

Sommaire

I - Installation	p.2
II - Contenu	p.3
III - Structure général	
1 - Articulation du code	p.4
<i>.ini file</i>	p.5
2 - Les trois classes de Pycoffee	p.5
<i>Class Custom_TestBed</i>	p.6
<i>Class data_simulator</i>	p.7
<i>Class Estimator</i>	p.11
<i>Diagramme UML de séquence de TestPhaseDiv.py</i>	p.12
3 - Criètrre.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 déroulé 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. (des objets en entiers, pas juste des fits.)</i>

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 peut "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)

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

DOCUMENTATION PYCOFFEE

II - Contenu

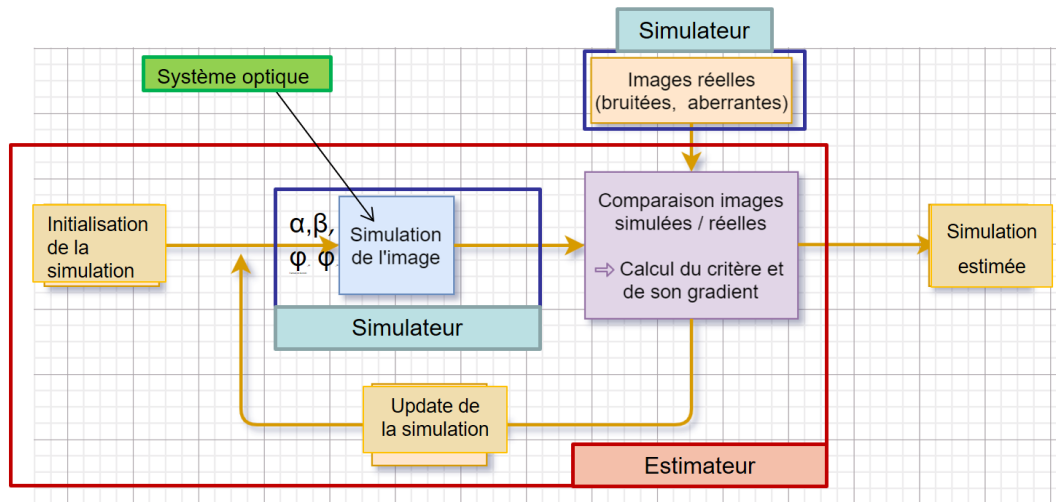
CoffeeLibs	
<u>CoffeeLibs</u>	Coffee library
coffee.py	Contient les trois classes de Pycoffee
critère.py	Fonctions de critère appelé par le minimiseur
file_manager.py	utilitaire de fichier (pas très utile)
pzernike.py	polynome de zernike (copie conforme de IDL)
tools.py	Fonction pour opération souvent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

III - Structure général

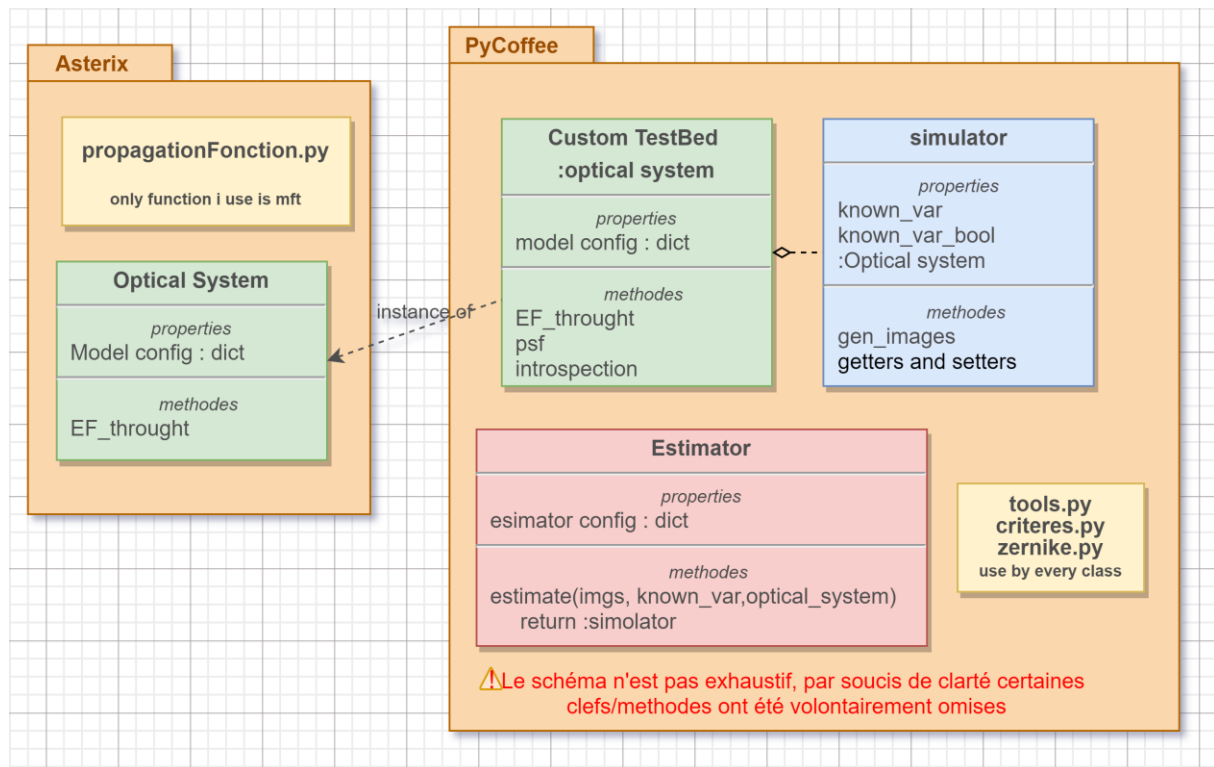
1 - Articulation du code

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 template 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, pznike ... 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.

DOCUMENTATION PYCOFFEE

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 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 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
zbais	bool	Si True, ajoutera la correction zbais dans le modèle de propagation (Par default : True 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 <u>avant l'arrivée sur le détecteur</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 détecteur. <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 détecteur
z_biais (internal)	aucun	zbais	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: <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
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. <u>Si div_factor float</u> : diversité défini comme $a \cdot \text{defoc}$ <u>Si div_factor array</u> : 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	<u>True</u> : phase upstream est un complexe <u>False</u> : phase upstream est un réel
myope	bool	<u>True</u> : estime erreur sur la diversité

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 ≠ 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` connu est différente que les autres paramètres parce qu'il est un peu particulier; le placer en tant que clefs de `known_var` n'était pas très pratique

DOCUMENTATION PYCOFFEE

Méthodes :

Cette classe contient BEAUCOUP de méthodes car :

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

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

2- On veut être sûr que quand on change quelque chose, toutes les propriétés qui doivent être modifiées aussi le soit.

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.

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

On peut 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 <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 certaines variables 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 getters pour certaines variables que je n'ai jamais eu besoin de changer.
* Ccheckers <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	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 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 (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 <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 (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

DOCUMENTATION PYCOFFEE

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 passer 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	<u>True</u> : gradient automatique -> fonctions gradient auto pour la minimisation <u>False</u> : gradient analytique -> fonctions gradient analytique pour la minimisation
cplx	bool	<u>True</u> : estime la phase complexe <u>False</u> : estime la phase réel
myope	bool	<u>True</u> : estime erreur sur la diversité

Méthode unique :

estimate	<p>Tbed : optical system System optique</p> <p>Imgs : 3D ndarray (N,N,nb_div) Images réels bout-à-bout</p> <p>Div_factor : list Liste de diversité connu</p> <p><u>Si liste de float</u> : diversité défini comme a*defoc <u>Si liste de array</u>: carte de diversité a ajouter a la phase focalisé.</p> <p>Knwo_var : dict Dictionnaire des entrés connues. Si une clef n'est pas rempli, elle sera considérée comme inconnue et sera estimée.</p> <p>Clefs possible: → peuvent être initialisées comme un nombre (pour carte constante) <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 fonds pour chaque image de diversité</p>	<p>e_sim : object data_simulator</p> <p>Cet objet contiendra tous le résultat de l'estimation (sim.get_phi).</p> <p>+ les propriétés info, info_grad, et info_div seront ajoutées à l'objets</p>	<p>Réalise l'estimation MAP.</p> <p><u>Etape 1</u> : cree un objet simulation avec tous les paramètres 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 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)</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.

Exemple avec TestPhaseDiv.py

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

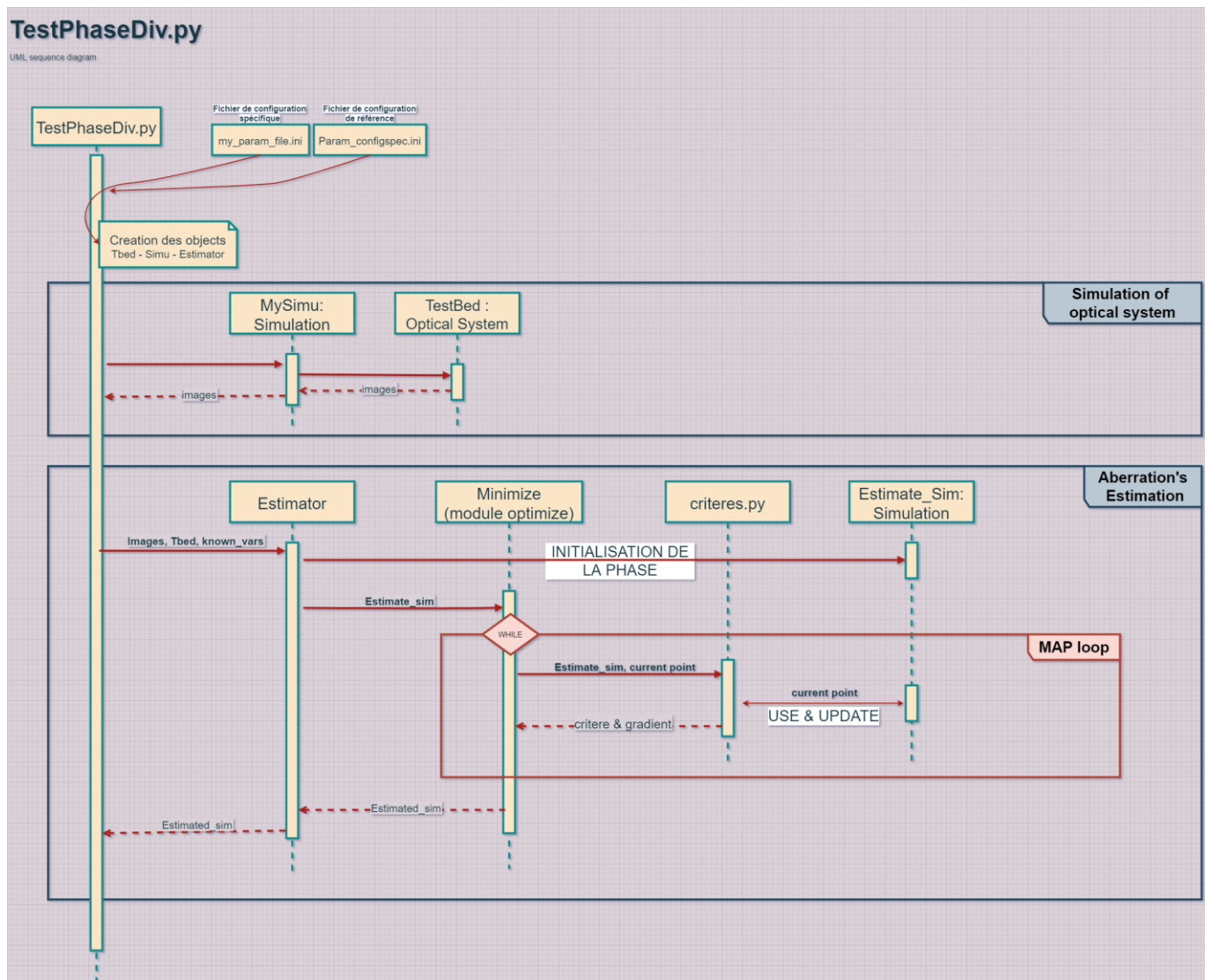
[2er encadré bleu] Déroulé de l'estimation

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.



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 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 <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	Fonctions appelées par optimize.minimize dans la méthode 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 déroulé (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 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 permettent d'afficher les résultats de la minimisation.

Ces fonctions vont chercher les données à afficher, les infos (pondération, temps de traitement), bref tous les résultats de la minimisation et les mets bien 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 et vérifier que ça marche

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. (ça ne marche pas/très mal)

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- J'ai ajouter un truc "NormA", pour normalisation amplifié. C'est pour que même quand il n'y a pas de flux, les images aient des valeurs élevé. C'est vraiment pas bien implémenté et c'est pas compatible avec l'estimation flux/fond

PQ? parce que comme ça $h(\phi)$ -img est plus élevé --> différence entre deux gradients consécutifs plus élevé aussi --> le point d'arrêt (qui basé sur ce critère) est atteint moins vite la précision machine.

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.

4- J'ai tendance à échanger les lettres des mots. Comme je retape quasiment jamais mes noms de variables je propage les erreurs. Je m'excuse d'avance pour toutes fautes (orthographe ou non) qui vont être pénible à corriger. Par exemple : info_gard au lieu de info_grad (pour les infos sur le gradient)

CE QUI EST IMPLEMENTE MAIS DONT JE GARANTI PAS LA FIABILITE :

Affichage : J'ai toujours pas réglé le problème pour les valeurs aberrantes pour l'affichage de la pondérations entre terme R / terme J (dans l'encadré jaune). C'est un peu aléatoire.. En plus ma pondération et pas bien réglé

Pondération : Dès que je change de mode je suis obligé de re-chercher c'est quoi la bonne pondération. J'ai pas été très rigoureuse sur ce point. Aussi les différents variables n'ont pas toujours de "réalité physique" (cf. disclaimer 1)

DOCUMENTATION PYCOFFEE

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. (mais pas adapté à Asterix car padding pup)

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

Tester rigoureusement ! : il y a un script de testes fonctionnelles pour ça mais tous ne marchera pas (cf. disclaimer 1). C'est le point d'amélioration majeur;

gradient downstream : pour qu'il marche vraiment bien, il faut mettre des bornes qui sont pas identique que celle de phi_foc je pense ?

Mode myope : Il faut une régularisation particulière pour que ça fonctionne.