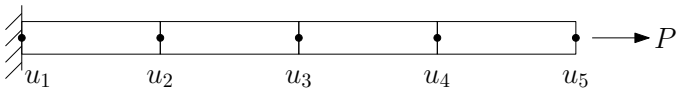


Figure 1: One dimensional axial rod with a tensile load.

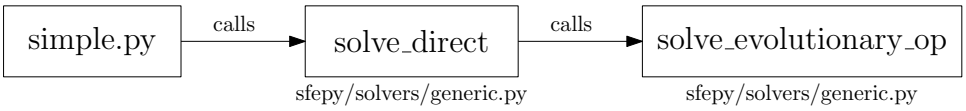


Figure 2: Initial flow of SfePy setup.

# 1 Introduction

This is an attempt to summarize the flow of SfePy as it solves a simple FEA problem (an axial rod with a traction load at its free end). The problem to be solved is a three dimensional version of the problem shown in Figure 1.

# 2 Initial Set Up

I am running SfePy from IPython using the command

```
run simple.py axial_rod_sfepy.py
```

The flow of the initial setup portion of the code is summarized in Figure 2. simple.py calls the function solve_direct which is in the module `sfepy/solvers/generic.py`. solve_direct calls the functions solve_evolutionary_op, because my problem is time varying.

# 3 Time Step Iteration

solve_evolutionary_op calls the __call__ method of SimpleTimeSteppingSolver as a generator with this line:

```
for ts, state in time_solver( state0 ):
```

SimpleTimeSteppingSolver.__call__ iterates over its time vector self.ts with this code:

```
for step, time in self.ts:
```

Each time through the **for** loop, step_fun is called. For my problem step_fun is time_step_function in `generic.py`.

The flow for each iteration in the **for** loop of SimpleTimeSteppingSolver.__call__ is summarized in the flow chart of Figure 3. SimpleTimeSteppingSolver.__call__ calls time_step_function. time_step_function calls ProblemDefinition.solve from this line:

```
state = problem.solve( state0 = state0, ts = ts, **data )
```

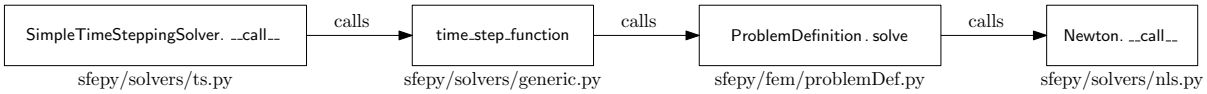


Figure 3: Interating through the ts array of SimpleTimeSteppingSolver.

near the end of time_step_function. ProblemDefinition.solve calls the __call__ method of the nonlinear solver Newton via this line:

```
state = solvers.nls( state )
```