

Sommaire

I - Installation	p.2
II - Contenu	p.3
III - Structure général	
1 - Articulation du code	p.4
.ini file	p.5
2 - Les trois classes de Pycoffee	p.5
Class Custom_TestBed	p.6
Class data_simulator	p.7
Class Estimator	p.11
Diagram UML de sequence de la methode estimate	p.12
3 - Criètre.py	p.13
IV - Outils spécifiques	p.14
V - Bilan	
1 - Problèmes	p.15
2 - Ce qui marche	p.16

DOCUMENTATION PYCOFFEE

I - Installation

INSTALLATIONS NECESSAIRE A PYCOFFEE :

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

MANUEL D'INSTALLATION :

Installation Package :

Attention, installer implique une copie des fichiers sur le système. Vous devrez re-installer 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'aurait 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 peut être importé depuis n'importe quel emplacement.

Installation développeur :

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

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

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

C'est pas "La bonne manière de faire". Ca permet de ranger les fichiers qui utilise coffee où on veut sans se poser de question de path et de working directory : quand je doit accéder à un fichier, je sais que je suis à l'emplacement du fichier que j'exécute.

La vraie bonne manière de faire c'est de configurer votre working directory au bon endroit.

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 prises en compte a chaque nouvelle exécution (sans avoir à re exécuter le `_ini_.py`)

Inconvénient : Fonctionne uniquement sur la console en cours d'utilisation. Il faudra exécuter le `__init__` à chaque nouvelle console démarrée.

II - Contenu

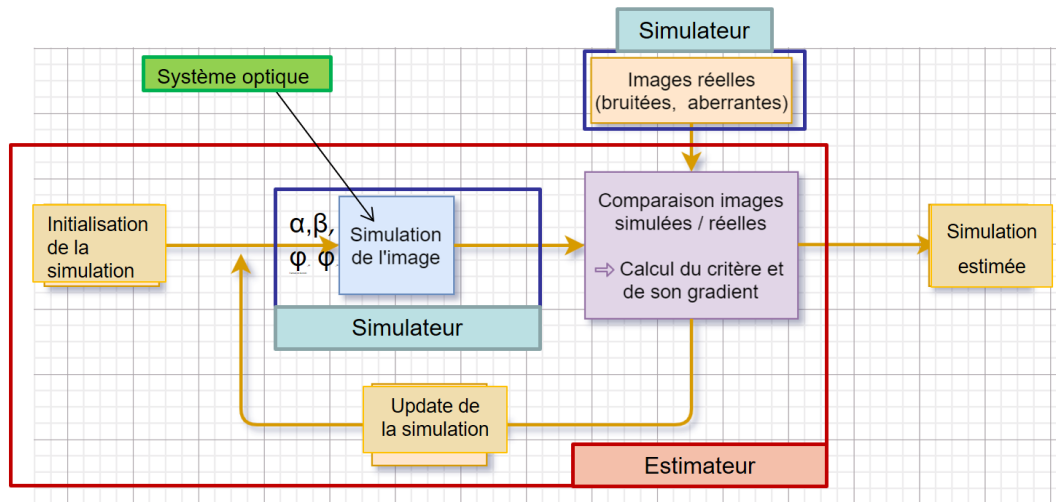
CoffeeLibs	
<u>CoffeeLibs</u>	Coffee library
coffee.py	Contient les trois classes de Pycoffee
critère.py	Fonctions de critère appelé par le minimiseur
file_manager.py	utilitaire de fichier (pas très utile)
pzernike.py	polynome de zernike (copie conforme de IDL)
tools.py	Fonction pour opération souvent répété (opération sur les matrices, template plots,..)
<u>Demos_Test</u>	Demos et tests
TestPhaseDiv.py	Exemple d'utilisation de Pycoffe sur données simulé
noCorono_donne_reel.py	Exemple d'utilisation de Pycoffe sur données 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
<u>Img_processing_tools</u>	Pré-traitements des images
my_script.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
<u>__init__.py</u> <u>setup.py</u>	Necessaire a l'installation en tant que module

III - Structure général

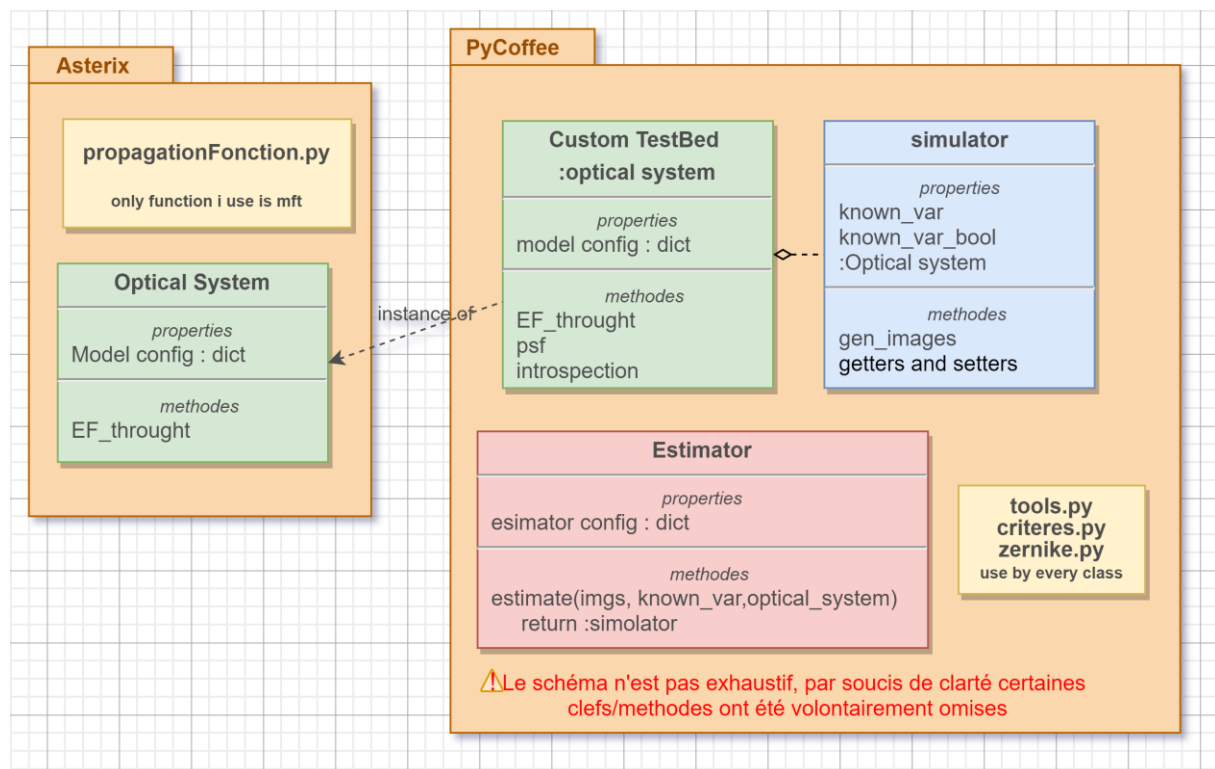
1 - Articulation du code

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 représentées les trois région 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 class de PyCoffee est représenté ci-dessous, en reprenant le même code couleurs. La liste des propriétés et des méthodes est exhaustivement développé dans la section suivante.



III - Structure général

.INI FILE : (dev Johan Mazoyer)

Le fichier ini permet de définir les paramètres de construction des différents objects (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 tempate.

Fonctionnement : ConfigObj et Validator permet de vérifier 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 ensuite 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, pznike ... mais vous pouvez jeter 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.

DOCUMENTATION PYCOFFEE

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 peut 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 hérité de la super class Optical System	--	Ce référe 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
zbais	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 <u>avant l'arrivé sur le detecteur</u>
* 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. <u>sans module carre</u>
* 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 detecteur
z_biais	aucun	zbais	Revoie l'ajustement z_biais. Utilisé dans EF_through pour corrigé 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.

DOCUMENTATION PYCOFFEE

CLASS DATA_SIMULATOR

Definition :

Simulateur de donnée.

Ces propriétés 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 leur 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 faciliter son utilisation avec l'estimateur

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

Nom	type	description
tbed	Optical system	Système optique
cplx	bool	<u>True</u> : phase upstream est un complexe <u>False</u> : 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: <u>Downstream_EF</u> : carte de CHAMP d'aberration downstream. Peut être défini comme un float si phase constante. <u>flux</u> : liste des flux pour chaque image de diversité <u>fond</u> : 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'objet. <u>Si div_factor float</u> : diversité défini comme a*defoc <u>Si div_factor array</u> : carte de diversité tel quel.
* Nb_div	int	Nombre d'images de diversité
* N	int	Dimension (largeur) de la carte de phase

DOCUMENTATION PYCOFFEE

Constructeur :

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

Le constructeur a des mécanismes un peu spécifiques 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 clefs possibles)

Si une clef présente → `Known_var_bool["clef"] = True`. : Propriété définie connue

Si une clef manquante → `Known_var_bool["clef"] = False`. : Propriété définie inconnue + valeur mise à default

Intérêt : Pour l'estimation, les propriétés définies connues ne seront pas estimées.

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 la diversité

Exception : si longueur de liste flux ou fond \neq nombre d'images 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 défaut) : `phi_foc` considéré inconnu

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

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

DOCUMENTATION PYCOFFEE

Méthodes :

Cette classe contient BEAUCOUP de méthodes car :

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

C-a-d lorsque l'on va modifier l'objet à l'intérieur d'une fonction, on veut qu'elle soit modifiée partout.
(explication un peu grossière, ce réfère une doc)

2 - On veut être sûr que quand on change quelque chose, tous les propriétés qui doivent être modifiées avec le sont.

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

3 - on veut définir des méthodes pratiques, par exemple une méthode qui permet de définir phi foc directement à partir de zernike.

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

Parmi ces méthodes on va retrouver :

- Les setters and getters et checkers
- Les generators
- Les optimizers 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 <code>set_<proprietes>()</code>	Truc que on veut 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 <code>get_<proprietes>()</code>	Rien <u>généralement</u> div_id quand il s'agit de récupérer une carte de phase/champ de diversité particulière	Le truc que on veut 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.
* Checkers <code>Truc_is_know</code>	rien	bool	Va chercher si le propriété est défini comme connu ou inconnu. A savoir 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	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 variable 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 sépare 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 <u>estime flux est fond en plus</u> (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

DOCUMENTATION PYCOFFEE

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'appelle la fonction a l'interieur de la classe. mais il sera plus judicieux de passé une L deja consruit au!!

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 priori de la phase. Fait office d'hyper paramètre pour la regularisation
auto	bool	<u>True</u> : gradient automatique -> appelle des fonctions gradient auto pour la minimisation <u>False</u> : gradient analytique -> appelle des fonctions gradient analytique pour la minimisation
cplx	bool	<u>True</u> : estime la phase complexe <u>False</u> : estime la phase réel

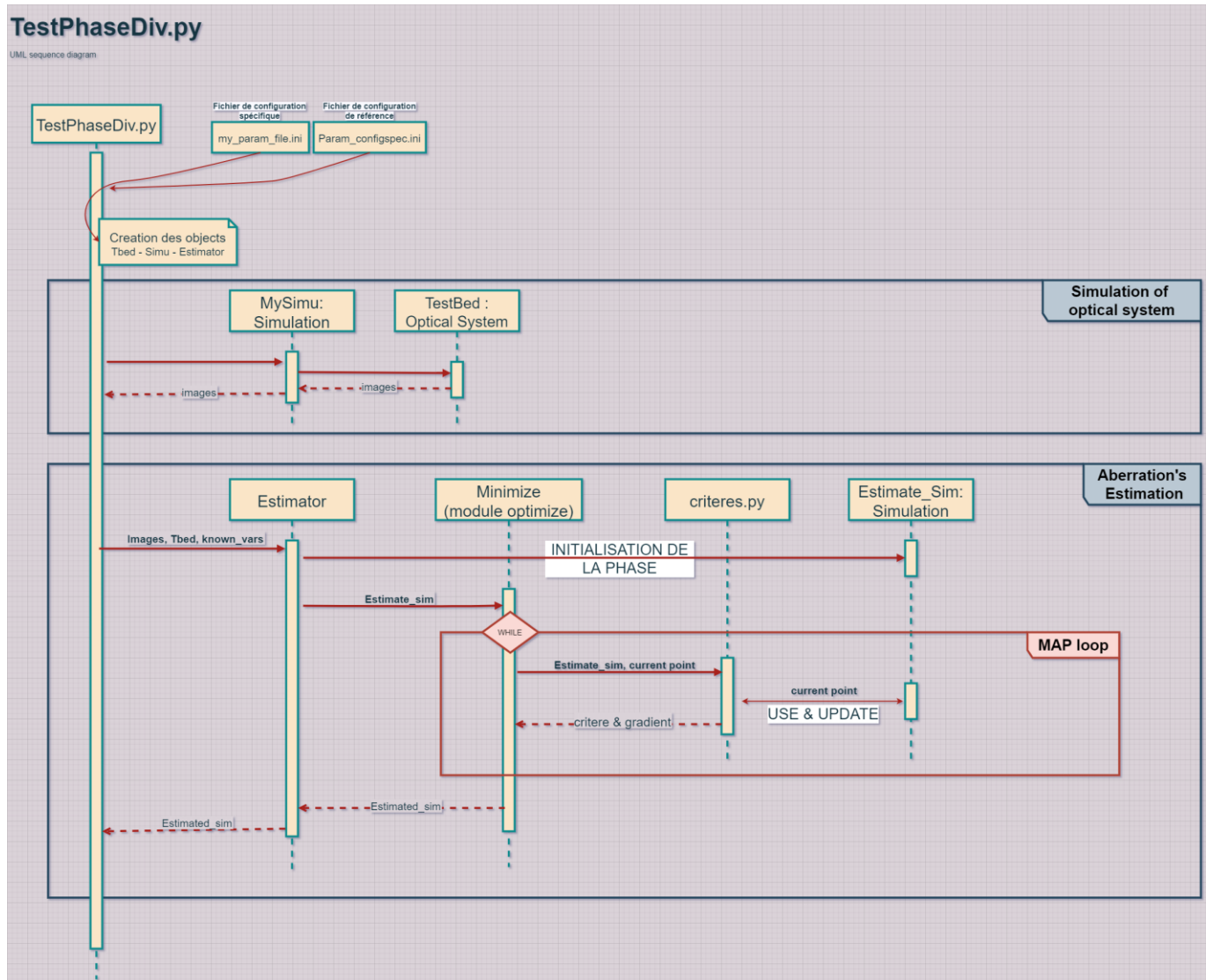
Méthode unique :

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

DOCUMENTATION PYCOFFEE

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.



DOCUMENTATION PYCOFFEE

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 appelle 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 resultat mis en forme correesctement 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 <u>Appelé dans sim.opti_uptadte</u> 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. <u>Etape 1</u> : uptate la simulation sim avec var (la sortie donnée par minimize 1D array) <u>Etape 2</u> : Calculer critère/gradient du critère pour chaque diversité, terme d'attache au donnée et regularisation <u>Etape 3</u> : la ponderation <u>Etape 4</u> : Enregister les infos pour le suivie de la simu -> ajoute les propriété info,info_gra,info_div a l'object sim <u>Etape 5 (optionel)</u> : sauvergarde les gradients pour suivi de la simu en affichant le dérouler (simGif) <u>Etape 6</u> : return (en adaptant a la syntax optimize si besion.)
Grad_diff_<truc>	/	grad	Gradient par différence fini. Cf commentaires et compar_grad.py dans demo_test

IV - Outils spécifique

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

TEST_FONCTIONNELLS.PY :

Dans demo_test : test_fonctionells.py

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

Le script 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, croqué, 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 ?

IV - Bilan

INTEGRATION 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. → Actuellement utilisable seulement avec Custom_bench de PyCoffee

DISCLAIMERS :

1- L'estimation (surtout pour le gradient Downstream) a un équilibre assez fragile : La pondération des gradients, des termes de régularisations et les profiles de phases upstream/downstream doivent être parfaitement cohérentes sinon ça ne marche pas, ou ça marchera mal.

C'est en particulier vrai quand on cumule les inconnus.

2- Borne de BFGS : Peux poser des problèmes quand elles ne sont pas assez large. Par soucis de simplicité pendant la phase de développement j'ai fait mes estimations sans bornes, donc je ne sais pas exactement dans quel mesure ça peut être problématique.

CE QUI EST IMPLÉMENTÉ MAIS QUI NE MARCHE PAS : Normalement, rien

CE QUI N'EST PAS IMPLÉMENTÉ :

1- Gradient analytique : Pas de surech de la pastille. Vrai pour coronograph Asterix mais gradient pas adapté.

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

3 - Mode myope (début d'implémentation, le gradient est similaire a celui de phi_up). Il faut juste quelques adaptations supplémentaire.

CE QUI EST IMPLÉMENTÉ MAIS DONT JE GARENTI PAS LA FIABILITÉ :

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

Pondération par la variance du bruit : ca marche mais avoir des valeurs trop faible c'est pas terrible donc parfois je désactive. (cf. disclaimer 1)

Regul grad auto avec Coronograph Asterix : Pour la régularisation, j'ai besoin de la taille de la pupille dans le plan overpaddé. Je vais la chercher comme : tbed.pupil.pup Mais si on change de tbed le chemin pourrait changer.

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
```

Grad auto : La régularisation et le mode phi complexe est aussi implémenté pour le gradient automatique.

Ce n'est pas le cas du gradient downstream, du flux et du fond.