PhreeqPy Documentation

Release 0.5.1

Mike Müller

http://www.hydrocomputing.com

Apr 18, 2022

CONTENTS

CHAPTER

ONE

PHREEQPY - PYTHON TOOLS FOR PHREEQC

1.1 Introduction

PhreeqPy is Open Source software and provides Python tools to work with PHREEQC.

Currently it provides access to the new IPhreeqc interface without the need to run a COM server and therefore also works on non-Windows systems. IPhreeqc is described in more detail in this publication.

Please let us know what you do with PhreeqPy or if things do not work as expected. There is a mailing list for PhreeqPy. Please subscribe to get your questions about PhreeqPy answered.

Download latest version of this documentation as PDF

License: BSD

1.2 Installation

Pythons (tested): Python 2.6, 2.7, 3.3, 3.4, 3.5, 3.6, 3.8, 3.7, 3.9, 3.10 PyPy

Pythons (untested but should work since using ctypes): IronPython, Jython 2.7

Platforms: Unix/Posix and Windows

PyPI package name: phreeqpy

Installation with *pip*:

```
pip install -U phreeqpy
```

You need an IPhreeqc shared libray for your operating system. PhreeqPy comes with shared libraries for 32-bit Windows, 32-bit Linux and 64-bit Mac OS X. To use the COMM server, please install the Windows COM 64-bit module of IPhreeqc and use `phreeqpy.iphreeqc.phreeqc_com as phreeqc_mod` instead of `import phreeqpy.iphreeqc.phreeqc_dll as phreeqc_mod` as shown in the example. If you use Anaconda or Miniconda make sure to:

conda install pywin32

before you install PhreeqPy.

You may download an appropriate library from here: https://www.usgs.gov/software/phreeqc-version-3

For example for Linux:: wget https://water.usgs.gov/water-resources/software/PHREEQC/iphreeqc-3.7.3-15968.tar.gz tar -xzvf iphreeqc-3.7.3-15968.tar.gz cd iphreeqc-3.7.3-15968 ./configure make make check sudo make install

Then either use the full path to the shared libray when making an instance of `phreeqc_dll.IPhreeqc`

```
phreeqc = phreeqpy.iphreeqc.phreeqc_dll.IPhreeqc('/full/path/to/libiphreeqc.so')
```

or copy the shared object into `phreeqpy/iphreeqc` replacing the existing one. For example:

```
sudo cp /usr/local/lib/libiphreeqc.so  /path/to/site-packages/phreeqpy/iphreeqc/
↪libiphreeqc.so.0.0.0
```

1.3 Benchmark Test Comparing PhreeqPy to External Processes, COM and C++

This publication demonstrates how PhreeqPy can be used for reactive transport modeling.

Müller M., Parkhurst D.L., Charlton S.R. (2011) Programming PHREEQC Calculations with C++ and Python - A Comparative Study , In: Maxwell R., Poeter E., Hill M., Zheng C. (2011) MODFLOW and More 2011 - Integrated Hydrological Modeling, Proceedings, pp. 632 - 636.

CHAPTER

TWO

CLASS IPHREEQC

This is the main class to work with the IPhreeqc interface.

You can *add your own shared library for IPhreeqc*.

Access PHREEQC-DLL via ctypes.

This is exchangeable with the COM interface.

`class phreeqpy.iphreeqc.phreeqc_dll.`**`IPhreeqc`**`(`*dll_path=None*`)`

Wrapper for the IPhreeqc DLL.

Connect to DLL and create IPhreeqc.

The optional *dll_path* takes a path to the IPhreeqc shared library. If not provided it tries to select an appropriate library. Make sure you have the right library for your operating system. You may download one from here: ftp://brrftp.cr.usgs.gov/pub/charlton/iphreeqc/

See the PhreeqPy documentation for help on compiling a IPhreeqc shared library.

`accumulate_line``(`*line*`)`

Put line in input buffer.

`add_error``(`*phc_error_msg*`)`

Add an error message to Phreeqc.

`add_warning``(`*phc_warn_msg*`)`

Add an warning message to Phreeqc.

`clear_accumlated_lines``()`

Clear the input buffer.

`property column_count`

Get number of columns in selected output.

`property component_count`

Return the number of components.

`create_iphreeqc``()`

Create a IPhreeqc object.

`destroy_iphreeqc``()`

Delete the current instance of IPhreeqc.

`get_component``(`*index*`)`

Get one component.

`get_component_list()`
: Return all component names.

`get_error_string()`
: Retrieves the error messages.

`get_selected_output_array()`
: Get all values from selected output.

`get_selected_output_column(`*col*`)`
: Get all values for one column from selected output.

`get_selected_output_row(`*row*`)`
: Get all values for one from selected output.

`get_selected_output_value(`*row*, *col*`)`
: Get one value from selected output at given row and column.

`load_database(`*database_name*`)`
: Load a database with given file_name.

`load_database_string(`*input_string*`)`
: Load a datbase from a string.

`static raise_ipq_error(`*error_code*`)`
: There was an error, raise an exception.

`raise_string_error(`*errors*`)`
: Raise an exception with message from IPhreeqc error.

`property row_count`
: Get number of rows in selected output.

`run_string(`*cmd_string*`)`
: Run PHREEQC input from string.

`set_output_file_off()`
: Turn on writing to selected output file.

`set_output_file_on()`
: Turn on writing to selected output file.

`set_selected_output_file_off()`
: Turn on writing to selected output file.

`set_selected_output_file_on()`
: Turn on writing to selected output file.

`exception phreeqpy.iphreeqc.phreeqc_dll.PhreeqcException`
: Error in Phreeqc call.

`class phreeqpy.iphreeqc.phreeqc_dll.VAR`
: Struct with data type and data values.

  See Var.h in PHREEQC source.

`class phreeqpy.iphreeqc.phreeqc_dll.VARUNION`
: Union with types.

  See Var.h in PHREEQC source.

CHAPTER THREE

EXAMPLES FOR PHREEQPY

3.1 Simple example

This is an example for the use PhreeqPy for one-dimensional advection. It is the example 11 from the PHREEQC users manual.

```python
"""Advection with DLL or COM server.

Using MODFIY we update the concentration on every
time step. We shift by one cell per step.
"""

from __future__ import print_function

import sys

# Simple Python 3 compatibility adjustment.
if sys.version_info[0] == 2:
    range = xrange

import os
import timeit

MODE = 'dll'  # 'dll' or 'com'

if MODE == 'com':
    import phreeqpy.iphreeqc.phreeqc_com as phreeqc_mod
elif MODE == 'dll':
    import phreeqpy.iphreeqc.phreeqc_dll as phreeqc_mod
else:
    raise Exception('Mode "%s" is not defined use "com" or "dll".' % MODE)


def make_initial_conditions():
    """
    Specify initial conditions data blocks.

    Uniform initial conditions are assumed.
    """
    initial_conditions = """
```

(continues on next page)

(continued from previous page)

```
    TITLE Example 11.--Transport and ion exchange.
    SOLUTION 0  CaCl2
        units            mmol/kgw
        temp             25.0
        pH               7.0     charge
        pe               12.5    O2(g)   -0.68
        Ca               0.6
        Cl               1.2
    SOLUTION 1  Initial solution for column
        units            mmol/kgw
        temp             25.0
        pH               7.0     charge
        pe               12.5    O2(g)   -0.68
        Na               1.0
        K                0.2
        N(5)             1.2
        END
    EXCHANGE 1
        equilibrate 1
        X                0.0011
    END
        """
    return initial_conditions


def make_selected_output(components):
    """
    Build SELECTED_OUTPUT data block
    """
    headings = "-headings    cb    H    O    "
    for i in range(len(components)):
        headings += components[i] + "\t"
    selected_output = """
    SELECTED_OUTPUT
        -reset false
    USER_PUNCH
    """
    selected_output += headings + "\n"
    #
    # charge balance, H, and O
    #
    code = '10 w = TOT("water")\n'
    code += '20 PUNCH CHARGE_BALANCE, TOTMOLE("H"), TOTMOLE("O")\n'
    #
    # All other elements
    #
    lino = 30
    for component in components:
        code += '%d PUNCH w*TOT(\"%s\")\n' % (lino, component)
        lino += 10
    selected_output += code
    return selected_output
```

(continues on next page)

(continued from previous page)

```python
def initialize(cells, first=False):
    """
    Initialize IPhreeqc module
    """
    phreeqc = phreeqc_mod.IPhreeqc()
    phreeqc.load_database(r"phreeqc.dat")
    initial_conditions = make_initial_conditions()
    phreeqc.run_string(initial_conditions)
    components = phreeqc.get_component_list()
    selected_output = make_selected_output(components)
    phreeqc.run_string(selected_output)
    phc_string = "RUN_CELLS; -cells 0-1\n"
    phreeqc.run_string(phc_string)
    conc = get_selected_output(phreeqc)
    inflow = {}
    initial = {}
    for name in conc:
        if first:
            inflow[name] = conc[name][0]
        else:
            inflow[name] = conc[name][1]
        initial[name] = conc[name][1]
    task = initial_conditions + "\n"
    task += "COPY solution 1 %d-%d\n" % (cells[0], cells[1])
    task += "COPY exchange 1 %d-%d\n" % (cells[0], cells[1])
    task += "END\n"
    task += "RUN_CELLS; -cells %d-%d\n" % (cells[0], cells[1])
    task += selected_output
    phreeqc.run_string(task)
    conc = get_selected_output(phreeqc)
    for name in conc:
        value = [initial[name]] * len(conc[name])
        conc[name] = value
    return phreeqc, inflow, conc


def advect_step(phreeqc, inflow, conc, cells):
    """Advect by shifting concentrations from previous time step.
    """
    all_names = conc.keys()
    names = [name for name in all_names if name not in ('cb', 'H', 'O')]
    for name in conc:
        # shift one cell
        conc[name][1:] = conc[name][:-1]
        conc[name][0] = inflow[name]
    modify = []
    for index, cell in enumerate(range(cells[0], cells[1] + 1)):
        modify.append("SOLUTION_MODIFY %d" % cell)
        modify.append("\t-cb      %e" % conc['cb'][index])
        modify.append("\t-total_h %f" % conc['H'][index])
```

(continues on next page)

(continued from previous page)

```
        modify.append("\t-total_o %f" % conc['O'][index])
        modify.append("\t-totals")
        for name in names:
            modify.append("\t\t%s\t%f" % (name, conc[name][index]))
    modify.append("RUN_CELLS; -cells %d-%d\n" % (cells[0], cells[1]))
    cmd = '\n'.join(modify)
    phreeqc.run_string(cmd)
    conc = get_selected_output(phreeqc)
    return conc


def get_selected_output(phreeqc):
    """Return calculation result as dict.

    Header entries are the keys and the columns
    are the values as lists of numbers.
    """
    output = phreeqc.get_selected_output_array()
    header = output[0]
    conc = {}
    for head in header:
        conc[head] = []
    for row in output[1:]:
        for col, head in enumerate(header):
            conc[head].append(row[col])
    return conc


def run(ncells, shifts, specie_names):
    """Do one run in one process.
    """
    cells = (1, ncells)
    phreeqc, inflow, conc = initialize(cells, first=True)
    outflow = {}
    for name in specie_names:
        outflow[name] = []
    for _counter in range(shifts):
        # advect
        conc = advect_step(phreeqc, inflow, conc, cells)
        for name in specie_names:
            outflow[name].append(conc[name][-1])
    return outflow


def write_outflow(file_name, outflow):
    """Write the outflow values to a file.
    """
    fobj = open(file_name, 'w')
    header = outflow.keys()
    for head in header:
        fobj.write('%20s' % head)
    fobj.write('\n')
```

(continues on next page)

(continued from previous page)

```
    for lineno in range(len(outflow[head])):
        for head in header:
            fobj.write('%20.17f' % outflow[head][lineno])
        fobj.write('\n')


def main(ncells, shifts):
    """Run different versions with and without multiprocessing
    """

    def measure_time(func, *args, **kwargs):
        """Convinience function to measure run times.
        """
        start = timeit.default_timer()
        result = func(*args, **kwargs)
        return result, timeit.default_timer() - start

    print('Dimensions')
    print('==========')
    print('number of cells:   ', ncells)
    print('number of shifts   ', shifts)
    specie_names = ('Ca', 'Cl', 'K', 'N', 'Na')
    outflow, run_time = measure_time(run, ncells, shifts, specie_names)
    if not os.path.exists('data'):
        os.mkdir('data')
    write_outflow('data/out.txt', outflow)
    print('run time:', run_time)
    print("Finished simulation\n")

if __name__ == '__main__':
    main(ncells=40, shifts=120)
```

3.2 Parallel Example

This is an example for the use PhreeqPy for one-dimensional advection running in parallel, using `multiprocessing`.

```
# -*- coding: utf-8 -*-

"""A coupled advection-reaction model using PhreeqPy.

The sketch below shows the setup. The coupled model contains a advection and a
reaction model. The reaction model can have one or more Phreeqc calculators.
The calculators can work in parallel using the module `multiprocessing`.

+------------------------------------------------------------------------------+
|                              CoupledModel                                    |
|  +-----------------------------------------------------------------------+  |
|  |                           AdvectionModel                              |  |
|  |                                                                       |  |
|  +-----------------------------------------------------------------------+  |
```

(continues on next page)

(continued from previous page)

```
|   |                         ReactionModel                              |   |
|   |  +--------------------+--------------------+--------------------+  |   |
|   |  | PhreeqcCalculator1 | PhreeqcCalculator2 | PhreeqcCalculator3 |  |   |
+---------------------------------------------------------------------------+

Author: Mike Müller, mmueller@hydrocomputing.com
"""


import multiprocessing
import os
import timeit

import matplotlib.pyplot as plt

MODE = 'dll'  # 'dll' or 'com'

if MODE == 'com':
    import phreeqpy.iphreeqc.phreeqc_com as phreeqc_mod
elif MODE == 'dll':
    import phreeqpy.iphreeqc.phreeqc_dll as phreeqc_mod
else:
    raise Exception('Mode "%s" is not defined use "com" or "dll".' % MODE)


class CoupledModel(object):
    """This is a coupled advection model.

    Since it is just a simple example, we use a 1D model.
    The same approach can be applied to a 2d or 3D model
    as long as the advection model supports it.
    The PHREEQC part is the same.
    Furthermore, instead of a simple advection model,
    we can have a more sophisticated transport model.
    """

    def __init__(self, ncells, nshifts, initial_conditions, processes):
        self.nshifts = nshifts
        self.reaction_model = ReactionModel(ncells, initial_conditions,
                                            processes)
        self.reaction_model.make_initial_state()
        init_conc = dict([(name, [value] * ncells) for name, value in
                          self.reaction_model.init_conc.items()])
        self.advection_model = AdvectionModel(init_conc,
                                              self.reaction_model.inflow_conc)
        self.component_names = self.reaction_model.component_names
        self.results = {}
        for name in self.component_names:
            self.results[name] = []

    def run(self):
        """Go over all time steps (shifts).
```

(continues on next page)

(continued from previous page)

```
        """
        for _ in range(self.nshifts):
            self.advection_model.advect()
            self.advection_model.save_results(self.results)
            self.reaction_model.modify(self.advection_model.conc)
            self.advection_model.update_conc(self.reaction_model.conc)
        self.reaction_model.finish()


class AdvectionModel(object):
    """Very simple 1D advection model.

    This model can be replaced by a more sophisticated transport
    model with two or three dimensions.
    This could be an external code such as MT3D or anything
    else that is moving concentrations through the subsurface.
    """

    def __init__(self, init_conc, inflow_conc):
        """Set the initial and inflow concentrations.

        Both concentrations are dictionaries with the specie names as keys.
        Values are 1D arrays for `init_conc` and scalars for `inflow_conc`.
        """
        self.conc = init_conc
        self.inflow_conc = inflow_conc
        self.outflow = {}

    def update_conc(self, new_conc):
        """Update the concentrations after the reactions step.

        This is very simple but could be more involved
        if the transport model is more complex.
        """
        self.conc = new_conc

    def advect(self):
        """Shift one cell.
        """
        for name in self.conc:
            self.outflow[name] = self.conc[name][-1]
            self.conc[name][1:] = self.conc[name][:-1]
            self.conc[name][0] = self.inflow_conc[name]

    def save_results(self, results):
        """Save the calculation results.

        Typically, we would write our results into a file.
        For simplicity we just add the current outflow that
        we stored in `self.outflow` and add it to `results`,
        which is a dictionary with all specie names as keys
        and lists as values.
```

(continues on next page)

(continued from previous page)

```
        """
        for name in self.conc:
            results[name].append(self.outflow[name])


class ReactionModel(object):
    """Calculate reactions using PHREEQC as computational engine.

    We have no direct contact with IPhreeqc here.
    We make one or more instances of `PhreeqcCalculator`
    that are actually using IPhreeqc.
    We can use more than one processor with `multiprocessing`.
    """

    def __init__(self, ncells, initial_conditions, processes):
        if processes > ncells:
            raise ValueError('Number of processes needs to be less or equal '
                             'than number of cells. %d processes %d cells.'
                             % (processes, ncells))
        if processes < 1:
            raise ValueError('Need at least one process got %d' % processes)
        self.parallel = False
        if processes > 1:
            self.parallel = True
        self.ncells = ncells
        self.initial_conditions = initial_conditions
        self.processes = processes
        self.inflow_conc = {}
        self.init_conc = {}
        self.conc = {}
        self.component_names = []
        self.calculators = []
        self.cell_ranges = []
        self._init_calculators()
        self.make_initial_state()

    def _init_calculators(self):
        """If we are going parallel we need several calculators.
        """
        if self.parallel:
            # Domain decomposition.
            slave_ncells, reminder = divmod(self.ncells, self.processes)
            root_ncells = slave_ncells + reminder
            current_cell = root_ncells
            root_calculator = PhreeqcCalculator(root_ncells,
                                                self.initial_conditions)
            self.calculators = [root_calculator]
            self.cell_ranges = [(0, root_ncells)]
            for _ in range(self.processes - 1):
                self.calculators.append(PhreeqcCalculatorProxy(slave_ncells,
                                                    self.initial_conditions))
                self.cell_ranges.append((current_cell,
```

(continues on next page)

(continued from previous page)

```
                                          current_cell + slave_ncells))
                current_cell += slave_ncells
            assert current_cell == self.ncells
            self.calculators.reverse()
            self.cell_ranges.reverse()
        else:
            root_calculator = PhreeqcCalculator(self.ncells,
                                                self.initial_conditions)
            # Just one calculator and the entire range but still use a list
            # to provide the same interface as the parallel case.
            self.calculators = [root_calculator]
            self.cell_ranges = [(0, self.ncells)]

    def make_initial_state(self):
        """Get the initial values from the calculator(s).
        """
        self.inflow_conc = self.calculators[0].inflow_conc
        self.init_conc = self.calculators[0].init_conc
        self.component_names = self.calculators[0].component_names
        if self.parallel:
            # Make sure all calculators are initialized the same.
            for calculator in self.calculators[1:]:
                assert self.inflow_conc == calculator.inflow_conc
                assert self.init_conc == calculator.init_conc
                assert self.component_names == calculator.component_names

    def modify(self, new_conc):
        """Pass new conc after advection to the calculator.
        """
        self.conc = {}
        for name in self.component_names:
            self.conc[name] = []
        for cell_range, calculator in zip(self.cell_ranges, self.calculators):
            current_conc = dict([(name, value[cell_range[0]:cell_range[1]]) for
                                  name, value in new_conc.items()])
            calculator.modify(current_conc)
        for calculator in self.calculators:
            conc = calculator.get_modified()
            for name in self.component_names:
                self.conc[name].extend(conc[name])

    def finish(self):
        """This is necessary for multiprocessing.

        Multiprocessing uses external processes. These need to be
        explicitly closed to avoid hanging of the program at
        the end.
        """
        for calculator in self.calculators:
            calculator.finish()
```

(continues on next page)

(continued from previous page)

```
class PhreeqcCalculator(object):
    """All PHREEQC calculations happen here.

    This is the only place where we interact wit IPhreeqc.
    Each instance of this class might run in a different
    process using `multiprocessing`.
    """

    def __init__(self, ncells, initial_conditions):
        """
        ncells - number of cells
        initial_conditions - string containing PHREEQC input for
                             solution and exchange, see example below
        """
        self.ncells = ncells
        self.initial_conditions = initial_conditions
        self.inflow_conc = {}
        self.init_conc = {}
        self.conc = {}
        self.phreeqc = phreeqc_mod.IPhreeqc()
        self.phreeqc.load_database(r"phreeqc.dat")
        self.components = []
        self.component_names = []
        self._make_initial_state()

    def _make_initial_state(self):
        """Copy solution to all cells and calculate initial conditions.
        """
        self.phreeqc.run_string(self.initial_conditions)
        self.components = self.phreeqc.get_component_list()
        start = 1
        end = self.ncells
        code = ''
        code += "COPY solution 1 %d-%d\n" % (start, end)
        code += "COPY exchange 1 %d-%d\n" % (start, end)
        code += "END\n"
        code += "RUN_CELLS; -cells %d-%d\n" % (start, end)
        code += self.make_selected_output(self.components)
        self.phreeqc.run_string(code)
        self.conc = self.get_selected_output()
        all_names = self.conc.keys()
        self.component_names = [name for name in all_names if name not in
                                ('cb', 'H', 'O')]
        code = ''
        code += self.make_selected_output(self.components)
        code += "RUN_CELLS; -cells 0-1\n"
        self.phreeqc.run_string(code)
        start_conc = self.get_selected_output()
        for name in self.component_names:
            self.inflow_conc[name] = start_conc[name][0]
            self.init_conc[name] = start_conc[name][1]
```

(continues on next page)

(continued from previous page)

```
    def modify(self, new_conc):
        """Set new concentration after advection and re-calculate.
        """
        conc = self.conc
        end = self.ncells + 1
        conc.update(new_conc)
        modify = []
        for index, cell in enumerate(range(1, end)):
            modify.append("SOLUTION_MODIFY %d" % cell)
            modify.append("\t-cb      %e" % conc['cb'][index])
            modify.append("\t-total_h %s" % conc['H'][index])
            modify.append("\t-total_o %s" % conc['O'][index])
            modify.append("\t-totals")
            for name in self.component_names:
                modify.append("\t\t%s\t%s" % (name, conc[name][index]))
        modify.append("RUN_CELLS; -cells %d-%d\n" % (1, self.ncells))
        code = '\n'.join(modify)
        self.phreeqc.run_string(code)
        self.conc = self.get_selected_output()

    def get_modified(self):
        """Return calculated conc.
        """
        return self.conc

    @ staticmethod # this is just a function but belongs here
    def make_selected_output(components):
        """
        Build SELECTED_OUTPUT data block.
        """
        headings = "-headings    cb    H    O    "
        headings += '\t'.join(components)
        selected_output = """
        SELECTED_OUTPUT
            -reset false
        USER_PUNCH
        """
        selected_output += headings + "\n"
        # charge balance, H, and O
        code = '10 w = TOT("water")\n'
        code += '20 PUNCH CHARGE_BALANCE, TOTMOLE("H"), TOTMOLE("O")\n'
        # All other elements
        lino = 30
        for component in components:
            code += '%d PUNCH w*TOT(\"%s\")\n' % (lino, component)
            lino += 10
        selected_output += code
        return selected_output

    def get_selected_output(self):
        """Return calculation result as dict.
```

(continues on next page)

(continued from previous page)

```
        Header entries are the keys and the columns
        are the values as lists of numbers.
        """
        output = self.phreeqc.get_selected_output_array()
        header = output[0]
        conc = {}
        for head in header:
            conc[head] = []
        for row in output[1:]:
            for col, head in enumerate(header):
                conc[head].append(row[col])
        return conc

    def finish(self):
        """Placeholder to give same interface as the multiprocessing version.
        """
        pass


class PhreeqcCalculatorProxy(object):
    """Proxy that communicates with other processes.

    We uses this proxy for parallel computations.
    All code that is specific for parallel computing is located
    in here.
    """

    def __init__(self, ncells, initial_conditions):
        """Go parallel.
        """
        self.in_queue = multiprocessing.JoinableQueue()
        self.out_queue = multiprocessing.JoinableQueue()
        self.process = multiprocessing.Process(
            target=process_worker,
            args=(ncells, initial_conditions, self.in_queue, self.out_queue))
        self.process.start()
        (self.inflow_conc,
         self.init_conc,
         self.component_names) = self.out_queue.get()

    def modify(self, new_conc):
        """Run PHREEQC in another process.
        """
        self.in_queue.put(new_conc)

    def get_modified(self):
        """Return calculated conc.
        """
        return self.out_queue.get()

    def finish(self):
        """Terminate the process.
```

(continues on next page)

(continued from previous page)

```
        """
        self.in_queue.put(None)
        self.process.join()


def process_worker(ncells, initial_conditions, in_queue, out_queue):
    """This runs in another process.
    """
    print('Started process with ID', os.getpid())
    calculator = PhreeqcCalculator(ncells, initial_conditions)
    out_queue.put((calculator.inflow_conc, calculator.init_conc,
                   calculator.component_names))
    while True:
        new_conc = in_queue.get()
        # None is the sentinel. We are done
        if new_conc is None:
            break
        calculator.modify(new_conc)
        out_queue.put(calculator.conc)


def plot(ncells, outflow, specie_names):
    """Plot the results.
    """
    colors = {'Ca': 'r', 'Cl': 'b', 'K': 'g', 'N': 'y', 'Na': 'm'}
    x = [i / float(ncells) for i in
         range(1, len(outflow[specie_names[0]]) + 1)]
    args = []
    for name in specie_names:
        args.extend([x, outflow[name], colors[name]])
    plt.plot(*args)
    plt.legend(specie_names, loc=(0.8, 0.5))
    plt.ylabel('MILLIMOLES PER KILOGRAM WATER')
    plt.xlabel('PORE VOLUME')
    plt.savefig("ex11.png")
    plt.show()


def measure_time(func, *args, **kwargs):
    """Convenience function to measure run times.
    """
    start = timeit.default_timer()
    result = func(*args, **kwargs)
    return result, timeit.default_timer() - start


def main(ncells, nshifts, processes=2, show_plot=False):
    """
    Specify initial conditions data blocks.

    Uniform initial conditions are assumed.
    """
```

(continues on next page)

(continued from previous page)

```
    initial_conditions = """
    TITLE Example 11.--Transport and ion exchange.
    SOLUTION 0  CaCl2
        units            mmol/kgw
        temp             25.0
        pH               7.0     charge
        pe               12.5    O2(g)   -0.68
        Ca               0.6
        Cl               1.2
    SOLUTION 1  Initial solution for column
        units            mmol/kgw
        temp             25.0
        pH               7.0     charge
        pe               12.5    O2(g)   -0.68
        Na               1.0
        K                0.2
        N(5)             1.2
        END
    EXCHANGE 1
        equilibrate 1
        X                0.0011
    END
        """

    def run():
        """Do the work.
        """
        model = CoupledModel(ncells, nshifts, initial_conditions,
                             processes)
        model.run()
        return model, model.results

    (model, outflow), run_time = measure_time(run)
    print('Statistics')
    print('==========')
    print('number of cells:    ', ncells)
    print('number of shifts:   ', nshifts)
    print('number of processes:', processes)
    print('run_time:           ', run_time)
    print()
    if show_plot:
        plot(ncells, outflow, model.component_names)


def benchmark(ncells=400, nshifts=1200, process_range=range(1,9)):
    """Benchmark for diffrent numbers of processes"""
    for processes in process_range:
        main(ncells=ncells, nshifts=nshifts, processes=processes)


def plot_result(ncells=400, nshifts=1200, processes=4):
    """Also show the plot"""
```

(continues on next page)

(continued from previous page)

```
    main(ncells=ncells, nshifts=nshifts, processes=processes, show_plot=True)


if __name__ == '__main__':
    benchmark()
    plot_result()
```

CHAPTER

FOUR

CHANGELOG HISTORY

4.1 Changes in version 0.5

- Updated documentation to PHREEQC3
- Added IPhreeqc 3.7.3 and mac arm64 version
- Removed download, use *pip* to install
- Added makefile to automate build steps
- Added parallel example

4.2 Changes in version 0.4

- Add *pywin32* dependency to *pip* install
- Removed setting of `doc`attribute on PhreeqPy objects because it is not supported by new IPhreeqcCOM versions.
- Hint that there is no 64-bit DLL for Windows. Instead the COM-Server version should be used.

4.3 Changes in version 0.3

- Added *set_output_file_off()`and `set_output_file_on()* that call IPhreeqc's *SetOutputFileOn*

4.4 Changes in version 0.2

- Added more IPhreeqc functions.
- Added support for Mac OS X.
- Added error handling turning IPhreeqc errors into Python exceptions.
- Added Python 3 compatibility. Tested with Python 3.3.
- Added documentation in addition to example on website.

CHAPTER

FIVE

CONTACT

hydrocomputing GmbH & Co. KG
Zur Schule 20
04158 Leipzig
Germany

Tel: +49 341 525 599 54
Fax: +49 341 520 4495
mail: info [at] hydrocomputing [dot] com
www: http://www.hydrocomputing.com

CHAPTER

SIX

DATENSCHUTZERKLÄRUNG

Diese Seite beschreibt die Erhebung, Verarbeitung und Nutzung Ihrer personenbezogenen Daten beim Besuch dieser Website. Ihre Daten werden im Rahmen der gesetzlichen Vorschriften geschützt. Sie erfahren hier, welche Daten während Ihres Besuchs auf der Website erfasst und wie diese genutzt werden.

Über diese Website erheben wir keine personenbezogenen Daten.

6.1 1. Datenerhebung und -protokollierung

Wir speichern von Ihrem Besuch Log-Dateien mit diesem Inhalt:

- pseudonymisierte IP-Adresse (ohne Personenbezug, siehe Erklärung unten)
- Datum und Uhrzeit des Zugriffs
- Art des Zugriffes (z.B. GET oder POST)
- besuchten Seiten innerhalb unseres Angebots
- Protokoll (z.B. HTTP 1.1)
- übertragene Datenmenge
- Meldung über erfolgreichen Abruf
- anfragende Domain (z.B. google.com)
- Webbrowser und Betriebssystem

Bei uns werden nur pseudonymisierte IP-Adressen von Besuchern der Website gespeichert. Auf Webserver-Ebene erfolgt dies dadurch, dass im Logfile standardmäßig stat der tatsächlichen IP-Adresse des Besuchers z.B. 123.123.123.123 eine IP-Adresse 123.123.123.XXX gespeichert wird, wobei XXX ein Zufallswert zwischen 1 und 254 ist. Die Herstellung eines Personenbezuges ist nicht mehr möglich.

6.2 2. Verwendung von Cookies

Wir verwenden keine Cookies.

CHAPTER

SEVEN

DATA PROTECTION STATEMENT

English Version for your convenience. The German version below is the legally binding one.

This page describes the collection, processing, and usage of your personal data while visiting this website. Your data is protected within the legal regulations. This page explains what data are collected while visiting this website and how they are used.

We do not collect personal via this website.

7.1 1. Data Collection and Data Logging

We store log files about your visit with this content:

- pseudonymised IP (no person identity can be establish, see explain below)
- date and time of access
- type of access (e.g. GET or POST)
- visited page on website
- protocol (e.g. HTTP 1.1)
- amount of transmitted data
- referring domain (e.g. google.com)
- type of webbrowser

We do not save hostname or Ip address of your request. The collected data is not personal and don't allow to find out who is visiting our website.

We only store pseudonymised IP addresses of visitors to the website. At the web server level, this happens by default by storing an IP address 123.123.123.XXX in the log file instead of the visitor's actual IP address, for example, 123.123.123.123. The "XXX" is a random value between 1 and 254, so it is no longer possible to establish the true identity of the visitor.

7.2 2. Use of Cookies

We do not use cookies.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX