The Journal of Open Source Software

PyToughReact – A Python Package for automating reactive transport and biodegradation simulations.

Temitope Ajayi [*1] **and Ipsita Gupta**[1¶]

1 Louisiana State University, USA ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Matthew Feickert

Reviewers:

- @blakeaw
- @frank1010111

Submitted: 05 December 2023
Published: unpublished

Statement of need

The suite of TOUGH simulators by the Lawrence Berkeley National Laboratory (LBNL) are well known simulators for flow and transport simulations. In order to model chemical reactions coupled to flow for isothermal and non-isothermal systems, the TOUGHREACT simulator exists. In addition, the TMVOC software exists for the modeling of multicomponent mixtures of volatile organic compounds suitable for contamination problems. When biodegradation is required to be modeled, the tool used is known as TMVOC-BIO. These reaction simulators as of now do not provide a mechanism to automate simulations for tasks such as uncertainty quantification or sensitivity analysis which require multiple simulation runs. Users need to create many simulation folders to perform every sensitivity they intend to run with the simulator. This makes it cumbersome to use and users could get lost on the purpose of each folder if not correctly labeled. While PyTOUGH and TOUGHIO exists for fluid flow and transport simulations, no such tool exists for chemical and biodegradation reactions, `PyToughReact` fulfills this need for users familiar with python. Further, `PyTOUGHREACT` enables users familiar with python to incorporate language capabilities into their reaction simulation workflows.

Introduction

TOUGH suite of simulators requires users to write or modify a text file and parse the file to an executable to perform the simulation. Results from simulations are subsequently also saved to text files for interpretation by third party softwares. These makes it cumbersome for sensitivity analysis and uncertainty studies to be conducted as this would require a user to make multiple manual edits to files to extract required data. As would be imagined, this is also prone to human errors. Further, coupling the simulator with other simulators would be more involving. As a result of this, numerous tools have been created that wrap around the executables and make writing, reading and visualization of model properties easier to implement. Examples of such include PetraSim (Yamamoto, 2008) for which assist in creating the models and visualizing results, TOUGH2VIEWER (Bondua et al., 2012) and TECPLOT (Tecplot, 2013) for postprocessing and visualizations and IGMESH for building and integrating visualization tools suited especially for irregular gridding (Hu et al., 2016). For scripting purposes, PyTOUGH (Croucher, 2011) and TOUGHIO (Luu, 2020) are used for pre and post processing of TOUGH2 simulations. So far, no tool exists for scripting of the reaction softwares of TOUGH. Reactive processes as with other processes are subject to lots of uncertainties and need to be adequately accounted for in engineering studies. To enable this with the TOUGHREACT simulator, it is essential to create a scripting tool to accomplish this. This tool extends the work done by PyTOUGH in creating an automatic platform for running TOUGH flow simulations by creating a concurrent tool for automating chemical and biodegradation reactions from any python enabled terminal or IDE.

*Co-first author

The Journal of Open Source Software

Architecture

General Architecture

The software uses object-oriented programming principles to structure the code. The software can be thought of as composed of two main segments; the processing segment and the output segment (Figure 1). The processing segment contains three main sections, IO processing, BIO and REACT. The IO processing is responsible for most of the input and output processing such as reading, writing. The BIO section is responsible for the TMVOC section of the package where it contains classes for storing biomass and degradation information and processing it before passing to IO processing for read/write. Similarly, the react section assists in processing reaction parameters such as mineral, chemical, and solute information before passing to the IO processing segment for read/write. After the files have been written to or read from the appropriate file types, the executable is called from within PyTOUGHREACT and the simulation is performed using the executable. Thereafter, the output segment is called which can read the results of the simulations and contains methods and functions which assist the user in creating 2D or 3D plots through the plotting module.

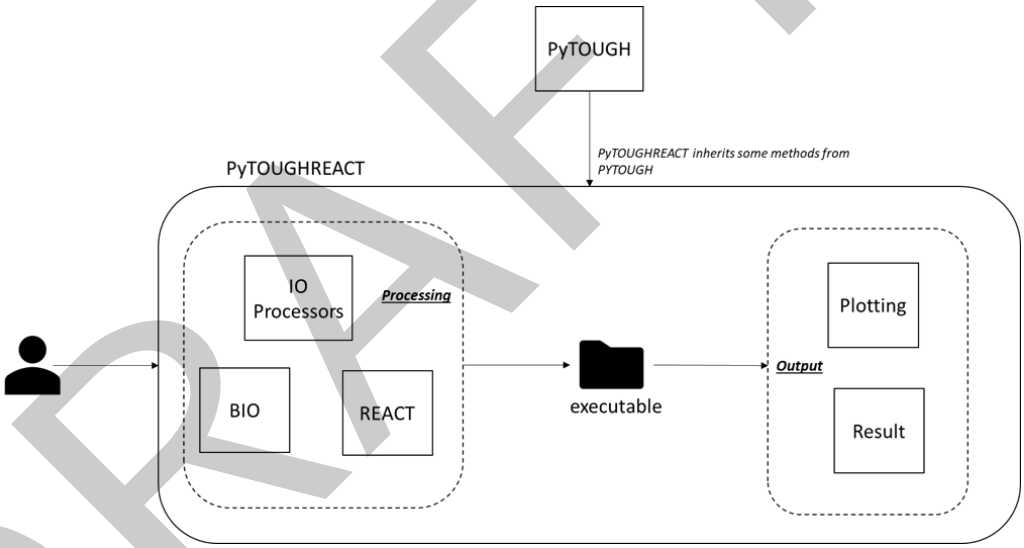

Figure 1: General Structure of the PyTOUGHREACT package.

BIO Architecture

The BIO section is responsible for processing the TMVOC inputs. It makes use of two subsections to achieve this (Figure 2). The first subsection is responsible for aggregating the unique constituents of a biodegradation simulation for TMVOC; these constituents include water, gas, biomass, solids, and components. The second subsection is responsible for unique biodegradation processes in the simulation. This is done via two classes Process class and the BIODG class. The process class defines the constituents for any biodegradation process and BIODG class combines all processes together with numerical parameters for that process.

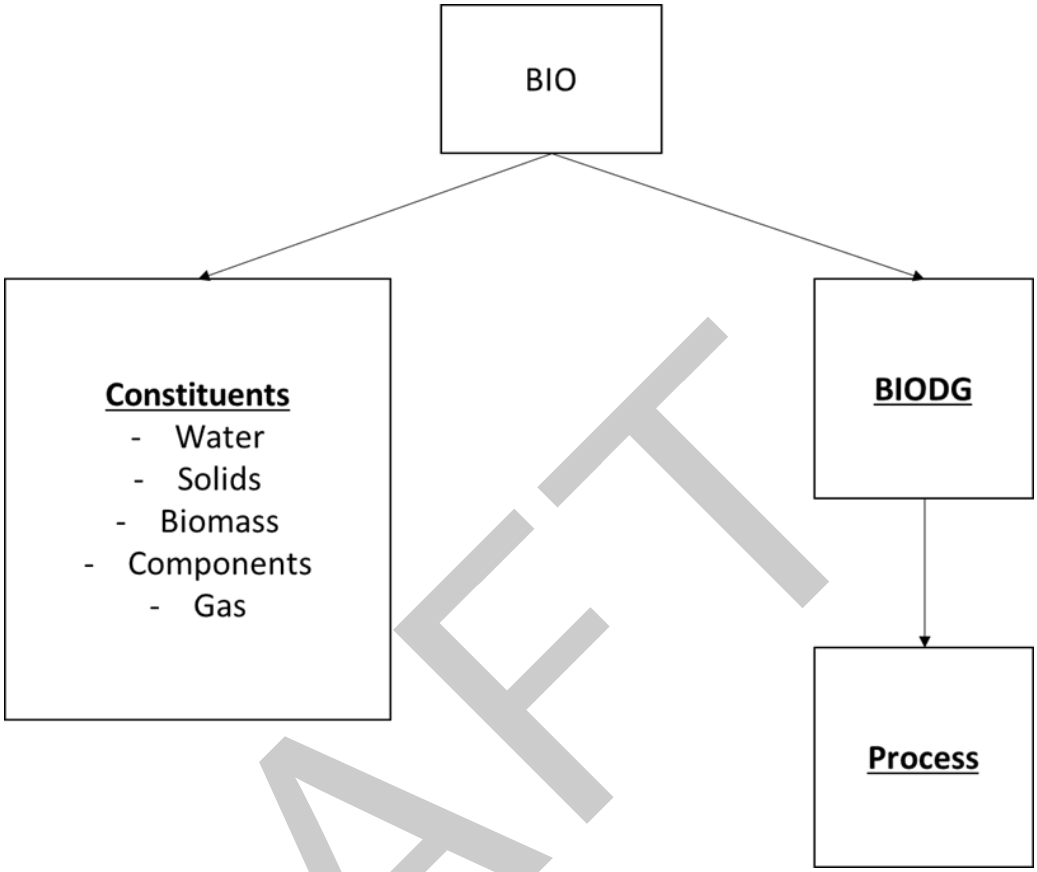

Figure 2: BIO Architecture Structure.

React Architecture

The react component consists of two main subcomponents: the chemical subcomponent and the solute subcomponent. The chemical subcomponent is responsible for defining the chemical constituents of the simulation such as the primary species, water, mineral, gas while the solute sub component is responsible for mapping each of the defined chemical constituents to the grid of the simulation and other functions such as what grids to write to output etc. This component contains information stored in the solute.inp file for TOUGHREACT simulations.

Result and Plotting Architecture

Though separate, the result and plotting architecture have very similar structures (Figure 3). The two sections are subdivided into single and multiple file processing. As the name suggests, the single file enables the processing of a single file. While the results section enables users to process simulation output further in python, the plotting sections uses the output from the results section to make line and 2D plots. Similarly, the multiple file processing of the results section makes use of multiple files to output results which are then used in the plotting section for line and 2D plots. The multiple file processing provides efficiency when multiple files are required to be processed or in the plotting section to see variations from one simulation to another in a single plot.

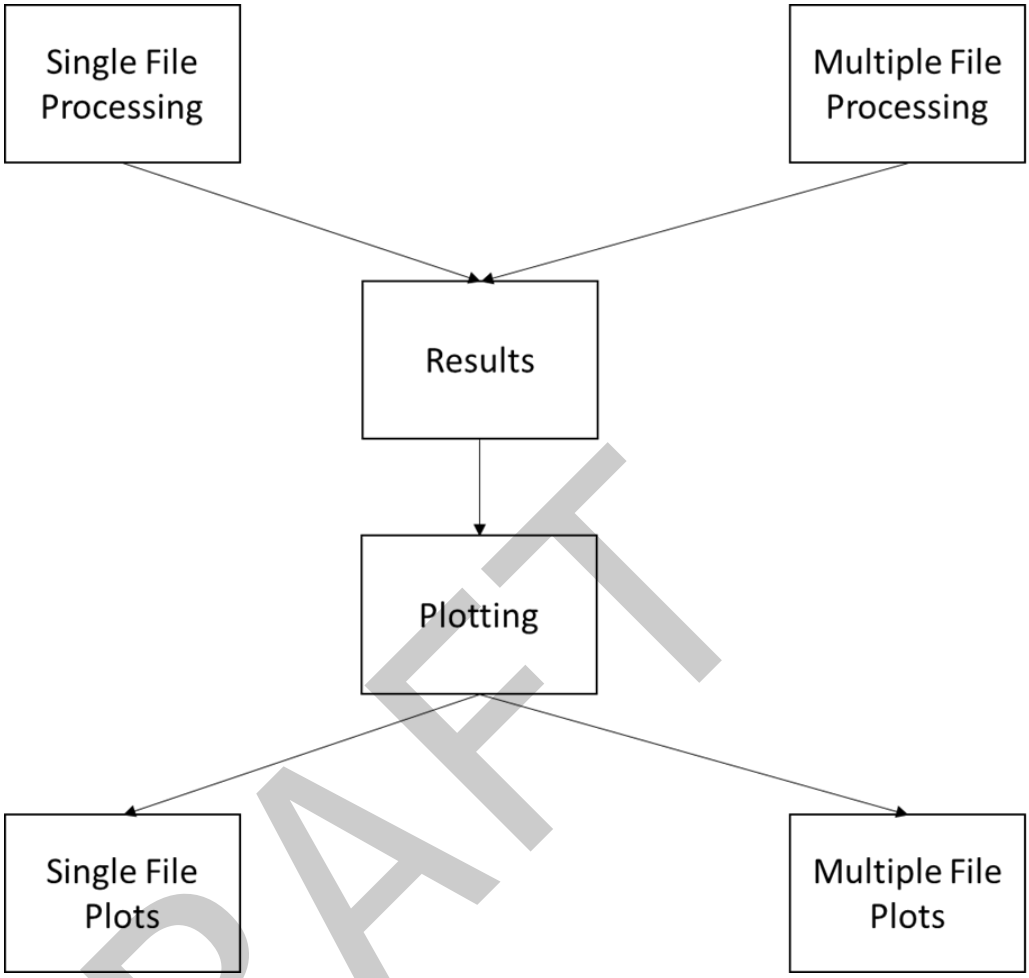

Figure 3: BIO Architecture Structure.

Sample Simulations

Two broad examples are shown in the next two sections to show how the package can be applied to both biodegradation and chemical reaction simulations using TOUGHREACT and TMVOCBIO

Chemical Reaction Example

```
import os
from mulgrids import mulgrid
from pytoughreact.writers.react_writing import t2react
from pytoughreact.wrapper.reactgrid import t2reactgrid
from pytoughreact.wrapper.reactzone import t2zone
from pytoughreact.chemical.chemical_composition import PrimarySpecies, WaterComp, Water,
from pytoughreact.chemical.mineral_composition import MineralComp
from pytoughreact.chemical.mineral_zone import MineralZone
from pytoughreact.constants.default_minerals import get_kinetics_minerals, get_specific_
from pytoughreact.writers.chemical_writing import t2chemical
from pytoughreact.chemical.perm_poro_zone import PermPoro, PermPoroZone
from pytoughreact.writers.solute_writing import t2solute
```

```
from t2grids import rocktype

# ___________________________________FLOW.INP_____________________________________________
length = 0.1
nblks = 1
dx = [length / nblks] * nblks
dy = [0.5]
dz = [0.5] * 1
geo = mulgrid().rectangular(dx, dy, dz)
geo.write('geom.dat')

react = t2react()
react.title = 'Reaction example'

react.multi = {'num_components': 1, 'num_equations': 1, 'num_phases': 2,
               'num_secondary_parameters': 6}

react.grid = t2reactgrid().fromgeo(geo)

react.parameter.update(
    {'print_level': 4,
     'max_timesteps': 9999,
     'tstop': 8640,
     'const_timestep': 10.,
     'print_interval': 1,
     'gravity': 9.81,
     'relative_error': 1e-5,
     'phase_index': 2,
     'default_incons': [1.013e5, 25]})

sand = rocktype('ROCK1', 0, 2600, 0.1, [6.51e-12, 6.51e-12, 6.51e-12], 0.0, 952.9)

react.grid.delete_rocktype('dfalt')
react.grid.add_rocktype(sand)

for blk in react.grid.blocklist[0:]:
    blk.rocktype = react.grid.rocktype[sand.name]

zone1 = t2zone('zone1')

react.grid.add_zone(zone1)

for blk in react.grid.blocklist[0:]:
    blk.zone = react.grid.zone[zone1.name]

react.start = True

react.write('flow.inp')

# ___________________________________CHEMICAL.INP_____________________________________________
h2o = PrimarySpecies('h2o', 0)
h = PrimarySpecies('h+', 0)
na = PrimarySpecies('na+', 0)
cl = PrimarySpecies('cl-', 0)
hco3 = PrimarySpecies('hco3-', 0)
```

```
ca = PrimarySpecies('ca+2', 0)
so4 = PrimarySpecies('so4-2', 0)
mg = PrimarySpecies('mg+2', 0)
h4sio4 = PrimarySpecies('h4sio4', 0)
al = PrimarySpecies('al+3', 0)
fe = PrimarySpecies('fe+2', 0)
hs = PrimarySpecies('hs-', 0)

all_species = [h2o, h, na, cl, hco3, ca, so4, mg, h4sio4, al, fe, hs]

h2o_comp1 = WaterComp(h2o, 1, 1.0000E+00, 1.000000E+00)
h_comp1 = WaterComp(h, 1, 1E-7, 1E-7)
na_comp1 = WaterComp(na, 1, 1E-10, 2.93E-2)
cl_comp1 = WaterComp(cl, 1, 1E-10, 1.08E-3)
hco3_comp1 = WaterComp(hco3, 1, 1E-10, 2.21E-08)
ca_comp1 = WaterComp(ca, 1, 1E-10, 5.9E-03)
so4_comp1 = WaterComp(so4, 1, 1E-10, 6.94E-3)
mg_comp1 = WaterComp(mg, 1, 1E-10, 2.54E-8)
h4sio4_comp1 = WaterComp(h4sio4, 1, 1E-10, 1E-10)
al_comp1 = WaterComp(al, 1, 1E-10, 9.96E-5)
fe_comp1 = WaterComp(fe, 1, 1E-10, 9.7E-9)
hs_comp1 = WaterComp(hs, 1, 1E-10, 1E-10)

initial_water_zone1 = Water([h2o_comp1, h_comp1, na_comp1, cl_comp1, hco3_comp1, ca_comp

mineral_list = ['c3fh6', 'tobermorite', 'calcite', 'csh', 'portlandite', 'ettringite', '
all_minerals = get_kinetics_minerals(mineral_list)


c3fh6_zone1 = MineralComp(get_specific_mineral(mineral_list[0]), 0.1, 0, 0.0E-00, 20000.
tobermorite_zone1 = MineralComp(get_specific_mineral(mineral_list[1]), 0.05, 0, 0.0E-00,
calcite_zone1 = MineralComp(get_specific_mineral(mineral_list[2]), 0.4, 1, 0.0E-00, 260.
csh_zone1 = MineralComp(get_specific_mineral(mineral_list[3]), 0.1, 1, 0.0E-00, 20000.0,
portlandite_zone1 = MineralComp(get_specific_mineral(mineral_list[4]), 0.1, 1, 0.0E-00,
ettringite_zone1 = MineralComp(get_specific_mineral(mineral_list[5]), 0.1, 1, 0.0E-00, 2
katoite_zone1 = MineralComp(get_specific_mineral(mineral_list[6]), 0.1, 1, 0.0E-00, 570.
hydrotalcite_zone1 = MineralComp(get_specific_mineral(mineral_list[7]), 0.05, 1, 0.0E-00

initial_co2 = ReactGas('co2(g)', 0, 1.1)
ijgas = [[initial_co2], []]

permporo = PermPoro(1, 0, 0)
permporozone = PermPoroZone([permporo])

zone1.water = [[initial_water_zone1], []]
zone1.gas = [[initial_co2], []]
mineral_zone1 = MineralZone([c3fh6_zone1, tobermorite_zone1, calcite_zone1, csh_zone1, p
zone1.mineral_zone = mineral_zone1
zone1.permporo = permporozone

writeChemical = t2chemical(t2reactgrid=react.grid)
writeChemical.minerals = all_minerals
writeChemical.title = 'Automating Tough react'
writeChemical.primary_aqueous = all_species
writeChemical.gases = initial_co2
```

```
writeChemical.write()

# ____________________________________SOLUTE.INP_____________________________________
writeSolute = t2solute(t2chemical=writeChemical)
writeSolute.nodes_to_write = [0]
masa = writeSolute.getgrid_info()
writeSolute.write()

# ____________________________________ RUN SIMULATION _____________________________________
print(os.path.dirname(__file__))
react.run(writeSolute, simulator='treacteos1.exe')
```

Biodegradation Reaction Example

```
import numpy as np
import os

from mulgrids import mulgrid
from pytoughreact.writers.bio_writing import t2bio
from pytoughreact.chemical.biomass_composition import Component, Biomass, Gas, Water_Bio
from pytoughreact.chemical.bio_process_description import BIODG, Process
from t2grids import t2grid
from t2data import rocktype, t2generator

# ____________________________________FLOW.INP_____________________________________
second = 1
minute = 60 * second
hour = 60 * minute
day = 24 * hour
year = 365. * day
year = float(year)
simtime = 100 * year

length = 1000.
xblock = 10
yblock = 1
zblock = 5
dx = [length / xblock] * xblock
dy = [1.0]
dz = [5] * zblock
geo = mulgrid().rectangular(dx, dy, dz, origin=[0, 0, -95])
geo.write('geom.dat')

bio = t2bio()
bio.title = 'Biodegradation Runs'

bio.grid = t2grid().fromgeo(geo)
bio.grid.delete_rocktype('dfalt')
shale = rocktype('shale', 0, 2600, 0.27, [6.51e-19, 6.51e-19, 6.51e-19], 1.5, 900)
bio.grid.add_rocktype(shale)

for blk in bio.grid.blocklist[0:]:
```

```
        blk.rocktype = bio.grid.rocktype[shale.name]

bio.multi = {'num_components': 3, 'num_equations': 3, 'num_phases': 3,
             'num_secondary_parameters': 8}

bio.parameter.update(
    {'print_level': 3,
     'max_timesteps': 9999,
     'tstop': simtime,
     'const_timestep': 100.,
     'print_interval': 1,
     'gravity': 9.81,
     'option': np.array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     'relative_error': 1e-5,
     'phase_index': 2,
     'default_incons': [9.57e+06, 0, 1e-6, 30.]})

# ____________________________________BIODEGRADATION_____________________________
bio.start = True
toluene = Component(1).defaultToluene()
bio.components = [toluene]
O2_gas = Gas('O2', 2)
bio.gas = [O2_gas]

water = Water_Bio('H2O')

biomass = Biomass(1, 'biom', 0.0153, 1.00e-6, 30, 2.3148e-07, 0.e-6)
oxygen_ks = 0.5e-6
oxygen_uptake = 1
water_uptake = -3

process1 = Process(biomass, 2, 1.6944e-04, 0.58, 0)
water.addToProcess(process1, water_uptake)
O2_gas.addToProcess(process1, oxygen_uptake, oxygen_ks)
toluene.addToProcess(process1, 1, 7.4625e-06)

biodegradation = BIODG(0, 1.e-10, 0, 0.2, 0.9, 0.9,
                       [process1],
                       [biomass])
bio.biodg = [biodegradation]

bio.diffusion = [
    [2.e-5, 6.e-10, 6.e-10],
    [2.e-5, 6.e-10, 6.e-10],
    [2.e-5, 6.e-10, 6.e-10],
    [2.e-5, 6.e-10, 6.e-10],
    [2.e-5, 6.e-10, 6.e-10],
    [2.e-5, 6.e-10, 6.e-10],
    [2.e-5, 6.e-10, 6.e-10],
    [2.e-5, 6.e-10, 6.e-10]
]

well = 'wl '
compo = ['COM3']
direction = 'x'
```

```
duration = [0, 1 * year, 101 * year]
# duration = np.linspace(0, simtime * 2, 7)
rate = np.array([1.00e-2, 0, 0])
rate_O2 = [1.00e-03, 0, 0]
energy = [5, 5, 5]

if direction == 'x':
    j = 0
    for i in range(0, xblock):
        # for i in range(xblock * (zblock - 1), xblock * (zblock)):
        for component in compo:
            if component == 'COM2':
                gen = t2generator(name=well + str(i), block=bio.grid.blocklist[i].name,
                                  ltab=len(duration),
                                  itab=str(3),
                                  time=duration, rate=rate_O2, enthalpy=energy)
                bio.add_generator(gen)
            else:
                gen = t2generator(name=well + str(i), block=bio.grid.blocklist[i].name,
                                  ltab=len(duration),
                                  itab=str(3),
                                  time=duration, rate=rate, enthalpy=energy)
                bio.add_generator(gen)
            j = j + 1

# ____________________________________RUN SIMULATION_____________________________________

runlocation = os.getcwd()
bio.write('INFILE', runlocation=os.getcwd())
bio.run(simulator='tmvoc', runlocation='')
```


Acknowledgements

Funding for this research is from the National Academy of Sciences, Engineering, and Medicine (NASEM) Gulf Research Program (GRP) grant on "Mitigating Risks to Hydrocarbon Release through Integrative Advanced Materials for Wellbore Plugging and Remediation" under award number 200008863 and the Early Career Research Fellowship of Ipsita Gupta.

References

Bondua, S., Berry, P., Bortolotti, V., & Cormio, C. (2012). A post-processing tool for interactive 3D visualization of locally refined unstructured grids for TOUGH2. *Computers & Geosciences*. https://doi.org/10.1016/j.cageo.2012.04.008

Croucher, A. (2011). PyTOUGH: a Python scripting library for automating TOUGH2 simulations. *New Zealand Geothermal Workshop*.

Hu, L., Zhang, K., Cao, X., Li, Y., & Guo, C. (2016). A convenient irregular-grid-based pre-and post-processing tool for TOUGH2 simulator. *Computers & Geosciences*. https://doi.org/10.1016/j.cageo.2016.06.014

Luu, K. (2020). toughio: Pre- and post-processing Python library for TOUGH. *Journal of Open Source Software*. https://doi.org/10.21105/joss.02412

Tecplot. (2013). *Tecplot360 user's manual*. Tecplot.

Yamamoto, H. (2008). PetraSim: A graphical user interface for the TOUGH2 family of multiphase flow and transport codes. *Groundwater*. https://doi.org/10.1111/j.1745-6584.2008.00462.x