

Fortran wrapper to Python library

1. GUI requirements

Requirements needed in order to use the application `fparser_GUI` to convert fortran modules to python libraries:

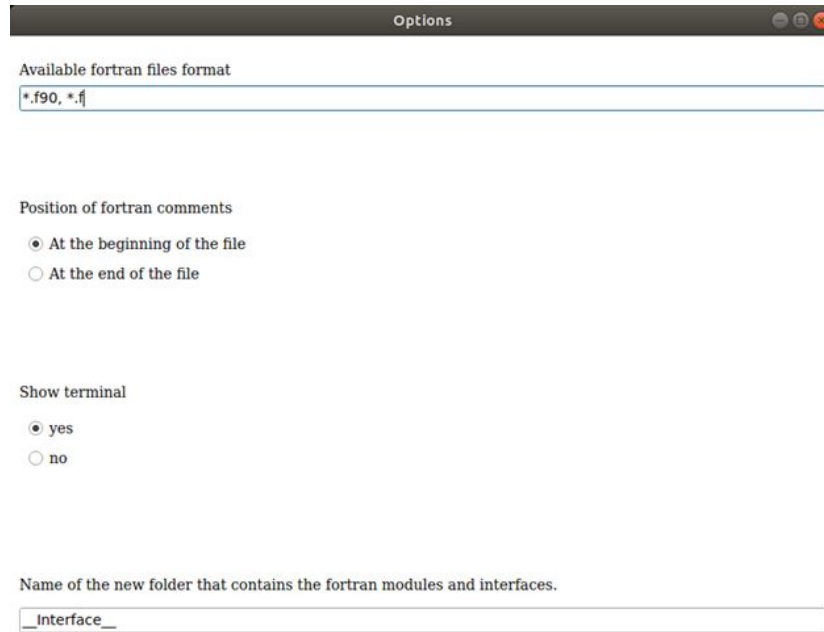
- Operating system:
 - a) Windows
 - b) Linux
- Download the file `fparser_GUI.exe` (Windows version) or `fparser_GUI` (Linux version) from Github. These files are inside a folder, which contains two more files that are also needed, so it is advisable to download the whole folder.
- Python and NumPy must be installed.
- Fortran compiler:
 - a) Intel Visual Fortran Compiler for 64-bit apps (`intelvem`)
 - b) GNU Fortran 95 compiler (`gfortran`)
- A fortran compiler must be found on the machine main search path. To check if it is on said path, run the executable in step 2 and click on search for fortran compiler on the second page. Alternatively manually run the following command on a python virtual environment with numpy installed: `"f2py -c --help-fcompiler"`. A long list should pop up with a statement such as this one stating if it has found a compatible fortran compiler.

```
Fortran compilers found:
--fcompiler=intelvem Intel Visual Fortran Compiler for 64-bit apps
(19.0.5.281)
```

2. Execute `fparser_GUI`

- a) Open the terminal and look for the folder which contains `fparser_GUI` using the command: `cd path/exec_folder/`
- b) If Python and NumPy are installed inside a virtual environment, it is necessary to activate it.
- c) Run the application with the following command and a window should pop up:
 - Windows: `fparser_GUI.exe`
 - Linux: `./fparser_GUI`
- d) Select the options for the compilation by clicking on `file->options` and a window should pop up:
 - i) Files types to search for

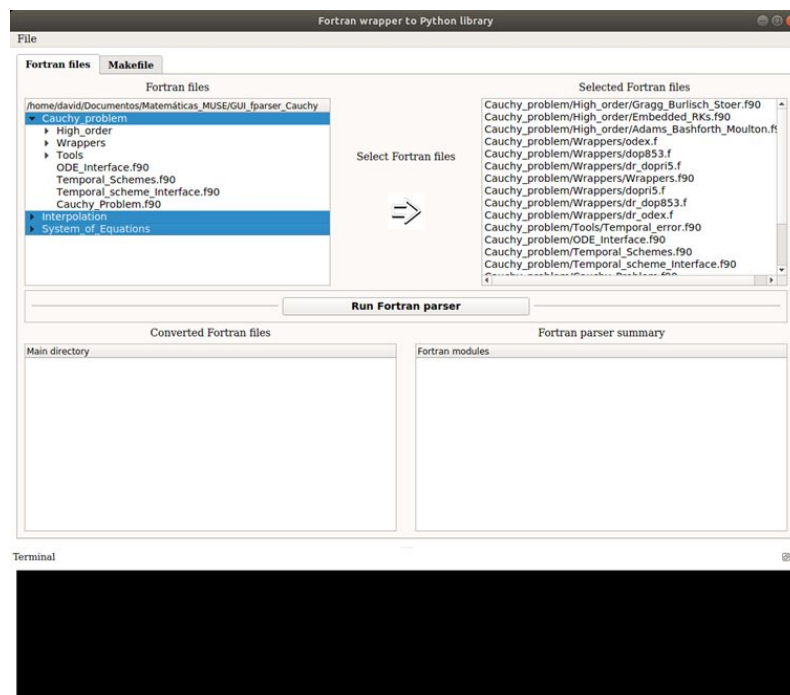
- ii) Whether the description of functions is above their declaration or below it.
 - iii) Whether you wish for the terminal with the outputs to be shown or not.
Note: All errors will be shown in this terminal.
 - iv) Name of the file in which the library will be generated.
- e) Select to
- by the
- f) Select files and
- g) Select within you from click
- which files open either directly or choosing folder. the fortran you wish to compile click on run fortran parser. the modules those files wish to access Python, ok.

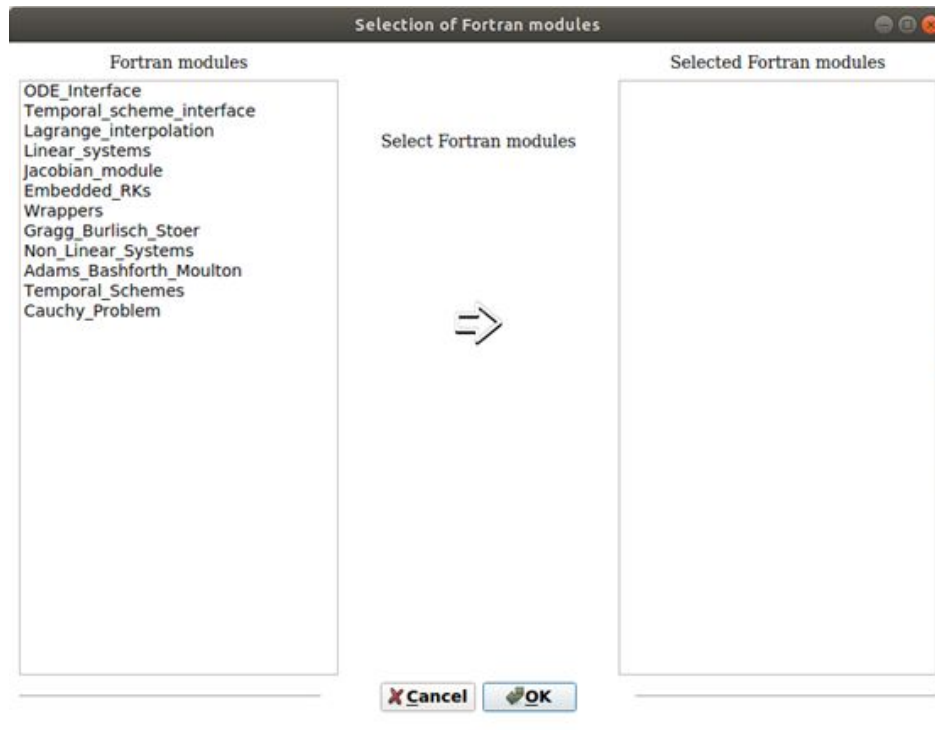


The 'Options' dialog box contains the following settings:

- Available fortran files format:** *.f90, *.f
- Position of fortran comments:**
 - ☒ At the beginning of the file
 - ☐ At the end of the file
- Show terminal:**
 - ☒ yes
 - ☐ no
- Name of the new folder that contains the fortran modules and interfaces:** _Interface_

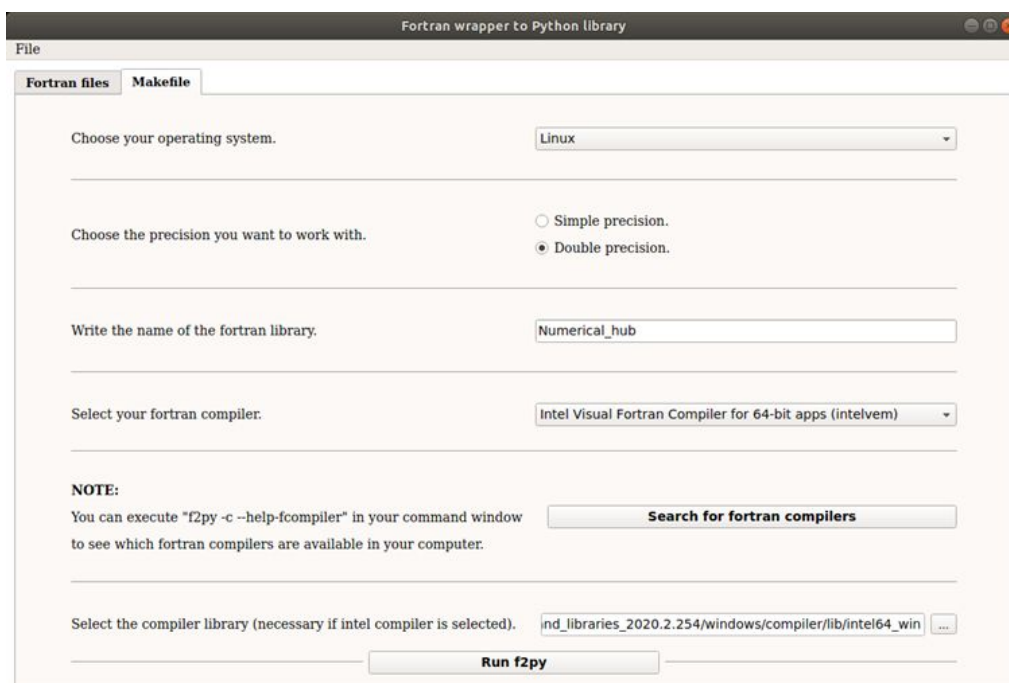
Cancel OK





h) Go to the makefile tab and select the following options before clicking run f2py:

- i) The Operating system of your machine.
- ii) Whether you wish to work with double real precision or simple.
- iii) The name of the library you wish to generate.
- iv) A fortran compiler in your machine.
 - If an intel compiler is selected you may need to add the path to the compiler, it may be found in a directory such as this one.
`\IntelSWTools\compilers_and_libraries\windows\compiler\lib\intel64`



- i) If every process is successful, a folder called `__interface__` (default name which can be changed through the GUI options) will be created in the same path of the executable file. Inside `__interface__` the libraries needed to import the library from Python are:

- `Library_name.py`
- `Library_namef.pyd` (Windows) or `Library_namef.so` (Linux)

*Note: `Library_name` can be changed through the GUI, inside the tab "Makefile".

3. Python user notes

- **Import:** To connect to the library simply use `import library_name` and to run a certain function use: `library_name.module_name.function_name(args)`. Note: The python interface and the compiled library must remain in the same directory.
- **Arrays:** To pass an array to the interface it needs to be defined as a fortran array by **numpy** and to be defined as its equivalent python type of data. Example: `A=numpy.asfortranarray([[2, 1], [1, 2]], dtype=numpy.double)` will produce a type `real(8)` matrix 2x2 that can be passed to the interface.
- **Inputs:** All inputs to the interface function must be previously defined as with their corresponding dimensions as the example above, even if they will be rewritten by the function. Dimensions must correspond or an error will appear.
- Fortran functions will return a result while Subroutines will just edit the inputs if they happen to be rewritten.

4. Wrapper limitations

Fparser has been tested by parsing many complex fortran files but there are some limitations to its use. Those that have been found are listed below.

- Fparser needs a set of parentheses on all Subroutines and Functions declarations, even if there are no arguments within.

```
subroutine Systems_of_Equations_examples      Not Valid
```

```
subroutine Systems_of_Equations_examples() Valid
```

- Fparser is not able to parse Fortran objects declared such as those declared with "type".
- Naming a module, function or library after a reserved python command will result on the python interface not working. Note: Variable names are automatically changed if there is a coincidence.
- Fparser is not able to process Fortran files that contain a "module procedures" type declaration.
- Fparser will not be able to aid f2py to compile any subroutines or functions in which the dimensions of the input or output variables are defined with other variables that are not in its input. (Assume shape variables are processable with fparser).

- Subroutines and Functions that contain two or more procedures as inputs, and one if these procedures call on another as an input will be compilable. However, these will not work from the python interface. F2py is able to process python functions into fortran but not the other way around.
- Functions that depend on global variables with other functions that change them will remain unchanged in Python.
- A wrapped library may only be able to run on the machine that compiled it.

5. Developer Instructions

To run on the same python virtual environment as the other developers it is important to install the fparser environment found on the main GitHub project and to develop the program either in VS or VS Code. This environment may be installed with Anaconda/Miniconda, Visual Studio or other IDE that allows virtual environment installations. Note: The same environment name is not necessary.

a) Anaconda/Miniconda:

- Run `"conda env create -f /path/environment.yml"`, where `/path` is the absolute or relative path to the environment.yml file.

b) VS:

- Select add python environment from visual studio.
- Select conda environment.
- Change the name to fparser or other names.
- Select environment file and search for the file.
- Click on create.

Both methods are equally valid and it should take a couple of minutes to install all the necessary libraries.

Contents of the environment.yml:

```
name: fparser
channels:
  - anaconda
  - defaults
dependencies:
  - python=3.8
  - pip
  - numpy
  - pytest
  - qt
  - pip:
    - pyqt5
    - pyinstaller
```