# A Quick Guide on How to Use the Fortran-to-Python (F2PY) Module

Magnus H-S Dahle

February 2015

**Abstract**

If you're familiar with Python and Fortran, and wish to minimize reading: **Linux** users may jump to section 4.2, while **Windows** and **MAC OS** users to 4.1. You should in addition be aware that `F2PY` also works for wrapping C-code.

## 1 Introduction

The choice of programming language can be extensively difficult, especially considering the computational effectiveness against implementation time and effort. While scripting languages like **MATLAB** and **Python** may provide intuitive code which is fast to implement, compiled languages like **C/C++** and **Fortran** yield superior computational speed. Within this tutorial, we'll show that it is in fact possible to obtain the very best of both worlds. At least to some extent.

In the following example, we'll make use of the Fortran to Python package `F2PY` , in which enables a Python script to directly call a readily compiled Fortran module. This is a powerful tool giving us the possibility for developing easy written code in Python, but leaving more computational demanding parts of our script to more capable Fortran routines.

## 2 Installation

One should first note that `F2PY` is actually part of the `NUMPY` [1] (Numerical Python) package of Python. Thus, having successfully installed `NUMPY` should imply also having a functional `F2PY` -module. If your computer is running on a **Linux** distribution, using `F2PY` is rather straight forward and easy going, as Python with `NUMPY` is almost always integrated in the OS. Linux users may check if `F2PY` is already installed by just opening a terminal (by pressing $[CTRL] + [ALT] + [T]$) and then type the command `f2py` (with small letters). If it is installed, a short user manual will be printed in the terminal. If not, the terminal will complain that there exists no such command. For Linux users, `F2PY` is installed by installing `NUMPY` from official repositories. Evidently, this is also valid for computers running **Windows** or **MAC OS**! That is, download and install the `NUMPY` package. It is easy to find on the web, but if you're having issues, you may want to take a look at the solved problem given by the following footnote[2].

**NOTE:** If one struggles to make `F2PY` run correctly on one's own computer, there are plenty of machines at the university (NTNU) running **Linux Ubuntu**, where `NUMPY` and `F2PY` are both installed!

## 3 When should You Use `F2PY` ?

This is perhaps the ultimate question, and there is no definite answer. A good rule of thumb however, is to use `F2PY` , or compiled languages in general, when considering (nested) loops. Possibly the most typical example would be operations on elements in multidimensional matrices. Other good examples could be programs calculating integrals or conducting Monte Carlo Simulations. You might wonder if anyone has already made `F2PY` -modules before you, and the answer is most likely yes! Most of the functions and routines found in `NUMPY` and `SCIPY` are actually compiled Fortran routines which provides highly efficient and fast(!) solvers for multiple problems. Thus, we advice you to always check if one of these two modules already have a routine in which may be suitable for your problem. If not, then implement your solver in pure Python to see if efficiency really is an issue. If it is, then `F2PY` may possibly provide the best solution for your problem.

[1] or more precisely: `numpy.distutils`.

[2] `http://scientificcomputingco.blogspot.no/2013/02/f2py-on-64bit-windows-python27.html`

# 4 Usage and Syntax by a Marginal Working Example

In the following, we present a short example on how to use the `F2PY` module. For details and references, we strongly encourage you to take a look at the module's official documentation[3]. The `F2PY` module effectively compiles and incorporates Fortran code into a new Python module[4], readily importable to any Python script. We will next give an working example and obviously, we'll need some Fortran code! Furthermore, we'll apply **Fortran 90**, and for simplicity, we'll use the following, banal and trivial Fortran routine

```
1  ! This is the content of the file: fortran_file_name.f90
2
3  SUBROUTINE Fortran_Sum(A,B,Su)
4
5      real(8) :: A, B, Su ! Real, double precicion variables
6
7  ! The Fortran Compiler ignores all lines starting with '!'
8  ! The 'f2py' utilizes this fact by defining its own special
9  ! command line as '!f2py', such that this is ignored by
10 ! Fortran, but used by Python to distinguish in/output variables
11 ! Notice the use of small cased letters ..
12
13 !f2py intent(in) a
14 !f2py intent(in) b
15 !f2py intent(out) Su
16
17     Su = A + B
18
19 END SUBROUTINE
```

**NOTE:** that older versions of fortran (e.g. 77), uses file extension `.f`, not `.f90`, but even more crucial; the comment-character may be different! For Fortran77, it is an upper case '`C`', while for more modern versions of Fortran (e.g. 90) it is an exclamation mark '`!`'. This ultimately also affects the `!f2py` → `Cf2py` command. You may witness examples of this if you search for more tutorials elsewhere.

Finally, we want to compile this Fortran Subroutine and incorporate it into a python module to be imported in a python script.

- Create your Fortran file, `fortran_file_name.f90`. It may contain multiple subroutines.
- Choose a name for the Python module you are about to create, `chosen_module_name`.
- Create a directory in which you want to work in and move/copy-paste the Fortran file there.

Linux users may skip the next section, and go straight to section 4.2 in which explaines a somewhat easier Linux-approach.

## 4.1 Creating the Python Module from Fortran Code - The General Method

The following method uses/runs nothing but a pre-written Python script and is thus eligible for all users regardless of their Operating System.

One needs the arbitrarily named Python script `build_f2py_modules.py` which is given below and uses `numpy.distutils` to generate a new Python module from the Fortran code file.

[3] `http://docs.scipy.org/doc/numpy-dev/f2py/">documentation`

[4] Equivalent to `NUMPY` , `SciPy`, etc.

```
1
2  # The content of build_f2py_modules.py
3
4  from numpy.distutils.core import Extension
5
6  ext = Extension(name     = 'chosen_module_name',
7                  sources = ['fortran_file_name.f90'] )
8
9  if __name__ == "__main__":
10     from numpy.distutils.core import setup
11     setup(name = 'f2py_example',
12           ext_modules = [ext]
13           )
```

Running this Python script will result in the creation of the new directory `build/lib.linux-x86_64-3.4/`, where you will find the newly generated file `chosen_module_name.cpython-34m.so`. **This is a Python module**, to be imported equivalently like any other module within a python script, e.g. `NUMPY` or `SCIPY`!

## 4.2 Creating the Python Module from Fortran Code - The Easy Life of the Linux User

Having installed `F2PY` , you can call it directly from the terminal,

```
working_directory % f2py -c -m chosen_module_name fortran_file_name.f90
```

which will create the file `chosen_module_name.so`. **This is a Python module**, to be imported equivalently like any other module within a python script, e.g. `NUMPY` or `SCIPY`!

## 4.3 How to Use the Module You've Just Created

Move/copy-paste the Python module file you just created to a directory you wish to work in, Start a live Python console (or simply create a script) and import whatever modules you'd like, including your own self made. Print your self made module's documentation to see if it was imported correctly.

```
>>> import numpy
>>> import chosen_module_name
>>> print( chosen_module_name.__doc__ )
```

Which should result in,

```
This module 'chosen_module_name' is auto-generated with f2py (version:2).
Functions:
  su = fortran_sum(a,b)
.
>>>
```

**NOTE:** that in the process of creating the python module from the Fortran code, upper case letters are transformed to lower case. This is not that much of a big deal, however, you should note that Fortran is **not** case-sensitive. That is, variable/function names like `aAA`, `aaa`, `AAA` are equivalent and non-distinguishable. This is however not the case in Python.

Nevertheless, the function may now be called similar to any other python function/routine

```
>>> a = numpy.linspace(0,1,10)
>>> A = chosen_module_name.fortran_sum(1.2,4.2)
>>> print(A)
 5.4
>>>
```

**Hot Insider Tip:** If you're planning on using `F2PY` quite a lot, you might end up with a large amount of self made Python modules. We therefore suggest that you create a permanent library directory to store *all* modules you create this way. Thus, you can make a static path thread to import all your glorious modules from, whenever you need them. This way, you'll save a lot of space compared to having a copy of the module in every project you conduct.

# 5 Where to Go Next!?

As a starter, try to implement your own modules, which may also include multiple subroutines! We would also like to encourage you to redo the example, but this time with the source code written in **C** instead of **Fortran**. This tutorial was meant to be as short as possible, and while we do not necessarily claim to have succeeded on this point, we have given a marginal working example but neglected much of `F2PY` 's functionality and applications.

Hence, we'd like encourage you to take greater look in its details and documentation! We suggest you to try to write a generic integration routine (in fortran), in which takes a general (one-dimensional) function and integration boundaries as arguments and returns the integral. Nested loops are known to be an Achilles heel for all scripting language. You can also take a look at this numerical integration example[5] which explicity shows how powerful a tool F2PY can be, by reducing the total running time of a pure Python script by a factor of 300 by using F2PY.

If you have any comments, suggestions or even corrections you wish to add, please feel free to send me an email, `magnud(at)stud.ntnu.no` or `mhsd91(at)gmail.com`.

[5] `http://folk.ntnu.no/magnud/Projects/F2PY_integration.html`