# Fluid-Structure interaction and Python scripting capabilities in OpenSees

Minjie Zhu and Michael H. Scott

Civil & Construction Engineering

Oregon State University

*PEER Researchers' Workshop 2018*

August 8, 2018

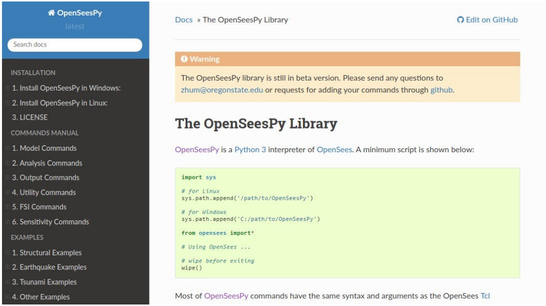
# Outline

# Background

1. OpenSees was developed and linked with Tcl for many years.
2. Tcl is difficult to learn because of its syntax.
3. Python has become popular as a scientific programming language.
4. Python syntax is clean and easy to use.
5. Many students learn Python in undergraduate.
6. Python has a very large ecosystem with many libraries for OpenSees users:
   `NumPy, SciPy, Pandas, Jupyter, Matplotlib, Mayavi, VTK, mpi4py, Sphinx, PyQt, wxPython, Django, Flask, and others on conda or pip.`

# OpenSeesPy

1. OpenSeesPy is a Python 3 interpreter of OpenSees. *Python 2 will be offcially deprecated in 2020.*
2. All model building (elements, materials, ...), analysis, output, utility, sensitivity, FSI commands have been implemented in OpenSeesPy.
3. `http://openseespydoc.readthedocs.io/`


OpenSeesPy
latest
Search docs
INSTALLATION
1. Install OpenSeesPy in Windows:
2. Install OpenSeesPy in Linux:
3. LICENSE
COMMANDS MANUAL
1. Model Commands
2. Analysis Commands
3. Output Commands
4. Utility Commands
5. FSI Commands
6. Sensitivity Commands
EXAMPLES
1. Structural Examples
2. Earthquake Examples
3. Tsunami Examples
4. Other Examples

Docs » The OpenSeesPy Library Edit on GitHub

Warning

The OpenSeesPy library is still in beta version. Please send any questions to zhum@oregonstate.edu or requests for adding your commands through github.

The OpenSeesPy Library

OpenSeesPy is a Python 3 interpreter of OpenSees. A minimum script is shown below:

```
import sys

# for Linux
sys.path.append('/path/to/OpenSeesPy')

# for Windows
sys.path.append('C:/path/to/OpenSeesPy')

from opensees import*

# Using OpenSees ...

# wipe before exiting
wipe()
```

Most of OpenSeesPy commands have the same syntax and arguments as the OpenSees Tcl

# Migration from Tcl to Python

Most Python commands have the same arguments as in the Tcl

Tcl node command:

```
node $nodeTag (ndm $crds) -mass (ndf $mass)
```

Python node command:

```
node(nodeTag, *crds, '-mass', *mass)
```

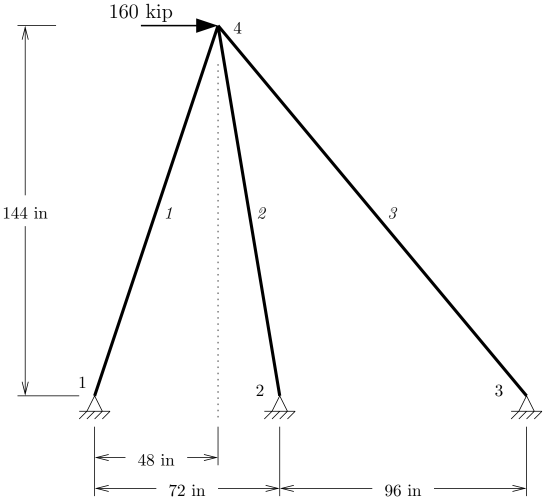
where (ndm $crds) is a list of Tcl variables, which can only be given individually,

```
node $nodeTag [lindex $crds 0] [lindex $crds 1] \
  [lindex $crds 2]
```

*crds is a Python list, which can be used in two ways,

```
node(nodeTag, crds[0], crds[1], crds[2])
node(nodeTag, *crds)
```

# A Truss Example



# A Truss Example

Tcl defining nodes

```
    set x {0.0 72.0 168.0 48.0}
    set y {0.0 0.0 0.0 144.0}

    for {set i 0} {$i<4} {incr i} {
      node [expr $i+1] [lindex $x $i] [lindex $y $i]
    }
```

Python defining nodes

```
    x = [0.0, 72.0, 168.0, 48.0]
    y = [0.0, 0.0, 0.0, 144.0]

    for i in range(4):
      node(i+1, x[i], y[i])
```

# A Truss Example

Tcl defining material

```
set A 4.0; set E 29000.0;
set alpha 0.05; set sigmaY 36.0

uniaxialMaterial Hardening 1 $E $sigmaY 0.0 [expr $alpha/(1- $alpha)* $E]


```

Python defining material

```
A = 4.0; E = 29000.0; alpha = 0.05; sigmaY = 36.0

uniaxialMaterial('Hardening', 1, E, sigmaY, 0.0, alpha/(1-alpha)*E)

```

# A Truss Example

Tcl defining elements:

```
foreach i {1 2 3} {
    element truss $i $i 4 $A 1
}

```

Python defining elements:

```
for i in [1,2,3]:
    element('truss',i,i,4,A,1)

```

# A Truss Example

Tcl defining load pattern:

```
    timeSeries Linear 1
    pattern Plain 1 1 {
      load 4 $Px $Py
    }

```

Python defining load pattern:

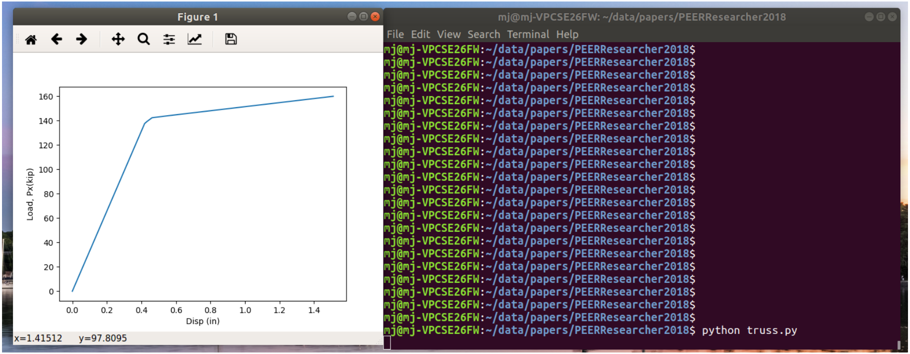
```
    timeSeries('Linear', 1)
    pattern('Plain', 1, 1)
    load(4, Px, Py)

```

Nested Tcl commands that use {}, e.g., load pattern, fiber section, have no braces in Python interpreter for OpenSees

# Generic Interface for OpenSees

```
void *
OPS_TrussElement()
{
  int numRemainingArgs = OPS_GetNumRemainingInputArgs();

  int numData = 3;
  if (OPS_GetInt(&numData, iData) != 0) { /*tag, iNode, jNode*/ }

  numData = 1;
  if (OPS_GetDouble(&numData, &A) != 0) { ... }

  ...

  UniaxialMaterial *theUniaxialMaterial =
       OPS_GetUniaxialMaterial(matTag);

  theElement = new Truss(iData[0], ndm, iData[1], iData[2],
     *theUniaxialMaterial, A, rho, doRayleigh, cMass);

  return theElement;
}
```

# Run Python Script in Terminal



# Run Python Script in Jupyter Notebook

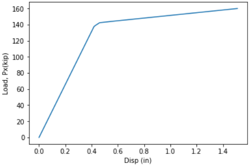
jupyter demo Last Checkpoint: Last Friday at 5:03 PM (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Markdown

## This is a note book for demonstrating OpenSeesPy

In [2]:
```
%run truss.py
```




In [ ]:
```
# %load truss.py
import sys
sys.path.append('/scratch/opensees/SRC/interpreter')
from opensees import *

import numpy as np
import matplotlib.pyplot as plt

# remove existing model
wipe()

# set modelbuilder
model('basic', '-ndm', 2, '-ndf', 2)

# variables
A = 4.0
E = 29000.0
alpha = 0.05
```

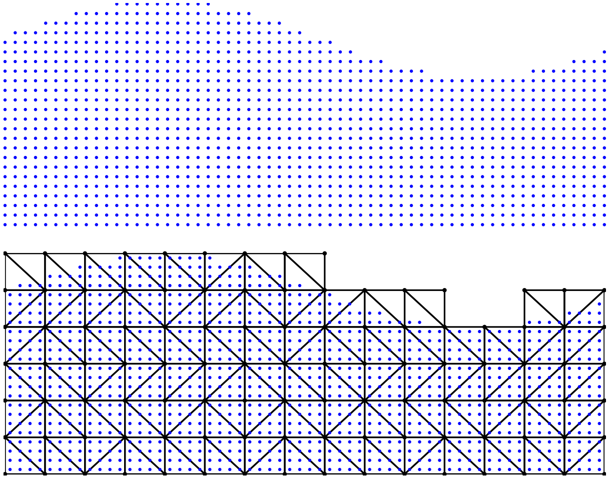
# Outline

# Background

- The FSI in OpenSees uses the Particle Finite Element Method.
- Finite Element Method with Lagrangian formulation
  - The position and physical properties of the particles are described in terms of the material coordinates and time
- Effective for solving free-surface flow problems
  - The changes in the position and physical properties as the material particles moves in free-surface are easy to capture
  - Fractional Step Method $\Rightarrow$ fast iterative solver
- Natural for solving structure-fluid interaction problems
  - Current configuration is chosen as the referential coordinates which is normally used in solid mechanics
  - The FSI in OpenSees can use all the elements in OpenSees.
- The ongoing research is using a new background mesh approach for more efficient FSI.

# A background mesh: Steps

1. Fixed mesh in fluid only area

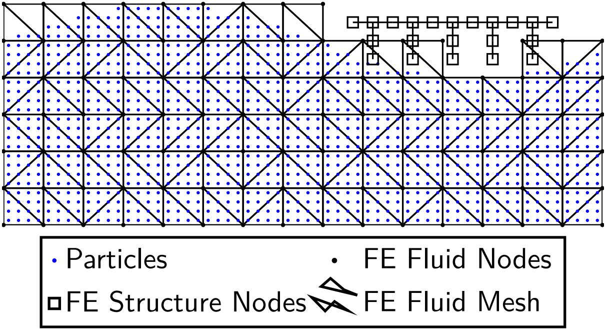


- Take only fluid particles.
- Create background fixed grid cells and FE nodes.

# A background mesh: Steps

2. Identify FSI area

(a) Structure on fixed mesh

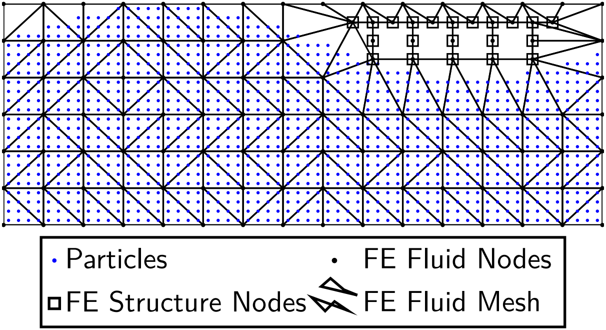



- Identify the location of structures.
- Identify the FSI area.
- Remove the closest grids nodes for all structural nodes.

# A background mesh: Steps

3. Delaunay Triangulation in FSI area

(b) DT around structure

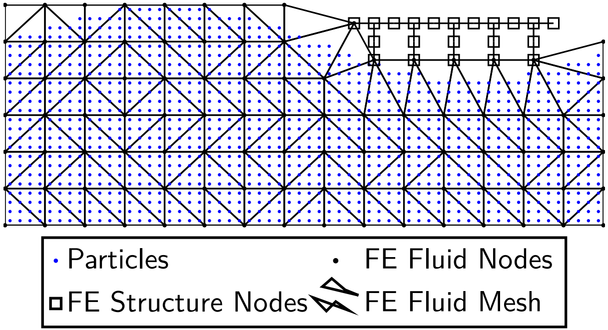



- Create DT with remaining grid nodes and structural nodes.
- Remove empty triangles in DT.

# A background mesh: Steps

4. The background mesh

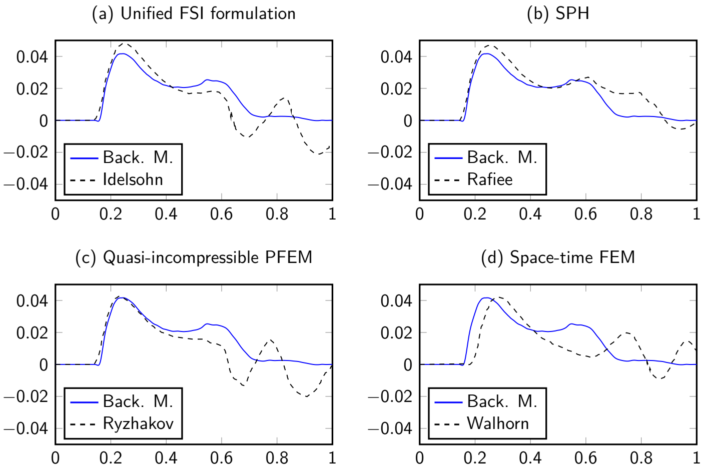
(c) Background mesh




- Create elements in the FSI area.
- Create elements in the Fluid area.

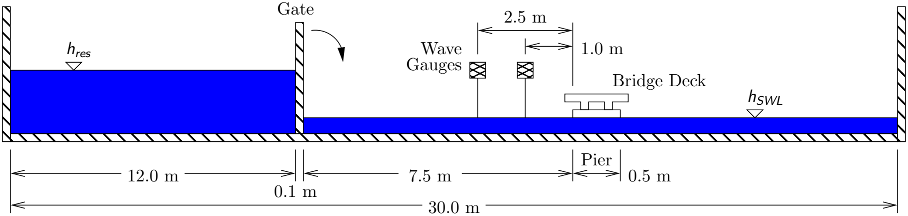
# Breaking dam on elastic column

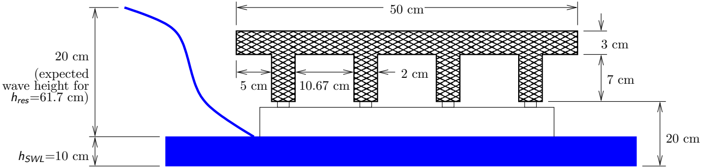
# Breaking dam on elastic column



# PWRI flume experiments

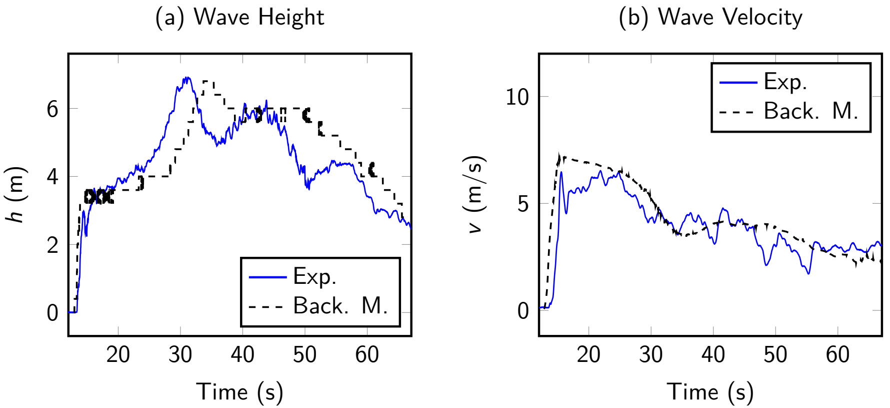
Public Works Research Institute 1:20 scale wave flume experiments in Japan (Hoshikuma et al 2013)



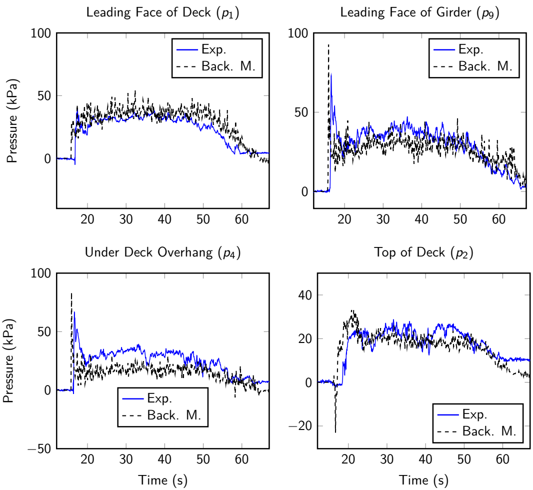



# PWRI flume experiments

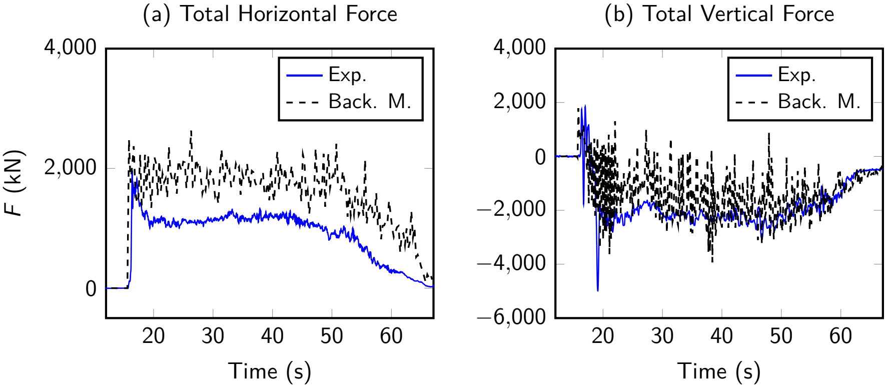
# Wave height and velocity



# Wave pressures on deck and girders



# Total forces on bridge deck



# 3D Breaking dam on obstacle

# Outline

# Summary

1. The OpenSeesPy has been implemented with most of the OpenSees Tcl commands.
2. The commands for parallel computing and reliability analysis will be added in the future.
3. Online user documentation is available and will be updated accordingly.
4. The OpenSeesPy is distributed as a Python module in Windows and Linux. The Mac version may be released in the future.
5. The FSI in OpenSees has been implemented for 2D problems.
6. The 3D FSI is under development.
7. A more efficient background mesh FSI is under development.

Zhu, M., Elkhetali, I., and Scott, M. (2017).
Validation of opensees for tsunami loading on bridge superstructures.
*Journal of Bridge Engineering*, 23(4):04018015.

Zhu, M., McKenna, F., and Scott, M. (2018).
Openseespy: Python library for the opensees finite element framework.
*SoftwareX*, 7:6–11.

Zhu, M. and Scott, M. H. (2014).
Improved fractional step method for simulating fluid-structure interaction using the pfem.
*International Journal for Numerical Methods in Engineering*, 99(12):925–944.