dfnWorks Documentation

Release 2.5

Subsurface Flow and Transport Team LANL LA-UR-17-22216

Apr 13, 2021

CONTENTS

Contents:

CHAPTER

ONE

WELCOME TO DFNWORKS

dfnWorks is a parallelized computational suite to generate three-dimensional discrete fracture networks (DFN) and simulate flow and transport. Developed at Los Alamos National Laboratory, it has been used to study flow and transport in fractured media at scales ranging from millimeters to kilometers. The networks are created and meshed using dfnGen, which combines FRAM (the feature rejection algorithm for meshing) methodology to stochastically generate three-dimensional DFNs with the LaGriT meshing toolbox to create a high-quality computational mesh representation. The representation produces a conforming Delaunay triangulation suitable for high-performance computing finite volume solvers in an intrinsically parallel fashion. Flow through the network is simulated with dfnFlow, which utilizes the massively parallel subsurface flow and reactive transport finite volume code PFLOTRAN. A Lagrangian approach to simulating transport through the DFN is adopted within dfnTrans to determine pathlines and solute transport through the DFN. Applications of the dfnWorks suite include nuclear waste repository science, hydraulic fracturing and CO_2 sequestration.

To run a workflow using the dfnWorks suite, the pydfnworks package is highly recommended. pydfnworks calls various tools in the dfnWorks suite with the aim to provide a seamless workflow for scientific applications of dfnWorks.

1.1 Obtaining dfnWorks

dfnWorks 2.5 can be downloaded from https://hub.docker.com/r/ees16/dfnworks

dfnWorks 2.5 can be downloaded from https://github.com/lanl/dfnWorks/

v1.0 can be downloaded from https://github.com/dfnWorks/dfnWorks-Version1.0

1.2 Citing dfnWorks

Hyman, J. D., Karra, S., Makedonska, N., Gable, C. W., Painter, S. L., & Viswanathan, H. S. (2015). dfnWorks: A discrete fracture network framework for modeling subsurface flow and transport. Computers & Geosciences, 84, 10-19.

BibTex:

```
  @article{hyman2015dfnWorks,
    title={dfnWorks: A discrete fracture network framework
for modeling subsurface flow and transport},
    author={Hyman, Jeffrey D and Karra, Satish and Makedonska,
Nataliia and Gable, Carl W and Painter, Scott L
and Viswanathan, Hari S},
    journal={Computers \& Geosciences},
    volume={84},
    pages={10--19},
```

(continues on next page)

(continued from previous page)

```
    year={2015},
    publisher={Elsevier}
  }
```

1.3 Versions

1.3.1 v2.5

- New Generation parameters, family orientation by trend/plunge and dip/strike
- Define fracture families by region
- Updated output report

1.3.2 v2.4

- New meshing technique (Poisson disc sampling)
- Define fracture families by region
- Updated output report
- Well Package

1.3.3 v2.3

- Bug fixes in LaGrit Meshing
- Bug fixes in dfnTrans checking
- Bug fixes in dfnTrans output
- Expanded examples
- Added PDF printing abilities

1.3.4 v2.3

- pydfnWorks updated for python3
- Graph based (pipe-network approximations) for flow and transport
- Bug fixes in LaGrit Meshing
- Increased functionalities in pydfnworks including the path option
- dfn2graph capabilities
- FEHM flow solver
- Streamline routing option in dfnTrans
- Time Domain Random Walk in dfnTrans

1.3.5 v2.1

- Bug fixes in LaGrit Meshing
- Increased functionalities in pydfnworks including the path option

1.3.6 v2.0

- New dfnGen C++ code which is much faster than the Mathematica dfnGen. This code has successfully generated networks with 350,000+ fractures.
- Increased functionality in the pydfnworks package for more streamlined workflow from dfnGen through visualization.

1.4 About this manual

This manual comprises of information on setting up inputs to dfnGen, dfnTrans and PFLOTRAN, as well as details on the pydfnworks module: *pydfnworks*. Finally, the manual contains a short tutorial with prepared examples that can be found in the `examples` directory of the dfnWorks repository, and a description of some applications of the dfnWorks suite.

1.5 Contact

Please email dfnworks@lanl.gov with questions about dfnWorks. Please let us know if you publish using dfnWorks and we'll add it to the *Publication Page*

1.6 Contributors

1.6.1 LANL

- Jeffrey D. Hyman
- Satish Karra
- Nataliia Makedonska
- Carl Gable
- Hari Viswanathan
- Matt Sweeney
- Shriram Srinivasan
- Aric Hagberg
- Yu Chen

1.6.2 External

- Quan Bui (now at LLNL)
- Jeremy Harrod (now at Spectra Logic)
- Scott Painter (now at ORNL)
- Thomas Sherman (University of Notre Dame)
- Johannes Krotz (Oregon State University)

1.7 Copyright Information

Documentation:

LA-UR-17-22216

Software copyright:

LA-CC-17-027

Contact Information : dfnworks@lanl.gov

(or copyright) 2018 Triad National Security, LLC. All rights reserved.

This program was produced under U.S. Government contract 89233218CNA000001 for Los Alamos National Laboratory (LANL), which is operated by Triad National Security, LLC for the U.S. Department of Energy/National Nuclear Security Administration.

All rights in the program are reserved by Triad National Security, LLC, and the U.S. Department of Energy/National Nuclear Security Administration. The Government is granted for itself and others acting on its behalf a nonexclusive, paid-up, irrevocable worldwide license in this material to reproduce, prepare derivative works, distribute copies to the public, perform publicly and display publicly, and to permit others to do so.

The U.S. Government has rights to use, reproduce, and distribute this software. NEITHER THE GOVERNMENT NOR TRIAD NATIONAL SECURITY, LLC MAKES ANY WARRANTY, EXPRESS OR IMPLIED, OR ASSUMES ANY LIABILITY FOR THE USE OF THIS SOFTWARE. If software is modified to produce derivative works, such modified software should be clearly marked, so as not to confuse it with the version available from LANL.

Additionally, this program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. Accordingly, this program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

Additionally, redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of Los Alamos National Security, LLC, Los Alamos National Laboratory, LANL, the U.S. Government, nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY LOS ALAMOS NATIONAL SECURITY, LLC AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL LOS ALAMOS NATIONAL SECURITY, LLC OR CONTRIBUTORS BE LIABLE FOR

ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Additionally, this program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. Accordingly, this program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

CHAPTER

TWO

EXAMPLE APPLICATIONS

2.1 Carbon dioxide sequestration

dfnWorks provides the framework necessary to perform multiphase simulations (such as flow and reactive transport) at the reservoir scale. A particular application, highlighted here, is sequestering CO_2 from anthropogenic sources and disposing it in geological formations such as deep saline aquifers and abandoned oil fields. Geological CO_2 sequestration is one of the principal methods under consideration to reduce carbon footprint in the atmosphere due to fossil fuels (Bachu, 2002; Pacala and Socolow, 2004). For safe and sustainable long-term storage of CO_2 and to prevent leaks through existing faults and fractured rock (along with the ones created during the injection process), understanding the complex physical and chemical interactions between CO_2 , water (or brine) and fractured rock, is vital. dfnWorks capability to study multiphase flow in a DFN can be used to study potential CO_2 migration through cap-rock, a potential risk associated with proposed subsurface storage of CO_2 in saline aquifers or depleted reservoirs. Moreover, using the reactive transport capabilities of PFLOTRAN coupled with cell-based transmissivity of the DFN allows one to study dynamically changing permeability fields with mineral precipitation and dissolution due to CO_2 –water interaction with rock.

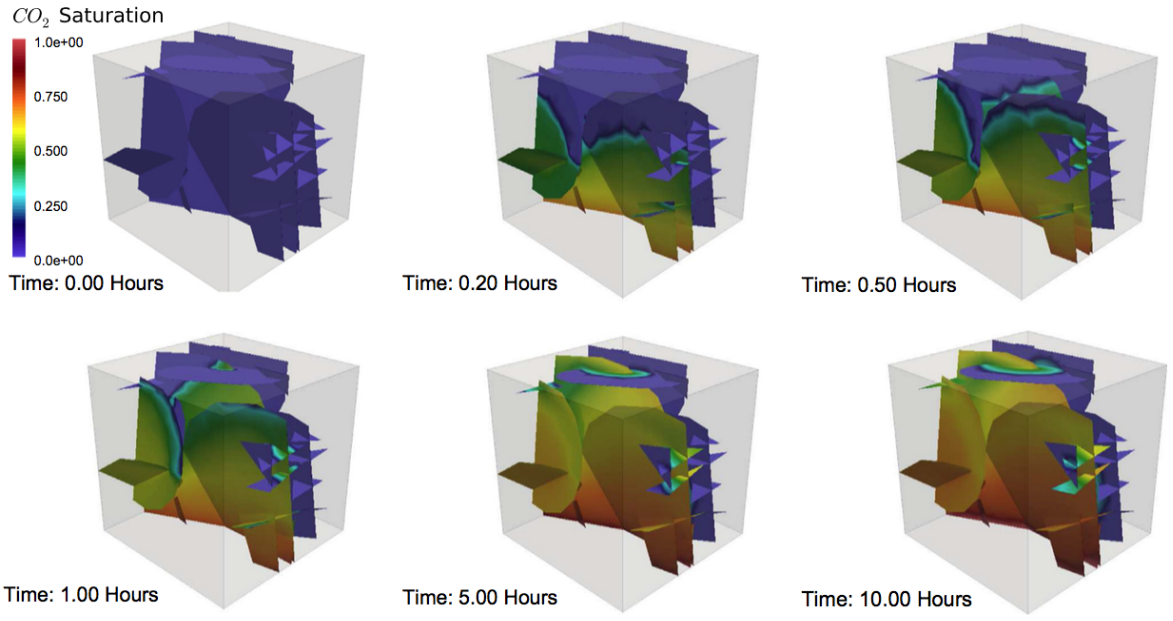


Fig. 1: *Temporal evolution of supercritical |CO2| displacing water in a meter cube DFN containing 24 fractures. The DFN is initially fully saturated with water, (top left time 0 hours) and supercritical |CO2| is slowly injected into the system from the bottom of the domain to displace the water for a total time of 10 h. There is an initial flush through the system during the first hour of the simulation, and then the rate of displacement decreases.*

2.2 Shale energy extraction

Hydraulic fracturing (fracking) has provided access to hydrocarbon trapped in low-permeability media, such as tight shales. The process involves injecting water at high pressures to reactivate existing fractures and also create new fractures to increase permeability of the shale allowing hydrocarbons to be extracted. However, the fundamental physics of why fracking works and its long term ramifications are not well understood. Karra et al. (2015) used dfnWorks to generate a typical production site and simulate production. Using this physics based model, they found good agreement with production field data and determined what physical mechanisms control the decline in the production curve.

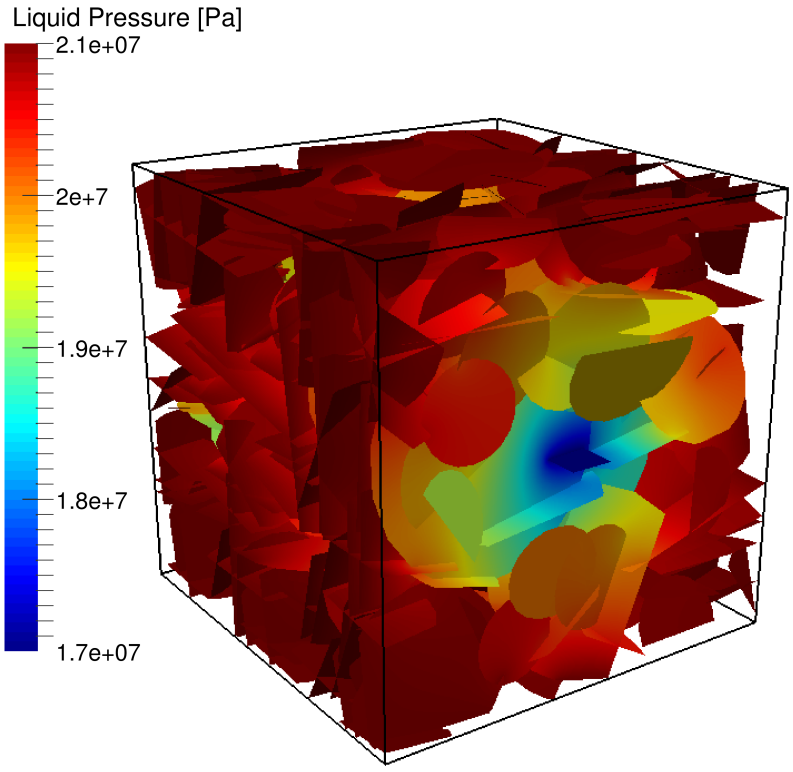


Fig. 2: *Pressure in a well used for hydraulic fracturing.*

2.3 Nuclear waste repository

The Swedish Nuclear Fuel and Waste Management Company (SKB) has undertaken a detailed investigation of the fractured granite at the Forsmark, Sweden site as a potential host formation for a subsurface repository for spent nuclear fuel (SKB, 2011; Hartley and Joyce, 2013). The Forsmark area is about 120 km north of Stockholm in northern Uppland, and the repository is proposed to be constructed in crystalline bedrock at a depth of approximately 500 m. Based on the SKB site investigation, a statistical fracture model with multiple fracture sets was developed; detailed parameters of the Forsmark site model are in SKB (2011). We adopt a subset of the model that consist of three sets of background (non-deterministic) circular fractures whose orientations follow a Fisher distribution, fracture radii are sampled from a truncated power-law distribution, the transmissivity of the fractures is estimated using a power-law model based on the fracture radius, and the fracture aperture is related to the fracture size using the cubic law (Adler et al., 2012). Under such a formulation, the fracture apertures are uniform on each fracture, but vary among fractures. The network is generated in a cubic domain with sides of length one-kilometer. Dirichlet boundary conditions are

imposed on the top (1 MPa) and bottom (2 MPa) of the domain to create a pressure gradient aligned with the vertical axis, and noflow boundary conditions are enforced along lateral boundaries.

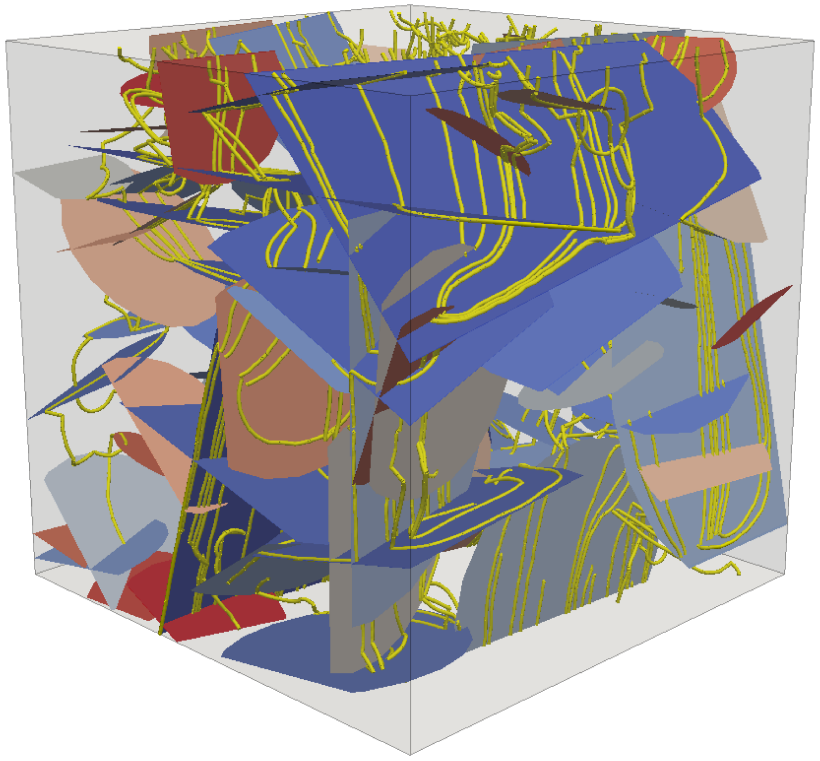


Fig. 3: *Simulated particle trajectories in fractured granite at Forsmark, Sweden.*

Sources:

- Adler, P.M., Thovert, J.-F., Mourzenko, V.V., 2012. Fractured Porous Media. Oxford University Press, Oxford, United Kingdom.
- Bachu, S., 2002. Sequestration of CO_2 in geological media in response to climate change: road map for site selection using the transform of the geological space into the CO_2 phase space. Energy Convers. Manag. 43, 87–102.
- Hartley, L., Joyce, S., 2013. Approaches and algorithms for groundwater flow modeling in support of site investigations and safety assessment of the Fors- mark site, Sweden. J. Hydrol. 500, 200–216.
- Karra, S., Makedonska, N., Viswanathan, H., Painter, S., Hyman, J., 2015. Effect of advective flow in fractures and matrix diffusion on natural gas production. Water Resour. Res., under review.
- Pacala, S., Socolow, R., 2004. Stabilization wedges: solving the climate problem for the next 50 years with current technologies. Science 305, 968–972.
- SKB, Long-Term Safety for the Final Repository for Spent Nuclear Fuel at Forsmark. Main Report of the SR-Site Project. Technical Report SKB TR-11-01, Swedish Nuclear Fuel and Waste Management Co., Stockholm,

Sweden, 2011.

CHAPTER

THREE

SETTING AND RUNNING UP DFNWORKS

3.1 Docker

The easiest way to get started with dfnWorks is using our docker container (https://hub.docker.com/r/ees16/dfnworks).

If you do not already have Docker installed on your machine, visit Getting Started with Docker.

The dfnWorks Docker image can be pulled from DockerHub using:

```
$ docker pull ees16/dfnworks:latest
```

3.1.1 Running the dfnWorks container

The base command for running the dfnWorks container is:

```
docker run -ti ees16/dfnworks:latest
```

However, to exchange files between the host and container, we will need to mount a volume.

The option `-v LOCAL_FOLDER:/dfnWorks/work` will allow all files present in the container folder `dfnWorks/work` to be exposed to `LOCAL_FOLDER`, where `LOCAL_FOLDER` is the absolute path to a folder on your machine.

With this is place, the final command for running the Docker container is:

On macOS:

```
docker run -ti -v <LOCAL_FOLDER>:/dfnWorks/work dfnworks:latest
```

3.2 Native build from github repository

This document contains instructions for setting up dfnWorks natively on your machine. To setup dfnWorks using Docker instead, see the next section.

3.2.1 Clone the dnfWorks repository

```
$ git clone https://github.com/lanl/dfnWorks.git
```

3.2.2 Fix paths in test directory

Fix the pathnames in files throughout pydfnworks. This can be done automatically by running the script `fix_paths.py`:

```
$ cd dfnWorks/pydfnworks/bin/
$ python fix_paths.py
```

3.2.3 Set the LagriT, PETSC, PFLOTRAN, Python, and FEHM paths

Before executing dfnWorks, the following paths must be set:

- dfnWorks_PATH: the dfnWorks repository folder
- PETSC_DIR and PETSC_ARCH: PETSC environmental variables
- PFLOTRAN_EXE: Path to PFLOTRAN executable
- PYTHON_EXE: Path to python executable
- LAGRIT_EXE: Path to LaGriT executable

```
$ vi dfnWorks/pydfnworks/pydfnworks/paths.py
```

For example:

```
os.environ['dfnWorks_PATH'] = '/home/username/dfnWorks/'
```

Alternatively, you can create a `.dfnworksrc` file in your home directory with the following format

```
{
    "dfnworks_PATH": "<your-home-directory>/src/dfnworks-main/",
    "PETSC_DIR": "<your-home-directory>/src/petsc",
    "PETSC_ARCH": "arch-darwin-c-debug",
    "PFLOTRAN_EXE": "<your-home-directory>/src/pflotran/src/pflotran/pflotran",
    "PYTHON_EXE": "<your-home-directory>/anaconda3/bin/python",
    "LAGRIT_EXE": "<your-home-directory>/bin/lagrit",
    "FEHM_EXE": "<your-home-directory>//src/xfehm_v3.3.1"
}
```

3.2.4 Setup the python package pydfnworks

Go up into the pydfnworks sub-directory:

```
$ cd dfnWorks/pydfnworks/
```

If the user has admin privelges:

```
$ python setup.py install
```

If the user DOES NOT have admin priveleges:

```
$ python setup.py install --user
```

3.2.5 Installation Requirements for Native Build

Tools that you will need to run the dfnWorks work flow are described in this section. VisIt and ParaView, which enable visualization of desired quantities on the DFNs, are optional, but at least one of them is highly recommended for visualization. CMake is also optional but allows faster IO processing using C++.

Operating Systems

dfnWorks currently runs on Macs and Unix machine running Ubuntu.

Python

pydfnworks uses Python 3. We recommend using the Anaconda 3 distribution of Python, available at https://www.continuum.io/. pydfnworks requires the following python modules: `numpy`, `h5py`, `scipy`, `matplotlib`, `multiprocessing`, `argparse`, `shutil`, `os`, `sys`, `networkx`, `subprocess`, `glob`, `networkx`, `fpdf`, and `re`.

LaGriT

The LaGriT meshing toolbox is used to create a high resolution computational mesh representation of the DFN in parallel. An algorithm for conforming Delaunay triangulation is implemented so that fracture intersections are coincident with triangle edges in the mesh and Voronoi control volumes are suitable for finite volume flow solvers such as FEHM and PFLOTRAN.

PFLOTRAN

PFLOTRAN is a massively parallel subsurface flow and reactive transport code. PFLOTRAN solves a system of partial differential equations for multiphase, multicomponent and multi-scale reactive flow and transport in porous media. The code is designed to run on leadership-class supercomputers as well as workstations and laptops.

FEHM

FEHM is a subsurface multiphase flow code developed at Los Alamos National Laboratory.

CMake

CMake is an open-source, cross-platform family of tools designed to build, test and package software. It is needed to use C++ for processing files at a bottleneck IO step of dfnWorks. Using C++ for this file processing optional but can greatly increase the speed of dfnWorks for large fracture networks. Details on how to use C++ for file processing are in the scripts section of this documentation.

Paraview

Paraview is a parallel, open-source visualisation software. PFLOTRAN can output in `.xmf` and `.vtk` format. These can be imported in Paraview for visualization. While not required for running dfnWorks, Paraview is very helpful for visualizing dfnWorks simulations.

Instructions for downloading and installing Paraview can be found at http://www.paraview.org/download/

CHAPTER

FOUR

EXAMPLES

This section contains a few examples of DFN generated using dfnWorks. All required input files for these examples are contained in the folder dfnWorks/examples/. The focus of this document is to provide visual confirmation that new users of dfnWorks have the code set up correctly, can carry out the following runs and reproduce the following images. All images are rendered using Paraview, which can be obtained for free at http : //www.paraview.org/. The first two examples are simplest so it is recommended that the user proceed in the order presented here.

All examples are in the examples/ directory. Within each subdirectory are the files required to run the example. The command line input is found in notes.txt. Be sure that you have created ~/test_output_files prior to running the examples.

4.1 4_user_defined_rects

Location: examples/4_user_defined_rects/

This test case consists of four user defined rectangular fractures within a a cubic domain with sides of length one meter. The network of four fractures, each colored by material ID. The computational mesh is overlaid on the fractures. This image is created by loading the file full_mesh.inp. located in the job folder into Paraview.

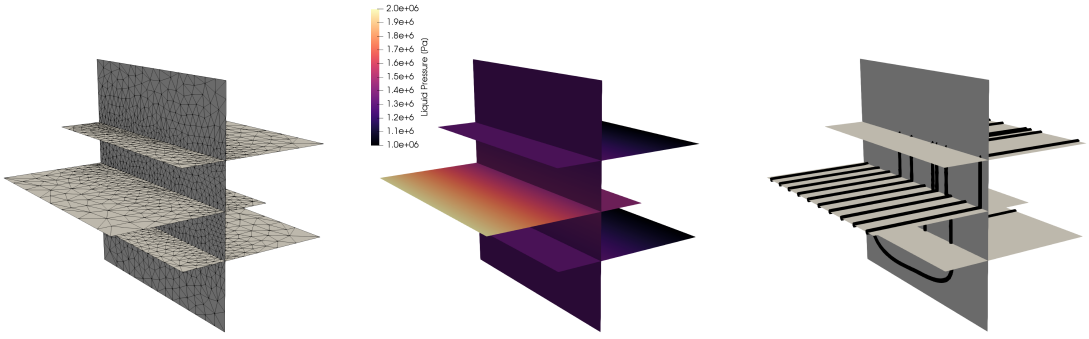


Fig. 1: *The meshed network of four rectangular fractures.*

High pressure (red) Dirichlet boundary conditions are applied on the edge of the single fracture along the boundary x = -0.5, and low pressure (blue) boundary conditions are applied on the edges of the two fractures at the boundary x = 0.5. This image is created by loading the file parsed_vtk/dfn_explicit-001.vtk into Paraview.

Particles are inserted uniformly along the inlet fracture on the left side of the image. Particles exit the domain through the two horizontal fractures on the right side of the image. Due to the stochastic nature of the particle tracking algorithm, your pathlines might not be exactly the same as in this image. Trajectories are colored by the current

velocity magnitude of the particle's velocity. Trajectories can be visualized by loading the files part_*.inp, in the folder 4_user_rectangles/traj/trajectories/

We have used the extract surface and tube filters in paraview for visual clarity.

4.2 4_user_defined_ell_uniform

Location: examples/4_user_defined_ell_uniform/

This test case consists of four user defined elliptical fractures within a a cubic domain with sides of length one meter. In this case the ellipses are approximated using 8 vertices. We have set the meshing resolution to be uniform by including the argument slope=0 into the mesh_networks function in run_explicit.py.

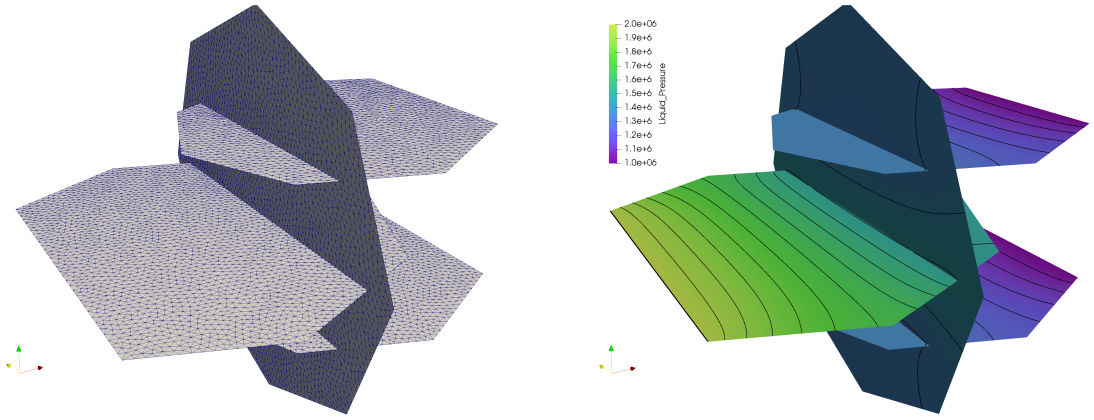


Fig. 2: *The uniformly meshed network of four circular fractures.*

4.3 exp: Exponentially Distributed fracture lengths

Location: examples/exp/

This test case consists of a family of fractures whose size is exponentially distributed with a minimum size of 1m and a maximum size of 50m. The domain is cubic with an edge length of 10m. All input parameters for the generator can be found in tests/gen_exponential_dist.dat. We have changed the flow direction to be aligned with the y-axis by modifying the PFLOTRAN input card dfn_explicit.in

4.4 TPL: Truncated Power-Law

Location: examples/TPL/

This test case consists of two families whose sizes have a truncated power law distribution with a minimum size of 1m and a maximum size of 5m an exponent 2.6. The domain size is cubic with an edge length of 15m.

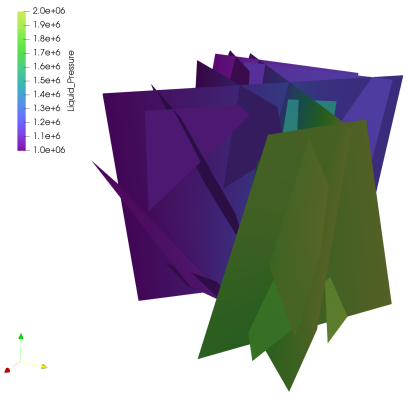
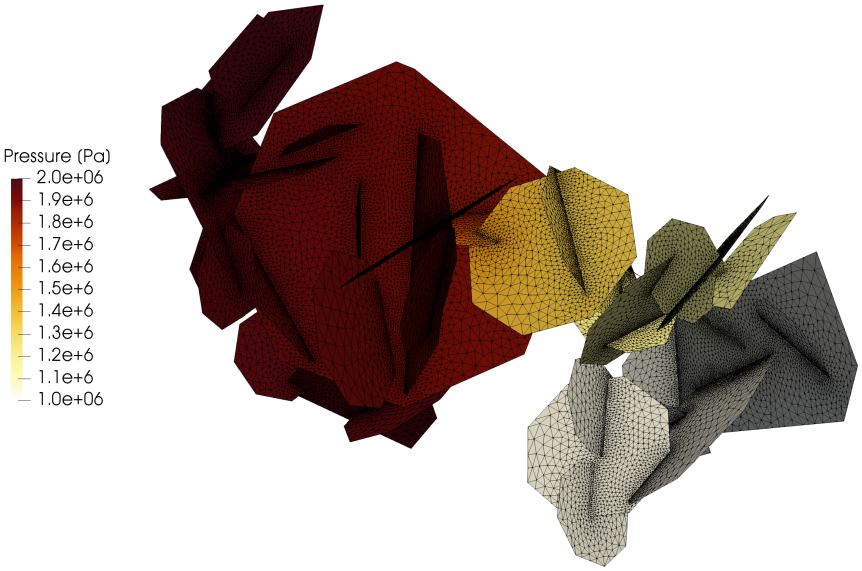


Fig. 3: *Pressure solution on with rectangular fractures whose lengths following a exponential distribution. Gradient is aligned with the Y-Axis*



4.5 Graph-based pruning

Location: examples/pruning/

This example uses a graph representation of a DFN to isolate the 2-core. The pruned DFN has all dead end fractures of the network are removed. This example has two run_explicit.py scripts. The first creates the original DFN and identifies the 2-core using networkx (https://networkx.github.io/). The second meshes the DFN corresponding to the 2-core of the graph and then runs flow and transport. The 2 core network is in a sub-directory 2-core. The original network has 207 fractures and the 2-core has 79 fractures.

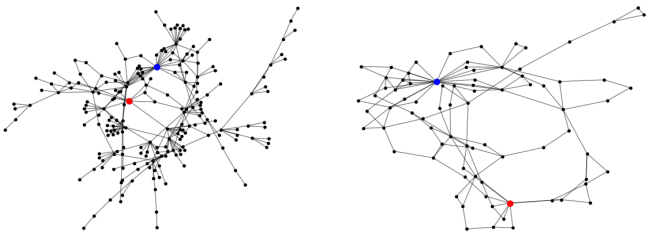


Fig. 4: *(left) Graph based on DFN topology. Each vertex is a fracture in the network. The inflow boundary is colored blue and the outflow is colored red. (right) 2-Core of the graph to the left.*

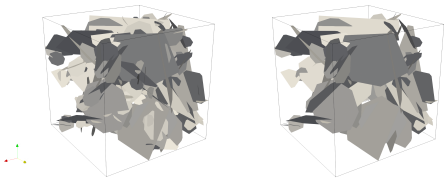


Fig. 5: *(left) Original DFN (right) DFN corresponding to the 2-core of the DFN to the left.*

CHAPTER

FIVE

PYDFNWORKS: THE DFNWORKS PYTHON PACKAGE

The pydfnworks package allows the user to run dfnWorks from the command line and call dfnWorks within other python scripts. Because pydfnworks is a package, users can call individual methods from the package.

The pydfnworks must be setup by the user using the following command in the directory `dfnWorks/pydfnworks/` :

`python setup.py install` (if the user has admin privileges), OR:

`python setup.py install --user` (if the user does not have admin privileges):

The documentation below includes basic information about the DFN class and pydfnworks setup.

5.1 pydfnWorks : Modules

Information about the various pieces of pydfnworks is found in

pydfnGen - Network generation, meshing, and analysis

pydfnFlow - Flow simulations using PFLOTRAN and FEHM

pydfnTrans - Particle Tracking

pydfnGraph - Graph-based analysis and pipe-network simulations

Well-Package - Well simulations

5.1.1 DFN Class and setup

`pydfnworks.general.dfnworks.`**`create_dfn`**`()`

Parse command line inputs and input files to create and populate dfnworks class

Parameters **`None`** –

Returns **DFN** – DFN class object populated with information parsed from the command line. Information about DFN class is in dfnworks.py

Return type object

Notes

None

5.1.2 General Work-flow functions

5.1.3 Set up run paths

pydfnworks.general.paths.**define_paths**()

Defines environmental variables for use in dfnWorks. The user must change these to match their workspace. :param None:

Returns

Return type None

Notes

Environmental variables are set to executables

5.1.4 Helper functions

pydfnworks.general.general_functions.**dump_time**(*self*, *function_name*, *time*)

Write run time for a funcion to the jobname_run_time.txt file

Parameters

- **self** (*object*) – DFN Class
- **function_name** (*string*) – Name of function that was timed
- **time** (*float*) – Run time of function in seconds

Returns

Return type None

Notes

While this function is working, the current formulation is not robust through the entire workflow

pydfnworks.general.general_functions.**print_run_time**(*self*)

Read in run times from file and and print to screen with percentages

Parameters **self** (*object*) – DFN Class

Returns

Return type None

Notes

This will dump out all values in the run file, not just those from the most recent run

5.2 Detailed Doxygen Documentation

Doxygen

CHAPTER

SIX

PYDFNWORKS: DFNGEN

DFN Class functions used in network generation and meshing

6.1 dfnGen

6.1.1 Processing generator input

6.1.2 Running the generator

pydfnworks.dfnGen.generation.generator.**create_network**(*self*)

Execute dfnGen

Parameters **self** (*object*) – DFN Class

Returns

Return type None

Notes

After generation is complete, this script checks whether the generation of the fracture network failed or succeeded based on the existence of the file params.txt.

pydfnworks.dfnGen.generation.generator.**dfn_gen**(*self*, *output=True*, *visual_mode=None*)

Wrapper script the runs the dfnGen workflow:

1) make_working_directory: Create a directory with name of job
2) check_input: Check input parameters and create a clean version of the input file
3) create_network: Create network. DFNGEN v2.0 is called and creates the network
4) output_report: Generate a PDF summary of the DFN generation
5) mesh_network: calls module dfnGen_meshing and runs LaGriT to mesh the DFN

Parameters

- **self** (*object*) – DFN Class object
- **output** (*bool*) – If True, output pdf will be created. If False, no pdf is made
- **visual_mode** (*None*) – If the user wants to run in a different meshing mode from what is in params.txt, set visual_mode = True/False on command line to override meshing mode

Returns

Return type None

Notes

Details of each portion of the routine are in those sections

pydfnworks.dfnGen.generation.generator.**make_working_directory**(*self*, *delete=False*)

Make working directory for dfnWorks Simulation

Parameters **self** (*object*) – DFN Class object

Returns

Return type None

Notes

If directory already exists, user is prompted if they want to overwrite and proceed. If not, program exits.

6.1.3 Analysis of generated DFN

filename gen_output.py

synopsis Main driver for dfnGen output report

version 1.0

maintainer Jeffrey Hyman

moduleauthor Jeffrey Hyman <jhyman@lanl.gov>

pydfnworks.dfnGen.generation.output_report.gen_output.**output_report**(*self*, *verbose=True*, *output_dir='dfnGen_output_report'*)

Creates a PDF output report for the network created by DFNGen. Plots of the fracture lengths, locations, orientations are produced for each family. Files are written into "output_dir/family_{id}/". Information about the whole network are also created and written into "output_dir/network/"

Parameters

- **self** (*object*) – DFN Class object
- **verbose** (*bool*) – Toggle for the amount of information printed to screen. If true, progress information printed to screen
- **output_dir** (*string*) – Name of directory where all plots are saved

Returns

Return type None

Notes

Final output report is named "jobname"_output_report.pdf User defined fractures (ellipses, rectangles, and polygons) are not supported at this time.

6.1.4 Modification of hydraulic properties of the DFN

pydfnworks.dfnGen.generation.hydraulic_properties.**dump_hydraulic_values**(*self*, *b*, *perm*, *T*, *prefix=None*)

Writes variable information to files.

Parameters

- **prefix** (*string*) – prefix of aperture.dat and perm.dat file names prefix_aperture.dat and prefix_perm.dat
- **b** (*array*) – aperture values
- **perm** (*array*) – permeability values
- **T** (*array*) – transmissivity values

Returns

Return type None

Notes

pydfnworks.dfnGen.generation.hydraulic_properties.**generate_hydraulic_values**(*self*, *variable*, *relationship*, *params*, *radii_filename='radii*, *family_id=None*)

Generates hydraulic property values.

Parameters

- **self** (*object*) – DFN Class
- **relationship** (*string*) – name of functional relationship for apertures. options are log-normal, correlated, semi-correlated, and constant
- **params** (*dictionary*) – dictionary of parameters for functional relationship
- **family_id** (*int*) – family id of fractures

Returns

- **b** (*array*) – aperture values

- **perm** (*array*) – permeability values
- **T** (*array*) – transmissivity values
- **idx** (*array of bool*) – true / false of fracture families requested. If family_id = None, all entires are true. Only family members entires of b, perm, and T will be non-zero

Notes

See Hyman et al. 2016 "Fracture size and transmissivity correlations: Implications for transport simulations in sparse three-dimensional discrete fracture networks following a truncated power law distribution of fracture size" Water Resources Research for more details

6.2 Meshing - LaGriT

6.2.1 Primary DFN meshing driver

pydfnworks.dfnGen.meshing.mesh_dfn.**mesh_network**(*self*, *prune=False*, *uniform_mesh=False*, *production_mode=True*, *coarse_factor=8*, *slope=0.1*, *min_dist=1*, *max_dist=40*, *concurrent_samples=10*, *grid_size=10*, *visual_mode=None*, *well_flag=False*)

Mesh fracture network using LaGriT

Parameters

- **self** (*object*) – DFN Class
- **prune** (*bool*) – If prune is False, mesh entire network. If prune is True, mesh only fractures in self.prune_file
- **uniform_mesh** (*bool*) – If true, mesh is uniform resolution. If False, mesh is spatially variable
- **production_mode** (*bool*) – If True, all working files while meshing are cleaned up. If False, then working files will not be deleted
- **visual_mode** (*None*) – If the user wants to run in a different meshing mode from what is in params.txt, set visual_mode = True/False on command line to override meshing mode
- **coarse_factor** (*float*) – Maximum resolution of the mesh. Given as a factor of h
- **slope** (*float*) – slope of variable coarsening resolution.
- **min_dist** (*float*) – Range of constant min-distance around an intersection (in units of h).
- **max_dist** (*float*) – Range over which the min-distance between nodes increases (in units of h)
- **concurrent_samples** (*int*) – number of new candidates sampled around an accepted node at a time.
- **grid_size** (*float*) – side length of the occupancy grid is given by H/occupancy_factor
- **well_flag** (*bool*) – If well flag is true, higher resolution around the points in

Returns

Return type None

Notes

1. For uniform resolution mesh, set slope = 0
2. All fractures in self.prune_file must intersect at least 1 other fracture

6.2.2 Meshing helper methods

pydfnworks.dfnGen.meshing.mesh_dfn_helper.**create_mesh_links**(*self*, *path*)
Makes symlinks for files in path required for meshing

Parameters

- **self** (*DFN object*) –
- **path** (*string*) – Path to where meshing files are located

Returns

Return type None

Notes

None

pydfnworks.dfnGen.meshing.mesh_dfn_helper.**inp2gmv**(*self*, *inp_file=''*)
Convert inp file to gmv file, for general mesh viewer. Name of output file for base.inp is base.gmv

Parameters

- **self** (*object*) – DFN Class
- **inp_file** (*str*) – Name of inp file if not an attribure of self

Returns

Return type None

Notes

pydfnworks.dfnGen.meshing.mesh_dfn_helper.**inp2vtk_python**(*self*)
Using Python VTK library, convert inp file to VTK file.

Parameters **self** (*object*) – DFN Class

Returns

Return type None

Notes

For a mesh base.inp, this dumps a VTK file named base.vtk

pydfnworks.dfnGen.meshing.mesh_dfn_helper.**run_lagrit_script**(*lagrit_file*, *output_file=None*, *quiet=False*)

Runs LaGriT

Parameters **----------** –

lagrit_file [string] Name of LaGriT script to run

output_file [string] Name of file to dump LaGriT output

quiet [bool] If false, information will be printed to screen.

Returns **failure** – If the run was successful, then 0 is returned.

Return type int

pydfnworks.dfnGen.meshing.add_attribute_to_mesh.**add_variable_to_mesh**(*self*, *variable*, *variable_file*, *mesh_file_in*, *mesh_file_out=None*, *cell_based=None*)

Adds a variable to the nodes of a mesh. Can be either fracture (material) based or node based.

Parameters

- **self** (*object*) – DFN Class
- **variable** (*string*) – name of variable
- **variable_file** (*string*) – name of file containing variable files. Must be a single column where each line corresponds to that node number in the mesh
- **mesh_file_in** (*string*) – Name of source mesh file
- **mesh_file_out** (*string*) – Name of Target mesh file. If no name if provide, mesh_file_in will be used
- **cell_based** (*bool*) – Set to True if variable_file contains cell-based values, Set to False if variable_file provide fracture based values

Returns **lagrit_file** – Name of LaGriT output file

Return type string

6.3 UDFM

6.3.1 Creating an upscaled mesh of the DFN

pydfnworks.dfnGen.meshing.udfm.map2continuum.**map_to_continuum**(*self*, *l*, *orl*, *path='./'*, *dir_name='octree'*)

This function generates an octree-refined continuum mesh using the reduced_mesh.inp as input. To generate the reduced_mesh.inp, one must turn visualization mode on in the DFN input card.

Parameters

- **self** (*object*) – DFN Class
- **l** (*float*) – Size (m) of level-0 mesh element in the continuum mesh
- **orl** (*int*) – Number of total refinement levels in the octree
- **path** (*string*) – path to primary DFN directory
- **dir_name** (*string*) – name of directory where the octree mesh is created

Returns

Return type None

Notes

octree_dfn.inp [Mesh file] Octree-refined continuum mesh

fracX.inp [Mesh files] Octree-refined continuum meshes, which contain intersection areas

pydfnworks.dfnGen.meshing.udfm.upscale.**upscale**(*self*, *mat_perm*, *mat_por*, *path='../'*)

Generate permeabilities and porosities based on output of map2continuum.

Parameters

- **self** (*object*) – DFN Class
- **mat_perm** (*float*) – Matrix permeability (in m^2)
- **mat_por** (*float*) – Matrix porosity

Returns

- **perm_fehm.dat** (*text file*) – Contains permeability data for FEHM input
- **rock_fehm.dat** (*text file*) – Contains rock properties data for FEHM input
- **mesh_permeability.h5** (*h5 file*) – Contains permeabilites at each node for PFLOTRAN input
- **mesh_porosity.h5** (*h5 file*) – Contains porosities at each node for PFLOTRAN input

Notes

None

pydfnworks.dfnGen.meshing.udfm.false_connections.**check_false_connections**(*self*, *path='../'*)

Parameters

- **self** (*object*) – DFN Class
- **fmc_filname** (*string*) – name of the pickled dictionary of mesh and fracture intersections

Returns

- **num_false_connections** (*int*) – number of false connections
- **num_cell_false** (*int*) – number of Voronoi cells with false connections
- **false_connections** (*list*) – list of tuples of false connections created by upscaling

Notes

map2continuum and upscale must be run first to create the fracture/mesh intersection dictionary. Thus must be run in the main job directory which contains connectivity.dat

CHAPTER

SEVEN

PYDFNWORKS: DFNFLOW

DFN Class functions used in flow simulations (PFLOTRAN and FEHM)

7.1 Running Flow : General

pydfnworks.dfnFlow.flow.**create_dfn_flow_links**(*self*, *path='../'*)

Create symlinks to files required to run dfnFlow that are in another directory.

Parameters

- **self** (*object*) – DFN Class
- **path** (*string*) – Absolute path to primary directory.

Returns

Return type None

Notes

1. Typically, the path is DFN.path, which is set by the command line argument -path
2. Currently only supported for PFLOTRAN

pydfnworks.dfnFlow.flow.**dfn_flow**(*self*, *dump_vtk=True*, *effective_perm=True*)

Run the dfnFlow portion of the workflow

Parameters

- **self** (*object*) – DFN Class
- **dump_vtk** (*bool*) – True - Write out vtk files for flow solutions False - Does not write out vtk files for flow solutions

Notes

Information on individual functions is found therein

pydfnworks.dfnFlow.flow.**set_flow_solver**(*self*, *flow_solver*)
Sets flow solver to be used

Parameters

- **self** (*object*) – DFN Class
- **flow_solver** (*string*) – Name of flow solver. Currently supported flow sovlers are FEHM and PFLOTRAN

Notes

Default is PFLOTRAN

7.2 Running Flow: PFLOTRAN

functions for using pflotran in dfnworks

pydfnworks.dfnFlow.pflotran.**lagrit2pflotran**(*self*, *inp_file=''*, *mesh_type=''*, *hex2tet=False*)
Takes output from LaGriT and processes it for use in PFLOTRAN. Calls the functuon write_perms_and_correct_volumes_areas() and zone2ex

Parameters

- **self** (*object*) – DFN Class
- **inp_file** (*str*) – Name of the inp (AVS) file produced by LaGriT
- **mesh_type** (*str*) – The type of mesh
- **hex2tet** (*bool*) – True if hex mesh elements should be converted to tet elements, False otherwise.

Returns

Return type None

Notes

None

pydfnworks.dfnFlow.pflotran.**parse_pflotran_vtk_python**(*self*, *grid_vtk_file=''*)
Adds CELL_DATA to POINT_DATA in the VTK output from PFLOTRAN. :param self: DFN Class :type self: object :param grid_vtk_file: Name of vtk file with mesh. Typically local_dfnFlow_file.vtk :type grid_vtk_file: string

Returns

Return type None

Notes

If DFN class does not have a vtk file, inp2vtk_python is called

pydfnworks.dfnFlow.pflotran.**pflotran**(*self*, *transient=False*, *restart=False*, *restart_file=''*)

Run PFLOTRAN. Copy PFLOTRAN run file into working directory and run with ncpus

Parameters

- **self** (*object*) – DFN Class
- **transient** (*bool*) – Boolean if PFLOTRAN is running in transient mode
- **restart** (*bool*) – Boolean if PFLOTRAN is restarting from checkpoint
- **restart_file** (*string*) – Filename of restart file

Returns

Return type None

Notes

Runs PFLOTRAN Executable, see http://www.pflotran.org/ for details on PFLOTRAN input cards

pydfnworks.dfnFlow.pflotran.**pflotran_cleanup**(*self*, *index_start=0*, *index_finish=1*, *filename=''*)

Concatenate PFLOTRAN output files and then delete them

Parameters

- **self** (*object*) – DFN Class
- **index** (*int*) – If PFLOTRAN has multiple dumps use this to pick which dump is put into cellinfo.dat and darcyvel.dat

Returns

Return type None

Notes

Can be run in a loop over all pflotran dumps

pydfnworks.dfnFlow.pflotran.**write_perms_and_correct_volumes_areas**(*self*)

Write permeability values to perm_file, write aperture values to aper_file, and correct volume areas in uge_file

Parameters **self** (*object*) – DFN Class

Returns

Return type None

Notes

Calls executable correct_uge

pydfnworks.dfnFlow.pflotran.**zone2ex**(*self*, *uge_file=''*, *zone_file=''*, *face=''*, *boundary_cell_area=0.1*)

Convert zone files from LaGriT into ex format for LaGriT

Parameters

- **self** (*object*) – DFN Class
- **uge_file** (*string*) – Name of uge file
- **zone_file** (*string*) – Name of zone file
- **Face** (*Face of the plane corresponding to the zone file*) –
- **zone_file** – Name of zone file to work on. Can be 'all' processes all directions, top, bottom, left, right, front, back
- **boundary_cell_area** (*double*) – should be a large value relative to the mesh size to force pressure boundary conditions.

Returns

Return type None

Notes

the boundary_cell_area should be a function of h, the mesh resolution

7.3 Running Flow: FEHM

pydfnworks.dfnFlow.fehm.**correct_stor_file**(*self*)

Corrects volumes in stor file to account for apertures

Parameters **self** (*object*) – DFN Class

Returns

Return type None

Notes

Currently does not work with cell based aperture

pydfnworks.dfnFlow.fehm.**fehm**(*self*)

Run FEHM

Parameters **self** (*object*) – DFN Class

Returns

Return type None

Notes

See https://fehm.lanl.gov/ for details about FEHM

7.4 Processing Flow

pydfnworks.dfnFlow.mass_balance.**effective_perm**(*self*)

Computes the effective permeability of a DFN in the primary direction of flow using a steady-state PFLOTRAN solution.

Parameters **self** (*object*) – DFN Class

Returns

Return type None

Notes

1. Information is written to screen and to the file self.local_jobname_effective_perm.txt
2. Currently, only PFLOTRAN solutions are supported
3. Assumes density of water

CHAPTER

EIGHT

PYDFNWORKS: DFNTRANS

DFN Class functions used in particle transport simulations (DFNTrans)

8.1 Running Transport Simulations

pydfnworks.dfnTrans.transport.**check_dfn_trans_run_files**(*self*)
Ensures that all files required for dfnTrans run are in the current directory

Parameters **self** (*object*) – DFN Class

Returns

Return type None

Notes

None

pydfnworks.dfnTrans.transport.**copy_dfn_trans_files**(*self*)
Creates symlink to dfnTrans Execuateble and copies input files for dfnTrans into working directory

Parameters **self** (*object*) – DFN Class

Returns

Return type None

pydfnworks.dfnTrans.transport.**create_dfn_trans_links**(*self*, *path='../'*)
Create symlinks to files required to run dfnTrans that are in another directory.

Parameters

- **self** (*object*) – DFN Class
- **path** (*string*) – Absolute path to primary directory.

Returns

Return type None

Notes

Typically, the path is DFN.path, which is set by the command line argument -path

pydfnworks.dfnTrans.transport.**dfn_trans**(*self*)
Primary driver for dfnTrans.

Parameters **self** (*object*) – DFN Class

Returns

Return type None

pydfnworks.dfnTrans.transport.**run_dfn_trans**(*self*)
Execute dfnTrans

Parameters **self** (*object*) – DFN Class

Returns

Return type None

CHAPTER

NINE

PYDFNWORKS: DFNGRAPH

DFN Class functions used in graph analysis and pipe-network simulations

9.1 General Graph Functions

pydfnworks.dfnGraph.dfn2graph.**add_fracture_source**(*self*, *G*, *source*)

Returns the k shortest paths in a graph

Parameters

- **G** (*NetworkX Graph*) – NetworkX Graph based on a DFN
- **source_list** (*list*) – list of integers corresponding to fracture numbers
- **remove_old_source** (*bool*) – remove old source from the graph

Returns G

Return type NetworkX Graph

Notes

bipartite graph not supported

pydfnworks.dfnGraph.dfn2graph.**add_fracture_target**(*self*, *G*, *target*)

Returns the k shortest paths in a graph

Parameters

- **G** (*NetworkX Graph*) – NetworkX Graph based on a DFN
- **target** (*list*) – list of integers corresponding to fracture numbers

Returns G

Return type NetworkX Graph

Notes

bipartite graph not supported

pydfnworks.dfnGraph.dfn2graph.**create_graph**(*self*, *graph_type*, *inflow*, *outflow*)
Header function to create a graph based on a DFN

Parameters

- **self** (*object*) – DFN Class object
- **graph_type** (*string*) – Option for what graph representation of the DFN is requested. Currently supported are fracture, intersection, and bipartitie
- **inflow** (*string*) – Name of inflow boundary (connect to source)
- **outflow** (*string*) – Name of outflow boundary (connect to target)

Returns **G** – Graph based on DFN

Return type NetworkX Graph

Notes

pydfnworks.dfnGraph.dfn2graph.**dump_fractures**(*self*, *G*, *filename*)
Write fracture numbers assocaited with the graph G out into an ASCII file inputs

Parameters

- **self** (*object*) – DFN Class
- **G** (*NetworkX graph*) – NetworkX Graph based on the DFN
- **filename** (*string*) – Output filename

Notes

pydfnworks.dfnGraph.dfn2graph.**dump_json_graph**(*self*, *G*, *name*)
Write graph out in json format

Parameters

- **self** (*object*) – DFN Class
- **G** (*networkX graph*) – NetworkX Graph based on the DFN
- **name** (*string*) – Name of output file (no .json)

Notes

pydfnworks.dfnGraph.dfn2graph.**greedy_edge_disjoint**(*self*, *G*, *source='s'*, *target='t'*, *weight='None'*, *k=''*)
Greedy Algorithm to find edge disjoint subgraph from s to t. See Hyman et al. 2018 SIAM MMS

Parameters

- **self** (*object*) – DFN Class Object
- **G** (*NetworkX graph*) – NetworkX Graph based on the DFN
- **source** (*node*) – Starting node

- **target** (*node*) – Ending node
- **weight** (*string*) – Edge weight used for finding the shortest path
- **k** (*int*) – Number of edge disjoint paths requested

Returns **H** – Subgraph of G made up of the k shortest of all edge-disjoint paths from source to target

Return type NetworkX Graph

Notes

1. Edge weights must be numerical and non-negative.
2. See Hyman et al. 2018 "Identifying Backbones in Three-Dimensional Discrete Fracture Networks: A Bipartite Graph-Based Approach" SIAM Multiscale Modeling and Simulation for more details

pydfnworks.dfnGraph.dfn2graph.**k_shortest_paths_backbone**(*self*, *G*, *k*, *source='s'*, *target='t'*, *weight=None*)

Returns the subgraph made up of the k shortest paths in a graph

Parameters

- **G** (*NetworkX Graph*) – NetworkX Graph based on a DFN
- **k** (*int*) – Number of requested paths
- **source** (*node*) – Starting node
- **target** (*node*) – Ending node
- **weight** (*string*) – Edge weight used for finding the shortest path

Returns **H** – Subgraph of G made up of the k shortest paths

Return type NetworkX Graph

Notes

See Hyman et al. 2017 "Predictions of first passage times in sparse discrete fracture networks using graph-based reductions" Physical Review E for more details

pydfnworks.dfnGraph.dfn2graph.**load_json_graph**(*self*, *name*)

Read in graph from json format

Parameters

- **self** (*object*) – DFN Class
- **name** (*string*) – Name of input file (no .json)

Returns **G** – NetworkX Graph based on the DFN

Return type networkX graph

pydfnworks.dfnGraph.dfn2graph.**plot_graph**(*self*, *G*, *source='s'*, *target='t'*, *output_name='dfn_graph'*)

Create a png of a graph with source nodes colored blue, target red, and all over nodes black

Parameters

- **G** (*NetworkX graph*) – NetworkX Graph based on the DFN
- **source** (*node*) – Starting node

- **target** (*node*) – Ending node
- **output_name** (*string*) – Name of output file (no .png)

Notes

Image is written to output_name.png

9.2 Graph-Based Flow and Transport

pydfnworks.dfnGraph.graph_flow.**run_graph_flow**(*self*, *inflow*, *outflow*, *Pin*, *Pout*, *fluid_viscosity=0.00089*, *G=None*)

Run the graph flow portion of the workflow

Parameters **self** – DFN Class

Returns **Gtilde** – Grtilde is updated with vertex pressures, edge fluxes and travel times

Return type NetworkX graph

Notes

Information on individual functions in found therein

pydfnworks.dfnGraph.graph_transport.**run_graph_transport**(*self*, *Gtilde*, *nparticles*, *partime_file*, *frac_id_file*, *frac_porosity=1.0*, *tdrw_flag=False*, *matrix_porosity=0.02*, *matrix_diffusivity=1e-11*)

Run particle tracking on the given NetworkX graph

Parameters

- **self** (*object*) – DFN Class
- **Gtilde** (*NetworkX graph*) – obtained from graph_flow
- **nparticles** (*int*) – number of particles
- **partime_file** (*string*) – name of file to which the total travel times and lengths will be written for each particle
- **frac_id_file** (*string*) – name of file to which detailed information of each particle's travel will be written
- **frac_porosity** (*float*) – porosity of fracture, default is 1.0
- **tdrw_flag** (*Bool*) – if False, matrix_porosity, matrix_diffusivity are ignored
- **matrix_porosity** (*float*) – default is 0.02
- **matrix_diffusivity** (*float*) – default is 1e-11 in SI units

Notes

Information on individual functions is found therein

CHAPTER

TEN

PYDFNWORKS: WELL PACKAGE

DFN Class functions used for well package

10.1 dfnWorks - Well Package

pydfnworks.dfnGen.well_package.wells.**cleanup_wells**(*self*, *wells*)

Moves working files created while making wells into well_data directory

Parameters

- **self** (*object*) – DFN Class
- **well** – dictionary of information about the well. Contains the following:

  well["name"] [string] name of the well

  well["filename"] [string]

  filename of the well coordinates. "well_coords.dat" for example. Format is : x0 y0 z0 x1 y1 z1 . . . xn yn zn

  well["r"] [float] radius of the well

Returns

Return type None

Notes

Wells can be a list of well dictionaries

pydfnworks.dfnGen.well_package.wells.**combine_well_boundary_zones**(*self*, *wells*)

Processes zone files for particle tracking. All zone files are combined into allboundaries.zone

Parameters **None** –

Returns

Return type None

Notes

None

pydfnworks.dfnGen.well_package.wells.**find_well_intersection_points**(*self*, *wells*)

Identifies points on a DFN where the well intersects the network. These points are used in meshing the network to have higher resolution in the mesh in these points. Calls a sub-routine run_find_well_intersection_points.

Parameters

- **self** (*object*) – DFN Class
- **well** – dictionary of information about the well. Contains the following:

  well["name"] [string] name of the well

  well["filename"] [string]

  filename of the well coordinates. "well_coords.dat" for example. Format is : x0 y0 z0 x1 y1 z1 ... xn yn zn

  well["r"] [float] radius of the well

Returns

Return type None

Notes

Wells can be a list of well dictionaries. Calls the subroutine run_find_well_intersection_points to remove redundant code.

pydfnworks.dfnGen.well_package.wells.**tag_well_in_mesh**(*self*, *wells*)

Identifies nodes in a DFN for nodes the intersect a well with radius r [m]

1. Well coordinates in well["filename"] are converted to a polyline that are written into "well_{well['name']}_line.inp"
2. Well is expanded to a volume with radius well["r"] and written into the avs file well_{well["name"]}_volume.inp
3. Nodes in the DFN that intersect with the well are written into the zone file well_{well["name"]}.zone
4. If using PFLOTRAN, then an ex file is created from the well zone file

Parameters

- **self** (*object*) – DFN Class
- **well** – Dictionary of information about the well that contains the following attributes

  well["name"] [string] name of the well

  well["filename"] [string] filename of the well coordinates with the following format x0 y0 z0

  x1 y1 z1

  ...

  xn yn zn

Returns

Return type None

Notes

Wells can be a list of well dictionaries

CHAPTER

ELEVEN

DFNGEN

dfnGen creates the discrete fracture networks using the feature rejection algorithm for meshing (FRAM). Fractures can be created stochastically or as deterministic features.

The detailed description of FRAm and implemented methodology is in J. D. Hyman, C. W. Gable, S. L. Painter, and N. Makedonska. Conforming Delaunay triangulation of stochastically generated three dimensional discrete fracture networks: A feature rejection algorithm for meshing strategy. SIAM J. Sci. Comput., 36(4):A1871–A1894, 2014.

11.1 Documentation

Doxygen

CHAPTER

TWELVE

DFNFLOW

dfnFlow involves using flow solver such as PFLOTRAN or FEHM. PFLOTRAN is recommended if a large number of fractures (> O(1000)) are involved in a network. Using the function calls that are part of pydfnworks, one can create the mesh files needed to run PFLOTRAN. This will involve creating unstructured mesh file `*uge` as well as the boundary `*ex` files. Please see the PFLOTRAN user manual at http://www.pflotran.org under unstructured *explicit* format usage for further details. An example input file for PFLOTRAN is provided in the repository. Please use this as a starting point to build your input deck.

Below is a sample input file. Refer to the PFLOTRAN user manual at http://www.pflotran.org for input parameter descriptions.

```
# Jan 13, 2014
# Nataliia Makedonska, Satish Karra, LANL
#================================================

SIMULATION
  SIMULATION_TYPE SUBSURFACE
  PROCESS_MODELS
    SUBSURFACE_FLOW flow
      MODE RICHARDS
    /
  /
END
SUBSURFACE

DFN

#========================== discretization ===================================
GRID
  TYPE unstructured_explicit full_mesh_vol_area.uge
  GRAVITY 0.d0 0.d0 0.d0
END


#========================== fluid properties =================================
FLUID_PROPERTY
  DIFFUSION_COEFFICIENT 1.d-9
END

DATASET Permeability
  FILENAME dfn_properties.h5
END

#========================== material properties ==============================
MATERIAL_PROPERTY soil1
```

(continues on next page)

(continued from previous page)

```
  ID 1
  POROSITY 0.25d0
  TORTUOSITY 0.5d0
  CHARACTERISTIC_CURVES default
  PERMEABILITY
    DATASET Permeability
  /
END


#========================== characteristic curves ===========================
CHARACTERISTIC_CURVES default
  SATURATION_FUNCTION VAN_GENUCHTEN
    M 0.5d0
    ALPHA  1.d-4
    LIQUID_RESIDUAL_SATURATION 0.1d0
    MAX_CAPILLARY_PRESSURE 1.d8
  /
  PERMEABILITY_FUNCTION MUALEM_VG_LIQ
    M 0.5d0
    LIQUID_RESIDUAL_SATURATION 0.1d0
  /
END

#========================== output options ==================================
OUTPUT
  TIMES s 0.01 0.05 0.1 0.2 0.5 1
#  FORMAT TECPLOT BLOCK
  PRINT_PRIMAL_GRID
  FORMAT VTK
  MASS_FLOWRATE
  MASS_BALANCE
  VARIABLES
    LIQUID_PRESSURE
    PERMEABILITY
  /
END

#========================== times ===========================================
TIME
  INITIAL_TIMESTEP_SIZE  1.d-8 s
  FINAL_TIME 1.d0 d==
  MAXIMUM_TIMESTEP_SIZE 10.d0 d
  STEADY_STATE
END

# REFERENCE_PRESSURE 1500000.

#========================== regions =========================================
REGION All
  COORDINATES
    -1.d20 -1.d20 -1.d20
    1.d20 1.d20 1.d20
  /
END

REGION inflow
```

(continues on next page)

(continued from previous page)

```
  FILE pboundary_left_w.ex
END

REGION outflow
  FILE pboundary_right_e.ex
END

#========================== flow conditions ===================================
FLOW_CONDITION initial
  TYPE
     PRESSURE dirichlet
  /
  PRESSURE 1.01325d6
END


FLOW_CONDITION outflow
  TYPE
     PRESSURE dirichlet
  /
  PRESSURE 1.d6
END

FLOW_CONDITION inflow
  TYPE
    PRESSURE dirichlet
  /
  PRESSURE 2.d6
END

#========================== condition couplers ================================
# initial condition
INITIAL_CONDITION
  FLOW_CONDITION initial
  REGION All
END


BOUNDARY_CONDITION INFLOW
  FLOW_CONDITION inflow
  REGION inflow
END

BOUNDARY_CONDITION OUTFLOW
  FLOW_CONDITION outflow
  REGION outflow
END

#========================== stratigraphy couplers =============================
STRATA
  REGION All
  MATERIAL soil1
END

END_SUBSURFACE
```

CHAPTER

THIRTEEN

DFNTRANS

dfnTrans is a method for resolving solute transport using control volume flow solutions obtained from dfnFlow on the unstructured mesh generated using dfnGen. We adopt a Lagrangian approach and represent a non-reactive conservative solute as a collection of indivisible passive tracer particles. Particle tracking methods (a) provide a wealth of information about the local flow field, (b) do not suffer from numerical dispersion, which is inherent in the discretizations of advection–dispersion equations, and (c) allow for the computation of each particle trajectory to be performed in an intrinsically parallel fashion if particles are not allowed to interact with one another or the fracture network. However, particle tracking on a DFN poses unique challenges that arise from (a) the quality of the flow solution, (b) the unstructured mesh representation of the DFN, and (c) the physical phenomena of interest. The flow solutions obtained from dfnFlow are locally mass conserving, so the particle tracking method does not suffer from the problems inherent in using Galerkin finite element codes.

dfnTrans starts from reconstruction of local velocity field: Darcy fluxes obtained using dfnFlow are used to reconstruct the local velocity field, which is used for particle tracking on the DFN. Then, Lagrangian transport simulation is used to determine pathlines through the network and simulate transport. It is important to note that dfnTrans itself only solves for advective transport, but effects of longitudinal dispersion and matrix diffusion, sorption, and other retention processes are easily incorporated by post-processing particle trajectories.

The detailed description of dfnTrans algorithm and implemented methodology is in Makedonska, N., Painter, S. L., Bui, Q. M., Gable, C. W., & Karra, S. (2015). Particle tracking approach for transport in three-dimensional discrete fracture networks. Computational Geosciences, 19(5), 1123-1137.

13.1 Documentation

Doxygen

CHAPTER

FOURTEEN

OUTPUT FILES

dfnWorks outputs about a hundred different output files. This section describes the contents and purpose of each file.

14.1 dfnGen

aperture.dat:

connectivity.dat:

Fracture connection list. Each row corresponds to a single fracture. The integers in that row are the fractures that fracture intersects with. These are the non-zero elements of the adjacency matrix.

convert_uge_params.txt:

Input file do conver_uge executable.

DFN_output.txt:

Detailed information about fracture network. Output by DFNGen.

families.dat:

Information about fracture families. Produced by DFNGen.

input_generator.dat:

Input file for DFN generator.

input_generator_clean.dat:

Abbreviated input file for DFN generator.

normal_vectors.dat:

Normal vector of each fracture in the network.

params.txt:

Parameter information about the fracture network used for meshing. Includes number of fractures, h, visualmode, expected number of dudded points, and x,y,z dimensions of the domain.

poly_info.dat:

Fracture information output by DFNGen. Format: Fracture Number, Family number, rotation angle for rotateln in LaGriT, x0, y0, z0, x1, y1, z1 (end points of line of rotation).

user_rects.dat:

User defined rectangle file.

radii:

Subdirectory containing fracture radii information.

radii.dat:

Concatentate file of fracture radii. Contains fractures that are removed due to isolation.

radii_Final.dat:

Concatentated file of final radii in the DFN.

rejections.dat:

Summary of rejection reasons.

rejectsPerAttempt.dat:

Number of rejections per attempted fracture.

translations.dat:

Fracture centriods.

triple_points.dat:

x,y,z location of triple intersection points.

warningFileDFNGen.txt:

Warning file output by DFNGen.

intersection_list.dat:

List of intersections between fractures. Format is fracture1 fracture2 x y z length. Negative numbers correspond to intersections with boundaries.

14.2 LaGrit

bound_zones.lgi:

LaGriT run file to identify boundary nodes. Dumps zone files.

boundary_output.txt:

Output file from bound_zones.lgi.

finalmesh.txt:

Brief summary of final mesh.

full_mesh.inp:

Full DFN mesh in AVS format.

full_mesh.lg:

Full DFN mesh in LaGriT binary format.

full_mesh.uge:

Full DFN mesh in UGE format. NOTE volumes are not correct in this file. This file is processed by convert_uge to create full_mesh_vol_area.uge, which has the correct volumes.

full_mesh_viz.inp:

intersections:

Directory containing intersection avs files output by the generator and used by LaGrit.

lagrit_logs:

Directory of output files from individual meshing.

logx3dgen:

LaGriT output.

outx3dgen:

LaGriT output.

parameters:

Directory of parameter*.mgli files used for fracture meshing.

polys:

Subdirectory contiaining AVS file for polygon boundaries.

tri_fracture.stor:

FEHM stor file. Information about cell volume and area.

user_function.lgi:

Function used by LaGriT for meshing. Defines coarsening gradient.

14.3 PFLOTRAN

Fracture based aperture value for the DFN. Used to rescale volumes in full_mesh_vol_area.uge.

cellinfo.dat:

Mesh information output by PFLOTRAN.

dfn_explicit-000.vtk:

VTK file of initial conditions of PFLOTRAN. Mesh is not included in this file.

dfn_explicit-001.vtk:

VTK file of steady-state solution of PFLOTRAN. Mesh is not included in this file.

dfn_explicit-mas.dat:

pflotran information file.

dfn_explicit.in:

pflotran input file.

_dfn_explicit.out:

pflotran output file.

dfn_properties.h5:

h5 file of fracture network properties, permeability, used by pflotran.

Full DFN mesh with limited attributes in AVS format.

full_mesh_vol_area.uge:

Full DFN in uge format. Volumes and areas have been corrected.

materialid.dat:

Material ID (Fracture Number) for every node in the mesh.

parsed_vtk:

Directory of pflotran results.

perm.dat:

Fracture permeabilities in FEHM format. Each fracture is listed as a zone, starting index at 7.

pboundary_back_n.ex:

Boundary file for back of the domain used by PFLOTRAN.

pboundary_bottom.ex:

Boundary file for bottom of the domain used by PFLOTRAN.

pboundary_front_s.ex:

Boundary file for front of the domain used by PFLOTRAN.

pboundary_left_w.ex:

Boundary file for left side of the domain used by PFLOTRAN.

pboundary_right_e.ex:

Boundary file for right of the domain used by PFLOTRAN.

pboundary_top.ex:

Boundary file for top of the domain used by PFLOTRAN.

14.4 dfnTrans

allboundaries.zone:

Concatenated file of all zone files.

darcyvel.dat:

Concatenated file of darcy velocities output by PFLOTRAN.

dfnTrans_output_dir:

Outpur directory from DFNTrans. Particle travel times, trajectories, and reconstructed Velocities are in this directory.

PTDFN_control.dat:

Input file for DFNTrans.

pboundary_back_n.zone:

Boundary zone file for the back of the domain. Normal vector (0,1,0) +- pi/2

pboundary_bottom.zone:

Boundary zone file for the bottom of the domain. Normal vector (0,0,-1) +- pi/2

pboundary_front_s.zone:

Boundary zone file for the front of the domain. Normal vector (0,-1,0) +- pi/2

pboundary_left_w.zone:

Boundary zone file for the left side of the domain. Normal vector (-1,0,0) +- pi/2

pboundary_right_e.zone:

Boundary zone file for the bottom of the domain. Normal vector (1,0,0) +- pi/2

pboundary_top.zone:

Boundary zone file for the top of the domain. Normal vector (0,0,1) +- pi/2

CHAPTER

FIFTEEN

DFNWORKS PUBLICATIONS

The following are publications that use *dfnWorks*:

1. J. D. Hyman, C. W. Gable, S. L. Painter, and N. Makedonska. Conforming Delaunay triangulation of stochastically generated three dimensional discrete fracture networks: A feature rejection algorithm for meshing strategy. SIAM J. Sci. Comput., 36(4):A1871–A1894, 2014.
2. R.S. Middleton, J.W. Carey, R.P. Currier, J. D. Hyman, Q. Kang, S. Karra, J. Jimenez-Martınez, M.L. Porter, and H.S. Viswanathan. Shale gas and non-aqueous fracturing fluids: Opportunities and challenges for supercritical CO2. Applied Energy, 147:500–509, 2015.
3. J. D. Hyman, S. L. Painter, H. Viswanathan, N. Makedonska, and S. Karra. Influence of injection mode on transport properties in kilometer-scale three-dimensional discrete fracture networks. Water Resources Research, 51(9):7289–7308, 2015.
4. S. Karra, Nataliia Makedonska, Hari S Viswanathan, Scott L Painter, and Jeffrey D. Hyman. Effect of advective flow in fractures and matrix diffusion on natural gas production. Water Resources Research, 51(10):8646–8657, 2015.
5. J. D. Hyman, S. Karra, N. Makedonska, C. W Gable, S. L Painter, and H. S Viswanathan. dfnWorks: A discrete fracture network framework for modeling subsurface flow and transport. Computers & Geosciences, 84:10–19, 2015.
6. H. S. Viswanathan, J. D. Hyman, S. Karra, J.W. Carey, M. L. Porter, E. Rougier, R. P. Currier,Q. Kang, L. Zhou, J. Jimenez-Martınez, N. Makedonska, L. Chen, and R. S. Middleton. Using Discovery Science To Increase Efficiency of Hydraulic Fracturing While Reducing Water Usage, chapter 4, pages 71–88. ACS Publications, 2016.
7. N. Makedonska, S. L Painter, Q. M Bui, C. W Gable, and S. Karra. Particle tracking approach for transport in three-dimensional discrete fracture networks. Computational Geosciences, 19(5):1123–1137, 2015.
8. D. O'Malley, S. Karra, R. P. Currier, N. Makedonska, J. D. Hyman, and H. S. Viswanathan. Where does water go during hydraulic fracturing? Groundwater, 54(4):488–497, 2016.
9. J. D. Hyman, J Jiménez-Martínez, HS Viswanathan, JW Carey, ML Porter, E Rougier, S Karra, Q Kang, L Frash, L Chen, et al. Understanding hydraulic fracturing: a multi-scale problem. Phil. Trans. R. Soc. A, 374(2078):20150426, 2016.
10. G. Aldrich, J. D. Hyman, S. Karra, C. W. Gable, N. Makedonska, H. Viswanathan, J.Woodring, and B. Hamann. Analysis and visualization of discrete fracture networks using a flow topology graph. IEEE Transactions on Visualization and Computer Graphics, 23(8):1896–1909, Aug 2017.
11. N. Makedonska, J. D. Hyman, S. Karra, S. L. Painter, C.W. Gable, and H. S. Viswanathan. Evaluating the effect of internal aperture variability on transport in kilometer scale discrete fracture networks. Advances in Water Resources, 94:486 – 497, 2016.

12. J. D. Hyman, G. Aldrich, H. Viswanathan, N. Makedonska, and S. Karra. Fracture size and transmissivity correlations: Implications for transport simulations in sparse three-dimensional discrete fracture networks following a truncated power law distribution of fracture size. Water Resources Research, 2016.

13. H. Djidjev, D. O'Malley, H. Viswanathan, J. D. Hyman, S. Karra, and G. Srinivasan. Learning on graphs for predictions of fracture propagation, flow and transport. In 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pages 1532–1539, May 2017.

14. J. D. Hyman, A. Hagberg, G. Srinivasan, J. Mohd-Yusof, and H. Viswanathan. Predictions of first passage times in sparse discrete fracture networks using graph-based reductions. Phys. Rev. E, 96:013304, Jul.

15. T Hadgu, S. Karra, N. Makedonska, J. D. Hyman, K. Klise, H. S. Viswanathan, and Y.Wang. A comparative study of discrete fracture network and equivalent continuum models for simulating flow and transport in the far field of a hypothetical nuclear waste repository in crystalline host rock. J. Hydrology, 2017.

16. V. Romano, J. D. Hyman, S. Karra, A. J. Valocchi, M. Battaglia, and S. Bigi. Numerical modeling of fluid flow in a fault zone: a case of study from majella mountain (Italy). Energy Procedia, 125:556 – 560, 2017.

17. M. Valera, Z. Guo, P. Kelly, S. Matz, A. Cantu, A.G. Percus, J. D. Hyman, G. Srinivasan, and H.S. Viswanathan. Machine learning for graph-based representations of three-dimensional discrete fracture networks. Computational Geosciences, 2018.

18. M. K. Mudunuru, S. Karra, N. Makedonska, and T. Chen. Sequential geophysical and flow inversion to characterize fracture networks in subsurface systems. Statistical Analysis and Data Mining: The ASA Data Science Journal, 10(5):326–342, 2017.

19. J. D. Hyman, Satish Karra, J. William Carey, Carl W. Gable, Hari Viswanathan, Esteban Rougier, and Zhou Lei. Discontinuities in effective permeability due to fracture percolation. Mechanics of Materials, 119:25 – 33, 2018.

20. S. Karra, D. O'Malley, J. D. Hyman, H.S. Viswanathan, and G. Srinivasan. Modeling flow and transport in fracture networks using graphs. Phys. Rev. E, 2018.

21. J. D. Hyman and J. Jimenéz-Martínez. Dispersion and mixing in three-dimensional discrete fracture networks: Nonlinear interplay between structural and hydraulic heterogeneity. Water Resources Research, 54(5):3243–3258, 2018.

22. D. O'Malley, S. Karra, J. D. Hyman, H. Viswanathan, and G. Srinivasan. Efficient Monte Carlo with graph-based subsurface flow and transport models. Water Resour. Res., 2018.

23. G. Srinivasan, J. D. Hyman, D. Osthus, B. Moore, D. O'Malley, S. Karra, E Rougier, A. Hagberg, A. Hunter, and H. S. Viswanathan. Quantifying topological uncertainty in fractured systems using graph theory and machine learning. Scientific Reports, 2018.

24. H. S. Viswanathan, J. D. Hyman, S. Karra, D. O'Malley, S. Srinivasan, A. Hagberg, and G. Srinivasan. Advancing graph-based algorithms for predicting flow and transport in fractured rock. Water Resour. Res., 2018.

25. S. Srinivasan, J. D. Hyman, S. Karra, D. O'Malley, H. Viswanathan, and G. Srinivasan. Robust system size reduction of discrete fracture networks: A multi-fidelity method that preserves transport characteristics. Computational Geosciences, 2018.

26. J. D. Hyman, Aric Hagberg, Dave Osthus, Shriram Srinivasan, Hari Viswanathan, and Gowri Srinivasan. Identifying backbones in three-dimensional discrete fracture net- works: A bipartite graph-based approach. Multiscale Modeling & Simulation, 16(4):1948– 1968, 2018.

27. G. Aldrich, J. Lukasczyk, J. D. Hyman, G. Srinivasan, H. Viswanathan, C. Garth, H. Leitte, J. Ahrens, and B. Hamann. A query-based framework for searching, sorting, and exploring data ensembles. Computing in Science Engineering, 2018.

28. T. Sherman, J. D. Hyman, D. Bolster, N. Makedonska, and G. Srinivasan. Characterizing the impact of particle behavior at fracture intersections in three-dimensional discrete fracture networks. Physical Review E, 99(1):013110, 2019.

29. J. D. Hyman, M. Dentz, A. Hagberg, and P. Kang. Linking structural and transport properties in three-dimensional fracture networks. J. Geophys. Res. Sol. Ea., 2019.

30. S. Srinivasan, S. Karra, J. D. Hyman, H. Viswanathan, and G. Srinivasan. Model reduction for fractured porous media: A machine-learning approach for identifying main flow pathways. Computational Geosciences, 2018.

31. N. Makedonska, J.D, Hyman, E. Kwicklis, K. Birdsell, Conference Proceedings, Discrete Fracture Network Modeling and Simulation of Subsurface Transport for the Topopah Spring Aquifer at Pahute Mesa, 2nd International Discrete Fracture Network Engineering, 2018..

32. N. Makedonska, C.W. Gable, R. Pawar, Conference Proceedings, Merging Discrete Fracture Network Meshes With 3D Continuum Meshes of Rock Matrix: A Novel Approach, 2nd International Discrete Fracture Network Engineering, 2018..

33. A. Frampton, J.D, Hyman, L. Zou, Advective transport in discrete fracture networks with connected and disconnected textures representing internal aperture variability, Water Resources Research. 2019.

34. J.D. Hyman, J. Jiménez-Martínez, C. W. Gable, P. H. Stauffer, and R. J. Pawar. Characterizing the Impact of Fractured Caprock Heterogeneity on Supercritical CO2 Injection. Transport in Porous Media: 2019..

35. J.D. Hyman, H. Rajaram, S. Srinivasan, N. Makedonska, S. Karra, H. Viswanathan, H., & G. Srinivasan, (2019). Matrix diffusion in fractured media: New insights into power law scaling of breakthrough curves. Geophysical Research Letters, 46. 2019.

36. J.D. Hyman, M. Dentz, A. Hagberg, & P. K. Kang, (2019). Emergence of Stable Laws for First Passage Times in Three-Dimensional Random Fracture Networks. Physical Review Letters, 123. 248501.

37. M. R. Sweeney, C. W. Gable, S. Karra, P. H. Stauffer, R. J. Pawar, J. D. Hyman (2019). Upscaled discrete fracture matrix model (UDFM): an octree-refined continuum representation of fractured porous mediaComputational Geosciences 2019.

38. T. Sherman, J. D. Hyman, M. Dentz, and D. Bolster. Characterizing the influence of fracture density on network scale transport. J. Geophys. Res. Sol. Ea., 2019.

39. D. Osthus, J. D. Hyman, S. Karra, N. Panda, and G. Srinivasan. A probabilistic clustering approach for identifying primary subnetworks of discrete fracture networks with quantified uncertainty. SIAM/ASA Journal on Uncertainty Quantification, 2020 8(2), pp.573-600..

40. V. Romano, S. Bigi, F. Carnevale, J. D. Hyman, S. Karra, A. Valocchi, M. Tartarello, and M. Battaglia. Hydraulic characterization of a fault zone from fracture distribution. Journal of Structural Geology, 2020.

41. S. Srinivasan, E. Cawi, J. D. Hyman, D. Osthus, A. Hagberg, H. Viswanathan, and G. Srinivasan. Physics-informed machine-learning for backbone identification in discrete fracture networks. Comput. Geosci., 2020.

42. N. Makedonska, S. Karra, H.S. Viswanathan, and G.D. Guthrie,. Role of Interaction between Hydraulic and Natural Fractures on Production. Journal of Natural Gas Science and Engineering 2020, p.103451..

43. H. Pham, R. Parashar, N. Sund, and K. Pohlmann. A Method to Represent a Well in a Three-dimensional Discrete Fracture Network Model. Groundwater. 2020.

44. M.R. Sweeney, and J.D. Hyman. Stress effects on flow and transport in three-dimensional fracture networks. Journal of Geophysical Research: Solid Earth, 125, e2020JB019754. 2020.

45. J.D. Hyman. Flow Channeling in Fracture Networks: Characterizing the Effect of Density on Preferential Flow Path Formation. Water Resources Research 56.9 (2020): e2020WR027986..

46. H. Pham, R. Parashar, N. Sund, and K. Pohlmann. Determination of fracture apertures via calibration of three-dimensional discrete-fracture-network models: application to Pahute Mesa, Nevada National Security Site, USA. Hydrogeol J (2020)..

47. S. Srinivasan, D. O'Malley, J. D. Hyman, s. Karra, H. S. Viswanathan, and G. Srinivasan Transient flow modeling in fractured media using graphs. Physical Review E..

48. S. Srinivasan, D. O'Malley, J. D. Hyman, s. Karra, H. S. Viswanathan, and G. Srinivasan Transient flow modeling in fractured media using graphs. Physical Review E..

49. P. K. Kang, J. D. Hyman, W. S. Han, & M. Dentz, Anomalous Transport in Three-Dimensional Discrete Fracture Networks: Interplay between Aperture Heterogeneity and Injection Modes. Water Resources Research, e2020WR027378..

INDEX

K

L

M

O

P

R

S

T

U

W

Z