DOCUMENTATION PYCOFFEE

TODO j'ai pas du tout mentionné les constructeurs. Après le code est commenté, toute les infos sont dedans

Sommaire

I - Installation

INSTALATIONS NECESSAIRE A PYCOFFEE :

ASTERIX → setup.py install	Necessite astropy et ConfigObj
OPTIMIZE	Module pour minimisation
PIL	(Python Imagaing Library) *pour la sauvgarder du dérouler de la simulation as a gif.*

MANUEL D'INSTALLATION :

Installation Package :

Attention, installé implique une copie des fichiers sur le système. Vous devrez re-installé à chaque fois si vous changez le code. Installation uniquement pour les utilisateurs.

Etape 1 : Vérifié que on a bien installé toute les dépendances

NB : Si j'avais bien fait mon setup.py comme il faut, ça n'aurai pas été la peine. Mais je ne l'ai pas fait.

Etape 2 : setup.py install

Le package installé, il s'utilise comme n'importe quel module. Il peux être importé depuis n'importe quel emplacement.

Installation développeur :

Etape 1 : Vérifié que on a bien installé toute les dépendances

NB : Si j'avais bien fait mon setup.py comme il faut, ça n'aurai pas été la peine. Mais je ne l'ai pas fait.

Tant que le package n'est pas installé, `Import CoffeeLibs` n'est reconnu que a la racine de l'arborescence. On peux l'utiliser en définissant le working directory a cette endroit. (emplacement du __init__.py)

(Etape 2 optionnel mais conseillé) : Exécuter le fichier __init__.py.

moi c'est comme ça que j'aime bien faire, c'est personnel donc c'est comme vous voulez. Ca permet de ranger les fichiers qui utilise coffee où on veux et pas de tous les avoir au même endroit avec le __ini__ et le setup. Moins de question de path et de working directory a se poser.

Résultat : La console sur laquelle vous travaillez enregistrera le chemin de CoffeeLibs. Ca fonctionnera comme un package installé : reconnu partout, pas de question d'emplacement à ce posé. Et ça ne créera pas de copie donc toutes les modifications seront prisent en compte a chque nouvelle execution (sans avior a re executer le _ini__.py)

Inconvénient : Fonctionne uniquement sur la consol en cours d'utilisation. Il faudra exécuter le __init__ a chaque nouvelle console démarré.

II - Contenu

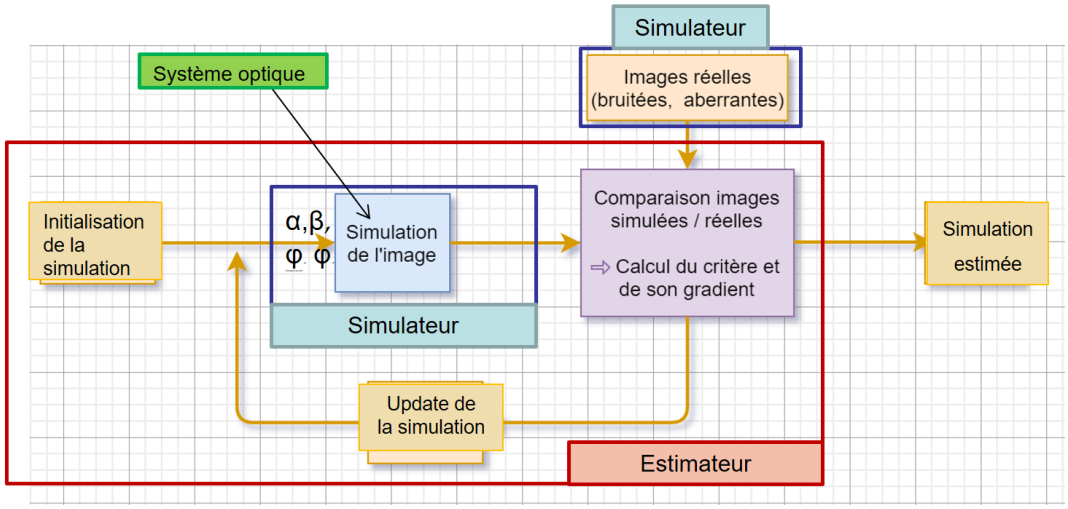

CoffeeLibs
CoffeeLibs
Coffee library
coffee.py
Contient les trois classes de Pycoffee
critère.py
Fonctions de critère appelé par le minimseur
file_manager.py
utilitaire de fichier (pas très utile)
pzernike.py
polynome de zernike (copie conforme de IDL)
tools.py
Fonction pour opération souvant répété
(opération sur les matrices, template plots,..)
Demos_Test
Démos et tests
TestPhaseDiv.py
Exemple d'utilisation de Pycoffe sur données simulé
noCorono_donne_reel.py
Exemple d'utilisation de Pycoffe sur donées expérimentals
test_fonctionelles.py
test Pycoffe avec Unit_test
my_param_file.ini
le fichier ini que j'utilise pour toutes les démos et tests
imgs
Emplacement de mes images expérimental
balbla.fits ...
save
Emplacement des sauvgarde de simulation
Img_ processing_ tools
Pré-traitements des images
my_scipt.py
Désolé pour le nom pas consistant
Script pour traité les images, qui apelle les fonction de process_img.py
process_img.py
Fonction de prétraitement des images
__init__.py
setup.py
Necessaire a l'installation en tant que module

III - Structure général

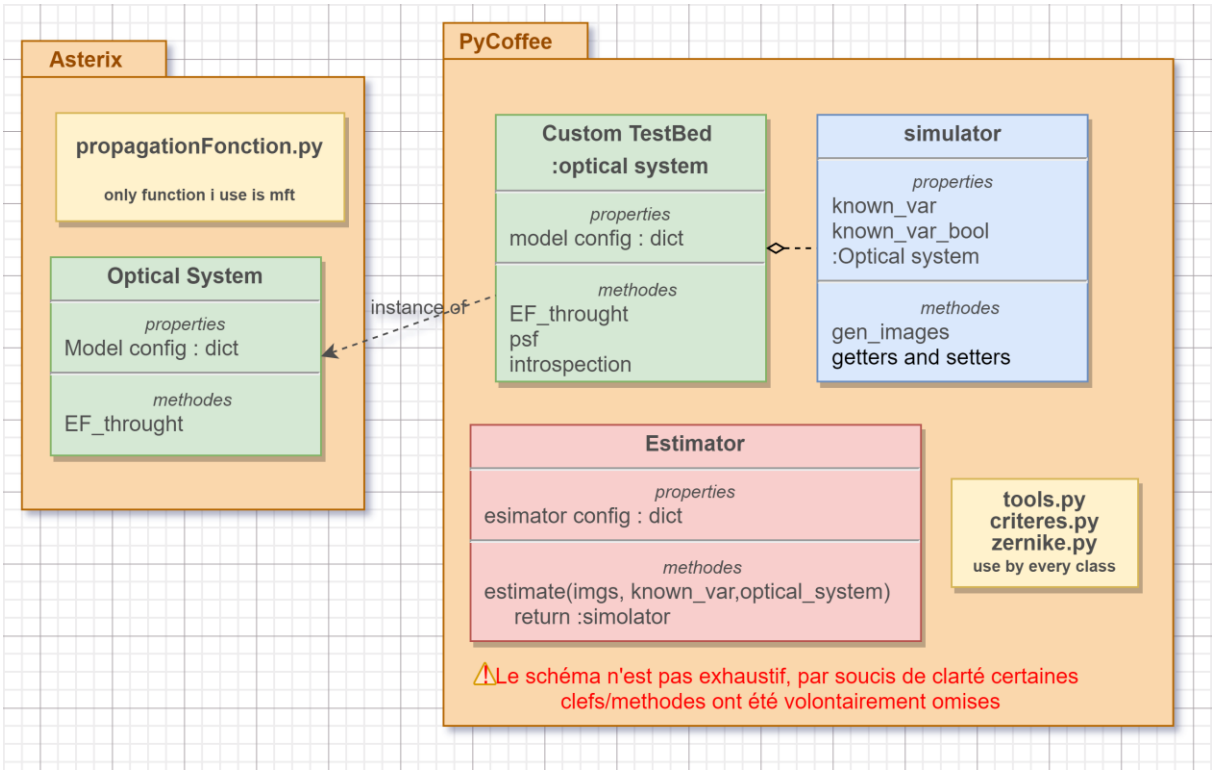
1 - Articulation du code

Le schéma ci-dessous présente le concept général d'une estimation de phase par minimisation d'un critère de maximum a posteriori.Les trois classes de pycoffee représentes les trois région de code encadré en rouge vert et bleu.

Rappel du concept général du programme mis en place de minimisation du critère de maximum a posteriori



Le diagramme UML de class de PyCoffee est représenté ci-dessous, en reprenant le même code couleurs.La liste des propriété et des méthodes est exhaustivement développé dans la section suivante.



III - Structure général

.INI FILE :

Le fichier ini permet de définir les paramètres de construction des différent object (opitcal system, estimator ..) .

Un fichier ini "template" se trouve dans le package asterix.

Pour voir les clefs et les valeurs possible regarder les exemples et le fichier tempate.

Fonctionnement : ConfigObj et Validator permet de vérifié que votre fichier ini match bien les spécification du fichier template.

→ Dans CoffeeLibs.file_manager, j'ai défini une fonction get_ini(path) qui va chercher le path du tempate dans Asterix, et appelle ConfigObj et Validator.

Elle renvoie un dictionnaire, cette objet peu ensuit être utilisé par les constructeurs.

2 - Les trois grande class PyCoffee.

Cette parti fait office de manuel. Elle décrit toute les méthodes et propriétés des classe le plus exhaustivement possible.

Je ne décrirais pas ici les fonctions dans tools, pzernike ... mais vous pouvez jetter un coup d'oeil, c'est commenté et c'est des opération assez "élémentaire".

Critère.py en revanche contient des fonction assez complexe qui méritent d'être expliquées. Elles le sont dans la section suivante.

CLASS CUSTOM_BENCH

Instance de **OPTICAL SYSTEM** : Super class d'Asterix

Si mode gradient automatique : Interchangeable avec n'importe quel autre instance de **OPTICAL SYSTEM** a condition que la propagation du champ soit défini linéaire.

Si mode gradient analytique : Non interchangeable, ou il faut adapter le gradient du critère au modèle de propagation.

Definition :

Classe qui représente le système optique du coronographe. On peux y propager un champs électromagnétique.

Custom_tested a aussi une méthode `introspection` pour suivre plan par plan la traversé du champ.

Propriété :

Nom	type	description
Propriétés hérité de la super class Optical System	--	Ce référé à la documentation d'Asterix
pup	array	Carte pupille
corono	array	Carte coronographe
offset	dict	Paramètre d'offset pour que les mft soit centré sur 4 pixels. Utilisé chaque appel de mft
zbiais	bool	Si True, ajoutera la correction zbais dans le modèle de propagation

Méthodes : *méthode de la super class, redéfini spécifiquement a Custom_Bench

Nom	argument	return	description
* EF_through	entrance_EF=1, EF_aberrations_LS=1	EF_out	Prend en argument une carte de champ upstream et downstream. Renvoie la propagation du champ a travers le système optique avant l'arrivé sur le detecteur
* todetector	entrance_EF=1, EF_aberrations_LS=1	mft(EF_out)	Prend en argument une carte de champ upstream et downstream. Renvoie la propagation du champ a travers le système optique jusqu'au detecteur. sans module carre
* todetector_ intensity	entrance_EF=1, EF_aberrations_LS=1	$\|mft(EF\_out)\|^2$	Comme en haut mais avec le module carre -> on renvoie l'intensité recu sur le detecteur
z_biais	aucun	`zbiais`	Revoie l'ajustement z_biais. Utilisé dans EF_through pour corrrigé le champ avec un 4q si la propriété `zbais`=True
introspect	entrance_EF=1, EF_aberrations_LS=1	`void`	Ouvre une fenêtre matplotlib "Introspection" qui affiche la propagation du champ électromagnétique a travers le banc, plan par plan.

CLASS DATA_SIMULATOR

Definition :

Simulateur de donnée.

Ces propriété sont les valeurs d'entrées de la simulation et le système optique. Se souviens aussi des variables qui sont supposé connues et celles qui sont supposé inconnu, même si on leurs assigne une valeur a posteriori.

Outil multitâche pour contenir tout les différent paramètre d'une simulation qui ne sont pas propre au système optique en lui même.

Contient des outils pour facilité son utilisation avec l'estimateur

Propriété : *Accessible autrement mais défini comme ça car utilisé fréquemment.

Nom	type	description
tbed	Optical system	Systeme optique
cplx	bool	`True` : phase upstream est un complexe `False` : phase upstream est un réel
phi_foc	2D array	Carte de phase focalisé
*EF_foc	2D	Carte de champ focalisé. Mis a jour si phi_foc est modifié. ATTENTION seulement dans ce sense.
Knwon_var	dict	**Clefs possible:** Downstream_EF : carte de CHAMP d'aberration downstream. Peut être defini comme un float si phase constante. flux : liste des flux pour chaque image de diversité fond : liste des fond pour chaque image de diversité
Div_map	List of 2D array	List des carte de phase de diversité. Genere a partir d'une liste div_factor en argument lors de la construction de l'object. Si div_factor float : diversité défini comme a*defoc Si div_factor array: carte de diversité tel quel.
* Nb_div	int	Nombre d'images de diversité
* N	int	Dimension (largeur) de la carte de phase

Méthodes :

Cette class contient BEAUCOUP de méthode car :

1 - On va exploiter les propriétés de mutabilité des classes en python.

C-a-d lorsque l'on va modifié l'object à l'intérieur d'une fonction, on veux qu'elle soit modifié partout. (expliquation un peu grossière, ce référé une doc)

2- On veux être sur que quand on change quelquechose, tous les propriétés qui doivent être modifié avec le soit.

Du coup, on va préféré -> sim.set_phi_foc(my_phi) **plutôt que** -> sim.phi_foc = my phi

3- on veux définir des méthode pratique, par exemple une méthode qui permet de définir phi foc directement a partir de zernike.

Conclusion : Beaucoup de méthode pour une classe flexible, mutable et pratique.

Parmis ces méthode on va retrouver :

- Les setters and getters et checkers
- Les generators
- Les optimizes wrappers
- Les Tbed wrappers
- (les internal tools, comme genre set_default_value())

Setters & getters : * utile pour l'estimateur

Nom	argument	return	description
Setters set_<propreties>()	Truc que on veux set	`rien`	Set la propriété voulu. Regarder spécifiquement le setters voulu si paramètre particulier. La plus part n'on pas de paramètre particulier ATTENTION Il manque des setters pour certain variable que je n'ai jamais eu besoin de changer.
Getters get_<propreties>()	Rien généralement div_id quand il s'agit de récupéré une carte de phase/champ de diversité particulière	`Le truc que on veux get`	Get la propriété voulu. Regarder spécifiquement le setters voulu si paramètre particulier. La plus part n'on pas de paramètre particulier ATTENTION Il manque des setters pour certain variable que je n'ai jamais eu besoin de changer.
* Ckeckers Truc_is_know	rien	`bool`	Va chercher si le propriété est défini comme connu ou incconu. A saoir est que il faut l'estimé ou pas.

Generators : -> Fontion pratique utilisé souvent a l'extérieur de la classe

Gen_zernike_phi_foc	Coeff : list des coeff du polynome de zernike	rien	Genere un polynome de zernike à partir des coeff en entrée et défini phi_foc comme tel.
Gen_zernike_phi_do	Coeff : list des coeff du polynome de zernike	rien	Genere un polynome de zernike à partir des coeff en entrée et défini phi_do comme tel.
Gen_zernike_phi	Coeff : list des coeff du polynome de zernike	carte du zernike	Renvoie l'array 2D qui du zernike genere a partir des coeff en entrée
Gen_div_phi	rien	3D array	Renvoie l'array 3D de toute les carte de phase de diversité
Gen_img_div	RSB : None -> pas de bruit	3D array	Renvoie l'array 3D de toute les image de diversité. Avec ou sans bruit (RSB = rapport signal/bruit)

Optimize wrappers : -> permet d'adapter les variables a la syntaxe d'optimize

Opti_unpack	Pack : 1D array	(optionel) revoie les varaible unpacké	Prend en argument la list 1D des inconnus de la simulation (tel que la syntax d'optimize nous l'oblige) Unpack les variables, les sépares si il y en a plusieurs (phi_up,phi_d,flux,fond) les remet sous forme de tableau 2D (eventuellement sous forme de complexe).
Opti_pack	rien	Pack : 1D array	Prend en argument la list 1D des inconnus de la simulation (tel que la syntax d'optimize nous l'oblige) c-a-d en concatenant en array 1D toute les inconnu de la simulation
Opti_uptade	Pack : 1D array imgs	Rien	Similaire a opti_unpack MAIS estime flux est fond en plus (si besoin de les estimé uniquement)

Tbed wrapper : -> appelle les méthodes du Testbed avec les propriétés de la simulation

EF_trough To_detector To_detecor intensité psf	/	/	Appelle ces fonctions avec les propriétés de la simu Pour plus de détail, regardé les fonctions spécifique, code commenté.
To_detector_loop		3D array	Genere toute les images de diversité (en appelant to_detecor_intensité bouclé sur le nombre de diversité). Renvoie un array 3D des images (N x N x Nb_div)
Phi_to_img	3D arrays	3D array	Prend en argument la carte des phase de diversité Renvoie les images

CLASS ESTIMATEUR

Definition :

Réalise une estimation des paramètre d'entree du systeme par critère de MAP. Minimisation L-BFGS-B avec optimize.minimize

!! ATTENTION option grad auto. Une fonction permet de calculer L a partir d'un systeme optique. Pour l'instant j'apelle la fonction a l'interieur de la classe. mais il sera plus judicieux de passé une L deja consruit au minimiseur a l'initialisation ou a l'appelle de la méthode estimate !!

Propriété :

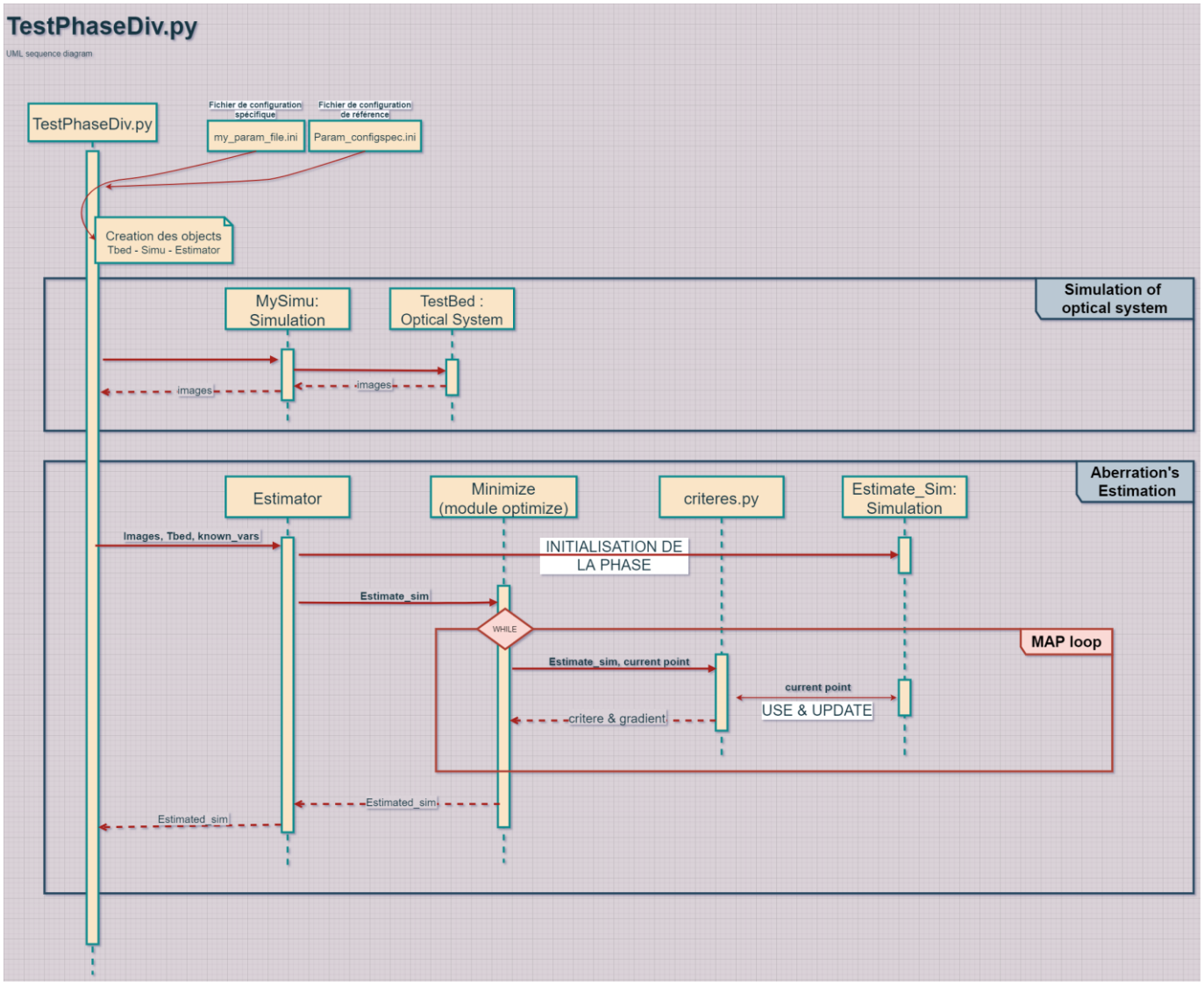
Nom	type	description
Propriétés "option"de minimize	--	Ce référé à la documentation de optimize.minimize Clefs : disp, gtol,eps,maxiter.
var_phi	float	Variance a pripori de la phase. Fait office d'hyper paramètre pour la regularisation
auto	bool	`True` : gradient automatique -> appelle des fonctions gradient auto pour la minimisation `False` : gradient analytique -> appelle des fonctions gradient analytique pour la minimisation
cplx	bool	`True` : estime la phase complexe `False` : estime la phase réel

Méthode unique :

estimate	Tbed : optical system Systeme optique Imgs : 3D ndarray (N,N,nb_div) Image reel Div_factor : list Liste de diversité connu Si liste de float : diversité défini comme a*defoc Si liste de array: carte de diversité a ajouter a la phase focalisé. Knwo_var : dict Dictionnaire des entrée connu. Si une clef s n'est pas rempli, elle sera considéré comme inconnu et sera estimé. **Clefs possible:** Downstream_EF : carte de **CHAMP** d'aberration downstream. Peut être defini comme un float si phase constante. flux : **liste** des flux pour chaque image de diversité fond : **liste** des fond pour chaque image de diversité	`e_sim` : object data_simulator Cet object contiendra tous le resultat de l'estimation (sim.get_phi). + les propriétés info, info_grad, et info_div seront ajouté à l'objets	Réalise l'estimation MAP. Etape 1 : cree un objet simulation avec tout les paramèter a 0. Etape 2 : défini les paramètres de l'estimation Etape 3 : Appel Optimize.minimize -> Prend en argument deux fonctions : une pour calculer le critère, une pour calculer le gradient. -> A chaque passage de ces fonction, la simulation vide est mise a jour avec la current estimation des paramètre a estimé. (cf. opti_update et les fonction préfixé par 'V_' dans critère.py) Renvoie l'objet data_simulator

DIAGRAMME UML DE SEQUENCE D'UNE ESTIMATION.

Conseil le code est commenté, le mieux pour comprendre, c'est d'aller voir dans la méthode estimate d'estimator. Le shéma ne vous donnera qu'un aperçu global.



3 - Critere.py

Bibliothèque de fonctions qui permet de calculer le critères.

Calcule de critère, du gradient, estimation flux/fond, gradient par différence fini

Les fonctions préfixé par 'V_' sont les fonctions destiné a etre appelé par le minimiseur.

Ces fonctions appelles les autres fonction opur calculer J, dJ, R.. Elles s'occupe de mettre a jour la simulation avec les nouvelles valeurs, calculer les trucs dont en a besoin, de géré la pondération et de renvoyé le resultas mis en forme corresctement pour optimize.

Nom	argument	return	description
meanSquare	Hx y : arrays	float	Renvoie la distance mean square entre Hx et y (==> calcule du maximum de vraisemblance J)
regul	Phi : 2D array	float	Renvoie le gradient spatial de l'entré
DJmv_up	Div_id Img Sim : data_simulator	2Darray	Renvoie le gradient du maximum de vraisemblance A partir des donne de l'object sim donné en entré et pour un indice de diversité donné (aka un entier !!), l'image de diversité correspondante
DJmv_dp	Div_id Img Sim : data_simulator	2Darray	Meme chose que DJmv_up mais gradient par rapport a phase downstream
Dregul	Phi : s2Darray	2Darray	Renvoie la laplacien spacial (derivé 2nd) de l'entré
Estime_fluxfond	Sim : data_simulator Imgs : 3D array images reel	2D array : 2xNb_div	Renvoie l'estimation du flux et du fond a partir des paramètre actuel de la simulation et des images Appelé dans sim.opti_uptadte Renvoie 2D array : 2xNb_div pour flux et fond associé a chaque diversité
V_map_J V_grad_J V_map_J_auto V_grad_J_auto	Var Sim Imgs Hypp Simfig L (pour auto)	1D array	Fonction appelé par optimize.minimize dans la methode estimate de Estimator. Etape 1 : uptate la simulation sim avec var (la sortie donnée par minimize 1D array) Etape 2 : Calculer critère/gradient du critère pour chaque diversité, terme d'attache au donnée et regularisation Etape 3 : la ponderation Etape 4 : Enregister les infos pour le suivie de la simu -> ajoute les propriété info,info_gra,info_div a l'object sim Etape 5 (optionel) : sauvergarde les gradients pour suivi de la simu en affichant le dérouler (simGif) Etape 6: return (en adaptant a la syntax optimize si besion.)
Grad_diff_<truc>	/	grad	Gradient par différance fini. Cf commentaires et compar_grad.py dans demo_test

IV - Outils spécifique

J'ai fait quelques "Outils" qui peuvent être pratique.

TEST_FONCTIONELLS.PY :

Dans demo_test : test_fonctionells.py

Avec unit_test de python, permet de tester différent cas de figure de simulation.

Le scipt est séparé en section.

Section 1-2-3 : Initialisation des choses dont on aura besoin

Section 4 : Fonctions qui seront appelé dans les test unitaire.

do_a_test, rest_estimator sont pour genere une simulation

fontchoice, init_worksheet et new_row sont pour la generation du fichier exels

Section 5 : definition des tests

Section 6 : main qui lance les tests.

PROCESS_IMG :

Dans le dossier img_processing_tools se trouve deux fichiers.py. Process_img.py contient des fonction pour moyenné, cropé, centré les fits qui se trouvent dans un dossier.

Script.py les utilisent

TEMPLATE PLOTS :

Dans CoffeeLibs.tools: Les fonctions template plots permet d'afficher les resultats de la minimisation.

Ces fonctions vont chercher les données a affiché, les infos, bref tous les resultat de la minimisation et les met en forme.

Avantage : Vous avez un affichage complet que vous pouvez réutilisés. Plusieurs template plots en fonction de se que vous voulez voir.

COMPARE TRUC :

Dans demo_test : compare_grad.py, comparSimu, grad auto_test.

Script pour comparé des trucs. peut-être réutilisé.

En particulier compare_grad, qui compare un gradient par rapport au gradient par différence fini pourrai peut-être servir dans le future ?