phreeqc=True*, *inplace=True*, ***kwargs*)

adds or returns the partial pressure of a gas using phreeqc. It is an alias for *get_saturation_index* so look at that method for details. gas column is pp_<gas_name>

get_phreeqpython_solutions(*equilibrate_with='none'*, *inplace=True*)

Return a series of phreeqpython solutions derived from the (row)data in the *SamplesFrame*.

Parameters

- **equilibrate_with** (*str, default 'none'*) – Ion to add for achieving charge equilibrium in the solutions.
- **inplace** (*bool, default True*) – Whether the result is returned as a *pd.Series* or is added to the *pd.DataFrame* as column *pp_solutions*.

Returns Returns None if *inplace=True* and *pd.Series* with *PhreeqPython.Solution* instances for every row in *SamplesFrame* if *inplace=False*.

Return type pandas.Series or None

get_ratios(**args*, ***kwargs*)

get_saturation_index(*mineral_or_gas*, *use_phreeqc=True*, *inplace=True*, ***kwargs*)

adds or returns the saturation index (SI) of a mineral or the partial pressure of a gas using phreeqc. The column name of the result is si_<mineral_name> in lower case (if inplace=True).

Parameters

- **mineral_or_gas** (*str*) – the name of the mineral of which the SI needs to be calculated
- **use_phreeqc** (*bool*) –

  whether to return use phreeqc as backend or fall back on internal hgc-routines to calculate SI or partial pressure

  inplace: bool, optional, default=True whether the saturation index should be added to the *pd.DataFrame* (inplace=True) as column *si_<mineral_name>* or returned as a *pd.Series* (inplace=False).

returns Returns None if *inplace=True* and *pd.Series* with the saturation index of the mineral for each row in *SamplesFrame* if *inplace=False*.

rtype pandas.Series or None

get_specific_conductance(*use_phreeqc=True*, *inplace=True*, ***kwargs*)
returns the specific conductance (sc) of a water sample using phreeqc. sc is also known as electric conductivity (ec) or egv measurements.

Parameters

- **use_phreeqc** (*bool, optional*) – whether to return use phreeqc as backend or fall back on internal hgc-routines to calculate SI or partial pressure
- **inplace** (*bool, optional, default=True*) – whether the specific conductance should be added to the *pd.DataFrame* (inplace=True) as column *sc* or returned as a *pd.Series* (inplace=False).
- ****kwargs** – are passed to the method *get_phreeqpython_solutions*

Returns Returns None if *inplace=True* and *pd.Series* with specific conductance for each row in *SamplesFrame* if *inplace=False*.

Return type pandas.Series or None

get_stuyfzand_water_type(**args*, ***kwargs*)

get_sum_anions(**args*, ***kwargs*)

get_sum_cations(**args*, ***kwargs*)

hgc_cols
Return the columns that are used by hgc

is_valid
returns a boolean indicating that the columns used by hgc have valid values

make_valid()
Try to convert the DataFrame into a valid HGC-*SamplesFrame*.

select_phreeq_columns(**args*, ***kwargs*)

hgc.samples_frame.**requires_ph**(*func*)
Decorator function for methods in the SamplesFrame class that require a column *ph* with valid values (non-zero and non-NaN).

CHAPTER 7

About

HGC is developed by KWR Water Research Institute. The code is based on the original Excel-based HGC program by Pieter Stuyfzand.

7.1 License

This package is MIT licensed. See License File .

Python Module Index

h

Index