LA-UR-14-25932
Approved for public release; distribution is unlimited.

Title: Fehmpytests: Developing a Usable and Modular Test Suite for FEHM

Author(s): Lange, Mark

Intended for: Report

Issued: 2014-07-29

# Fehmpytests:

## Developing a Usable and Modular Test Suite for FEHM

**by Mark Lange and Dylan Harp**

# 1. PROJECT REPORT

### Introduction

The Earth and Environmental Sciences Division at Los Alamos National Laboratory uses the Finite Element Heat and Transfer Code (FEHM) to simulate subsurface processes. To ensure accuracy, FEHM needs to be tested on a regular basis. The original test suite for FEHM tested a large portion of FEHM features. However, the suite was difficult to run and modify. Fehmpytests attempts to solve these problems by creating a new test suite using Python's unittest module and a general test method. Python's unittest module addresses the problem of executing the test suite by allowing anyone with Python 2.7 to run the suite through a command line interface. The general test method addresses the problem of modifying the test suite by removing the common details of simulation testing. Future plans for Fehmpytests include adding more test coverage, integrating the suite into the FEHM build process, and improving the developer interface.

**Background**

Because it models complex physical systems, FEHM has a vast spectrum of functionality. The original test suite, written in Perl and Fortran, tested a large portion of the  FEHM functionality. However, this test coverage was incomplete. Because of the complexity of the old test suite's structure, adding new tests to FEHM was not easy for developers. In addition to the incomplete coverage, FEHM was only tested periodically by a code administrator and rarely by code developers during code development.

These two factors, incomplete test coverage and infrequent testing, motivated the development of a new test suite, Fehmpytests, written in Python. The goal of Fehmpytests was to address the problems with the old test suite and ultimately inspire better developer practices.

Python was chosen as the language of the new test suite because the language is platform independent, maintainable, has support for a unit testing framework, and works with PyFEHM, a Python based front end for FEHM.

**Developing FEHM**

When I first arrived at Los Alamos National Laboratory, my mentor,

Dylan Harp, had already written a few tests under the Python unit test framework. Using this framework, the structure of Fehmpytests was already simpler than the old test suite. The code was easier to read and understand.

For this project, I made my own personal goal to program generalized code that could be applied to several different scenarios. This would make not only my own task easier but any future developer's task easier as well.

One problem Fehmpytests had when I arrived was that, throughout each unit test, the algorithms were noticeably similar. This added unnecessary code bulk that could potentially scare away developers from adding new tests.

Once I understood where the repeated steps were, I started to design a general test method whose only parameter was the name of the folder containing the necessary FEHM files. This would create a simple developer interface that removed unnecessary details anytime a new test was created.

In an attempt to generalize all tests, I needed to know the spectrum of tests from the old suite. Specifically, I needed to know what data

files were associated with each FEHM function so that I could compare simulated data to correct data. During this summer project, I identified 4 types of data files produced during FEHM simulations:

1. Contour
2. History
3. Output
4. Tracer

Each file type needed similar but slightly different algorithms for testing the data inside. Going back to my goal of creating modular and generalized code, I handled these different file formats with the concept of closure.

In Python, methods are first class objects. This means methods can return other methods and variables can store them. I used closure to create a template method for test cases. In the future, anyone who discovers a new file type only needs to change a smaller part of the code to do so.

With the code in a maintainable state, the last thing I needed to do was define a folder structure for developers to provide the necessary files for each test. To keep the structure simple and intuitive, I divided all test case directories into compare and input folders.

With this structure, the developer only needs to organize their comparison and input files for the test case they want to add.

**Future Ideas**

The bigger goal of Fehmpytests was to inspire better developer practices. These practices include frequent testing and complete test coverage. There are two ideas that could be implemented into Fehmpytests to help better meet these goals.

To achieve frequent testing, fehmpytests could be incorporated into the FEHM build process by including its execution into the FEHM makefile. This would force developers to run the suite anytime they compile FEHM.

To achieve complete test coverage, developers need to be motivated to add their own tests. The process for adding more tests could be improved by adding a command line or graphical interface.

# 2. INSTALLATION

Once you have an account on the FEHM development site and have been given read/write permissions, follow these steps for installation.

1. **Obtain the FEHM repository.** Fempytests are included with the FEHM repository. To obtain the FEHM repository, type the following command into a terminal:

```
hg clone https://ancho.lanl.gov/fehm/hg/fehm-open
```

2. **Build FEHM.** In a terminal, navigate to *fehm-open/source* and type the following command:

```
gmake xfehm -f Makefile.fehm
```

# 3. TESTING FEHM

### 3.1 Testing in Default Mode

To test the default suite:

1. Navigate the folder fehmpytests in a terminal.
2. Type the following command into the terminal:

```
python fehmpytests.py <fehm-path>
```

Where <*fehm-path*> is the path to the FEHM executable.

### 3.2 Testing in Admin Mode

To test the admin suite:

1. Navigate to the folder fehmpytests in a terminal.
2. Type the following command into the terminal:

```
python fehmpytests.py --admin <fehm-path>
```

where <*fehm-path*> is the path to the FEHM executable.

### 3.3 Testing in Developer Mode

To test the developer suite:

1. Navigate to the folder fehmpytests in a terminal.
2. Type the following command into the terminal:

```
python fehmpytests.py --dev <fehm-path>
```

where <*fehm-path*> is the path to the FEHM executable.

### 3.4 Testing in Solo Mode

The process for testing a single test case in solo mode is similar to testing a suite in the other modes. There is an additional command line argument needed.

To test a singe test case:

1. Navigate to the folder fehmpytests in a terminal.

2. Type the following command into the terminal:

```
python fehmpytests.py <fehm-path> <test-case>
```

   where <*fehm-path*> is the location of the FEHM executable and <*test-case*> is the name of the test case method.

| **Warning:** Developers must run in default, admin, or developer mode before committing new code. |
|---|

### 3.5 Creating an Error Log

An error log can be created to show details about an error and where it occurred. To generate an error log, add the switch *log* after *fehmpytests.py* and before <*fehm-path*>. Here is an example:

```
python fehmpytests.py --admin --log <fehm-path>
```

# 4. CREATING NEW TEST CASES

A developer can add new test cases to the suite. There are two steps to adding new test cases:

1. Create the test case folder.

2. Add the test method.

### 4.1 Create the Test Case Folder

This is the folder structure of a test-case:

```
[test-case]
  |
  |_[input]
  |   |
  |   |_[control]
  |
  |_[compare]
```

All input files needed to run the FEHM functionality of the test-case go inside the input folder. All control files go inside the control folder. All compare files (contour, history, and output) that are known to be correct go inside the compare folder.
To set up a new test-case folder:

1. Go into the folder fehmpytests.
2. Create a folder <*test-case*> where <*test-case*> is the name of the new case.
3. Inside <*test-case*>, create two folders called input and compare.
4. Inside the input folder, create a folder called control.
5. In the control folder, place all control files.
6. If there is only one control file, rename it to **fehmn.files**.
7. If there are more than one control file, rename each file to <*subcase*>.**files** where <*subcase*> is the name of the subcase.
8. In the input folder, place all input files needed for the FEHM run.
9. In the compare folder, place all comparison files known to be correct.

### 4.2 Add the Test Method

To add the test method:

1. Open **fehmpytests.py**.
2. Inside the **class Tests**, write a method **test_**<*name*> where <*name*> is the name of the test case. Here is an example for the **avdonin** test method:

```
class fehmTest(unittest.TestCase):
    …
    #This is the new test method for avdonin.
    def test_avdonin(self):
        …
    …
```

3. Inside this test method, call

```
self.test_case('<test-case>')
```

   where *<test-case>* is the name of the folder you created for the new test-case. See ***4.4 Documentation on test_case Method*** below for details on the general test case method. Here is an example for the test method for avdonin:

```
class fehmTest(unittest.TestCase):
    …
    def test_avdonin(self):
        #Add this line to test avdonin.
        self.test_case('avdonin')
    …
```

4. Inside the class Suite, under the condition all, add the following line:

```
suite.addTest(fehmTest('<method-name>'))
```

   where <method-name> is the name of the test method you just defined. Here is an example for adding the avdonin test to the test-suite:

```
def suite(case, test_case):
    suite = unittest.TestSuite()

    if case == 'all':
        suite.addTest(Tests('test_saltvcon'))
        suite.addTest(Tests('test_dissolution'))
        …
        #Add avdonin to suite.
        suite.addTest(Tests('test_avdonin'))
        …
```

Running fehmpytests will now include the new test case.

### 4.3 Customizing a Test-Case

By default, test_case() will check for a maximum difference less than 1.e-4 on all outputs of the FEHM simulation. Passing a dictionary into test_case() as the second argument allows a developer to specify how these tests are performed.

The following keywords are recognized by test_case():

| Keyword | Python Type |
|---|---|
| variables | list |
| nodes | list |
| components | list |
| maxerr | float |
| test_measure | string |

The following is an example for specifying the components, variables, and format for the saltvcon test:

```
 #Pass a dictionary into test_case() with keywords
specified.
 def test_saltvcon(self):
     arguments = {}
     arguments['components'] = ['water']
     arguments['variables'] = ['Kx']
     arguments['format'] = 'relative'

     self.test_case('saltvcon', arguments)
```

### 4.4 Documentation on test_case() Method

The following is documentation for the use of the test_case method which is used to test FEHM functionality.

fehmTest.**test_case**(*name, parameters={}*)

Performs a test on a FEHM simulation and raises an AssertError if it fails the test.

**Parameters**

- name (str) – The name of the test-case folder.
- parameters (dict) – Attribute values that override default values.

The folder *name* in fehmpytests must exist with correct structure. If parameters are not passed into this method, all simulated outputs will be checked for a relative difference of less than 1.e-4.

# 5. Test Case Descriptions

The following is a list of the 13 test cases currently included in Fehmpytests and descriptions of what is tested. By default, all simulated results are compared to those that are provided in the compare folder. Most test cases currently included are customized using optional arguments for **fehmTest.test_case()**.

**Test the Radial Heat and Mass Transfer Problem**

*fehmTest.test_avdonin()*

Compares the generated contour and history files to old contour and history files that are known to be correct. For contour files, only the temperature values at time 2 are tested. For history files, all temperature values are tested.

**Test the boun test results (Flow Macro vs Boun Macro)**

*fehmTest.test_boun()*

Compares the generated contour files to old contour files that are known to be correct. Only the pressure and hydraulic head values at time 2 are tested.

**Test the Concentration Dependent Brine Density Functionality**

*fehmTest.test_cden()*

Compares generated history files to old history files that are known to be correct. Only the density values are tested.

**Test the Dissoultion Macro**

*fehmTest.test_dissolution()*

A one-dimensional transport simulation of calcite (CaC03(s)) dissolution is tested. Profiles of concentration versus reactor length, at selected times, will be compared against the analytical solution.

*Details of this test are described in the FEHM V2.21 Validation Test Plan on pages 93-95 (STN: 10086-2.21-00, Rev.No. 00, Document ID: 10086-VTP-2.21-00, August 2003)*

**Test the DOE Code Comparison Project, Problem 5, Case A**

*fehmTest.test_doe()*

Compares the generated contour and history files to old contour and history files that are known to be correct. For contour files, only the pressure, temperature, and saturation values at time 3 are tested. For history files, all pressure, temperature, and saturation values are tested.

**Test Dry-Out of a Partially Saturated Medium**

*fehmTest.test_dryout()*

Compares the generated contour files to old contour files known to be correct. The saturation is tested for all times.

**Test Head Pressure Problem**

*fehmTest.test_head()*

Compares the generated contour files to old contour files that are known to be correct. Only the pressure values at day 2 are tested.

**Test Multi-Solute Transport with Chemical Reaction**

*fehmTest.test_multi_solute()*

Compares the generated tracer files to old tracer files known to be correct. All concentration values are tested.

**Test Temperature in a Wellbore Problem**

*fehmTest.test_ramey()*

Compares the generated contour and history files to old contour and history file that are known to be correct. For the contour files, only the temperature values at time 2 are tested. For the history files, all temperature values are tested.

**Test the Salt Permeability and Porosity Macro**

*fehmTest.test_salt_perm_poro()*

The porosity-permeability function for compacted salt from *Cinar et at. (2006)* is tested using a six node problem with porosities from 0.01 to 0.2.

The excel spreadsheet in information/salt-permporo.xlsx contains calculations of the perm-poro function. Cinar, Y, G Pusch and V Reitenbach (2006) Petrophysical and capillary properties of compacted salt. Transport in Porous Media. 64, p. 199-228, doi:10.1007/s11242-005-2848-1

**Test the Salt Variable Conductivity Macro**

*fehmTest.test_saltvcon()*

Tests the calculations of thermal conductivity of crushed and intact salt.

**Intact salt:** kxi = k_{t-300}(300/T)^1.14

*Munson et al. (1990) Overtest for Simulate Defense High-Level Waste (Room B): In Situ Data Report. WIPP. Sandia National Laboratories, SAND89-2671*

**Thermal conductivity of crushed salt from Asse mine:**

kx_asse = -270*phi^4+370*phi^3-136*phi^2+1.5*ph

*Bechtold et al. (2004) Backfilling and sealing of underground respositories for radioactive waste in salt*(BAMBUS II project), EUR 20621, ISBN 92-894-7767-9*

kx = (k_{t-300}/kx_asse)*(300/T)^1.14

if kx is less then 1.e-6, set to 1.e-6.

*The excel spreadsheet /information/saltvcon.xlsx contains the associated calculations.*

**Test One Dimensional Reactive Solute Transport**

*fehmTest.test_sorption()*

Compares the generated tracer files to old tracer files known to be correct. All concentraction values are tested.

**Test Pressure Transient Analysis Problem**

*fehmTest.test_theis()*

Compares the generated contour files to old contour files known to be correct. Only the pressure values at time 2 are tested.

# Acknowledgements

I would like to thank Dylan Harp for his help and mentoring. I would also like to thank Shaoping Chu, Zora Dash, David Dempsey, Terry Miller, Phil Stauffer, and Cameron Tauxe for their help and feedback on this project. This project was funded by the DOE Office of Science SULI program.