DOCUMENTATION PYCOFFEE

Sommaire

DOCUMENTATION PYCOFFEE

I - Installation

INSTALATIONS NECESSAIRE A PYCOFFEE :

ASTERIX	→ setup.py install pour installer Asterix
OPTIMIZE	Module pour minimisation
PIL	(Python Imaging Library) *Pour la sauvegarde du déroulé de la simulation as a gif.*
PICKEL (OPTIONEL)	Utilisé dans un des tests -> ne sert pas au fonctionnement de PyCoffee *permet de sauvegarder dans un fichier, au format binaire, n'importe quel objet Python*

MANUEL D'INSTALLATION :

Installation Package :

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

Etape 1 : Vérifier 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, on peux "import CoffeeLibs".

Installation développeur :

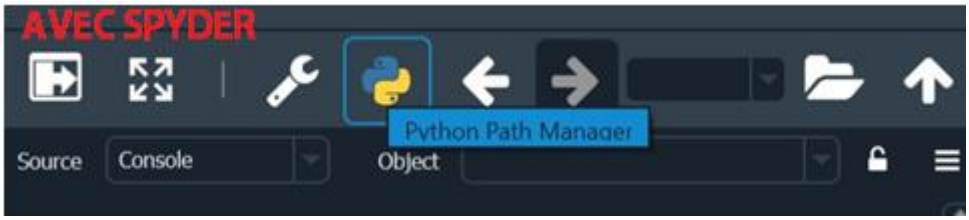
Etape 1 : Vérifier que on a bien installé toutes les dépendances

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

Etape 2 au choix :

1 - (BEST) Add CoffeeLib path to PYTHONPATH

→ DOC ICI : http://physicalmodelingwithpython.blogspot.com/2015/10/paths-in-python.html



2 - Set your working directory (a l'endroit du ini de CoffeeLibs) (can be set permanently in settings)

3 - (WORST) exécuter le ini de CoffeLibs : Fonctionne uniquement sur la console en cours d'utilisation. Il faudra executer le __init__ à chaque nouvelle console démarrée.

Résultat : Le chemin de CoffeeLibs sera enregistré. On pourra donc faire "import CoffeeLibs".

II - Contenu

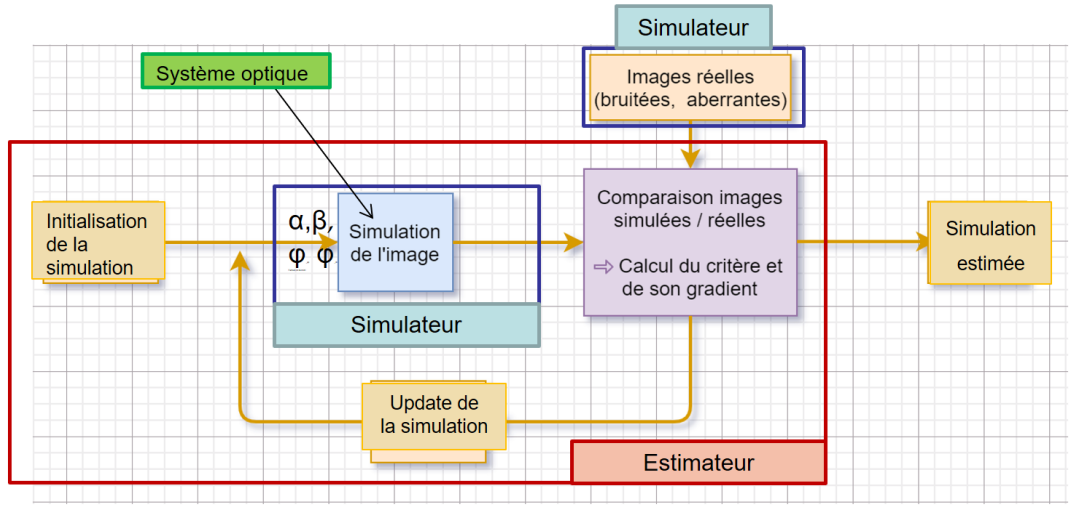

CoffeeLibs
CoffeeLibs
Coffee library
coffee.py
Contient les trois classes de Pycoffee
critère.py
Fonctions de critère appelé par le minimseur
file_manager.py
utilitaire de fichier (pas très utile)
pzernike.py
polynome de zernike (copie conforme de IDL)
tools.py
Fonction pour opération souvant répété
(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

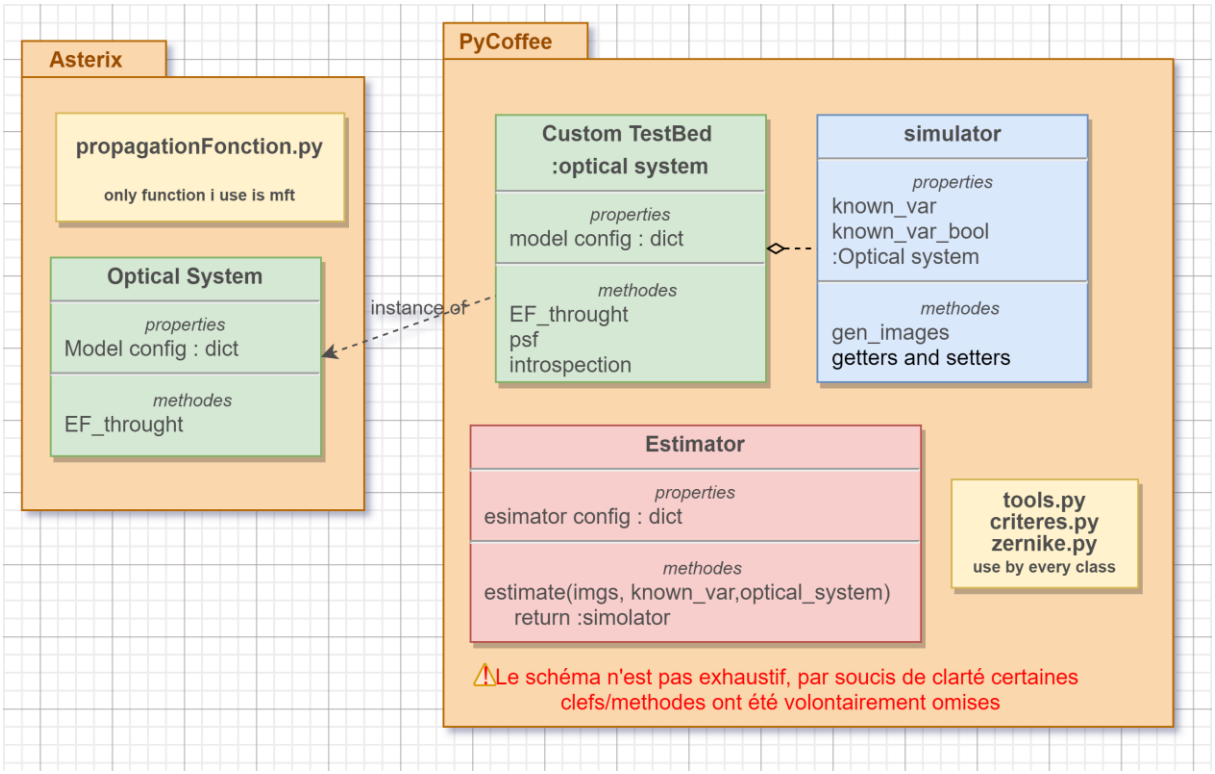
1 - Articulation du code

Le schéma ci-dessous présente le concept général d'une estimation de phase par minimisation d'un critère MAP. Les trois classes de pycoffee représentent les trois régions de code encadrées 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 classes de PyCoffee est représenté ci-dessous, (même code couleurs). La liste des propriétés et des méthodes est exhaustivement développé dans la section suivante.



III - Structure général

1 - .ini file (*dev Johan Mazoyer)*

Le fichier ini permet de définir les paramètres de construction des différents objets (Optical 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 "Template".

Fonctionnement : ConfigObj et Validator permettent de vérifier que votre fichier ini match bien les spécifications 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 (equiv structure IDL), cet objet peut ensuite être utilisé par les constructeurs.

2 - Les trois grande class PyCoffee.

Cette partie fait office de manuel. Elle décrit toutes les méthodes et propriétés des classes le plus exhaustivement possible.

Je ne décrirais pas ici les fonctions dans tools, pzernike ... mais vous pouvez jeter un coup d'œil, c'est commenté et c'est des opérations assez "élémentaires".

Critère.py en revanche contient des fonctions assez complexes 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 quelle autre instance de **OPTICAL SYSTEM** à 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 champ électromagnétique.

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

Propriété :

Nom	type	description
Propriétés héritées de la super classe Optical System	--	Ce référer à la documentation d'Astérix
pup	array	Carte pupille
corono	array	Carte coronographe
offset	dict	Paramètres d'offset pour que les mft soit centrées sur 4 pixels. Utilisés à chaque appel de mft
zbiais	bool	Si True, ajoutera la correction zbais dans le modèle de propagation (Par default : Ture si corono_type = 4q)

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 détecteur
* 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 reçu sur le détecteur
z_biais (internal)	aucun	`zbiais`	Revoie l'ajustement z_biais. Utilisé dans EF_through pour corriger 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 à travers le banc, plan par plan.

CLASS DATA_SIMULATOR

Definition :

Simulateur de données.

Ses propriétés sont les valeurs d'entrées de la simulation et le système optique. Se souviens aussi des variables qui sont supposées connues et celles qui sont supposées inconnues, même si on leurs assignent une valeur à posteriori.

Outil multitâche pour contenir tous les différents paramètres d'une simulation qui ne sont pas propre au système optique en lui même.

Contient des outils pour faciliter son utilisation avec l'estimateur

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

Nom	type	description
tbed	Optical system	Système optique
phi_foc	2D array	Carte de phase focalisé
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 = 3D	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.
Div_err	List of 2D array = 3D	Si mode myope, liste des cartes d'erreur sur la diversité. À 0 par default.
*EF_foc	2D	Carte de champ focalisé. Mis a jour si phi_foc est modifié. ATTENTION seulement dans ce sense.
* Nb_div	int	Nombre d'images de diversité
* N	int	Dimension (largeur) de la carte de phase
cplx	bool	True : phase upstream est un complexe False : phase upstream est un réel
myope	bool	True : estime erreur sur la diversité

Constructeur :

data_simulator(tbed,known_var="default",div_factors,phi_foc=None,cplx=False

Le constructeur a des mécanises un peu spécifique pour gérer les paramètres.

Sinon pour voir la définition des arguments, ce référer à la section propriété.

Known_var & known_var bool : (cf Propriété pour voir les clef possible)

Si une clef présente → Known_var_bool["clef"] = True. : Propriété défini connu

Si une clef manquante → Known_var_bool["clef"] = False. : Propriété défini inconu + valeur mise a default

Intérêt : Pour l'estimation, les propriété défini connu ne seront pas estimé.

Flux et fond : Clefs de Known_var bool

Si flux/fond est une liste -> Une valeur de flux/fond par image de diversité

Si flux/fond est un nombre - > Une valeur de flux/fond égale pour toute les diversité

Exception : si longueur de liste flux ou fond ≠ nombre d'image de diversité

div_factors :

Si div_factors est un array 2D -> Défini tel quel la carte de diversité

Si div_factors est un nombre - > Défini la carte de diversité 2D comme facteur * defoc

(Pas de gestion d'exception si vous mettez n'importe quoi)

phi_foc :

Si phi_foc = None (paramètre par défault) : phi_foc considéré inconnu

si phi_foc à une valeur : phi_foc considéré connu est ne sera pas estimé.

Pourquoi : La gestion de phi_foc connu est différente que les autres paramètres parce qu'il est un peu particulier; le placer en tant que clefs de knwon_var n'était pas très pratique

Méthodes :

Cette classe contient BEAUCOUP de méthodes car :

1 - On va exploiter les propriétés de mutabilité. Et c'est facile de ce tromper quand on fait pas attention.

(doc : http://cours.thirion.free.fr/Cours/Python-Liste-Dico/Python-Mutabilite.html)

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

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

3- on veux définir des méthodes pratique.

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

On peux classer ces méthodes en 5 catégories :

- Les setters and getters et checkers
- Les generators
- Les optimizes wrappers
- Les Tbed wrappers
- (les internal tools, comme genre set_default_value() que je ne vais pas décrire).

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.

DOCUMENTATION PYCOFFEE

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

Gen_zernike_phi_foc	Coeff : list des coeff du polynome de zernike	rien	Génère un polynôme 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	Génère un polynôme 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 génère 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 (for estimator)	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 (éventuellement sous forme de complexe).
Opti_pack (for estimator)	rien	Pack : 1D array	Prend en argument la list 1D des inconnus de la simulation (tel que la syntaxe d'optimize nous l'oblige) c-a-d en concaténant en array 1D toute les inconnu de la simulation
Opti_uptade (for estimator)	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 (internal)		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 (internal)	3D arrays	3D array	Prend en argument la carte des phase de diversité Renvoie les images

CLASS ESTIMATEUR

Definition :

Réalise une estimation des paramètres du système par critère de MAP. Minimisation L-BFGS-B avec optimize.minimize

!! ATTENTION option grad auto. Une fonction permet de calculer L à partir d'un système optique. Pour l'instant j'appelle la fonction a l'intérieur de la classe. mais il sera plus judicieux de passé une L déjà construit !!

Propriété :

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

Méthode unique :

estimate	Tbed : optical system System optique Imgs : 3D ndarray (N,N,nb_div) Images réels bout-à-bout 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 entrés connues. Si une clef n'est pas rempli, elle sera considérée comme inconnue et sera estimée. **Clefs possible**: → peuvent être initialisées comme un nombre (pour carte constante) Downstream_EF : carte de **CHAMP** d'aberration downstream. Peut être défini comme un float si phase constante. flux : **liste** des fluxs pour chaque image de diversité fond : **liste** des fonds pour chaque image de diversité	`e_sim` : object data_simulator Cet objet contiendra tous le résultat de l'estimation (sim.get_phi). + les propriétés info, info_grad, et info_div seront ajoutées à l'objets	Réalise l'estimation MAP. Etape 1 : cree un objet simulation avec tous les paramètes 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 dans les fonctions de critère, la simulation est mise à jour avec la current estimation des paramètres à estimer. (cf. opti_update et les fonctions préfixées 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 schéma ne vous donnera qu'un aperçu global.

Exemple avec TestPhaseDiv.py

[1er encadré bleu] Simulation de données (pour tester le simulateur)

[2er encadré bleu] Déroulé de l'estimation

Estimator

→ Initialisation de la phase (object data_simulator a 0)

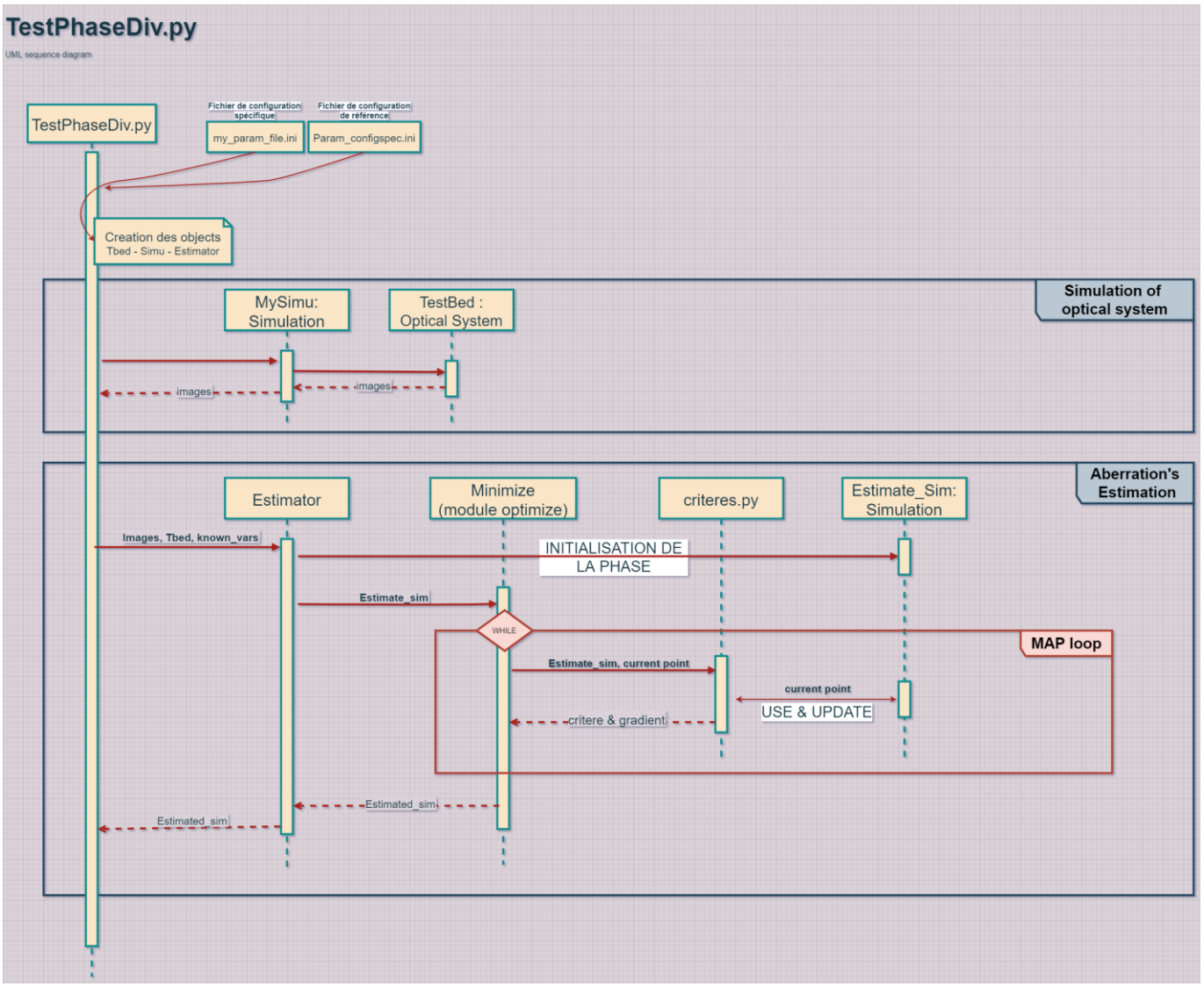
→ Appel Minimize

→ Appel les fonctions qui calcules le gradient et le critère (fct de critère.py préfixé par 'V_')

→ Update & use l'objet simulation pour calculer gradient/critère

Fin minimize

Fin Estimator return l'objet simulation dont les paramètres ont été estimés.



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 préfixées par 'V_' sont les fonctions destinées à être appelées par le minimiseur.

Ces fonctions appellent les autres fonctions pour calculer J, dJ, R.. Elles s'occupent de mettre à jour la simulation avec les nouvelles valeurs, calculer les trucs dont on a besoin, de gérer la pondération et de renvoyer le résultat mis en forme correctement 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'entrée
DJmv_up	Div_id Img Sim : data_simulator	2Darray	Renvoie le gradient du maximum de vraisemblance À partir des donne de l'object sim donné en entré et pour un indice de diversité donné (indice = entier), l'image de diversité correspondante
DJmv_dp	Div_id Img Sim : data_simulator	2Darray	Même chose que DJmv_up mais gradient par rapport a phase downstream
Dregul	Phi : s2Darray	2Darray	Renvoie la laplacien spatial (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 paramètres actuels 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	Fonctions appelées par optimize.minimize dans la méthode 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 déroulé (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 appelées dans les testes.

do_a_test, rest_estimator sont pour génère une simulation

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

Section 5 : définition des tests

Section 6 : main qui lance les tests.

ZERNIKE :

pzernike.py : génération de Zernike, identique au zernike IDL

PROCESS_IMG :

Dans le dossier img_processing_tools ce trouve deux fichiers.py.

Process_img.py contient des fonctions pour moyenner, croper, centrer, darker des fits.

Script.py les utilisent

TEMPLATE PLOTS :

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

Ces fonctions vont chercher les données à afficher, les infos (pondération, temps de traitement), bref tous les résultat de la minimisation et les met en forme.

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

COMPARE TRUC :

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

Scripts pour comparer des trucs.

IV - Bilan

Integration Asterix :

Si mode gradient automatique : Interchangeable avec n'importe quel autre instance de **OPTICAL SYSTEM** à 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. → Actuellement utilisable seulement avec Custom_bench de PyCoffee

Disclaimers :

1- Plus on cumule les inconnus, plus l'équilibre de l'estimation est fragile : La pondération des gradients, des termes de régularisations et les profiles de phases upstream/downstream doivent être cohérentes sinon ça ne marche pas, ou ça marchera mal. (J'ai pas trop le sens des ordres de grandeurs)

2- Borne de BFGS : Par soucis de simplicité je n'ai quasiment jamais mis de bornes, donc je ne sais pas exactement dans quel mesure ça peux être problématique.

Ce qui n'est pas implemente :

1- Gradient analytique : Pas de surech de la pastille.

Problème qui serait résolu en adaptant le gradient au coronographe d'Astérix

2- Il n'existe qu'un seule type de régularisation.

3 - Mode myope totalement implémenté MAIS la séparation err_div / phi_foc est pas bonne (images bien reconstruites cependant). Probablement une petite astuce dans coffee du genre soustraire le gradient upstream non ?

Ce qui est implemente mais dont je garanti pas la fiabilite :

Affichage : J'ai toujours pas réglé le problème pour les valeurs aberrantes dans le calcule des pondérations entre terme R / terme J.

Pondération par la variance du bruit : C'est implémenté mais parfois je désactive. (cf. disclamer 1)

Grad auto optimisation : La manière dont c'est codé n'est pas optimal niveau vitesse et utilisation mémoire

CE QUI MARCHE :

* mode grad automatique ~~ ok
* phi complex ~~ ok
* Regularisation qui marche ~~ ok
* L-BfGS-B ~~ ok
* PyCoffee = package ~~ ok
* phi_foc known ~~ in known_var_bool
  traitement un peu différent des autres
* div list prend des cartes ~~ ok
* centrage 4pix ~~ ok
* zbiaz ~~ ok
* plusieurs i_div ~~ ok
* estimation flux fond ~~ ok
* tests ~~ ok
* zernike identique à coffee ~~ ok

* Régularisation avec gradient automatique.

CE QUE J'AURAI AIME IMPLEMENTER MAIS J'AI PAS LE TEMPS:

Phi complex pour grad auto : adapter gradient, comme grad analytique

mode myope pour grad auto : exactement le même mécanisme qu'en mode analytique, juste a adapter. (gradient upstream/gradient myope identique)