

UNIVERSITÉ DE NEUCHÂTEL

TRAVAIL DE BACHELOR

Hypy - Un module Python pour l'interprétation des essais de pompage

Auteur :

Loïc PIANARO

Superviseurs :

P. RENARD

et H. MERCIER

17 décembre 2017

Table des matières

1	Introduction	2
2	Contexte détaillé	3
2.1	Disponibilité	3
2.2	Hytool	3
2.3	Tableau des fonctions disponibles	6
2.4	Choix de Python	6
3	Aspects techniques du portage	7
3.1	Objectif	7
3.2	Structure du package	7
3.3	Comment construire un package	8
3.4	Validation du code Python	9
3.5	Distribution du code	10
3.6	Comment étendre Hypy	11
3.7	Utilisation de Git	12
4	Guide utilisateur	13
4.1	Installation	13
4.2	Comment utiliser le code actuel	16
5	Conclusion	19

1 Introduction

Le Bachelor en Systèmes Naturels se termine par un projet personnel, qui consiste en un travail qui peut s'effectuer sous forme de stage en entreprise, étude d'un problème concret ou projet dans un laboratoire. Ce dernier doit être effectué dans l'option choisie par l'étudiant, ici l'informatique. Dans ce cas, il a été décidé, avec le Professeur Philippe Renard, d'être effectué au centre d'Hydrogéologie de l'université de Neuchâtel. Il consiste en la traduction de Hytool, une toolbox codé en Matlab, en Hypy. La différence vient du fait qu'Hypy est codé en Python. Ce travail s'effectue sous la direction du Prof. Renard, créateur de la toolbox et d'Hugues Mercier, maître-assistant à l'Université de Neuchâtel, à l'institut d'informatique.

Hytool est une toolbox qui a été développé dans l'environnement Matlab pour l'interprétation de tests hydrauliques dans des puits. Cette toolbox contient des solutions analytiques utilisées pour décrire l'écoulement souterrain autour des puits, ainsi que des fonctions pour importer des données et ensuite afficher et ajuster un modèle à ces données. [1] Cette toolbox est limitée dans son utilisation par les licences de Matlab. Pour remédier à ce problème, une solution consiste à traduire cette toolbox dans un autre langage de programmation, qui serait libre d'accès.

Ainsi, le but du projet présenté dans ce rapport est de traduire une partie de cette toolbox en Python, afin d'en faciliter l'accès aux utilisateurs. Python est un langage de programmation sous licence libre, qui fonctionne sur toutes les plate-formes informatiques et qui offre des outils de haut niveau particulièrement adaptés à ce type de travaux d'analyse de données. À cause du temps limité, il a été décidé de se focaliser sur plusieurs fonctions clés, en rapport avec les différentes méthodes d'analyses : Theis interpretation (ths), Theis no-flow interpretation (thn), Theis in a confined aquifer interpretation (thc), Drawdown in the well interpretation (pcw), Boulton interpretation (blt) et Warren and Root interpretation (war).

Le rapport est structuré en trois parties. La première partie du rapport contient une description détaillée du contexte dans lequel le travail a été réalisé. Cette partie comprend une description de la structure actuelle d'Hytool et il y sera également question du choix de Python. La deuxième partie présente les aspects techniques principaux concernant le portage entre Matlab et Python. Cette partie décrit les modifications apportées afin que la nouvelle toolbox (ou package) soit compatible avec Python et la validation du code. Enfin, la dernière partie du rapport contient un guide destiné à l'utilisateur.

2 Contexte détaillé

2.1 Disponibilité

Hytool est disponible en ligne, sur Github à l'adresse suivante : <https://github.com/UniNE-CHYN/hytool>. La toolbox est en libre accès et peut facilement être téléchargée et installée sur Matlab. Cependant, il faut posséder une licence pour utiliser Matlab (ou y avoir accès via une entreprise ou une université). En se connectant à Github, il est ainsi facile de voir comment elle est organisée et de comprendre les exemples cités ci-dessous.

2.2 Hytool

Pour un certain modèle analytique, Hytool est organisé en groupes de fonctions. Pour chaque groupe, la syntaxe du nom de la fonction est toujours la même : `nom_utilité`. Par exemple, pour la modélisation de Theis, chaque fonction commencera par le nom `ths` et sera suivi par son contenu (en anglais et raccourci). Si le but est d'estimer rapidement (guess) un paramètre, la fonction s'appelle `ths_gss`. Pour effectuer une démonstration de la fonction elle se nomme `ths_dmo`. Chaque fonction possède ses propres paramètres et sur matlab, afin d'appeler et d'exécuter la fonction, il suffit d'écrire dans la commande `_utilité(paramètres)`. Si les paramètres d'entrées sont inconnus, il est possible, sous matlab, d'appeler l'aide dans la fenêtre de commande en tapant `help ths_gss` (pour cette fonction spécifique).

En plus des groupes, Hytool possède également des fonctions génériques. Elles sont utilisées de deux manières différentes : appelées directement dans la fenêtre de commande ou à l'intérieur d'une autre fonction générique ou d'un groupe. leur nom fait également référence à leur utilité. Par exemple, afin de charger vos données, il faut appeler la fonction `ldf(nomdufichier.extension)` qui fait référence à `load file`. Pour effectuer un premier diagnostic de données, la fonction "diagnostic(paramètres)" sera utilisée. La fonction `diagnostic` effectue des calculs différentiels. Pour cela, elle appelle `ldiff(paramètres)`.

Le code sous matlab est donc très bien structuré. Les fonctions génériques sont indépendantes des fonctions en groupe et chaque groupe est également autonome (sauf dans des cas extrêmement précis). Cela amène de nombreux avantages, comme la correction du code ou l'ajout d'une nouvelle fonction. Il est également facile d'effectuer une traduction de matlab à un autre langage, car il suffit de coder les fonctions génériques et ensuite de programmer les groupes de fonctions, comme `ths`, les uns après les autres pour créer une toolbox fonctionnelle.

Afin d'expliciter ces propos, le tableau 1 présente la structure d'un groupe de fonctions, en l'occurrence `ths`.

TABLE 1 – Liste des fonctions associées à la solution analytique de Theis pour un écoulement radial convergent en deux dimensions dans un aquifère confiné.

<code>ths_der</code>	Derivative of the Theis Function in Lapalce domain
<code>ths_dim</code>	Compute drawdown with the Theis (1935) solution
<code>ths_dls</code>	Dimensionless drawdown of the Theis model
<code>ths_dmo</code>	Theis (1935) interpretation
<code>ths_drw</code>	Type cruves of the Theis model
<code>ths_gss</code>	First guess for the parameters of the Theis model
<code>ths_jac</code>	Jacobian matrix of the Theis function
<code>ths_lap</code>	Theis Function in Laplace domain
<code>ths_rpt</code>	Reports graphically the results of a pumping test

Ci-dessous se trouve un tableau avec les modèles existants et s'ils ont été traduit ou non au moment de l'écriture du rapport.

TABLE 2 – Liste détaillée des fonctions de Hytool avec leur statut dans Hypy
Légende : V = traduit, F : traduit, mais non-testé, X = non-traduit, O = partiellement traduit

Model Names		
Interference tests :		Statut :
ths	Theis (1935) -Ideal confined aquifer	V
thc	Theis (1941) -Linear constant head boundary	V
thn	Theis (1941) -Linear no flow boundary	V
pca	Papadopulos and Cooper (1967) -Large diameter well : 1 storativity	X
pcb	Papadopulos and Cooper (1967) -Large diameter well : 2 storativities	X
aga	Agarwal et al (1980) -Large diameter well with skin	X
jcb	Cooper and Jacob (1946) -Ideal confined aquifer	O
del	Delay (2004) -Fractal with anomalous diffusion	X
grf	Barker (1988) -General Radial Flow	X
htj	Hantush (1956) -Leaky aquifer	X
war	Warren and Root (1965) -Double porosity	F
blt	Boulton (1963) -Unconfined aquifer with delayed yield	V
grg	Gringarten and Ramey (1974) -Vertical fracture of infinite conductivity	V
Single well test :		
pcw	Papadopoulos and Cooper (1967) -Large diameter well	V
waw	Warren and Root + Wll-bore storage -large diameter well + double porosity	X
eha	Eden-Hazel (1980) -Step drawdown test -Quadratic head losses	X
Variable rate interference tests :		
tmr	Theis (1935) -Stepwise variable rate	X
tmc	Theis (1941) -Linear constant head boundary with stepwise variable rate	X
Slug and other tests :		
cls	Copper et al. (1967) -Slug test confined aquifer	X
nsi	Neuzil (1982) -Pulse test confined aquifer	X
jlq	Jacob and Lohman (1952) -Constant head test	X
dsq	Doe and Geier (1990) -Constant head test with skin	X

2.3 Tableau des fonctions disponibles

À Partir de ces modèles, voici une liste plus détaillée de ceux qui ont été traduits :

TABLE 3 – Liste des fonctions traduites

Fonctions/Modules	Etat	Validé(e)s	Fonction(s) à coder
Fonctions génériques	Complet à 80%	Oui	dehoog, hycoop, hyerr, hyfilter, hyplot, hysampling, hyselect, hyver
Module ths	Complet à 100%	Oui	-
Module thn	Complet à 100%	Oui	-
Module thc	Complet à 100%	Oui	-
Module pcw	Complet à 95%	Oui	draw
Module blt	Complet à 95%	Oui	draw
Module war	Complet à 95%	Non	draw
Module jcb	Complet à 40%	Non	dim, dmo, rpt

Lorsqu’une fonction ou un module est validé, cela signifie deux choses : le résultat a été comparé avec la version de Matlab et le prof. Renard l’a approuvé. Les fonctions qu’il reste à coder n’ont pas été faites simplement par choix. Le prof. Renard a estimé qu’elle n’était pas nécessaire pour le bon fonctionnement de Hypy et qu’il était préférable d’utiliser le temps à disposition pour avancer dans la traduction.

2.4 Choix de Python

Avant de commencer le projet, il a fallu choisir le langage de programmation. Il était impératif qu’il soit dans un environnement libre. De ce fait, plusieurs options ont été envisagées : C, C++, Julia, Python et R notamment. Tous ont leurs points positifs et négatifs. C reste l’un des langages les plus utilisés, il possède de ce fait une littérature fournie et beaucoup de forums dédiés à la résolution de problèmes. De plus, il permet une maximisation de la performance grâce, par exemple, à l’utilisation de pointeurs. Le C++, par sa programmation orientée objet, était également un bon choix. Sa bibliothèque standard est en grande partie basée sur celle du C. Julia est un mélange, entre autres, de Matlab, R, Python. Il correspond aux exigences pour traduire une toolbox qui vient de Matlab. Cependant, étant relativement nouveau, il manque de littérature et d’aide pour résoudre d’éventuelles difficultés. R aurait également été pertinent, mais n’a pas été pris en considération de manière purement subjective. Finalement, le choix s’est porté sur Python. Ayant déjà été utilisé pour d’autres travaux, il avait cet avantage sur les autres et était également la suggestion du prof. Renard.

Python est facile à prendre en main et puissant. Il possède beaucoup de fonctionnalités directement intégrées dans le langage. De plus, il est open source, ce qui signifie qu'il ne faut pas posséder de licence pour l'utiliser. Enfin, Python contient de nombreuses librairies qui facilitent grandement le codage. Cela commence par les basiques comme Math, qui possède toutes les fonctions mathématiques, ou Matplotlib qui permet de créer aisément des graphiques simples ou complexes. Il en existe également des plus complexes, tel numpy, qui permet de travailler avec des tableaux ou bien scipy, qui possède un panel de fonctions complexes.

Toutes ces librairies ont été utiles lors de la programmation. Non seulement elles permettent de gagner du temps en facilitant le travail, mais elles limitent les erreurs et rendent le code plus efficace. De plus, étant souvent utilisé par les programmeurs Python, il existe beaucoup de tutoriels sur internet pour prendre en main efficacement ces fonctions. Ceci est aussi un avantage de Python. Étant un langage courant, les exemples de code ou les résolutions de problèmes sont nombreux sur les divers forums. Pour finir, il existe un notebook, Jupyter, très simple à installer et en libre accès, pour une utilisation optimale de Python.

3 Aspects techniques du portage

3.1 Objectif

Sur Matlab, Hytool est une toolbox. Il faut comprendre cela comme un ensemble de fonctions. Elle peut être téléchargée, installée, utilisée, voir modifiée. L'équivalent en Python s'appelle un package. Dans Hytool, chaque fonction était écrite séparément et ajoutée dans la toolbox. Sur Python, il y a moyen de créer des modules qui contiennent ces fonctions. Il est important de distinguer module et package. Un module est un fichier contenant des définitions et des instructions qui se termine par l'extension `.py`. Un package est un ensemble de modules. [4] Le but était donc de créer un package qui soit, en terme de fonctionnalité, équivalent à la toolbox de Matlab.

3.2 Structure du package

Vu que la structure de la toolbox matlab était bien organisée, il a été décidé de partir sur le même modèle, avec quelques changements. Plutôt que d'écrire chaque fonction dans un module séparé, les groupes de fonctions ont été rassemblés dans un même module. Par exemple, toutes les fonctions ths montrées dans le tableau 1 sont dans le module `ths.py`. De plus, les fonctions génériques ont été assemblées dans le module `hypy.py`.

Cette structure présente l'avantage de pouvoir retrouver rapidement chaque fonction. En pratique, les fonctions génériques ont été codées et testées en premier, afin de pouvoir les utiliser par la suite dans les modules spécifiques. Ensuite, les modules pour chaque solution analytique ont été programmés les uns après les autres. En agissant de la sorte, il était certain qu'un module était fonctionnel avant de passer au suivant. De plus, lorsqu'un module avait besoin d'une fonction générique pas encore codée, il était simple de l'ajouter dans le module `hypy.py` sans mettre en péril la structure existante.

3.3 Comment construire un package

Le premier module à avoir été codé fut `ths`. Lorsqu'il fut déclaré opérationnel, il a fallu créer le package Hypy.

La création d'un package est extrêmement simple. Pour ce faire, il faut rassembler les modules dans un dossier. Le nom de celui-ci sera le nom du package. En l'occurrence, ici, le dossier s'appelle `hypy`. En l'appelant ainsi, il y avait crainte de conflits entre le module `hypy.py` et le package `hypy`. Cependant, Python est capable de différencier les deux lorsqu'une commande y fait référence. Pour finaliser le package, il faut créer un module s'appelant `--init--.py` qui contient le nom des modules du package de manière spécifique. Par exemple, pour le module `ths.py`, il faut utiliser la ligne de code suivante :

Cela signifie que depuis le module `ths.py`, le package importe toutes les fonctions de ce module. Le module `init` ressemble à l'image ci-dessous, dépendant du nom des modules :

```
from .hypy import *
from .ths import *
from .jcb import *
from .script import *
from .thc import *
from .thn import *
from .pcw import *
from .blt import *
from .war import *
```

Bien sûr, il faut ajouter ce module dans le même dossier que les autres. Pour mettre à jour le package avec un nouveau module, il suffit d'ajouter le fichier `.py` dans le dossier et mettre à jour le module `init`. Après cela, le package est opérationnel et peut être utilisé sur les plate-formes Python.

3.4 Validation du code Python

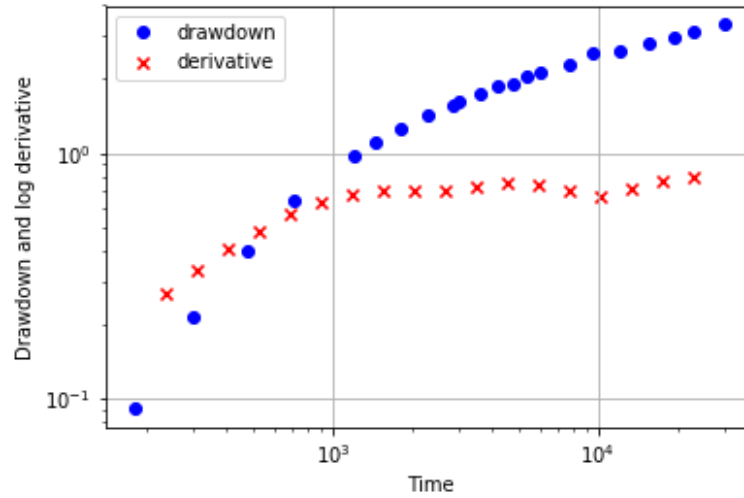
Le but de Hypy est d'interpréter des tests hydrauliques. Il faut donc s'assurer que le code est valide et fonctionnel avant de le distribuer à des utilisateurs.

Python possède des tests unitaires, qui permettent de vérifier le code et garantir son bon fonctionnement. C'est un outil puissant, mais pas toujours évident à mettre en place. Il faudrait que cela soit effectué dès le début du codage, sinon le temps nécessaire pour la mise en place une fois que le code est déjà bien avancé devient conséquent. Pour divers raisons, ce n'est pas la solution qui a été utilisée pour tester le code. Comme le temps était un facteur clé, il a été utilisé pour coder le package. De plus, ces tests n'étaient pas connus au départ du projet et leur existence n'a été apprise que plusieurs jours après le début du codage. Il aurait donc fallu reprendre tout depuis le début de la programmation pour les ajouter aux fonctions déjà existantes. Il n'a pas été jugé nécessaire de le faire, bien que leur présence ajouterait plus de valeur au package.

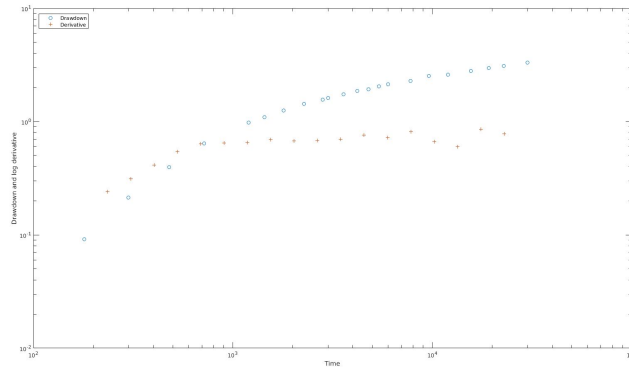
Comme Hypy est une traduction de Hytool, un bon moyen de vérifier le code Python est de comparer le résultat d'une fonction ou d'un module avec sa version matlab. Cette méthode possède toutefois un gros désavantage. Si la fonction agit correctement sur certaines données testées, cela ne veut pas dire qu'elle fonctionnera sur toutes les entrées possibles. Cependant, c'est tout de même la méthode qui a été préférée et appliquée dans ce projet.

Afin d'être efficace dans la comparaison, la méthodologie utilisée fut toujours la même. Plutôt que de tester le module complet une fois terminé, chaque fonction était vérifiée une fois finie. La raison est évidente. le module appelle plusieurs fonctions à la fois, et si le résultat final est erroné, il est beaucoup plus complexe de trouver l'erreur dans une multitude de fonctions que dans une seule. Ainsi, l'avancement se faisait pas à pas, mais en étant sûr que les fondations étaient solides.

Par exemple, toujours dans le module `ths`, la fonction `ths_rpt` a besoin de la fonction `ths_dim`. Ce n'est qu'après s'être assuré que `ths_dim` fonctionnait correctement que `ths_rpt` a été codée. Pour tester, les entrées étaient tout d'abord aléatoires, puis de vraies mesures étaient utilisées.



(a) HyPy



(b) Hytool

FIGURE 1 – Comparaison HyPy et Hytool

Voici ci-dessus un exemple de comparaison entre Hytool et HyPy avec un résultat satisfaisant.

3.5 Distribution du code

Les deux outils nécessaires à l'utilisation d'HyPy sont le package lui-même et une plateforme Python. Le package est disponible sur [Github](https://github.com/UniNE-CHYN/hypy) en libre accès à l'adresse suivante : <https://github.com/UniNE-CHYN/hypy>. Github est un point clé pour ce projet. C'est un service d'hébergement, qui permet également la gestion de développement de logiciels, en utilisant **Git**. Attention à ne pas confondre les deux. La principale fonction de **Git** est de gérer l'évolution du contenu d'une arborescence, tandis que **Github** héberge ce contenu. Le choix d'utiliser **Github** fut facile, tant cela est pratique.

Il permet de stocker le code, sans risque de le perdre si un quelconque problème survenait. De plus, cela se fait efficacement grâce à **Git**. Ensuite, cela aide si le travail s'effectue à plusieurs endroits. Il ne faut pas transporter le code sur un quelconque support. S'il y a collaboration sur un projet, ce qui est le cas ici, il est facile de donner accès aux divers fichiers et cela permet aux collaborateurs de suivre l'évolution du projet, d'y apporter des commentaires ou des corrections. Et finalement, comme c'est déjà le cas pour Hytool, il permet de rendre public un package, ce qui permet à tout le monde d'y avoir accès. Pour la plate forme Python, le plus simple est d'utiliser le notebook **Jupyter** disponible sur le navigateur Anaconda. Ceci n'est pas un ordinateur portable, comme le nom pourrait le laisser croire, mais une application web qui permet de programmer. Les détails d'installation sont dans la section suivante. L'avantage de ce package par rapport à la toolbox matlab est son libre accès. En effet, le navigateur est lui aussi en accès libre et peut être téléchargé sur tous les systèmes d'exploitation. L'unique exigence est d'être capable une seule fois de télécharger le package et le navigateur, qui peuvent ensuite être utilisés sans connexion internet.

3.6 Comment étendre Hypy

La force des logiciels libres et Open Source est qu'ils peuvent être améliorés et étendus par une communauté d'utilisateurs. Dans le cas de Hypy, il existe deux façons d'étendre le package : en modifiant un module ou en ajoutant un module.

Dans les deux cas, il est facile d'effectuer ces modifications/ajouts. Il est possible de travailler sur un environnement de développement Python, comme **Spyder**, mais il est également possible de le faire directement sur le notebook. En allant dans le dossier **Hypy** via le notebook, un module peut être ouvert, modifié, puis enregistré. Cela peut être nécessaire si une erreur est découverte, ou simplement pour ajouter une nouvelle fonction. Toutefois, il faut faire attention en modifiant un module, car si une erreur est ajoutée (de type grammaticale), tout le package ne fonctionnera plus. Par contre, si la fonction contient des erreurs de calculs ou d'assignement de variables, seul le résultat sera faux.

Pour ajouter un module inexistant, il suffit de créer un nouveau notebook, dans le dossier **Hypy**, et de coder les fonctions à partir de là. Il est important de faire attention à deux choses : que l'extension du nouveau fichier soit en `.py` et de mettre à jour la fonction `init.py` en ajoutant la ligne `from .X import *`. Cela permet de dire au package que le module `X` en fait maintenant parti.

Il faut également faire attention au fait d'ajouter un module non-terminé ou erroné à Hypy, car celui-ci risque de poser problème lors de l'importation. Le mieux est de créer séparément le nouveau module et une fois qu'il est terminé et fonctionnel, l'ajouter au package, afin de minimiser les risques de bugs et de pouvoir continuer à l'utiliser tout en créant de nouveaux modules.

3.7 Utilisation de Git

Il est possible de faire ces changements directement depuis **Git**, sans aller sur **Github**. Pour cela, il faut d'abord télécharger et installer **Git**. En allant sur le site internet, il faut choisir selon le système d'exploitation (Windows, Mac OS ou Linux) : <https://git-scm.com/downloads>. L'utilisation de **Git** ne varie pas selon l'OS. Lors de l'installation, il est recommandé d'utiliser les paramètres de base (par défaut). La suite de l'explication se réfère à l'utilisation sous Windows. Dans le répertoire où l'installation a été effectuée, un nouveau dossier est apparu : **gitdemo**. En faisant un clique droit dessus, il faut ouvrir **Git Bash**. C'est un environnement qui permet d'effectuer des commandes **Git**. Sous Linux ou Mac OS, il faut également l'ouvrir. La différence vient de la façon dont il est lancé. La suite est identique. Voici la fenêtre obtenue :

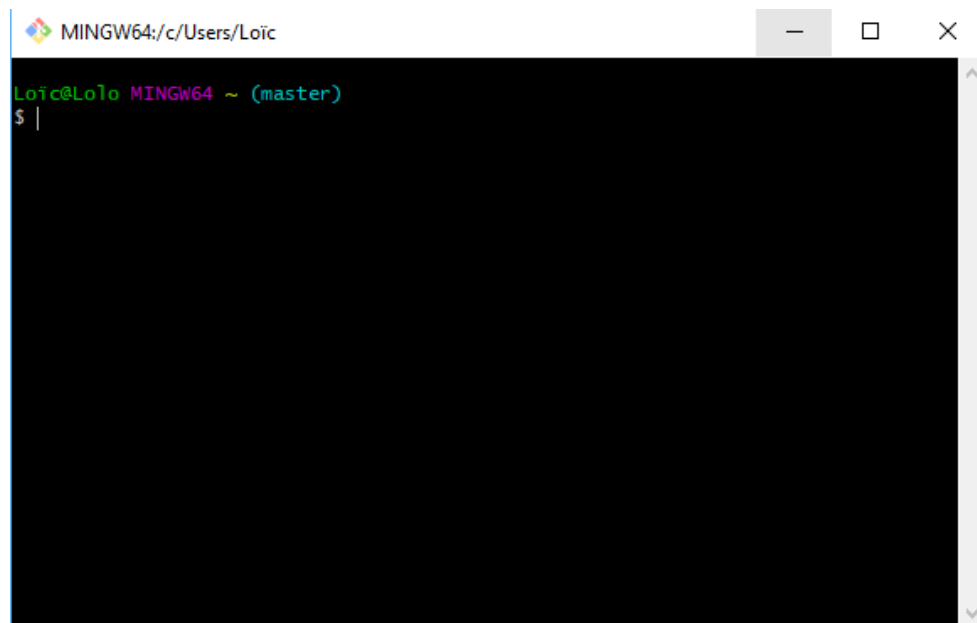


FIGURE 2 – Git fenêtre

La première commande est ensuite **git init** qui lance le programme. Une fois effectuée, un nouveau répertoire va se créer en **.git**. Ce dossier peut être envoyé, ce qui y donnera accès, pour autant que **git** soit installé. Dans le répertoire de travail actuel, a fichier texte **README.txt** (avec juste un mot) a été créé, afin de continuer l'exemple d'utilisation.

En effectuant la commande `git status`, une liste des fichiers/dossiers présents dans le répertoire s’affiche, mais avec la ligne **Untracked files** : , ce qui signifie qu’actuellement, le dépositaire est vide. Pour y ajouter un fichier, il suffit d’effectuer la ligne de commande `git add README.txt`. Le fichier est actuellement **ready to be committed**. Cela veut dire qu’il n’est pas encore ajouté au dépositaire, mais que la dernière version du fichier est enregistré comme telle. Si, avant de faire la commande suivante, une modification a lieu et que `git status` est appelé, alors le fichier sera en même temps **ready to be committed** et **not ready to be committed**, car un changement a eu lieu. Pour résoudre ce problème. Il suffit de taper à nouveau la commande `git add README.txt`. Pour terminer, il faut utiliser la commande `git commit` afin de finaliser l’opération. Pour plus de détails, il existe de nombreux tutoriels sur internet, notamment celui-là : https://www.youtube.com/watch?v=Y9XZQ01n_7c&t=338s.

4 Guide utilisateur

Le but du package est d’être utilisé par des étudiants, des hydrogéologues, mais également par des personnes externes qui en auraient besoin dans un cas spécifique. Tout le monde n’a pas la même affinité avec les outils informatiques, c’est pourquoi autant l’installation que l’utilisation ont été conçues pour être les plus simples possibles. Cette section sert de guide pour l’installation et l’usage du package. Il existe un User guide, inclus dans le package Hypy qui fournit plus de détails sur son utilisation.

4.1 Installation

Le premier pas est le téléchargement et l’installation du notebook Jupyter. Pour cela, il suffit de se rendre sur : <https://www.anaconda.com/download/> et de choisir **Windows**, **macOS** ou **linux** selon votre système d’exploitation. En suivant les instructions (ou en s’aidant des différents tutoriels ou vidéos disponibles sur internet, par exemple <https://www.youtube.com/watch?v=qRixkfuYp88>) et en lançant le navigateur Anaconda, la fenêtre suivante apparaît :

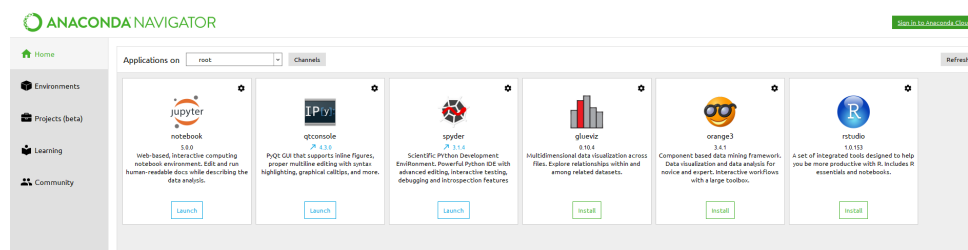


FIGURE 3 – Navigateur Anaconda

En haut à gauche se trouve le notebook Jupyter. Si le notebook n'est pas encore installé, à la place de **Launch** se trouvera **install**. En cliquant dessus, cela va lancer l'installation. Une fois finie, et en cliquant sur **Launch**, on obtient le menu représenté dans la figure ci-dessous : 4 .

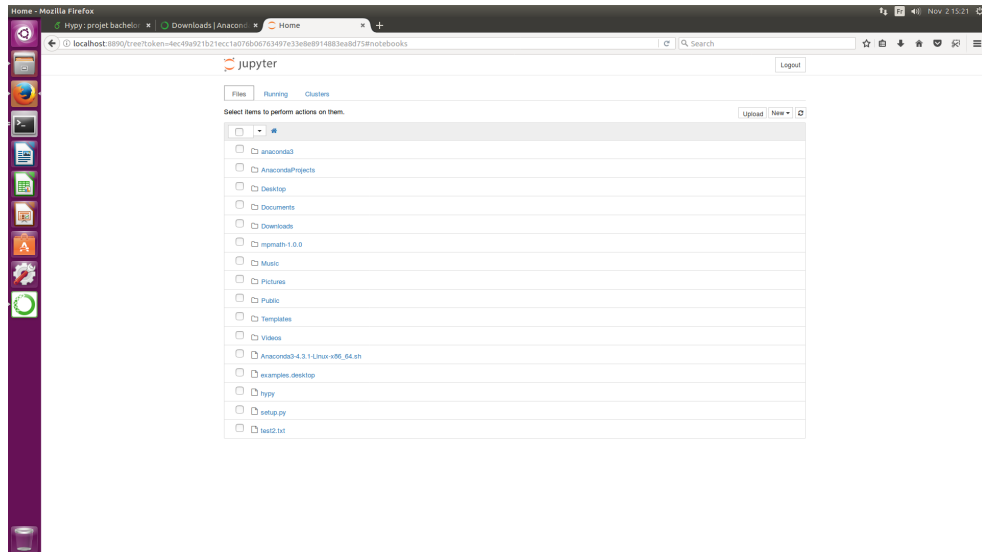


FIGURE 4 – notebook Jupyter

Contrairement au menu précédant, celui-ci est unique pour chaque utilisateur, car il représente les différents dossiers de l'ordinateur utilisé. Il faut ensuite télécharger le package sur github si ce n'est pas déjà fait en allant à l'adresse de la section "distribution du code". La figure suivante apparaît 5 :

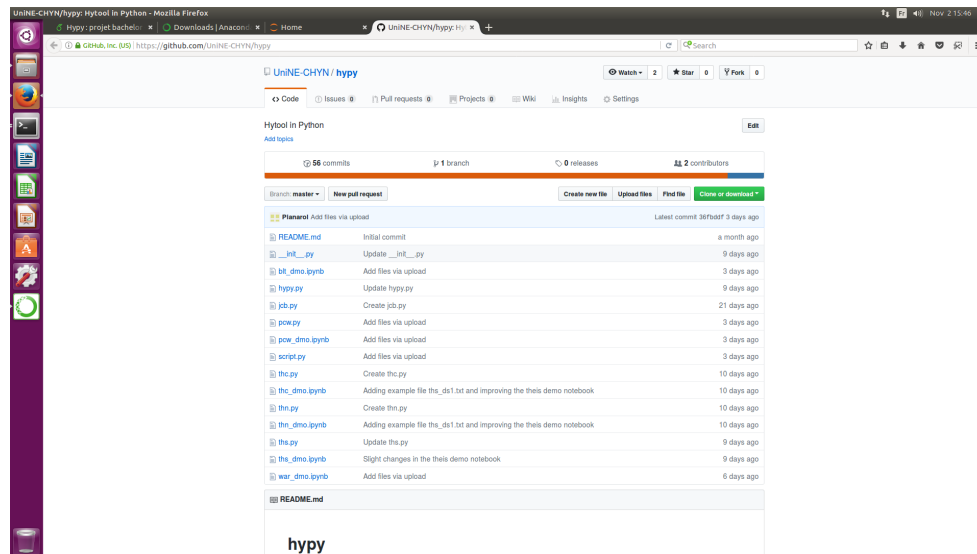


FIGURE 5 – Github hypy

En haut à droite se trouve l'icône "Clone or Download". En téléchargeant tous les fichiers, un dossier zip devra être décompressé pour avoir le package sur l'ordinateur. Dans la section suivante, il sera expliqué comment l'utiliser.

4.2 Comment utiliser le code actuel

Pour commencer, il faut se rendre dans le dossier où se trouve le package sur le notebook :

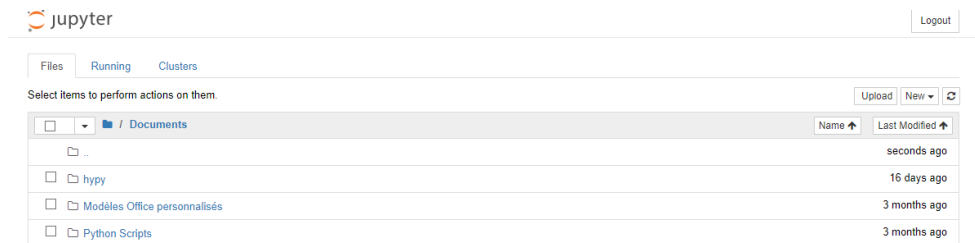


FIGURE 6 – Utilisation de Hypy

Ensuite, en haut à droite, sous le menu "new", il faut créer un nouveau notebook sous Python 3 :



FIGURE 7 – Création d'un nouveau notebook

Après cela, arrive la fenêtre suivante 8, qui permet de travailler dans un environnement Python :

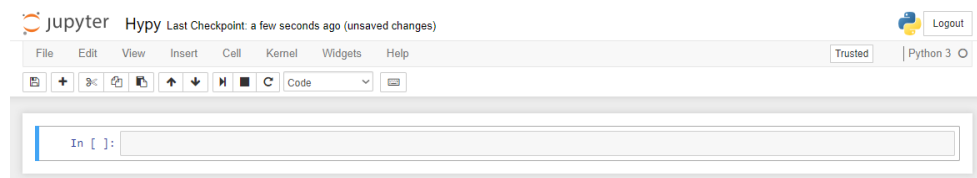
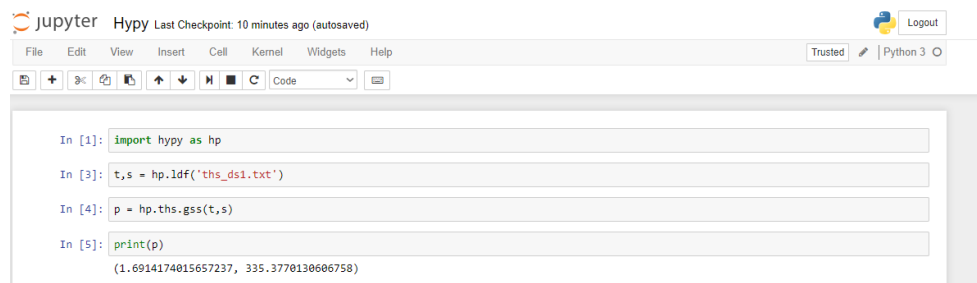


FIGURE 8 – Fenêtre de travail Python

Ici, il est possible de travailler comme sur un environnement classique, en assignant des variables, appelant les fonctions de base comme `print`, etc. L'avantage, c'est que cela permet également d'importer les modules comme `numpy` ou `matplotlib`, ou dans ce cas précis, `hypy`. Pour cela, il suffit d'écrire la ligne de commande suivante : `import hypy` (ou le nom donné au package). Comme `hypy` sera appelé chaque fois qu'une fonction doit être utilisée, Python propose de simplifier le nom du package pour qu'il prenne moins de place et soit plus rapide à appeler avec le mot-clé `as`. Ainsi, la commande devient `import hypy as hp`. Le `hp` peut être transformé en tout et n'importe quoi (sauf les chaînes de caractères qui font parties intégrantes de Python comme `print` ou `str`). Comme la dénomination sera tout le temps appelé, il faut que ce soit simple, mais tout de même explicite, afin ne pas confondre.

À partir de là, le package peut être utilisé. Chaque appel de fonction du package commencera par (ici, en prenant en compte que le package a été renommé **hp**) **hp**. Ensuite, pour permettre à Python d'aller chercher dans le panel de fonctions, il faut suivre cette dénomination par un `."`. Après ce point, il y a deux possibilités. Soit faire appel à une fonction générique, auquel cas il sera suivi par le nom de celle-ci avec les paramètres. Par exemple : `hp.ldf("nom du fichier")`. Ou alors, faire appelle à un groupe de fonctions. Dans ce cas, `hp` sera suivi par le nom du groupe et ce même nom par celui de la fonction avec les paramètres. Par exemple `theis` (`ths`) et la fonction `guess` (`gss`) : `hp.ths.gss(t,s)`. Si le nom d'une fonction est oubliée, il est possible d'utiliser la touche tabulation pour voir apparaître toutes celles disponibles. Par exemple, en faisant `hp. + tabulation`, toutes les fonctions génériques, ainsi que tous les modules seront visibles. En faisant `hp.ths. + tabulation`, uniquement les fonctions du module `ths` apparaîtront.

Voici un exemple d'une prise en main rapide :



```

In [1]: import hypy as hp

In [3]: t,s = hp.ldf('ths_ds1.txt')

In [4]: p = hp.ths.gss(t,s)

In [5]: print(p)
(1.6914174015657237, 335.3770130606758)

```

FIGURE 9 – Exemple d'utilisation du notebook

Pour plus de détails sur l'utilisation et des exemples, se référer au module de démonstration et au guide de l'utilisateur inclus dans le package.

5 Conclusion

La traduction de Hytool depuis Matlab vers Python fonctionne. Les fonctions d'Hypy opérationnelles fournissent les mêmes résultats que celles d'Hytool. Grâce aux différentes bibliothèques disponibles avec l'environnement de développement Python, le code a été grandement simplifié. Malgré le fait d'avoir dû s'adapter à plusieurs contraintes, une fois la structure générale comprise, la traduction d'un module s'effectue de façon linéaire. De plus, une collaboration directe et fréquente avec le professeur P. Renard a permis de résoudre les problèmes les plus difficiles, que ce soit dans la compréhension du code Matlab ou l'écriture de celui en Python. Enfin, Un tel projet a également permis de découvrir le fonctionnement de Matlab, tout en améliorant les compétences en Python.

L'objectif initial de permettre aux utilisateurs d'Hytool d'offrir le même produit sur Hypy est partiellement atteint. En partie seulement, car si le package en soi fonctionne bien, toutes les fonctions initiales n'ont pas été traduites et certaines améliorations, notamment l'ajout de tests, seraient bénéfiques. Cependant, comme le produit est en open source, il n'est pas exclus que ces ajouts et ces améliorations se fassent dans le futur. De plus, l'éventualité existe qu'une autre personne continue ce projet pour rendre Hypy encore plus performant.

Références

- [1] Renard, Philippe (2017). *Hytool : an open source matlab toolbox for the interpretation of hydraulic tests using analytical solutions*. Journal of Open Source Software, 2(19), 441, doi :10.21105/joss.00441
- [2] Documentation Python
- [3] Apprendre Python
- [4] Stackoverflow : différences entre Module et Package