

Créer un package Python

9 novembre 2018

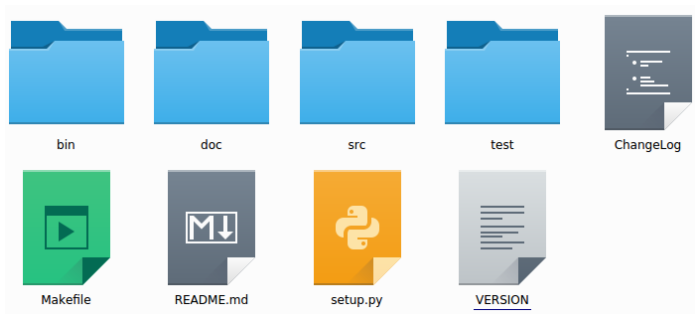
Antoine Dumas
dumas@phimeca.com

Du script vers le package...

- ❶ Il faut **regrouper vos fonctions et classes** Python dans un ou plusieurs **modules** (fichiers), eux-même regroupés dans un dossier commun pour créer un **package**. Un package peut contenir d'autres packages contenant d'autres modules dans des sous-dossiers. Exemple `import numpy.linalg`, `linalg` est un package du package `numpy`.
- ❷ Dans chaque dossier, il faut ajouter un fichier `__init__.py`, cela indique à Python que ce dossier est un package et on pourra alors **importer** les objets contenus dans les modules.
- ❸ Le fichier `__init__.py` peut être vide mais il peut également contenir du code pour **initialiser** votre module (import direct de fonction, test de version des dépendances, ...). Le code contenu dans ce fichier est appelé lors de l'import du package.

... installable

- ❏ Afin d'utiliser votre package comme n'importe quel package Python, il faut pouvoir l'**installer** dans votre distribution Python.
- ❏ L'outil le plus commun est d'utiliser le paquet Python **setuptools**.
- ❏ Il faut créer un **fichier setup.py** contenant des informations sur le package que vous souhaitez installer. Mais d'autres fichiers peuvent venir en complément.
- ❏ L'**arborescence** de votre package est importante.



L'arborescence



- ❏ Le dossier `src` contient l'ensemble des modules et si besoin des sous packages.
- ❏ Le dossier `doc` contient la structure de la documentation, pouvant être compilée avec `sphinx`.
- ❏ Le dossier `test` contient les tests unitaires du module.
- ❏ Des dossiers supplémentaires peuvent exister suivant les besoins du développement (`bin` contient des exécutables ici, `data` pour des fichiers de données, ...). Le but est d'avoir une organisation claire et structurée.

L'arborescence



Les différents fichiers :

- ④ `setup.py` : un fichier de code Python qui est notre **installateur** (obligatoire).
- ④ `README` : un fichier décrivant notre module (fortement conseillé)
- ④ `VERSION` : ce fichier contient le numéro de version de notre package. Afin de faciliter sa mise à jour, il est plus simple de récupérer le numéro contenu dans le fichier partout où il est nécessaire de renseigner la version (notamment dans `setup.py`).
- ④ D'autres fichiers :
 - ▶ `MANIFEST.in` : instructions pour inclure ou exclure des fichiers à installer.
 - ▶ `ChangeLog` : un fichier explicitant les changements majeurs ou corrections de bug entre les différentes versions.
 - ▶ `Makefile` : ce fichier est généré par sphinx et permet de compiler la documentation.

Le fichier setup.py

Exemple de fichier setup.py :

```
#!/usr/bin/env python3
#-*- coding: utf-8 -*-

from setuptools import setup
from distutils.version import LooseVersion
import openturns as ot

# check the version of openturns
ot_version_require = '1.12rc1'
if LooseVersion(ot.__version__) != ot_version_require:
    raise Exception('Version of openturns must be : {}, found {}'.format(ot_version_require, ot.__version__))

# load the version from the file
with open("VERSION", 'r') as fic:
    version = fic.read()
```

Le fichier setup.py

```
# set the parameter of the setup
setup(name='quantile', # define the name of the package
      version=version,
      description='Module for quantile estimation methods',
      author='Antoine Dumas',
      author_email='dumas@phimeca.com',
      # define some scripts as executable
      scripts=['bin/launch_sampling.py'],
      packages=['quantile'], # namespace of the package
      # define where the package "quantile" is located
      package_dir={'quantile':'src'},
      test_suite='test', # subclass for unittest
      # some additional data included in the package
      data_files = [(('.',), ['VERSION']),
                    ('bin', ['bin/Loads_VEB.blk.template'])],

# List of dependancies
      install_requires= ['numpy>=1.13.3',
                          'matplotlib>=2.0.2',
                          'pandas>=0.22.0',
                          'joblib>=0.12']
    )
```

Installation

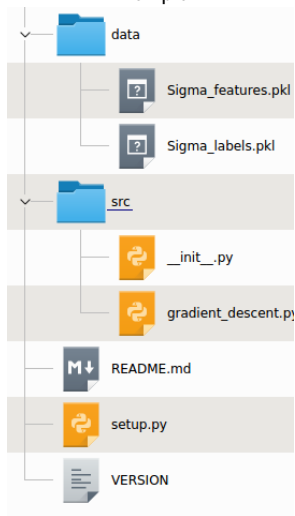
Plusieurs commandes sont possibles pour installer le module :

- ❏ `python setup.py install`
- ❏ `python setup.py install --user` : installation en mode utilisateur
- ❏ `python setup.py develop` : installation en mode développement, ceci crée un lien symbolique entre votre dossier source du package et le dossier d'installation.
- ❏ De nombreuses autres commandes existent pour créer votre package, pour les afficher : `python setup.py --help-commands`.

Travaux pratiques : créer votre package

- ❏ Créer un dossier avec le nom de votre package.
- ❏ Créer un dossier `src` dans votre dossier package : mettre votre module contenant la classe de descente de gradient ainsi qu'un fichier `__init__.py`.
- ❏ Dans le dossier du package : ajouter les fichier `setup.py`, `README.md` et `VERSION`.
- ❏ Dans le fichier `setup.py` : définissez les noms correspondant à votre module, ajouter des données supplémentaires si besoin, lister les dépendances s'il y en a.
- ❏ Installer le module avec la commande `python setup.py install --user`.
- ❏ Importer votre module dans jupyter-notebook et réaliser un calcul.

Exemple:



La documentation avec Sphinx

- ❏ **Sphinx** est un module Python qui permet de créer une documentation sous divers format dont html et pdf.
- ❏ Pour commencer il suffit d'installer sphinx et d'exécuter la commande **sphinx-quickstart**. Il faut ensuite suivre les instructions pour créer automatique les fichiers de configurations.
- ❏ Des modules supplémentaires peuvent être nécessaires pour compiler votre documentation comme **numpydoc**, **nbconvert** **sphinx-argparse**, ...
- ❏ Un exemple vous est fourni avec la correction des travaux pratiques.