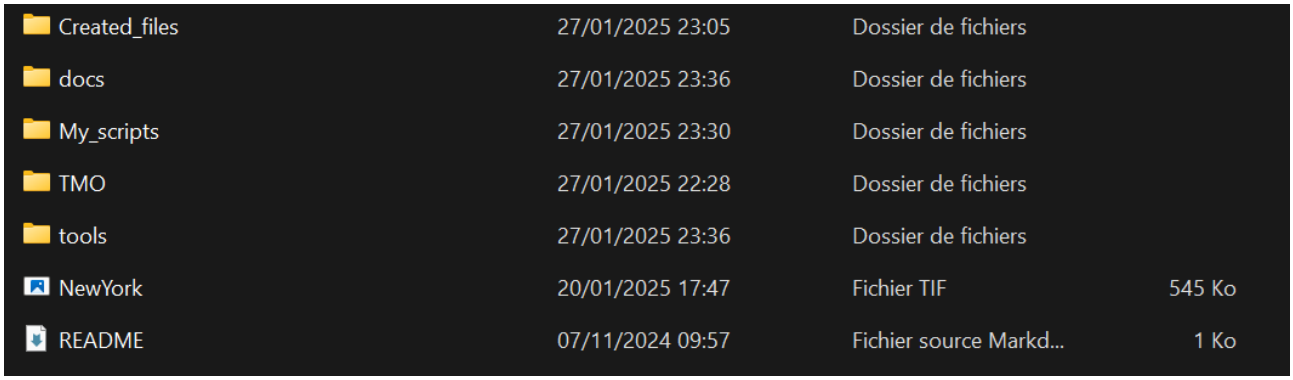
Documentation utilisateur :

Pour se servir de la toolbox, il suffit de télécharger le dossier ToolboxTMO, de le placer sur son ordinateur à un emplacement de votre choix. Pour accéder aux outils, il faut créer un fichier .py dans le dossier My_scripts. Dans ce dossier il existe déjà un fichier main.py qui peut être dupliqué pour servir de base. Les images obtenues après traitement seront automatiquement enregistées dans le dossier Created_files.



La première partie du fichier main.py correspond aux imports qui permettent de faire appel aux différentes fonctions implémentées. Cette partie ne doit pas être modifiée à moins qu'il y ait de nouveaux outils et TMO d'ajoutés à la toolbox et qui doivent être utilisés. Dans ce cas il suffit de reproduire le schéma d'import en ajoutant la ligne nécessaire aux parties **Importer les modules** et **Importer les classes.**

```
######################### Import des modules ################################
import os
import importlib.util
from osgeo import gdal
import numpy as np


# Définir les chemins des dossiers contenant les modules
current_script_dir = os.path.dirname(os.path.abspath(__file__))
parent_dir = os.path.dirname(current_script_dir)
tools_dir = os.path.join(parent_dir, 'tools')
tmo_dir = os.path.join(parent_dir, 'TMO')
cf_path = os.path.join(parent_dir, 'Created_files')

# # Vérification des chemins
# print(f"Chemin du script actuel : {current_script_dir}")
# print(f"Chemin vers tools : {tools_dir}")
# print(f"Chemin vers TMO : {tmo_dir}")
# print(f"Chemin vers Created_files : {cf_path}")

# Fonction pour charger un module de manière dynamique
def import_dynamic(module_name, module_path):
    assert os.path.exists(module_path), f"Module introuvable : {module_path}"
    spec = importlib.util.spec_from_file_location(module_name, module_path)
    module = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(module)
    print(f"Module '{module_name}' importé avec succès depuis {module_path}")
    return module


############# ajouter les outils nécessaires ici
# Importer les modules depuis 'tools'
tools_open = import_dynamic("open", os.path.join(tools_dir, 'open.py'))
tools_save = import_dynamic("save", os.path.join(tools_dir, 'save.py'))
tools_image_display = import_dynamic("ImageDisplay", os.path.join(tools_dir, 'ImageDisplay.py'))
# tools_meta = import_dynamic("modif_meta", os.path.join(tools_dir, 'modif_metadata.py'))


############# ajouter les TMO nécessaires ici ############################

# Importer les modules depuis 'TMO'
tmo_mantiuk = import_dynamic("Mantiuk", os.path.join(tmo_dir, 'Mantiuk.py'))
tmo_gamma = import_dynamic("Gamma", os.path.join(tmo_dir, 'Gamma.py'))
tmo_gamma_inv = import_dynamic("Gamma_Inv", os.path.join(tmo_dir, 'Gamma_Inverse.py'))


# Importer les classes et fonctions nécessaires des modules
OpenGDAL = tools_open.OpenGDAL
CreateImageFromDetails = tools_save.CreateImageFromDetails
ImageDisplay = tools_image_display.ImageDisplay
MantiukTMO = tmo_mantiuk.MantiukTMO
GammaTMO = tmo_gamma.GammaTMO
GammaInvTMO = tmo_gamma_inv.GammaInverseTMO
# MetadataLogger = tools_meta.MetadataLogger

# # Confirmation des imports
# print("Import réussi pour OpenGDAL, CreateImageFromDetails, ImageDisplay et MantiukTMO.")
```

La partie suivante correspond à un exemple de script exécutable. Il suffit de rentrer le chemin absolu d'une image et de remplacer 'nom_meta.txt' par le nom du document que l'on veut donner au document .txt contenant les métadonnées qui va être crée. meta_path correspondra au chemin

```
############################## Exemple de Script ########################


# !!!!!! Chemin de l'image d'entrée à remplir

im_NY = r"C:\Users\ablot\Documents\PPMD\TMO_Toolbox\TMO\TMO_Toolbox\tools\NewYork.tif"
meta_path = os.path.join(cf_path, "nom_meta.txt")


##### Ouverture image et sauvegarde métadonnées
image_metadata = OpenGDAL(im_NY, meta_path)

##### Récupérer la matrice correspondant à l'image
pixel_matrix = image_metadata.get_pixel_matrix(normalize=False)
ps = image_metadata.get_pixel_matrix(normalize=True)

####### Enregistre l'image de base dans le fichier Created_files avec les métadonnées intégrés
CreateImageFromDetails(pixel_matrix, meta_path, 'image_originale').create_image()


### Appliquer fonction Gamma et enregistrer Image
mat_mod2 = GammaTMO(pixel_matrix, gamma=2.2, metadata_file=meta_path).apply_correction()
CreateImageFromDetails(mat_mod2, meta_path, 'image_Gamma_TMO').create_image()

### Appliquer fonction Gamma Inverse et enregistrer l'image
mat_mod3 = GammaInvTMO(mat_mod2, metadata_file=meta_path).apply_inverse_correction()
CreateImageFromDetails(mat_mod3, meta_path, 'image_Gamma_Inv_TMO').create_image()
```

absolu vers ce fichier. OpenGDAL permet de créer l'objet image accompagné du fichier .txt. La fonction get_pixel_matrix récupère la matrice ou liste de matrice correspondant aux valeurs des pixels. Ensuite on peut appliquer les algorithmes de TMO sur les matrices obtenues. Ici GammaTMO avec en entrée les paramètres nécessaires. Les matrices, le chemin vers le fichier métadata qui sera modifié à chaque appel d'un TMO. Après cela on peut sauvegarder l'image nouvellement obtenue avec CreateImageFromDetails qui prend en entrée les nouvelles matrices, le chemin vers les métadonnées et le nom que l'on souhaite donner à l'image de sortie (sans extension puisqu'elle sera déterminée des métadonnées).

La suite du code dans main.py sert à afficher les résultats par plot mais aussi par affichage des matrices.