Comment contribuer au Frido

19 mai 2017

Table des matières

Tout commence par télécharger les sources à l'adresse

`https://github.com/LaurentClaessens/mazhe`

1 Structure LaTeXienne du document

1.1 Pourquoi ça va être compliqué

Le fichier principal est `mazhe.tex` et les entêtes sont dans `e_mazhe.tex`. Le fichier principal contient la structures des chapitres et les `input` des fichiers correspondants.

Le document complet contient plusieurs parties :

— Le Frido (partie pour l'agrégation)

— Des chapitres de niveau recherche (ce que j'ai étudié et fait pour ma thèse)
— Des exercices et des corrigés
— Un cours d'«outils mathématiques» que j'ai donné.

Afin de compiler seulement la partie «Frido» (le plus souvent les gens qui lisent ces lignes s'intéressent à cette partie), j'ai écrit quelque scripts de précompilation qui servent à extraire les bons `input`, à mettre la bonne couverture et effectuer quelque autres aménagements. C'est cela que nous allons voir à présent.

1.2 Les types de fichiers

1.2.1 Les exercices

Les énoncés sont les fichiers **./src_exocorr/exo*.tex**. Chacun est associé à une correction **./src_exocorr/corr*.tex**.

1.3 Les figures

Les figures sont toutes créés par phystricks qui sert à générer les codes Ti*k*Z.

Les fichiers **phystricks*.py** sont les «sources» de ces figures. C'est avec eux qu'il faut travailler. Le script **figures_mazhe.py** crée les fichiers **Fig_*.pstricks** qui contiennent le code Ti*k*Z des figures et un peu plus [1].

Ti*k*Z est gourmand en ressources et compiler (pdflatex) un fichier contenant beaucoup de figures Ti*k*Z peut vraiment prendre du temps. Les créateurs de Ti*k*Z ont alors eu l'idée d'un mécanisme intéressant.

Ajoutons ceci au fichier principal :

```
\usetikzlibrary{external}
\tikzexternalize
```

Maintenant au moment d'arriver à une figure Ti*k*Z, il va vérifier dans un fichier intermédiaire `.md5` si le code Ti*k*Z a déjà été rencontré. Si oui, il inclut directement le fichier `.pdf` correspondant. Si non, il génère les fichiers `.pdf` et `.md5`.

Si les fichiers `.md5` et `.pdf` sont présents, cela accélère énormément la compilation. Sinon... Sinon ...

Sinon, il génère le pdf et pour ce faire, il semble qu'il doive recompiler en interne l'entièreté du document pour chaque figure. Autant dire qu'avec n figures en Ti*k*Z, vous multipliez le temps de compilation par n.

Le principe est alors :
— Compiler sans `external` complètement le document pour que toutes les références et compteurs soient justes.
— Compiler avec `external` le document. Prévoyez une longue balade en forêt.
— Maintenant vous pouvez toujours compiler avec `external`. Les pdf générés sont utilisés automatiquement et ça va assez vite.

Pour compiler le Frido sans `external` :

```
pytex lst_frido.py --no-external
```

1. La génération des figures est un sujet assez complexe en soi, et comme d'habitude j'ai la vanité de prétendre que mon script est plus fort que tout ce qui existe.

En bref : les fichiers **tikzFIG*.pdf** et **tikzFIG*.md5** sont des sous-produits de la compilation des figures Ti*k*Z. Elles sont fournies avec le dépôt git pour éviter une énorme compilation et un jeu de `external`.

1.4 Ce qu'il faut télécharger

Vous devrez télécharger un certain nombre de choses.
Les choses standards :

1. Une distribution LaTeX.
2. Python3 fonctionnel.
3. Le module pygit2 pour python3.

En ce qui concerne les choses dédiées au Frido :

Le paquet `exocorr` Vous devez récupérer le paquet `exocorr` à l'adresse

`https://github.com/LaurentClaessens/exocorr`

Seul le fichier `.sty` vous intéresse a priori. Mettez-le là où vous mettez vos paquets LaTeX.

Le module `latexparser` Vous le téléchargez à l'adresse

`https://github.com/LaurentClaessens/latexparser`

et vous le mettez quelque part là où Python pourra le retrouver.

Le script `pytex` Il est à l'adresse

`https://github.com/LaurentClaessens/pytex`

Note. Il est conseillé de mettre tous ces fichiers dans des répertoires séparés, obéissant à une certaine logique : les paquets LaTeX avec les autres paquets LaTeX, les modules python avec les autres modules python. Cela surtout si vous comptez compiler souvent. Si votre but est seulement de compiler pour tester, vous pouvez tout aussi bien mettre `pytex.py` et `exocorr.sty` dans le répertoire courant.

1.5 Compiler tout le document

Lorsque tout est téléchargé et correctement configuré (`latexparser` doit être trouvable par python et `pytex` trouvable par votre terminal), vous compilez le Frido avec

```
pytex lst_frido.py
```

Le script s'occupe d'extraire de `mazhe.tex` les choses nécessaires au Frido, crée un fichier intermédiaire et le compile. Des passes de `bibtex` et `makeindex` sont également automatiquement effectuées.

Les `ref` et `eqref` ne correspondant à aucun `label` sont indiqués. Il ne devrait y en avoir aucun.

La compilation produit deux fichiers `pdf`. Le premier est `Inter_frido-mazhe_pytex.pdf` qui est créé par LaTeX lui-même durant la compilation. Le second est `0-lefrido.pdf` qui en est une simple copie effectuée après la compilation. Vous devriez ouvrir `0-lefrido.pdf` de façon à éviter que votre lecteur de `pdf` ne se mette en mode «rafraichissement» durant toute la durée de la compilation.

1.6 Compiler seulement une partie

Lisez le fichier `lst_exemple.py` :

```
#! /usr/bin/python
# -*- coding: utf8 -*-

from __future__ import unicode_literals

import latexparser
import latexparser.PytexTools
import commons
import plugins_agreg

myRequest = latexparser.PytexTools.Request("mesure")
myRequest.ok_hash=commons.ok_hash

myRequest.add_plugin(plugins_agreg.set_isAgreg,"before_pytex")

myRequest.original_filename="mazhe.tex"

myRequest.ok_filenames_list=["e_mazhe"]
myRequest.ok_filenames_list.extend(["gardeFrido"])
myRequest.ok_filenames_list.extend(["43_mesure"])
myRequest.ok_filenames_list.extend(["56_espace_vecto_norme"])
myRequest.ok_filenames_list.extend(["134_choses_finales"])


myRequest.new_output_filename="0-exemple.pdf"
```

A priori la seule chose qui vous intéresse est la liste des `ok_filename`. Je crois qu'elle est assez auto-explicative. Le fichier principal `mazhe.tex` contient une série de `input`. Seuls ceux de la liste seront effectués.

La ligne `new_output_filename` donne le nom du fichier de sortie. En l'omettant, ce sera un nom compliqué du type `0-Inter_...`. Pour compiler :

```
pytex lst_exemple.py
```

Il fera automatiquement autant de tours de `pdflatex`, `makeindex` et `bibtex` que nécessaire pour que toutes les références soient bien faites [2].

Après compilation, une liste des références incorrectes est donnée. Bien entendu si vous ne compilez qu'une partie, il risque d'y avoir beaucoup de références *manquantes*, mais il ne devrait n'y avoir aucune références *duplicate* !

2 Convention de nommage

Déjà il semble que «nommage» soit un anglicisme.

2. Bug connu : pour que la bibliographie soit correcte, il faut lancer deux fois `pytex`.

2.1 Les figures

Passons sur les figures. Si vous devez en inclure une, faites un `png` ou du `tikz`, n'importe quoi, mais qui fonctionne avec `pdflatex`.

2.2 Les fichiers `tex`

J'ai pris l'habitude de préfixer les noms par un nombre. Par exemple `139_EspacesVecto`. Le fait est qu'il est plus simple, pour ouvrir le ficher, de taper `139<TAB>` que de se souvenir si «EspacesVecto» est écrit avec une majuscule, en français, en anglais, ...

De plus un chapitre contenant plusieurs fichiers, on est vite avec `EspacesVecto1`, `EspacesVecto2`, etc.

Vous trouverez le prochain numéro disponible dans `réserve.tex`.

2.3 Le code à inclure

Pour inclure du code Sage, nous utilisons `\lstinputlisting` . Ici encore, le fichier `réserve.tex` contient le prochain disponible.

2.4 Pour les exercices

Les exercices sont tapés dans les fichiers déjà pré-remplis `src_exocorr/exo*.tex`. Les corrections sont dans le fichier `src_exocorr/corr*.tex` correspondant. Ces fichiers ne sont pas inclus directement, mais via la macro `\Exo`.

Le fichier `réserve.tex` contient le prochain disponible.

Exemple Vous voulez créer un exercice.

- Allez voir dans `réserve.tex` la prochaine ligne `Exo` disponible.
- Mettons que ce soit `\Exo{mazhe-0018}`
- Supprimez cette ligne de `réserve.tex`, et mettez la où vous voulez voir apparaître votre exercice.
- Tapez votre exercice dans `src_exocorr/exomazhe-0018.tex` et votre correction dans `src_exocorr/corrmazhe-0018.tex`. Ces fichiers sont déjà créés et pré-remplis. Ne changez pas le code qui y est.

3 Les politiques éditoriales

Certaines parties de ce texte ne respectent pas les politiques éditoriales. Ce sont des erreurs de jeunesse, et j'en suis le premier triste.

3.1 Licence libre

Je crois que c'est clair.

3.2 pdflatex

Tout est compilable avec pdfLaTeX. Pas de pstricks, de psfrag ou de ps<quoiquecesoit>.

3.3 utf8

Je crois que c'est clair.

3.4 Notations

On essaie d'être cohérent dans les notations et les conventions. Pour la transformée de Fourier par exemple, je crois que la définition du produit scalaire dans L^2, des coefficients de Fourier, de la transformation et de la transformation inverse sont cohérents. Cela demande, lorsqu'on suit un livre qui ne suit pas les conventions utilisées ici, de convertir parfois massivement.

3.5 De la bibliographie

On évite d'écrire en haut de chapitre «les références pour ce chapitre sont ...». Il est mieux d'écrire au niveau des théorèmes, entre parenthèses, les références.

Lorsqu'on écrit l'énoncé d'un théorème sans retranscrire la démonstration, il faut mettre une référence vers un document *en ligne* qui en contient la preuve. Il est vraiment fastidieux de chercher une preuve sur internet et de tomber sur des dizaines de documents qui donnent l'énoncé mais pas la preuve.

3.6 Faire des références à tout

Lorsqu'un utilise le théorème des accroissements finis, il ne faut pas écrire «d'après le théorème des accroissements finis, blablabla». Il faut écrire un `\ref{}` explicite vers le résultat. Cela alourdit un peu le texte, mais lorsqu'on joue avec un texte de plus de 1500 pages, il est parfois laborieux de trouver le résultat qu'on cherche (surtout s'il existe plusieurs versions d'un résultat et que l'on veut faire référence à une version particulière).

3.7 Des listes de liens internes

Le début du Frido contient une espèce d'index thématique. Il serait bon de l'étoffer.

3.8 Pas de références vers le futur

Dans le Frido, *aucune* preuve ne peut faire une référence vers un résultat prouvé plus bas. On n'utilise pas le théorème 10 dans la démonstration du théorème 7. Cela est une contrainte forte sur le découpage en chapitres et sur l'ordre de présentation des matières.

Il est bien entendu accepté et même encouragé de mettre des notes du type «Nous verrons plus loin un théorème qui ...». Tant que ce théorème n'est pas *utilisé*, ça va.

En faisant

```
pytex lst_frido.py --verif
```

vous aurez une liste des références vers le bas. Cette liste doit être vide ! Ce programme cherche tous les `\ref{}` et `\eqref{}` ainsi que les `\label{}` correspondants et vous prévient lorsque le `\label{}` est après le `\ref{}`.

Si vous pensez qu'une référence pointée doit être acceptée (par exemple parce c'est dans une des listes de liens internes), alors vous ajoutez son hash dans la liste du fichier `commons.py`. Si il s'agit vraiment d'une référence vers un résultat que vous utilisez, alors vous devez déplacer des choses. Soit votre résultat vers le bas, soit celui que vous utilisez vers le haut. Parfois cela demande de déplacer ou redécouper des chapitres entiers... Si il n'y a vraiment pas moyen, bravo, vous venez de prouver que la mathématique est logiquement inconsistante.

4 Vérifier si vous n'avez pas fait de bêtises

Lorsqu'on fait de lourdes modifications (déplacement de parties, fusion de théorèmes, etc) il est toujours possible de faire des bêtises d'au moins deux types : créer des références vers le futur et supprimer des parties (genre couper-coller en oubliant le coller). Pour s'en prémunir, le script suivant lance quelque compilations et vérifications :

```
./testing.sh
```

Aucune erreur ne devrait être signalée.

Attention : ce script fait quelque manipulations à base de `git stash` et crée une nouvelle branche (nom aléatoire assez long) pour tester votre dernière modification sans créer de commit.