Open-source software for coupling growth and remodeling of tissues with cell signaling pathways

Authors and developers:
Mohammadreza Soltany Sadrabadi, Ph. D.,
Amirhossein Arzani, Ph. D.

1-Introduction:

This software is an open-source nonlinear transient structural mechanics solver written in Python (FEniCS). The package couples the kinematic growth and remodeling (G&R) of a biological tissue/structure with systems biology and cell signaling pathways. Specifically, we developed open-source software to couple two modeling frameworks to capture associated multiscale connections between these two phenomena. At the cell scale, we implement a system of ordinary differential equations (ODEs) and partial differential equations representing the reactions between the biochemicals causing the growth. We implemented the weak form of the governing continuum mechanics equations using finite element analysis representing the G&R at the tissue level. Three test subjects (cube, tube, and aortic valve) are implemented in the package.

2- Installation:

The solver runs with MPI and interfaces, through FEniCS, to state-of-the-art linear algebra backends like PETSc. To work with this package, a user

should know the basic knowledge of how to solve PDEs through FEniCS and basic programming skills in Python. An excellent introduction to the FEniCS could be obtained by following the tutorial.
https://fenicsproject.org/tutorial/

- FEniCS Installation:

To run the software, users need to install FEniCS, an open-source finite element software written in Python: the best reference to install FEniCS is:
https://fenicsproject.org/download/
This solver is compatible with FEniCS 2018 and 2019. However, it is recommended to install the last version of FEniCS:

```
$ conda create -n fenicsproject -c conda-forge fenics=2019.2
$ source activate fenicsproject
```

- Some useful packages:
  1. scipy:

     scipy is a built-in python package that provides algorithms for optimization, integration, interpolation, eigenvalue problems, algebraic equations and etc. We used this package to solve the systems of ODEs for cell signaling pathways. Install this package in the FEniCS environment:

```
$ source activate fenicsproject
$ conda install scipy
```

  2. mshr:

     This package is a built-in python package and is a supplementary package in the FEniCS environment for making simple geometries and discretizing and meshing the geometries to tetrahedral elements. Install the package in the FEniCS environment:

```
$ conda install -c conda-forge mshr
```

- Growth and Remodeling Software:

The solver can be installed simply by cloning the GitHub repository to your own computer:

```
$ git clone https://github.com/amir-cardiolab/valve-growth.git
$ cd code
```

Also, you can just download software from the link below:

```
$ https://github.com/amir-cardiolab/valve-growth.git
$ unzip code.zip
$ cd code
```

3-Files and folders:

The software contains three folders for different examples. Each folder includes 8 python codes: All of the codes are implemented in Python, and the main executable code name is main_loop.py. To run the solver, you need to preset the assumptions:

a. **geometry.py**: In this code, you can define your geometry by either defining the geometry in the mshr package or importing pre-defined geometry. Also, the material properties of the tissue can be defined here.

b. **Geo.py**: In this code, the Function spaces, number of elements, and number of nodes are defined for the geometry. All of the functions have comments and are explained in the code. For more details, read the comments.

   Note: Definitions of global and local number of nodes for one processor and parallel multiprocessors are different.

c. **Flags.py:** This code gives you the option of whether you want to use the system of PDEs or the system of ODEs for your cell signaling pathway. Also, if you want to calculate the elastodynamics of tissue in the continuum framework, like the dynamic of the aortic valve, you can turn on the flag of the generalized alpha method for time integration.

d. **parameters:** The equations for the generalized alpha method and useful equations in continuum mechanics are defined as python functions.

   Note: you can add any other functions you need to use for your problem here.

e. **system_ODE.py:** you should define the system of ODEs for cell signaling pathways if the FLAG_ODE in Flags.py is True. Different systems of ODEs were implemented for different cases (cube, tube, and aortic valve). The function for the system of ODE should be defined as an array in a python environment. You can also define any variables you want to update in your systems of ODEs in the python function. Also, the growth rate at the tissue level is defined as the output of the system of ODEs.

f. **growth_cont.py:** The continuum mechanics governing equations for G&R at the tissue level is implemented here. F_growth is defined as a tensor that you can implement the growth rate and the growth direction. A simple one-term Mooney-Rivlin equation was implemented for the cube and tube case, and the anisotropic term was added to the strain energy function for the aortic valve. The weak form of Cauchy's equation is implemented in the reference configuration framework. Finally, Cauchy's stress and strain is defined to be calculated in the simulation.

g. **Num_prm.py:** The numerical solver parameters in PETSc are defined. You can change the parameters for better convergence in your case. For our examples, the nonlinear Newton solver is used for the nonlinear numerical problem, and tfqmr method is used for the linear solver.
   Note: To the best knowledge of us, these solvers that we have for growth and remodeling are the most stable solvers for these kinds of problems.

h. **main_loop.py:** This code solves the system of ODEs or the system of PDEs and also the G&R at the tissue level. Also, the G&R at the tissue level is coupled with cell signaling pathways for 1-way and 2-way coupling problems.
   Note: you should define the system of PDEs in this code if Flag_PDE in Flags.py is true.

main_loop.py is the executable code, which is included in the main loop where you can run your simulation. To run the solver, you need to run the main_loop.py:

```
$ python main_loop.py
```

If you want to run in parallel (recommended), you need to run the code as:

```
$ time mpi -n n_processors python main_loop.py
```

In the above, “n_processors” is the number of processors you want to use in parallel (MPI).

Note 1: for the aortic valve case, because the geometry was made in another software than the mshr, we uploaded the geometry in google drive.
https://drive.google.com/file/d/1QVIGgRxDCMKsZHVPIoSGhBz3DxKBMSlv/view?usp=sharing

5- Post-processing:

You can see the results in the open-source software Paraview. To download the software, you can use the link below:
https://www.paraview.org/download/

To see the displacement and the growth for each time span, you need to use the “warp by vector” option in Filters->alphabetical->warp by vector.