NM^3: Nonlinear MetaMaterial MPI Solver

This is the repository for the source codes used in paper:

High-Performance Large-Scale Simulation of Multi-stable Metastructures

M. Hwang, C. Scalo, and A. F. Arrieta, *Computer Physics Communications*, 2022.
DOI: https://doi.org/10.1016/j.cpc.2022.108365

Program overview

NM^3 is massively parallel code for solving general multi-stable mechanical metamaterials and metastructures. The code is written in **Fortran** and utilizes **MPI** (Message-passing interface) system calls to execute parallel computation. The code supports both explicit and implicit numerical methods: up to the 4^{th}-order Runge-Kutta (RK) methods are implemented for the explicit solver; and the Newmark-β (NB) method with constant average acceleration is implemented for the implicit solver. Currently, three types of multi-stable systems can be solved with **NM^3**: (1) 1D multi-stable lattice with coupled pendula (discrete sine-Gordon model), (2) 1D lattice with quartic on-site potentials (discerete φ-4 model), and (3) metabeam with a bi-stable microsctructure.

Before running the program

1. System prerequisites: **MPI**, **Fortran** complier, **GNU make**, **git**, **hdf5**, **Python** libraries (**numpy**, **matplotlib**, **h5py**).
    - Debian or Ubuntu-based Linux: Install the required packages with **apt** package management tool.

      ```
      $ sudo apt install gfortran openmpi-bin libopenmpi-dev make git
      $ sudo apt install python3-numpy python3-matplotlib python3-h5py
      $ sudo apt install libhdf5-openmpi-dev hdf5-tools
      ```

    - RedHat-based Linux (e.g., Fedora, Oracle): Install the required packages with **dnf** package management tool.

      ```
      $ sudo dnf install gcc-gfortran openmpi openmpi-devel make git
      $ sudo dnf install python3-numpy python3-matplotlib python3-h5py
      $ sudo dnf install hdf5-openmpi-devel
      ```

    - MacOS: **Fortran** compiler, **GNU make**, **git**, and **Python** are already installed in most cases. We recommend using **Homebrew** to install **MPI** and **hdf5**. First, go to https://brew.sh and follow the instruction to install **Homebrew**. Then, install the required packages with `brew` and **Python**'s `pip3` commands.

      ```
      $ brew install open-mpi hdf5-mpi
      $ pip3 install --user numpy matplotlib h5py
      ```

    - Windows: **NM^3** is not natively supported on Windows. The simplest way to use **NM^3** on Windows is through the Ubuntu app from Microsoft Store, which allows to use Ubuntu terminal. Once it is installed, run the app and follow the same procedure for the Debian or Ubuntu-based Linux system.
2. Clone **NM^3** repository.

   ```
   $ git clone https://github.com/wonnie87/NMCube
   ```

3. Compile and link the programs.

   ```
   $ cd NMCube
   $ make
   ```

4. NOTE: If **GNU make** gives an error saying that it cannot open 'hdf5.mod' and/or find '-lhdf5,' uncomment line 10 and line 15 in **Makefile** file, set the environment variables **DIR_TO_HDF5_HEADER** and **DIR_TO_HDF5_LIB** to the paths where **HDF5** headers and libraries are installed, and re-run `make` command. The following are the default directories on each operating system.
    - Debian or Ubuntu-based Linux:
        - Headers → /usr/include/hdf5/openmpi
        - Libraries → /usr/lib/x86_64-linux-gnu/hdf5/openmpi
    - RedHat-based Linux:
        - Headers → /usr/lib64/gfortran/modules/openmpi
        - Libraries → /usr/lib64/openmpi/lib
    - MacOS:

        - Headers → /usr/local/Cellar/hdf5-mpi/(version)/include
        - Libraries → /usr/local/Cellar/hdf-mpi/(version)/lib
    - Windows: Same as those for the Ubuntu-based Linux if using the Ubuntu app.

Running the program

For both NB and RK solvers, run the following command in the top-most (**NMCube/**) directory.

```
$ make run NP=(number of processes) INP=(input file name)
```

- (input file name) can be any input file in **inputs/** directory or a path to the input file.

For example, to run the input file **Input_prob4.py** in **inputs/** directory with 8 processes:

```
$ make run NP=8 INP=Input_prob4.py
```

Or, equivalently:

```
$ make run NP=8 INP=./inputs/Input_prob4.py
```

If parameters **NP** and **INP** are not specified, **make run** uses **NP**=4 and **INP**=./inputs/InputFile.py by default.

(Optional) Solution stability check

The implemented RK solver is not unconditionally stable so that the soution may be diverging or errorneous if **dt** is not set to be small enough for the problem. If needed, users may compare the obtained solution with that of NB solver by running `make check` in the top-most directory.

WARNING: `make check` invokes the NB solver, the simulation with which takes orders of magnitude longer than the RK solver.

Data storage

The simulated outputs are packaged in a **HDF5** file format (.h5). The time displacements, velocities, and accelerations are stored in the datasets **/u**, **/udot**, and **/uddot**, respectively. The output file is automatically saved in **outputs/** directory.

Postprocessing

Use **h5dump** or **h5ls** (default command line programs supplied with **hdf5** installation) for quick data retrieval. For example, to view the entire time displacements of a file **P4_d04_F0.1_f8.0_RK4.h5**:

```
$ h5dump -d "/u" P4_d04_F0.1_f8.0_RK4.h5
```

For more involved postprocessing, use more advanced tools, such as **Python**'s **h5py** library. Example **Python** scripts can be found in **outputs/** directory.

NM^3 can also generate a video of an output file by running `make movie` in the top-most directory. For example, the following command creates a video file named **P4_d04_F1.4_f35.0_RK4.mp4** in **outputs/** directory.

```
$ make movie OUT=P4_d04_F1.4_f35.0_RK4.h5 FSKIP=10
```

With `make movie`, the video play speed can be controlled with the **FSKIP** parameter. The higher **FSKIP**, the faster the video plays. By default, `make movie` uses **FSKIP**=100.

If finer control of the resulting video is desired, modify and run the supplied **Python** script in the **outputs/** directory to override the default behavior.

```
$ python3 MovieGen.py
```

Troubleshooting

1. `make movie` produces **TypeError: __init__() got an unexpected keyword argument 'extra_args'** → Install **ffmpeg** on your system
    - Debian or Ubuntu-based Linux:
      > $ sudo apt install ffmpeg

- RedHat-based Linux: Enable RPM fusion and then run the following command.
  > $ sudo dnf install ffmpeg
- MacOS:
  > $ brew install ffmpeg
- Windows: Same as those for the Ubuntu-based Linux if using the Ubuntu app.

2. On some systems, `make` gives an error "**make[1]: mpif90: No such file or directory**" during the compilation or "**make[2]: mpirun: No such file or directory**" during the runtime. This is because `mpif90` and `mpirun` are not in your system PATH. → Set **MPIF** and **MPIRUN** variable in Makefile to the exact path to `mpif90` and `mpirun`, respectively.

   - This error most likely occurs on Fedora. On Fedora, the default installation path is **/usr/lib64/openmpi/bin/mpif90** and **/usr/lib64/openmpi/bin/mpirun**.

File description

```
.
├── LICENSE            -> MIT License file
├── Makefile           -> Parent Makefile compiling and running the code
├── README.md          -> README file (User guide) in Markdown format
├── README.pdf         -> README file (User guide) in PDF format
├── inputs             -> [Directory] stores input files
│   ├── InputFile.py      -> Default input file
│   ├── Input_prob1.py    -> Example input for a discrete sine-Gordon model
│   ├── Input_prob1_nl.py -> Example input for a discrete sine-Gordon model (nonlinear response)
│   ├── Input_prob2.py    -> Example input for a discrete phi-4 model
│   ├── Input_prob4.py    -> Example input for a bi-stable metabeam
│   └── Input_prob4_nl.py -> Example input for a bi-stable metabeam (nonlinear response)
├── modules            -> [Directory] Python modules
│   ├── InpGen.py         -> Module containing classes and functions for input generation
│   ├── __init__.py       -> Initialization file for Python module folder
│   └── postprocessing.py -> Module containing functions for postprocessing
├── nb                 -> [Directory] containing NB solver
│   ├── Makefile          -> Makefile containing rules for compiling and running NB solver
│   ├── hdf5_helpers.f90  -> Helper module containing convenient subroutines for handling HDF5 files
│   ├── main_nb.f90       -> Main Fortran script for NB method
│   └── sub_nb.f90        -> Fortran module containing subroutines used in main_nb.f90
├── outputs            -> [Directory] stores output files
│   └── MovieGen.py       -> Example script generating a video of the output response
└── rk                 -> [Directory] containing RK solver
    ├── Makefile          -> Makefile containing rules for compiling and running RK solver
    ├── hdf5_helpers.f90  -> Helper module containing convenient subroutines for handling HDF5 files
    ├── main_rk.f90       -> Main Fortran script for RK method
    └── sub_rk.f90        -> Fortran module containing subroutines used in main_rk.f90
```