ROSSpy

Release 1

Andrew Philip Freiburger

May 22, 2023

CONTENTS

1 Theory **3**

2 Installation **5**

3 Citation **7**

4 Contents **9**

4.1 Usage 9

4.2 ROSSpy API 10

4.3 ROSSpy parameter files 14

4.4 iROSSpy 17

Desalinating ocean water is crucial for meeting the 6th UN Sustainable Development Goal of univeralizing potable water. Reverse Osmosis (RO) is the leading desalination technology, although, it remains hindered by membrane scaling, which lessens its energy efficiency and economic practicality. The geochemistry of mineral scaling evades experimental methods, and existing software programs to simulate scaling geochemistry – e.g. French Creek – are insufficient to explore all relevant variables and are furthermore inaccessible to many researchers.

Reverse Osmosis Scaling Software in Python (ROSSpy) satisfies this niche in RO research by implementating a one-dimesnional RO model through PHREEQpy, which is the Python version of PHREEQC. The `examples/scaling/scaling_validation` directory of the ROSSpy GitHub details validation studies and sensitivity analyses of ROSSPy via Notebook examples. We encourage users and developers to critique and improve the open-source (MIT License) ROSSpy package by creating new GitHub issues or emailing afreiburger@uvic.ca.

CHAPTER

ONE

THEORY

The ROSSpy framework represents RO desalination as a 1D reactive transport model of the membrane-solution interface in the RO feed channel. The feed solution is represented as a single, homogeneous, solution. The inlet boundary is defined by the Dirichlet condition, where the inlet concentrations are assumed to be independent of desalination, with the feed as an infinite reservoir. The outlet boundary is defined by the Cachy condition, where the effluent concentration is assumed to be dependent upon the reactive transport processes within the RO module.

Note: This project is under active development.

CHAPTER

TWO

INSTALLATION

ROSSpy is installed in a command prompt, Powershell, Terminal, or Anaconda Command Prompt via `pip`:

```
pip install rosspy
```

The IPHREEQC module must then be installed, since this is the source of geochemical calculations and data for ROSSpy. The appropriate version of IPHREEQC can be installed from the USGS .

Installation in **Linux** distributions, Ubuntu >=20 or an equivalent, requires addition steps (Ubuntu < 20 is currently unsupported):

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

CHAPTER

THREE

CITATION

Please cite this work:

```
Freiburger, Andrew P. and Molins, Sergi and Buckley, Heather L., A One-Dimensional␣
↪Reactive Transport Model of Geochemical Scaling in Reverse Osmosis Desalination. http:/
↪/dx.doi.org/10.2139/ssrn.4124149
```

CHAPTER

FOUR

CONTENTS

4.1 Usage

A complete ROSSpy simulation can be designed and executed through the following example sequence:

```
# conduct a ROSSpy simulation
from rosspy import ROSSPkg
ross = ROSSPkg(database_selection, simulation)
ross.reactive_transport(simulation_time, simulation_perspective, final_cf)
ross.feed_geochemistry(water_selection, water_characteristics)
ross.execute()
```

4.1.1 Accessible content

A multitude of values are stored within the `ROSSpy` object, and can be subsequently used in post-processing. The complete list of these values can be reviewed through the built-in `dir()` function, which is highlighted in the following list:

- *selected_output* & *processed_data* `DataFrame`: Pandas DataFrames that possesses the raw and processed simulation data, respectively, from the PHREEQ simulation.
- *elemental_masses* `dict`: A dictionary of the mass for each ion that constitutes scale. This is only determined for `scaling` simulations.
- *databases* & *feed_sources* `list`: The available databases and feed waters in the `rosspy/databases/` and `rosspy/water_bodies/` directories, respectively.
- *elements* & *minerals* `dict`: Dictionaries of the elements and minerals that are defined by the selected database, respectively.
- *cumulative_cf* `float`: The final effluent CF after the entire simulation
- *ro_module* & *water_body* `dict`: The complete sets of parameterized values for the simulated RO module and feed water, respectively.
- *input_file* `str`: The complete PHREEQ input file that is executed for the simulation.
- *predicted_effluent* `dict`: The predicted effluent concentrations of each feed ion.
- *parameters* & *variables* `dict`: Dictionaries with the simulation parameters and calculated variables, respectively.
- *results* `dict`: A dictionary with the simulation results and each block of the input file.
- *simulation_shifts* `float`: The number of simulation shifts.
- *water_mw* & *water_gL* `float`: The molecular weight and density of water, respectively.

- *chem_mw* `ChemMW`: The `ChemMW` object from the ChemW module, which allows users to calculate the molecular weight from any chemical formula or chemical common name. The formatting specifications are detailed in the README of the ChemW module.

4.2 ROSSpy API

4.2.1 ROSSPkg()

The only class in the ROSSpy module is `ROSSPkg`. The initial parameters for the class package are defined:

```
import rosspy
ross = rosspy.ROSSPkg(database_selection, simulation = 'scaling', simulation_type =
↪'transport', operating_system = 'windows',
export_content = True, domain_phase = None, quantity_of_modules = 1, simulation_title =␣
↪None, verbose = False, printing = True, jupyter = False)
```

- *database_selection* `str`: specifies which PHREEQ database file – `Amm`, `ColdChem`, `core10`, `frezchem`, `iso`, `llnl`, `minteq`, `minteq.v4`, `phreeqc`, `pitzer`, `sit`, `Tipping_Hurley`, or `wateq4f` – will be imported and simulated. These databases were all processed via `PHREEQdb` of the ChemW module (in this specific Notebook: here).
- *simulation* `str`: specifies whether the `scaling` or `brine` of the simulation data will be processed.
- *simulation_type* `str`: specifies whether RO reactive transport `transport` or simple `evaporation` will be simulated.
- *operating_system* `str`: specifies whether the user is using a `windows` or `unix` system, which directs importing PHREEQpy and commenting in the `PQI` PHREEQ input files.
- *export_content* `bool`: species whether the simulation contents will be exported.
- *domain_phase* `str`: specifies the simulated domain model, where `None` executes the single-domain model and `mobile` (i.e. bulk solution) or `immobile` (i.e. the CP solution layer) specify the respective solutions of the dual-domain model, where the latter two options are still under development.
- *quantity_of_modules* `int`: specifies the simulated number of in-series RO modules.
- *simulation_title* `str`: specifies the simulation title in the PHREEQC `PQI` input file.
- *verbose*, *printing*, & *jupyter* `bool`: The first two parameters specify whether simulation details and calculated values will be printed, respectively. The last parameter specifies whether the simulation is executed within a Jupyter Notebook, which allows `display()` to better illustrate tables and figures.

reactive_transport()

The spatiotemporal transport specifications are defined through the following parameters:

```
ross.reactive_transport(simulation_time, simulation_perspective = None, final_cf = None,␣
↪module_characteristics = {}, ro_module = 'BW30-400', permeate_efficiency = 1,
head_loss = 0.1, evaporation_steps = 15, cells_per_module = 12, coarse_timestep = True,␣
↪kinematic_flow_velocity = None, exchange_factor = 1e5)
```

- *simulation_time* `float`: specifies the total simulated time in seconds.

- *simulation_perspective* `str`: specifies whether the simulation data is slice a) at the final timestep (`all_distance`) or b) at the final module cell (`all_time`). These perspectives allow data to be two-dimensionally graphed either over the module or over the simulated time, respectively, where `None` defaults to `all_time` for brine simulations and `all_distance` for scaling simulations.
- *final_cf* `float`: specifies the permeate flux calculation method, where `None` signifies the `linear_permeate` method while any numerical value of the effluent CF signifies the `linear_cf` method. These methods differ only in that the former distributes less scale at the beginning of the module and more scale at the end of the module, relative to the latter.
- *module_characteristics* `dict`: specifies characteristics of the simulated RO module, which supplant default values from the DOW FILMTEC BW30-400 RO module. The optional dictionary keys – `module_diameter_mm`, `permeate_tube_diameter_mm`, `module_length_m`, `permeate_flow_m3_per_day`, `max_feed_flow_m3_per_hour`, `membrane_thickness_mm`, `feed_thickness_mm`, `active_m2`, `permeate_thickness_mm`, `polysulfonic_layer_thickness_mm`, `support_layer_thickness_mm` – are themselves dictionaries with at least a key-value pair of `value` and the value, in the proper units in the characteristic name:

```
{
"active_m2": {
   "value":37
   },
"permeate_thickness_mm": {
   "value":0.3
   },
"polysulfonic_layer_thickness_mm": {
   "value":0.05
   }
}
```

- *ro_module* `str`: specifies the RO module that will be simulated from the defined entries in the `ro_module.json` parameter file. This additionally provides the default parameters that supplement values from the `module_characteristics` argument.
- *permeate_efficiency* `float`: specifies the 0<=PE<=1 proportion of calculated permeate flux that is simulated, as a means of representing diminished efficacy from a fouled module: e.g. `PE=1` denotes a perfectly operational module and `PE=0.5` denotes a 50% operational module, etc.
- *head_loss* `float`: specifies the 0<=HL<=1 head loss of effluent pressure relative to the influent. The default value of 0.1 corresponds to an 10% pressure drop over the course of desalination through the module.
- *cells_per_module* `int`: specifies the quantity of cells into which the RO module is discretized, and thereby controls distance resolution.
- *coarse_timestep* `bool`: specifies whether a timestep that is 12x greater than the Courant condition minimum is used, where `False` uses the Courant condition. The smaller timestep marginally improves resolution, at the expense of ~6x greater execution time.
- *kinematic_flow_velocity* `float`: specifies the kinetic flow velocity of the feed solution, where `None` defaults to 9.33E-7 (m^2/sec).
- *exchange_factor* `float`: specifies the rate (1/sec) of solution exchange between the mobile (bulk) and immobile (concentration polarization) solutions of a dual-domain simulation.

feed_geochemistry()

The feed geochemistry is defined through the following parameters:

```
ross.feed_geochemistry(water_selection = '', water_characteristics = {}, solution_
↪description = '', ignored_minerals = [], existing_scale = {}, parameterized_ph_charge␣
↪= True)
```

- *water_selection* `str`: specifies a parameter file of a feed water from the *rosspy/water_bodies* directory, where the default options encompass natural waters – the `red_sea` and the `mediterranean_sea` – and produced waters of fracking oil wells – the `bakken_formation`, `marcellus_appalachian_basin`, `michigan_basin`, `north_german_basin`, `palo_duro_basin`, and `western_pennsylvania_basin`. Parameter files for other feed waters can be created by emulating the syntax of these default files and storing the created file in the aforementioned directory, which is elaborated in the `parameter_files` documentation page.
- *water_characteristics* `dict`: defines the geochemistry and conditions of the feed that can supplant values from the `water_selection`. The expected keys – `element`, `temperature (C)`, `pe`, `Alkalinity`, and `pH` – each possess a dictionary value, with the keys of `value` for the numerical value and optionally others to express metadata: e.g. `reference` to denote the source of the numerical value. The `element` key deviates slightly from this organization by using another sub-dictionary layer for each ion in the feed, where the keys are `concentration (ppm)` for its ppm concentration, optionally `form` for the mineral form or charge-state of the ion, and optionally `reference` with the same aforementioned purpose:

```
{
   "element": {
       "Mn": {
           "concentration (ppm)": 0.000734,
           "reference": "El Sayed, Aminot, and Kerouel, 1994"
       },
       "Si": {
           "concentration (ppm)": 95,
           "reference": "Haluszczak, Rose, and Kump, 2013",
           "form": "SiO2"
       }
   },
   "temperature (C)": {
       "value": 24,
       "reference": "Dresel and Rose, 2010"
   }
}
```

- *solution_description* `str`: a brief solution description that can replace the *water_selection* in the simulation folder name.
- *ignored_minerals* `list`: defines the minerals that will be excluded from the set of minerals that could hypothetically precipitate from the feed.
- *existing_scale* `dict`: specifies pre-existing scaling in the simulated module, where the keys are the corresponding minerals and the values are sub-dictionaries with `saturation` and `initial_moles` as keys – which represent the pre-existing saturation index and the moles of the mineral, respectively – and the corresponding values are the numerical values.
- *parameterized_ph_charge* `bool`: specifies whether the pH will be charge balanced, which is exclusive with parameterizing feed alkalinity.

parse_input()

This function can import, parse, and execute pre-existing PHREEQ input files:

```
ross.parse_input(input_file_path, water_selection = None, active_m2= None)
```

- *input_file_path* `str`: specifies the path of the existing input file that will be imported and parsed.
- *water_selection* `str`: describes the simulated feed water.
- *active_m2* `float`: defines the area of active filtration in the simulated RO module, where `None` defaults to 37 from the standard FILMTEC BW30-400 module.

execute()

The input file is executed through PHREEQ:

```
processed_data = ross.execute(simulation_name = None, selected_output_path = None,␣
↪simulation_directory = None, figure_title = None, title_font = 'xx-large',
label_font = 'x-large', x_label_number = 6, export_name = None, export_format = 'svg',␣
↪scale_ions = True, define_paths = True, selected_output_filename = None)
```

- *simulation_name* `str`: specifies the name of the folder that will be created and populated with simulation contents.
- *selected_output_path* `str`: specifies the path of a simulation output file that will be processed into data tables and figures, which does not execute a new file and thus can process old data, where `None` executes the parameterized PHREEQ input file.
- *simulation_directory* `str`: The path to where the simulation content will be saved, where `None` signifies the current working directory.
- *figure_title* `str`: specifies the title of the simulation figure, where `None` defaults to customized titles that incorporate unique simulation details: e.g. `scaling` or `brine`, the water body, and the total simulation time.
- *title_font* & *label_font* `str`: specifications of the MatPlotLib fonts – `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, or `xx-large` – for the figure title and axis labels, respectively.
- *x_label_number* `int`: quantifies the x-axis ticks in the simulation figure.
- *export_name* `str`: specifies the export name of the simulation figure. The default name for `brine` simulations is `brine`, while the default name for `scaling` simulations is `all_minerals`.
- *export_format* `str`: specifies the format of the exported simulation figure, from the MatPlotLib options – `svg`, `pdf`, `png`, `jpeg`, `jpg`, or `eps` – where `svg` is the default as a lossless and highly editable format: e.g. via Inkscape.
- *scale_ions* `bool`: specifies whether the scale from `scaling` simulations will be reduced into proportions of individual ions, which is exported as a JSON file.
- *define_paths* `bool`: specifies, for the iROSSpy Notebook, whether the simulation path will be determined to prevent redundant folder creation.
- *selected_output_filename* `str`: specifies the name of the SELECTED_OUTPUT file, where `None` constructs a name with important simulation parameters.

Returned *processed_data* `DataFrame`: A Pandas DataFrames that possesses the processed simulation data, as convenient access for post-processing.

test()

ROSSpy can execute a simple test simulation via the `test()` function:

```
import rosspy
ross = rosspy.ROSSPkg(database_selection, simulation)
ross.test()
```

4.3 ROSSpy parameter files

Simulation parameters may be more succinctly provided through JSON files, which are imported by the code through identified function arguments, than through dictionaries that are passed as function arguments. Argument parameters can be used synergistically by supplanting specific characteristics that are different in the parameter files.

4.3.1 ro_module

The RO module characteristics can be provided through `ro_module.json`. The default entry embodies the DOW FILMTEC BW30-400 RO module; however, other modules can be defined by emulating the structure of the default entry, where each characteristic possesses the respective units in the key name:

```
{
   "BW30-400":{
       "module_diameter_mm": {
           "value": 201
       },
       "permeate_tube_diameter_mm": {
           "value": 29
       },
       "module_length_m": {
           "value": 1.016
       },
       "permeate_flow_m3_per_hour": {
           "value": 1.667
       },
       "max_feed_flow_m3_per_hour": {
           "value": 15.9
       },
       "feed_thickness_mm": {
           "value": 0.8636
       },
       "active_m2": {
           "value": 37
       },
       "permeate_thickness_mm": {
           "value": 0.3
       },
       "membrane_thickness_mm": {
           "value": 0.25
       },
       "polysulfonic_layer_thickness_mm": {
```

(continues on next page)

(continued from previous page)

```
            "value": 0.05
        },
        "support_layer_thickness_mm": {
            "value": 0.15
        }
    }
}
```

- *module_diameter_mm* `dict`: specifies the total diameter of the RO module.
- *permeate_tube_diameter_mm* `dict`: specifies the diameter of the permeate tube within the RO module.
- *module_length_m* `dict`: specifies the total length of the RO module.
- *permeate_flow_m3_per_hour* & *max_feed_flow_m3_per_hour* `dict`: specify the permeate and maximum feed flows through the RO module.
- *feed_thickness_mm* `dict`: defines the thickness of the feed spacer through which the feed passes.
- *active_m2* `dict`: defines the total filtration area of the RO module.
- *permeate_thickness_mm* `dict`: defines the thickness of the permeate spacer.
- *membrane_thickness_mm* & *polysulfonic_layer_thickness_mm* & *support_layer_thickness_mm* `dict`: define the thickensses of each layer in the composite filtration membrane: the polyamide layer that filters the feed, and the polysulfonic and support layers that provide resiliency to the membrane structure, respectively.

Other key:value sub-dictionaries may be introduced as metadata for the parameters.

4.3.2 water_bodies

The feed geochemistry may be defined as a parameter file, in the `water_bodies` folder within the `rosspy` package directory, that supplements feed characteristics as a dictionary through the `feed_geochemistry` function. The default files in this folder embody curated experimental data from both natural and produced water sources, which can be emulated for constructing files for other water sources, which each characteristic is expressed with the respective units in its key name:

```
{
    "element": {
        "Mn": {
            "concentration (ppm)": 3000,
            "reference": "Haluszczak, Rose, and Kump, 2013 [estimated from another␣
↪Marcellus publication]"
        },
        "Fe": {
            "concentration (ppm)": 26.6,
            "reference": "Chapman et al., 2012"
        },
        "B": {
            "concentration (ppm)": 20,
            "reference": "Haluszczak, Rose, and Kump, 2013 [reported average form another␣
↪Marcellus publication]"
        },
        "Cl": {
            "concentration (ppm)": 81900,
```

(continues on next page)

(continued from previous page)


```
            "reference": "Chapman et al., 2012"
        },
        "Na": {
            "concentration (ppm)": 32800,
            "reference": "Chapman et al., 2012"
        },
        "S(6)": {
            "concentration (ppm)": 45,
            "reference": "Haluszczak, Rose, and Kump, 2013 [estimated from another␣
↪Marcellus publication]"
        },
        "Ca": {
            "concentration (ppm)": 8786,
            "reference": "Chapman et al., 2012"
        },
        "K": {
            "concentration (ppm)": 350,
            "reference": "Haluszczak, Rose, and Kump, 2013 [estimated from another␣
↪Marcellus publication]"
        },
        "Mg": {
            "concentration (ppm)": 841,
            "reference": "Chapman et al., 2012"
        },
        "Sr": {
            "concentration (ppm)": 2415,
            "reference": "Chapman et al., 2012"
        },
        "Ba": {
            "concentration (ppm)": 962,
            "reference": "Chapman et al., 2012"
        },
        "Li": {
            "concentration (ppm)": 95,
            "reference": "Haluszczak, Rose, and Kump, 2013 [reported average from another␣
↪Marcellus publication]"
        }
    },
    "temperature (C)": {
        "value": 24,
        "reference": "Dresel and Rose, 2010"
    },
    "pe": {
        "value": null,
        "reference": null
    },
    "Alkalinity": {
        "value": 71,
        "reference": "Haluszczak, Rose, and Kump, 2013 [reported average from another␣
↪Marcellus publication]",
        "form": "CaCO3"
    },
```


(continues on next page)

(continued from previous page)

```
    "pH": {
        "value": 7,
        "reference": "Haluszczak, Rose, and Kump, 2013 [estimated from another Marcellus␣
↪publication]"
    }
}
```

- *element* `dict`: specifies all of the elements that are present in the feed, with sub-dictionaries of their concentrations and metadata. Some of these elements will not be amenable with some databases, which ROSSpy will simply ignore when defining the input file for an incompatible database.
- *temperature (C)*, *pe*, *Alkalinity*, & *pH* `dict`: specify conditions and characteristics of the feed solution, with sub-directories of their respective value, chemical formula, and optionally metadata.

4.4 iROSSpy

The essential functionality of ROSSpy has been adapted into an interactive Notebook through MyBinder. The Notebook cell allows the investigator to execute the script and intuitively design, execute, and/or process a ROSSpy simulation through the Notebook console, where the simulation results are conveniently printed. This Binder is intended to support non-technical investigators who still desire to simulate scaling and brine formation from desalination without using an API.