`mpmfracture` Manual

Tito Adibaskoro

1 Important Files

1.1 `mpmfracture.tar.gz`

This is an archive containing `mpmfracture.exe`. Extract the executable by typing:

```
tar -zxvf mpmfracture.tar.gz
```

This is to avoid problems with pushing executables into github repository.

1.2 `mpmfracture.exe`

As mentioned in section 1.1, `mpmfracture.exe` is archived in `mpmfracture.tar.gz`.

The excutable solver, compiled C code from `mpmfracture.m` conversion via MATLAB coder. Reads the input file `input.txt` and produces output data at SimOutput folder.

If folder named "SimOutput" doesn't exist, `mpmfracture.exe` produces no output.

Expected use case: high performance run

1.3 `mpmfracture.m`

The "source code" of the executable solver `mpmfracture.exe`. Run in MATLAB, `mpmfracture.m` works the same way as its C-compiled executable counterpart.

`mpmfracture.m` is a paralellized MATLAB code, with known bug of having a small chance of big *parfor*s loop going over the defined upper limit. A *try-catch* and rollback procedure which could easily work around the issue renders the MATLAB script non-CODER-compliant, therefore deprecated. When left running via MATLAB for a long time, expect the run to crash at random point.

Expected use case: debugging and test runs

1.4 `mpmfractureplot.m`

Plots the run results. To be run in MATLAB, not intended to go through coder. This `mpmfractureplot.m` is essentially

Editing `mpmfractureplot.m` is unrecommended, instead edit `mpmfracture.m` and generate `mpmfractureplot.m` by running `generateplotter.m` (see subsection 1.5).

1.5 `generateplotter.m`

`generateplotter.m` activates plotting features suppressed in `mpmfracture.m` for coder compliance. Run in MATLAB, `generateplotter.m` generates `mpmfractureplot.m`, which, run in MATLAB, reads simulation output files and generates plot.

1.6 Input file `input.txt`

An input file called `input.txt` is included in the repository. See section 2 on the rules to write this input file, and section 3 for the available parameters.

2 How to write input file `input.txt`

2.1 Regular input

Variable input must conform with the following format:

```
<variable> = <value> | <type>
```

Spaces and indentations at the start of the line, around the equal sign, around the `|` sign, or at the end of the line is ignored.

For example, inputting the elastic modulus `E` can be done as:

```
E = 8000 | float
```

Which means the variable to be input is called `E`, with the value of `8000`, and with variable type `float`.

There are three variable types (`<type>`):

1. `long`: for long integer
2. `float`: for decimal number
3. `string`: for string input

Variable names need to match exactly with what the solver expects, and are **case sensitive**.

Variable types need to match exactly with what the solver expects. For example, elastic modulus `E` needs to be marked as `float`, even though the value is a round number, for example, `8000`.

Same goes with `long`, which should serve quantities that involes counting of round numbers. For example, number of grids (`NN_base_1` and `NN_base_2`) and `refinementfactor`

2.2 Comment

The symbol % and any text after, up to the end of the line, is ignored.

2.3 New line

Both ASCII characters 10 and 13 (and 10 followed by 13 and vice versa) are accepted as new line marker in the input file.

3 Parameters

3.1 General

3.1.1 refinementfactor

Determines the subdivision of cells for refinement purpose

3.1.2 E

Set the elastic modulus

3.1.3 nu

Set poisson's ratio

3.1.4 b_1

Constant body force in the x direction

3.1.5 b_2

Constant body force in the y direction

3.1.6 le_base

Determines the size of cell before refinement

3.1.7 NN_base_1

Number of grids in x direction before refinement

3.1.8 NN_base_2

Number of grids in y direction before refinement

3.1.9 NodexPosition_PredXvel

Determines the X position of where X displacement-controlled nodes are located. There's some tolerance to the number, so non-rounded decimals or irrational numbers won't be a problem

3.1.10 NodeyPosition_PredYvel

Determines the Y position of where Y displacement-controlled nodes are located. There's some tolerance to the number, so non-rounded decimals or irrational numbers won't be a problem

3.1.11 NodexPosition_PredXvel_rel

Determines the X position of where X displacement-controlled nodes are located. There's some tolerance to the number, so non-rounded decimals or irrational numbers won't be a problem

3.1.12 NodeyPosition_PredYvel_rel

Determines the Y position of where Y displacement-controlled nodes are located. There's some tolerance to the number, so non-rounded decimals or irrational numbers won't be a problem

3.1.13 endcostime

Determines when the cosine ramp ends (see figure 1)

3.1.14 dispload

Determines the displacement at when `endcostime` is reached (see figure 1)

3.1.15 ftime

Determines the time when simulation is considered finished

3.1.16 NodexPosition_pin

Determines the x position of nodes with pin support

3.1.17 NodeyPosition_pin

Determines the y position of nodes with pin support

3.1.18 NodexPosition_roller

Determines the x position of nodes with roller support

3.1.19 NodeyPosition_roller

Determines the y position of nodes with roller support

3.1.20 x_start

Determines the start of particle placement in x direction

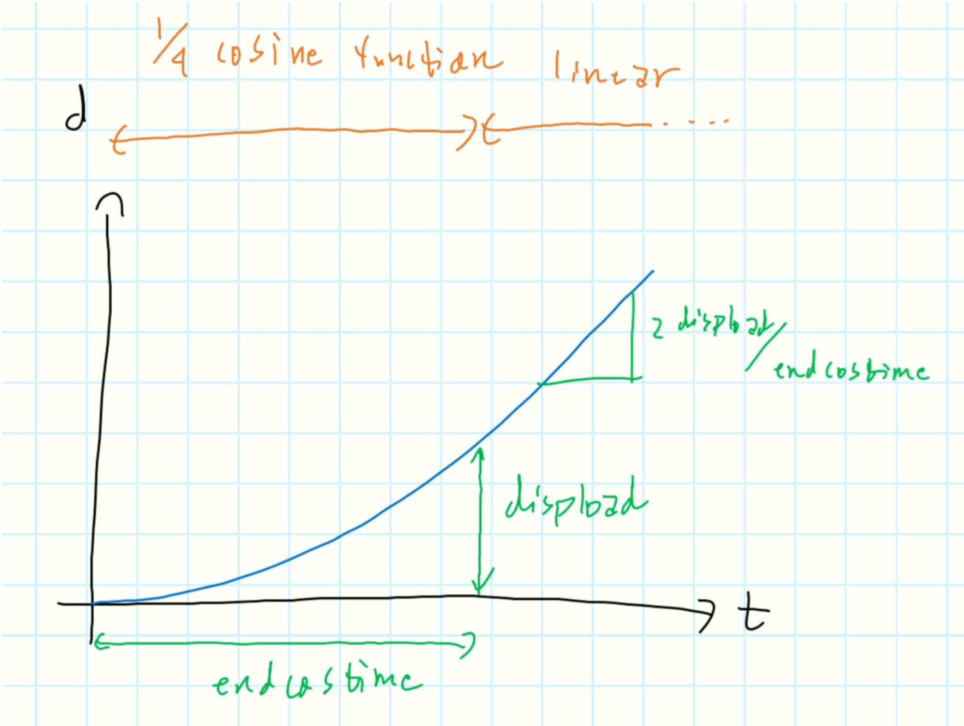


Figure 1: Definition of `endcostime` and `dispload`

3.1.21 x_end

Determines the end of particle placement in x direction

3.1.22 y_start

Determines the start of particle placement in y direction

3.1.23 y_end

Determines the end of particle placement in y direction

3.1.24 denyinitiation

Prevents any crack initiation

3.1.25 denypropagation

Prevents any crack propagation

3.1.26 t_halflife

Damping, how long does it take to half particle velocity (the smaller the number, the larger the damping). If not assigned, then no damping applies.

3.1.27 t0_plane

Sets the thickness of the 2D plane. Thickness stays constant under plane strain, may change under plane stress.

3.2 Erase Boxes

Particles can be deleted (erased) to, for example, create a notch. This is achieved by defining several boxes where particles can be deleted from

3.2.1 n_eraseboxes

Declare the number of boxes to be defined. This is important for memory preallocation.

3.2.2 x_erase_start(i)

Defines the left boundary of the box, where $i = 1, 2, 3, ...,$ `n_eraseboxes`

3.2.3 x_erase_end(i)

Same as `x_erase_start(i)` but for the right boundary of the box.

3.2.4 y_erase_start(i)

Same as `x_erase_start(i)` but for the bottom boundary of the box.

3.2.5 y_erase_end(i)

Same as `x_erase_start(i)` but for the top boundary of the box.

3.3 Initial Cracks

Initial cracks are defined by how many crack paths is desired, followed by the number of particles for each path.

3.3.1 n_crackpaths

Declares the number of crack paths as initial crack.

3.3.2 n_crackparticles(i_{path})

Declares the number of crack particles for path number i_{path}

3.3.3 x_crackparticle($i_{particle}$,i_{dim},i_{path})

Defines the position of crack particle number $i_{particle}$ of path i_{path}, in either x ($i_{dim} = 1$) or y ($i_{dim} = 2$) direction

3.3.4 tip_terminate(i_{path},i_{tip})

tip_terminate(i_{path},i_{tip})=1 prevents tip number i_{tip} of path i_{path} from propagating. $i_{tip} = 1$ designates the tip with the highest $i_{particle}$ number, while $i_{tip} = 2$ designates the tip with the lowest $i_{particle}$.

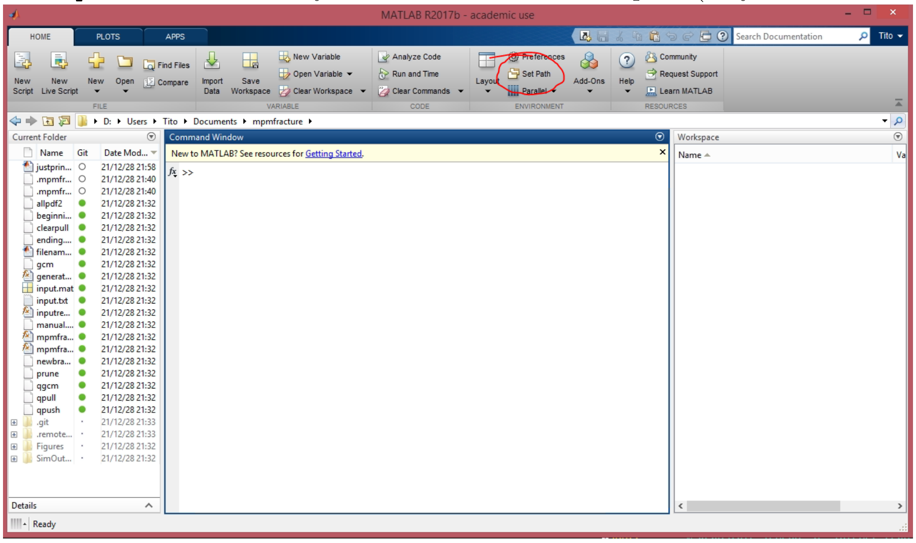
3.3.5 elasticstrength

Determines the strength of the elastic material, for the porpose of crack initiation and propagation.

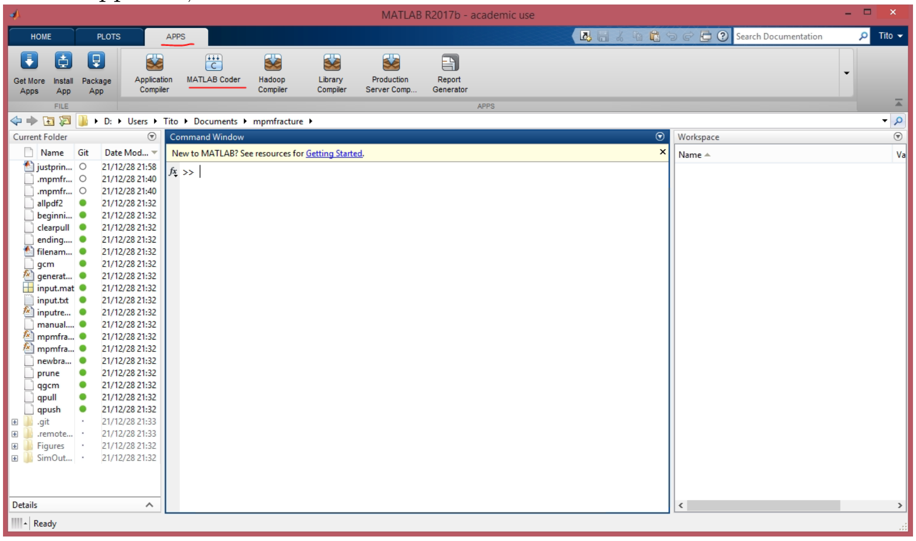
4 How to "compile" `mpmfracture.m` into an executable

4.1 MATLAB coder

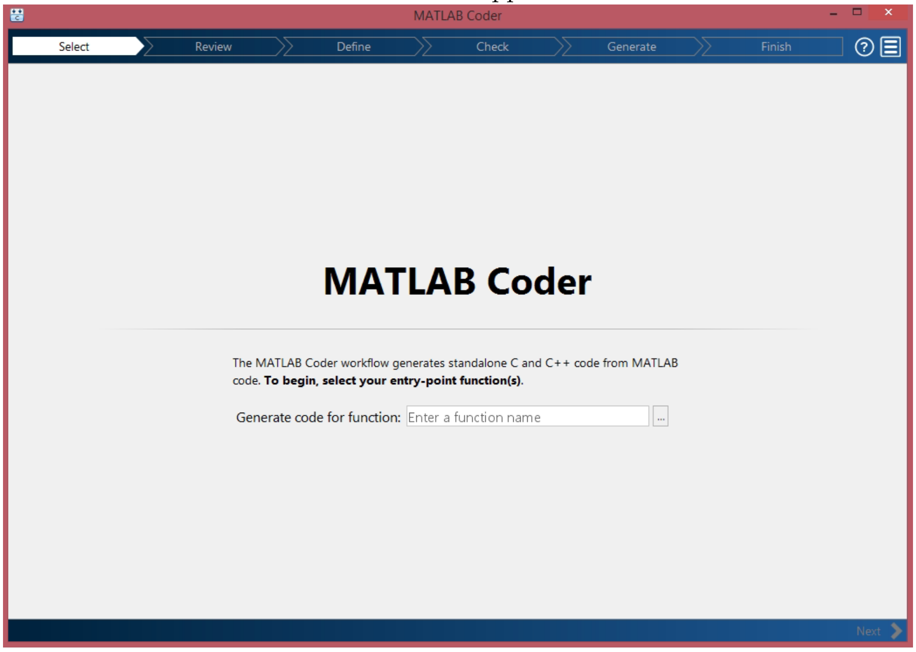
1. Add `mpmfracture` directory into MATLAB as known path (only need to do this once)



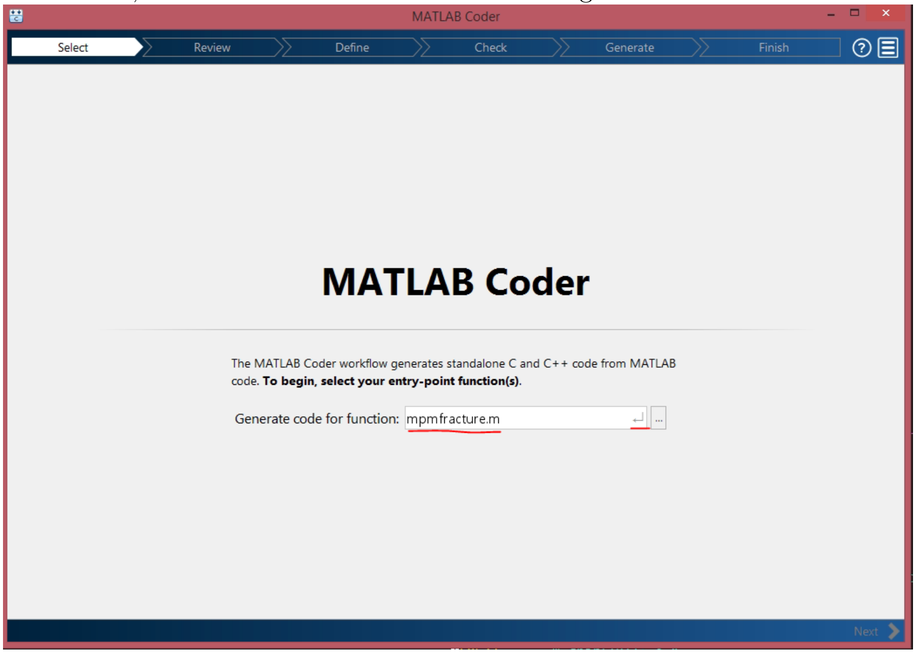
2. Go to Apps tab, then click *MATLAB Coder*



3. Wait until the *MATLAB Coder* window appears

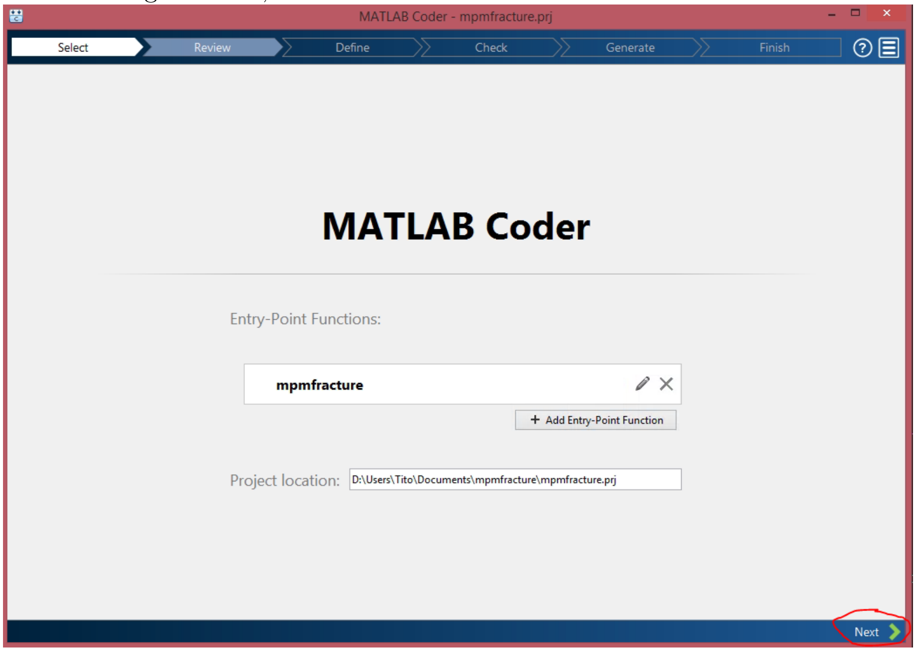


4. write `mpmfracture.m` in the *Enter a function name* textbox
5. Press enter, or click the small enter icon on the right end of the textbox

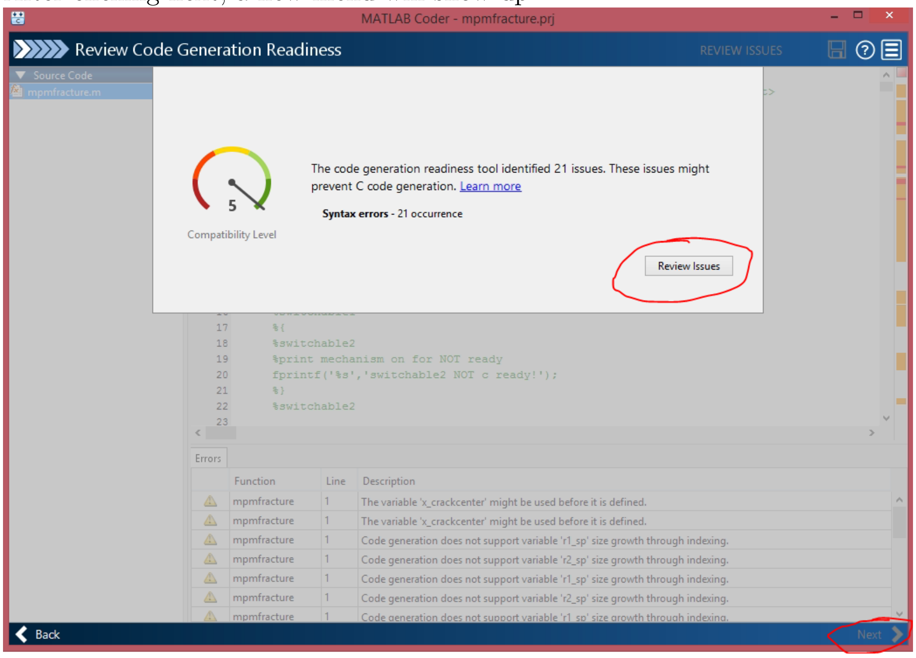


6. If there is a warning about project already existing, click *overwrite*

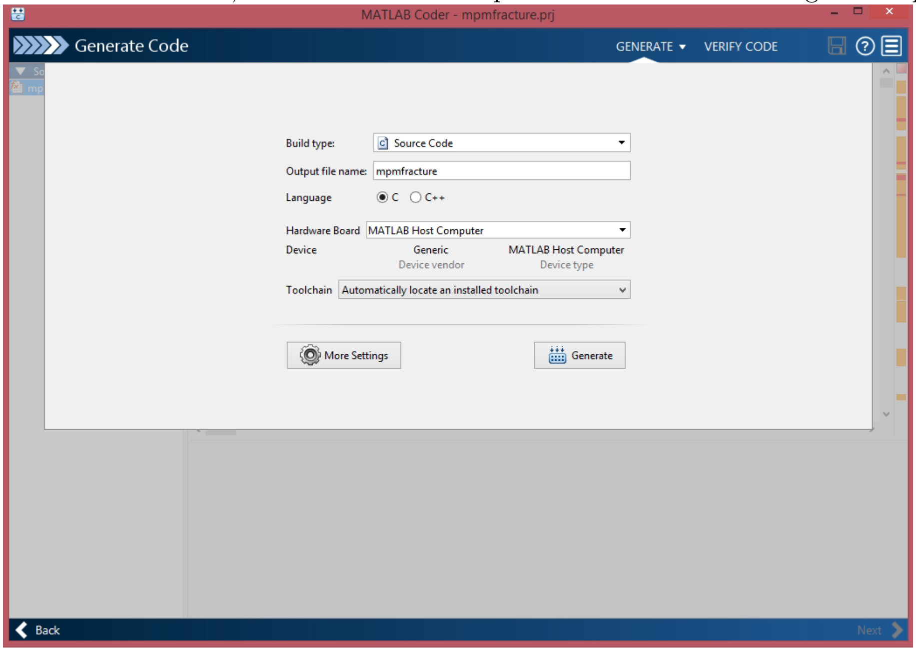
7. After waiting for a bit, the next button can be clicked



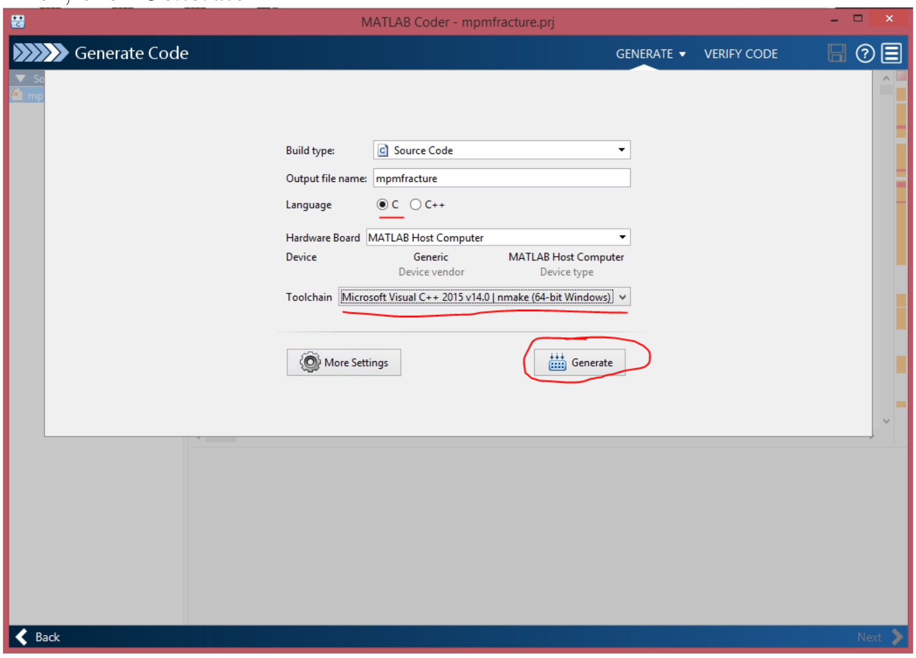
8. After clicking next, a new menu will show up



9. Click *Review Issues*, then click *Next* multiple times until the following menu appears:

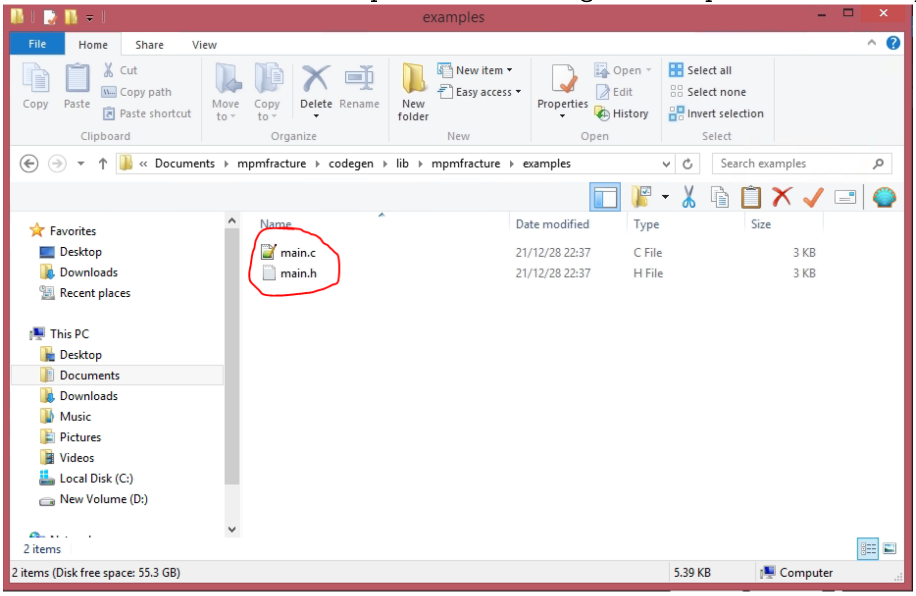


10. Make sure *C* is selected as the *Language*
11. For *Toolchain*, choose *Microsoft Visual C++ 2015 v14.0*
12. Then, click *Generate*



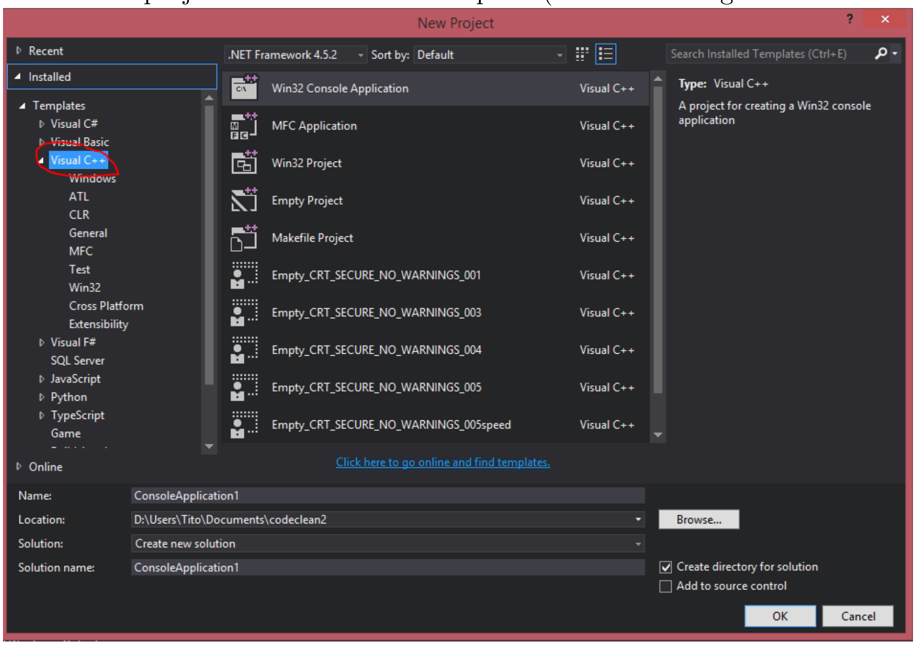
13. The process may take about 3 minutes until the C code is generated
14. Once done, a bunch of .c and .h files should be available in `mpmfracture\codegen\lib\mpmfracture`

15. move `main.c` and `main.h` from `mpmfracture\codegen\lib\mpmfracture\examples` to `mpmfracture\codegen\lib\`

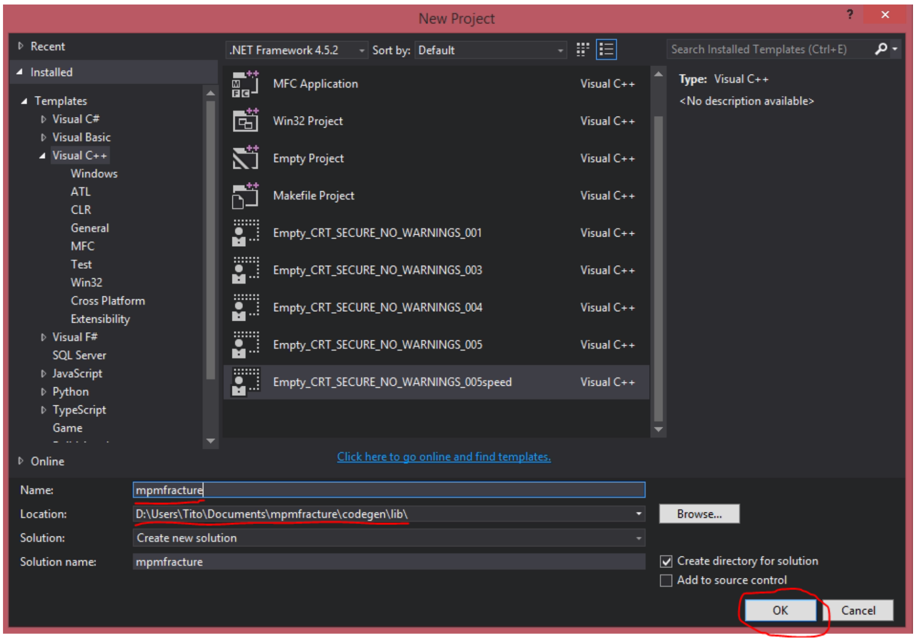


4.2 MS Visual Studio

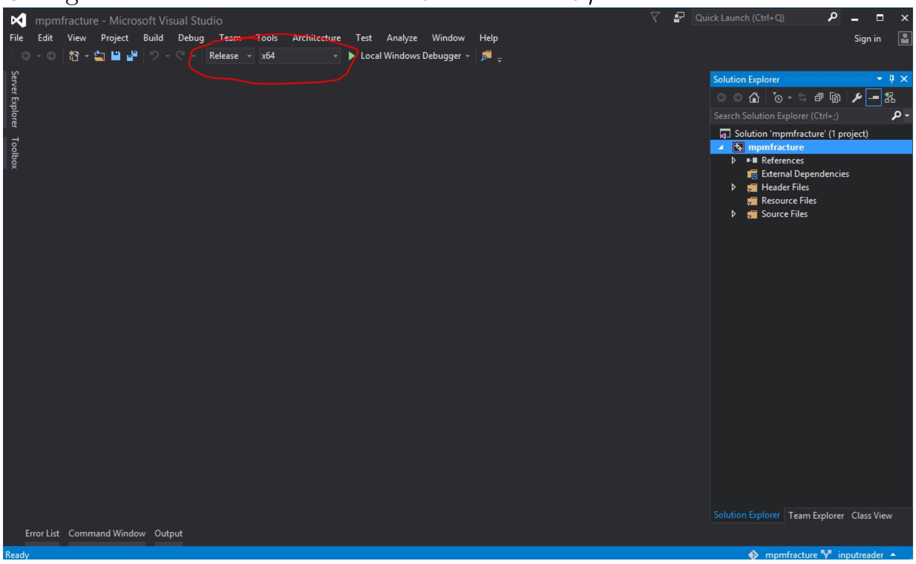
1. Open MS Visual Studio
2. Import the included template .zip file in `mpmfracture\MSVSTemplate`
3. Create new project with the custom template (should be categorized in *Visual C++*)



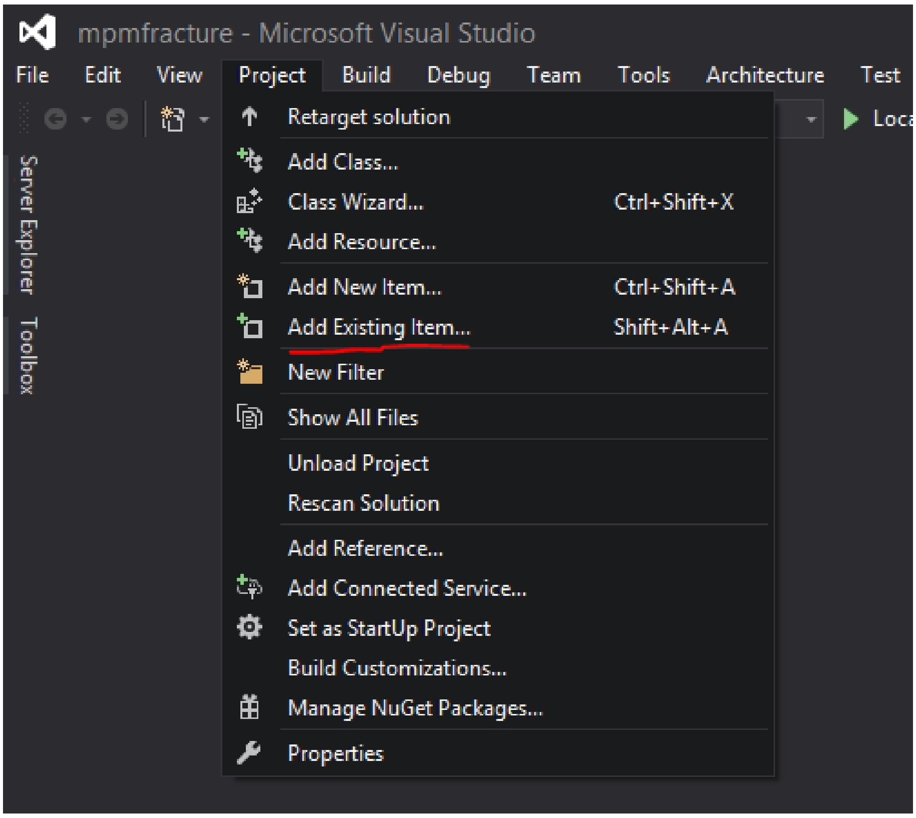
4. Put *mpmfracture* in the *Name* textbox, and the full directory of `mpmfracture\codegen\lib` in *Location* textbox, then click OK



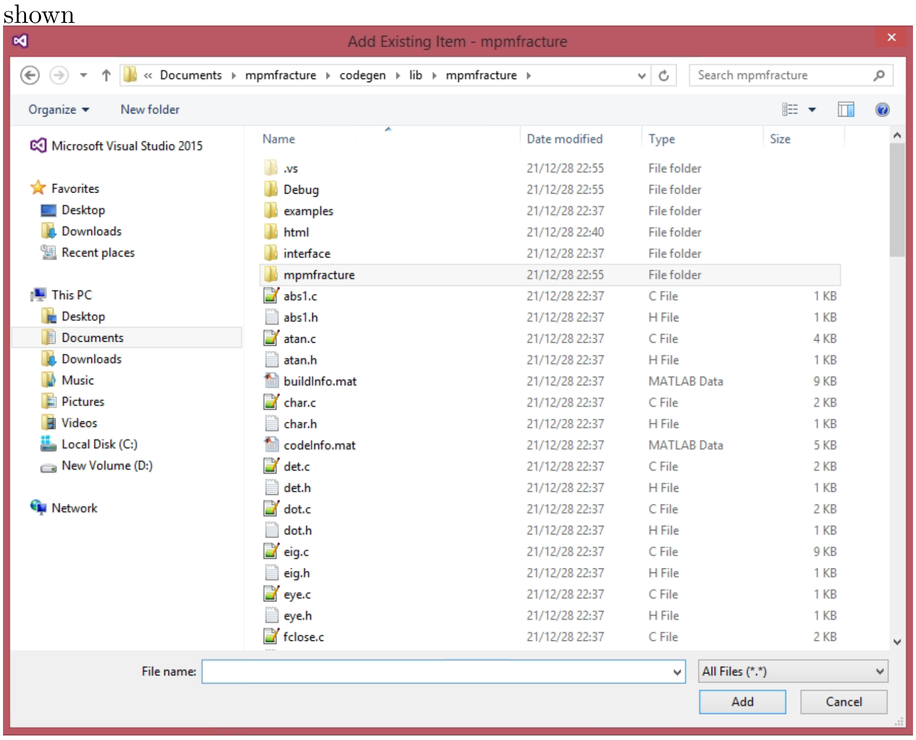
5. Change the two selectors above to *Release* and *x64*



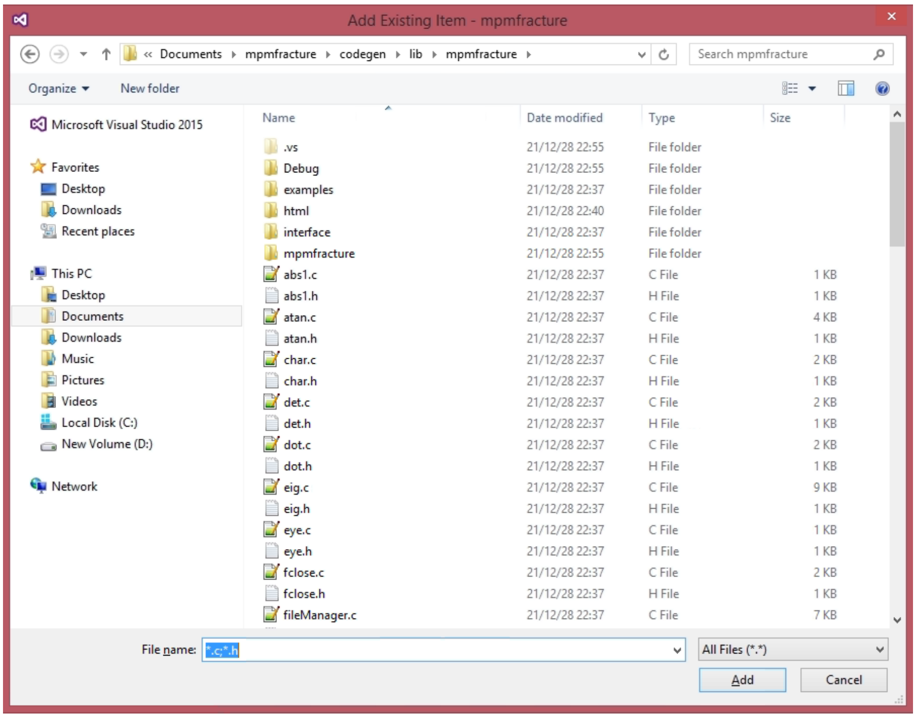
6. Go to *Project*, *Add Existing Item*



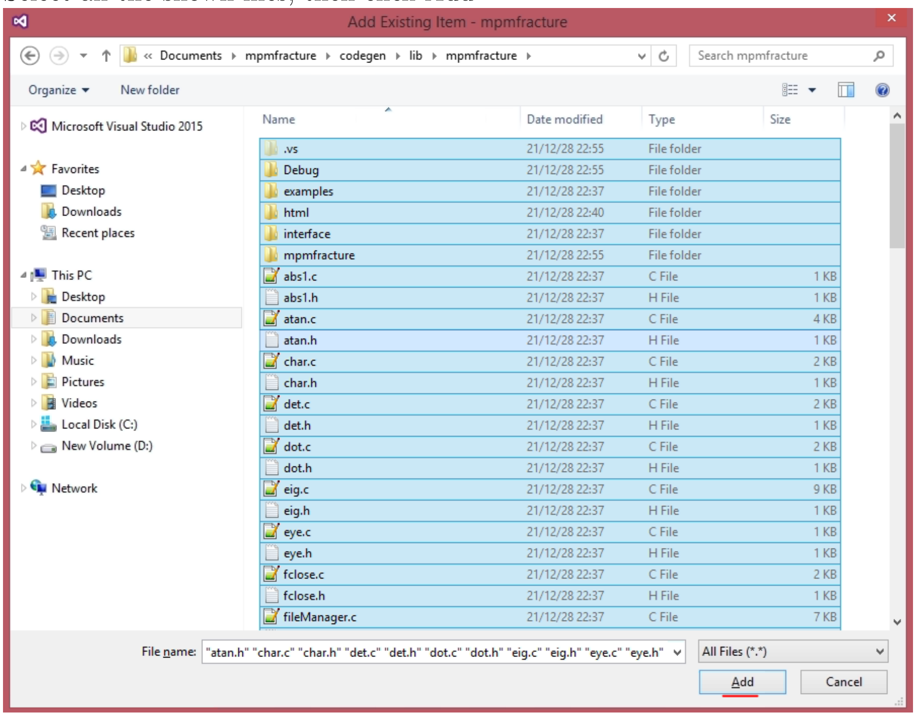
7. Go to the `mpmfracture\codegen\lib\mpmfracture` directory, a bunch of .c and .h files should be shown



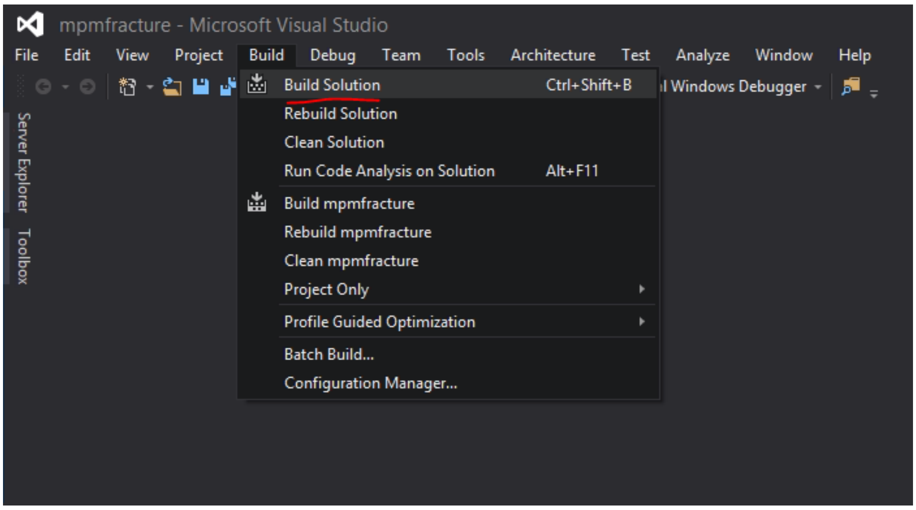
8. Type **.c;*.h* in the file name, then press enter (to show only .c and .h files)



9. Select all the shown files, then click Add



10. Go to *Build*, *Build Solution*, then wait until the build process is finished



11. After the build is finished, the executable is located in `mpmfracture\codegen\lib\mpmfracture\x64\Releasempmf` copy the executable to the root directory (`mpmfracture` folder)
12. Make sure `SimOutput` folder exists in the root directory
13. Run `mpmfracture.txt`, the output should be many .txt files in SimOutput directory
14. At any point of the simulation, the result can be printed by running `mpmfractureplot.m` in MATLAB. A new folder will be created within `SimOutput` containing printed frames of some timesteps.

5 `release` Versions

5.1 Current

- cleaner source code, same functionality
- no more warnings from MATLAB Coder (R2017b)

5.2 Commit Hash `fbda4ee2ac218c585358db592e3cd42e62494d67`

- support crack initiation, propagation, merging in elastic medium
- cohesion is inaccessible
- ability to create notches via particle deletion

5.3 Commit Hash `89884f2e7cea72df665e06d68f8b5be2a8fcbbaf`

- coder compliant
- no access to cracking features
- input file as `input.txt`

5.4 Commit Hash `cddb217154dcc657114a0b9c2850942a2f57f80e`

- not coder compliant
- parfor error handling via MATLAB run
- input file as `input.mat`

Appendix: Interacting with the github repository

The github repository is:

`https://github.com/titoadibaskoro/mpmfracture`

Only clone from `release` branch. For pushing, please use `dev` branch.

To clone `release` branch of the repository, type:

```
git clone -b release https://github.com/titoadibaskoro/mpmfracture.git
```

To create new branch, type:

```
git branch <new branch name>
```

To pull remote branch that is currently active locally:

```
git pull origin <branch name>
```

Typing <branch name¿ different from the current active branch may result in unexpected merging.

To pull remote branch that doesn't exist locally:

```
git fetch origin <branch name>:<branch name>
```

To switch branch:

```
git checkout <branch name>
```

To go to any commit hash:

```
git checkout <commit hash>
```

To drop any uncommitted chances:

```
git checkout .
```