ENSIMAG

P.P.M.M.F

Comparaison des algorithmes pour le fit de modèles RegArch

IF - MeQA

Voong Kwan &
Ruimy Benjamin &
Ibakuyumcu Arnaud.

10/03/2017

Table des matières

1 Introduction

Ce projet a pour but de tester les différents algorithmes d'optimisation utilisés lors du *fit* des modèles de type Arch.
Pour les différents modèles, nous avons testé la convergence, la vitesse de convergence ainsi que l'erreur quadratique lors de l'estimation des paramètres. Le but étant d'utiliser pour chaque modèle le 'meilleur' algorithme pour gagner en vitesse, gagner en précision et gagner en stabilité.
Nous avons automatisé les tests de performance avec des scripts R présent dans le dossier `testPerformance`

2 Algorithmes de GSL

Dans cette section, nous testerons sur les modèles ARCH et GARCH l'efficacité des algorithmes de la librairie GSL pour décider d'un 'meilleur' algorithme, en fonction de la taille du modèle, selon nos critères de :

- Vitesse : en temps de convergence.
- Erreur : somme des écarts au carré avec le vrai paramètre de chaque paramètre.
- Convergence : présence ou non de NaN.

Voici les étapes pour tester chacune des mesures d'efficacité ci-dessus :

1. Création d'un modèle étalon.
2. n simulation de trajectoire avec ce modèle.
3. Pour chaque simulation, estimation des paramètres du modèle avec des points initials différents des vrais paramètres pour chaque algorithme.
4. Moyenne des résultats sur les n simulations par algorithme.

2.1 Modèle de type Arch

Dans cette sous-section, nous traiterons l'efficacité des algorithmes pour des modèles Arch de différentes tailles et en faisant varier les points initiaux.

2.1.1 Taille des paramètres variables

Vitesse

Voici le graphique des temps de convergence selon le nombre de paramètres du modèle Arch.

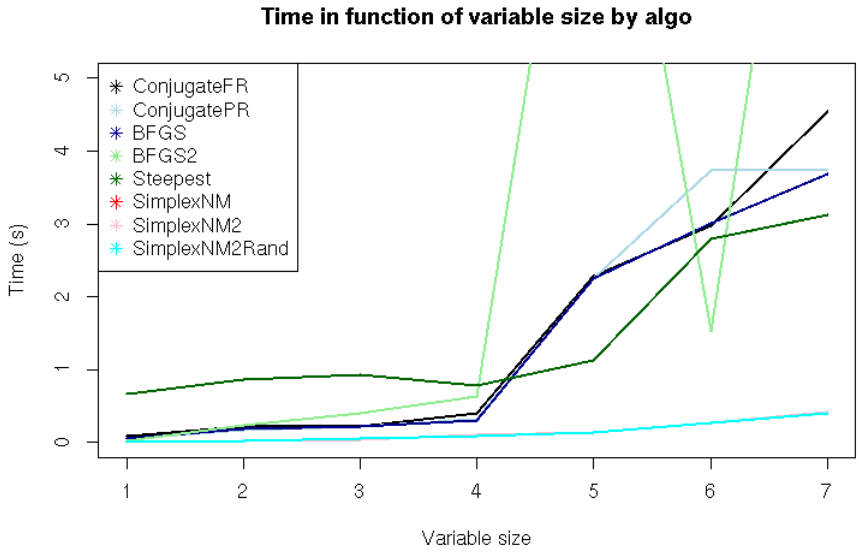

Figure 1 : Temps de convergence par taille et algorithme

On remarque que jusqu'à un modèle d'Arch de taille 4, les temps de convergence restent à peu près stable et inférieur à 1 seconde.
Après un nombre de 4 paramètres, les différents algorithmes se séparent en temps de convergence. Le `BFGS2` devient instable en moyenne, les autres algorithmes (outre les `Simplex`) semblent réagir de manière affine, quant au nombre de paramètres, sur le temps de convergence.
Dans tous les cas, les algorithmes de type simplex sont les plus performants sur le critère du temps de convergence.

Gagnant : `SimplexNM`

Erreur

Voici le graphique des erreurs en fonction du nombre de paramètres du modèle Arch.

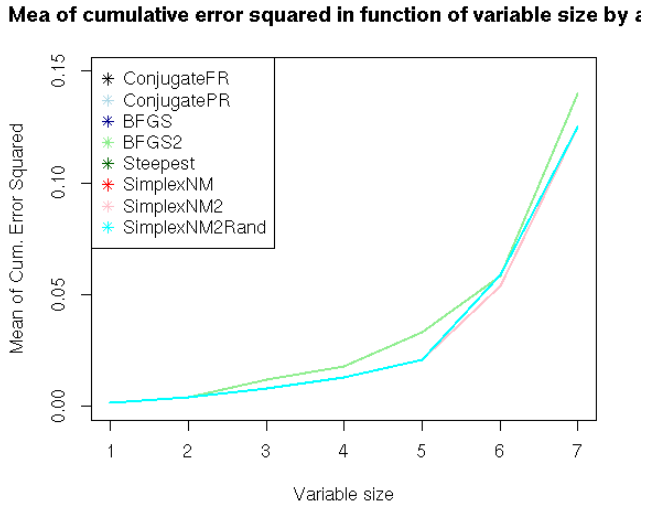

Figure 2 : Erreur par taille et algorithme

Avant de commenter ce graphique, nous rappelons que l'erreur calculé est la somme des écarts au carré entre paramètres étalons et paramètres estimés. Ainsi, s'il y a deux paramètres, le résultat sera la somme entre l'écart au carré sur le paramètre 1 et l'écart au carré sur le paramètre 2. Il est donc logique de voir une fonction croissante sur ce graphique. Ce qui est plus étonnant, c'est que nous nous attendions à voir une fonction affine alors que ce n'est pas du tout le cas. Ceci implique que pour un modèle de taille n, l'erreur par paramètre (en divisant par n) n'est pas égale à l'erreur d'un modèle de taille 1. Et donc que le fait d'ajouter des paramètres peut augmenter ou diminuer l'erreur par paramètre.

Nous pouvons remarquer ici que pour chaque algorithme l'erreur est à peu près identique et croît comme une fonction carré. Le `BFGS2` est toujours un peu moins efficace que les autres tandis que les algorithmes `simplex` semblent être les plus efficace en terme d'erreurs.

Gagnant : `SimplexNM`

2.1.2 Points initiaux

Ici, nous avons calculés le temps de convergence et l'erreur en fonction de la distance entre le point initial pour estimer et le point étalon pour un modèle Arch à 1 paramètre.

Vitesse

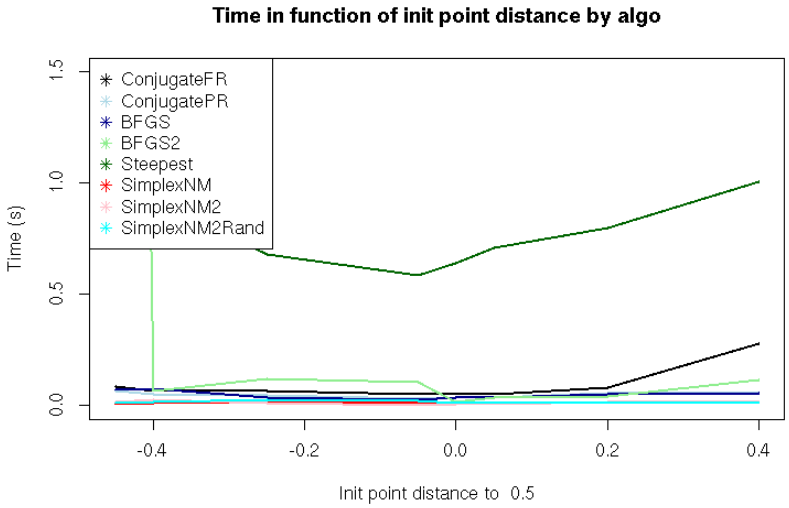

Figure 3 : Temps de convergence selon le point initial par algorithme

Conformément à la première figure, l'algorithme de `Steepest` est le plus long. L'algorithme de `BFGS2` semble être instable (vitesse pour un point initial de 0.1 lorsque le point étalon est de 0.5).
Pour séparer les autres algorithmes, nous avons 'zoomé' ce qui donne :

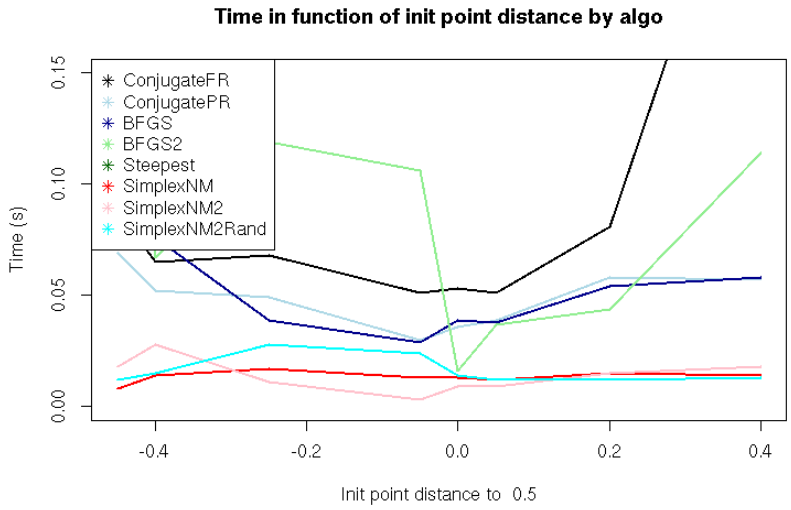

Figure 4 : Temps de convergence selon le point initial par algorithme

Les algorithmes de `ConjugateFR` et `BFGS2` semble donner une courbe quadratique par rapport à la distance au point étalon. Les algorithmes de `ConjugatePR` et `BFGS` semblent être stable pour une distance à droite du point étalon et quadratique à gauche. Les `simplex` restent stables et quasiment constants.

Gagnant : `SimplexNM`

Erreur

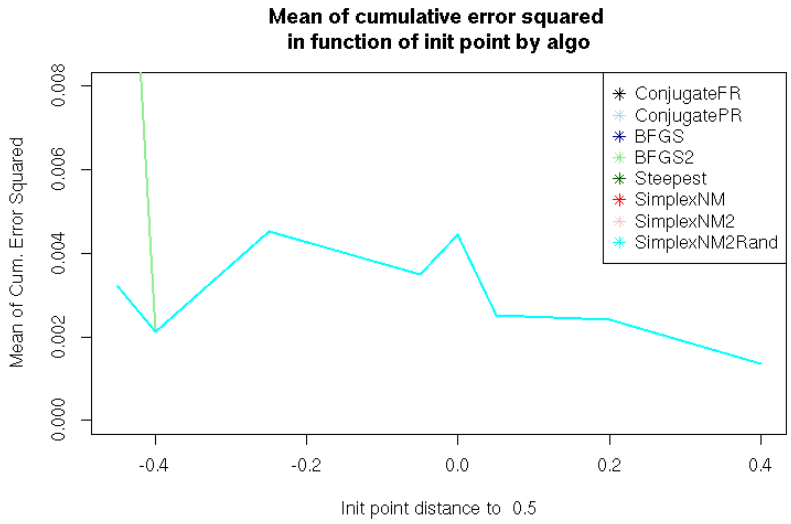

Figure 5 : Erreur selon le point initial par algorithme

Seuls les algorithmes `simplex` convergent pour chacun des points initiaux testés. Les autres ont des NaN pour certain points et `BFGS2` explose.

Gagnant : `SimplexNM`

2.2 Modèle de type Garch

Nous avons pu, compte tenu d'un temps très long d'estimation pour les modèles Garch, allé qu'à des tailles de 3 paramètres pour ce modèle

2.2.1 Taille des paramètres variables

Comme pour la section sur le modèle Arch, voici quelques indications de performance en faisant varier le nombre de paramètre du modèle Garch.

Vitesse

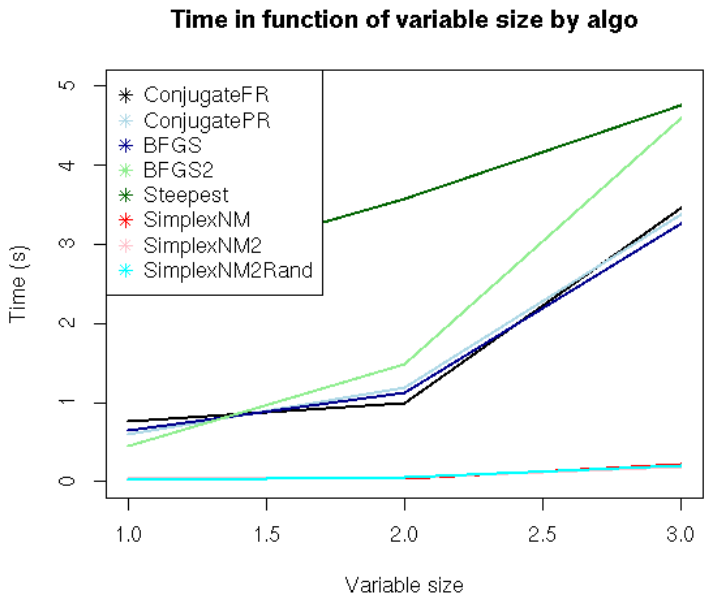

Figure 6 : Temps de convergence par taille et algorithme

De même que pour le modèle Arch, les algorithmes les plus stables et rapides pour le modèle Garch sont les algorithmes de `simplex`. Les autres 'explosent' jusqu'à des durées moyennes de $3-4$ secondes dès que le nombre de variable dépasse 2. L'algorithme de `Steepest` commence à 3 secondes pour un modèle à 1 variable mais semble stable pour des modèles plus grands.

Gagnant : `SimplexNM`

Erreur

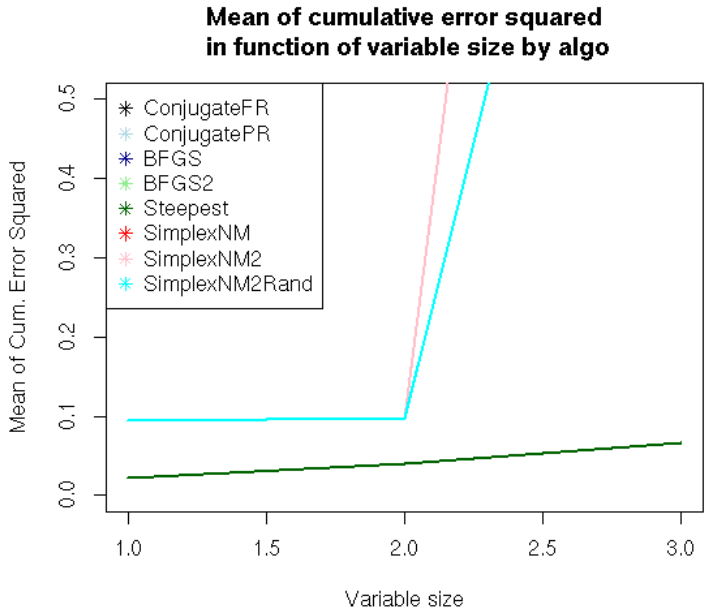

Figure 7 : Erreur par taille et algorithme

Pour les algorithmes du `Simplex` et `Conjugate` l'erreur explose après 3 paramètres. Les algorithmes `BFGS`, `BFGS2` contiennent de nombreux NaN. Seul l'algorithme du `Steepest` semble rester stable et précis pour des modèles Garch de toute taille.

Gagnant : `Steepest`

2.2.2 Points initiaux

Ici encore, nous testons l'efficacité des algorithmes sur un modèle Garch à un paramètre (de valeur 0.5) en jouant sur le point initial lors de l'estimation.

Vitesse

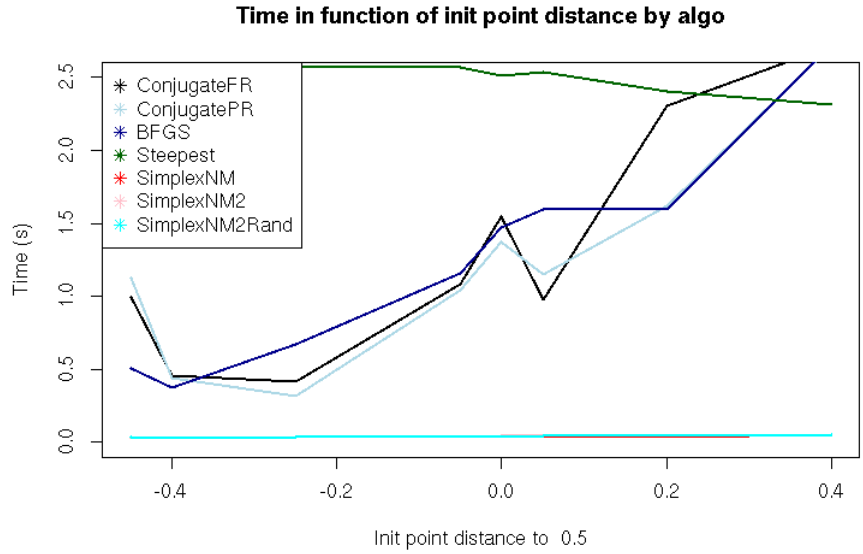

Figure 8 : Temps de convergence selon le point initial par algorithme

Les algorithmes `ConjugateFR/PR` sont sensibles, au niveau vitesse de convergence, au choix du point initial. L'algorithme de `steepest` est stable même si sa vitesse reste assez élevé (autour de 2.5 secondes).
Encore une fois, les algorithmes de `simplex` sont les plus rapides et restent plus ou moins stables même si on peut noter une augmentation du temps de convergence plus le point initial est à droite du point étalon comme le démontre cette version zoomée :

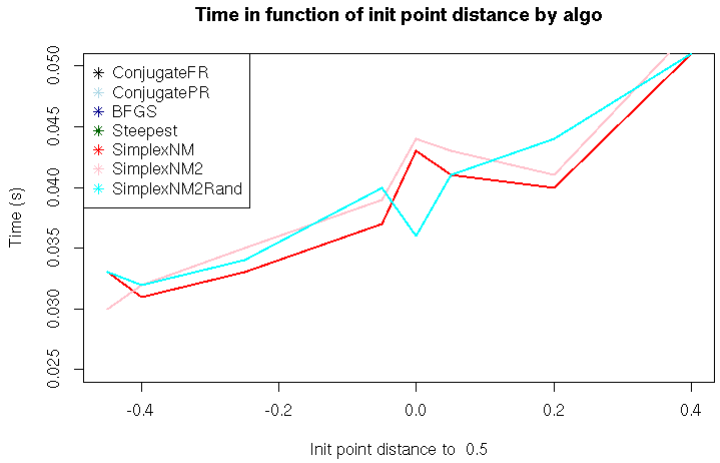

Figure 9 : Temps de convergence selon le point initial par algorithme

Gagnant : `SimplexNM`

Erreur

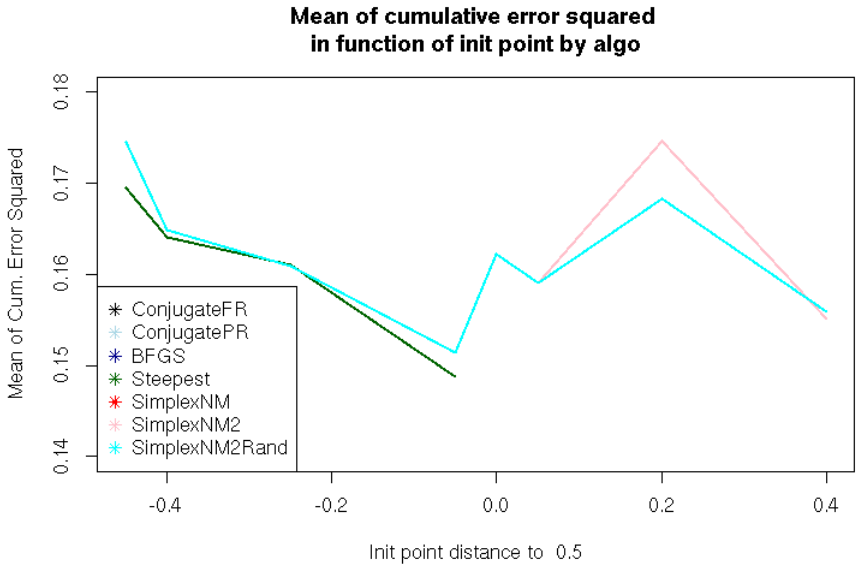

Figure 10 : Erreur selon le point initial par algorithme

Nous pouvons remarquer que les algortithmes de `simplex` restent stables pour n'importe quel point initial mais l'erreur est assez importante. L'algorithme du `steepest` connait la même précision que ceux du `simplex` mais ne converge plus lorsque le point initial est à droite du point étalon (une solution serait donc de prendre 0 comme point initial à chaque fois).
Les autres algorithmes ne convergent pas pour des points éloignés du point étalon.

Gagnant : `SimplexNM` et `Steepest` pour des points à droite

2.3 Conclusion

On a donc pu remarquer qu'en pratique, et avec les mesures d'efficacité que nous avons choisies, l'algorithme du `simplex`[1] est le 'meilleur' pour les modèles Arch et Garch pour des modèles en dessous de 3 paramètres parmi les algorithmes de la librairie GSL. Pour des modèles Garch à 3 ou plus de paramètres, c'est l'algorithme `steepest` qu'il faut choisir.
Il pourrait être intéressant de les comparer aux algorithmes de la librairie NLOPT (voir section suivante) et même de combiner plusieurs algorithmes (GSL et NLOPT) pour estimer de manière plus précise les paramètres tout en gardant un temps raisonnable de convergence.
Une autre idée d'amélioration serait de jouer sur les paramètres de tolérance de chaque algorithme.

Les scripts utilisés se trouvent dans le dossier.

3 Algorithmes de NLOpt

3.1 Introduction & Explication du script

Nous avons testés les algorithmes de NLOpt pour le **modèle ARCH** de moyenne nulle et avec uniquement une constante et un paramètre.
Il y a 44 algorithmes et une partie d'entre eux n'est pas fonctionnelle pour le modèle ARCH. D'autres sont très long pour une seule simulation ce qui ne nous permet pas réellement de tester les erreurs moyennes des différents algorithmes. Enfin parmi les algos qui sont fonctionnelles, certaines données ne sont pas disponible afin de pouvoir comparer les algos entre eux.
Cette partie est donc dédiée à la catégorisation des différents algorithmes pour le **modèle ARCH**.
La démarche que nous avons suivie peut être adaptée à n'importe quel modèle. Il convient simplement de récupérer notre script de test `testArchNLOPT.R` et de l'adapter aux nouveaux modèles.
Une liste d'algorithme appelée "ListAlgosNLOPT" qui est la concaténation de string contenant le nom des algorithme permet de tester l'ensemble des algorithmes sur la simulation courante de façon **automatisée**.

Il convient alors de remplir cette liste avec l'ensemble des algorithmes possible et de les catégoriser au fur et à mesure. On detecte alors facilement quels sont les algorithmes qui ne convergent pas, ou encore ceux qui entraînent des erreurs. On les retire de la liste au fur et à mesure en les catégorisants.
Il convient de faire bien attention aux nombres de paramètres N (en l'occurrence 2 dans notre cas), et au vecteur Param qui contient les valeurs des paramètres.
Enfin il ne faut pas oublier de changer, si on utilise un autre modèle, le modèle initiale (mod2 dans notre script de test).

3.2 Algorithmes qui entraînent des erreurs pour le modèle ARCH

Les algorithmes suivant entraînent tous l'erreur "division par 0" :

1. Une des versions parmi : `SimplexNM`, `SimplexNM2` et `SimplexNM2Rand`

- nlopt_gn_direct
- nlopt_gn_direct_l
- nlopt_gn_direct_l_rand
- nlopt_gn_direct_noscal
- nlopt_gn_direct_l_noscal
- nlopt_gn_direct_l_rand_noscal
- nlopt_gn_orig_direct

Ainsi, ces algorithmes ne peuvent pas être utilisés pour les modèles ARCH.

3.3 Algorithmes qui ne convergent pas pour le modèle ARCH

Les algorithmes qui ne convergent pas sont les suivants :

- nlopt_gn_crs2_lm
- nlopt_gn_mlsl
- nlopt_gd_mlsl
- nlopt_gn_mlsl_lds
- nlopt_gd_mlsl_lds
- nlopt_ld_mma
- nlopt_ln_newuoa
- nlopt_ln_sbplx
- nlopt_ln_auglag
- nlopt_ld_auglag
- nlopt_ln_auglag_eq
- nlopt_ln_bobyqa
- nlopt_ld_slsqp
- nlopt_ld_ccsaq

Ces algorithmes sont très très long et vont toujours jusqu'au time out, de plus ils n'arrivent pas à converger.

3.4 Algorithmes qui convergent lentement ($\geq$ 1mins) pour le modèle ARCH

Les algorithmes qui convergent pour le modèle ARCH mais légèrement au dessus d'une minute sont au nombre de deux. Ils ne doivent pas être utilisés pour les modèles ARCH tout simplement parce qu'il existe de bien meilleurs algorithmes. (voir plus loin)

- nlopt_ld_tnewton_precond_restart
- nlopt_ln_newuoa_bound

3.5 Algorithmes qui terminent rapidemment pour le modèle ARCH

Enfin les algos qui se terminent rapidement pour le modèle ARCH sont les suivants :

- nlopt_gn_orig_direct_l
- nlopt_gd_stogo_rand
- nlopt_gd_stogo
- nlopt_ld_lbfgs_nocedal
- nlopt_ld_lbfgs
- nlopt_ln_praxis
- nlopt_ld_var1
- nlopt_ld_var2
- nlopt_ld_tnewton
- nlopt_ld_tnewton_restart
- nlopt_ld_tnewton_precond
- nlopt_ln_neldermead
- nlopt_ld_auglag_eq
- nlopt_gn_isres
- nlopt_auglag
- nlopt_auglag_eq
- nlopt_g_mlsl
- nlopt_g_mlsl_lds
- nlopt_gn_esch
- nlopt_num_algorithms

Nous disons qu'ils se terminent rapidement, mais ce qui serait plus juste serait de dire "rapidement par rapport aux autres". En effet, nous n'avons pas toujours accès au temps de calcul lors du fitting d'un algorithme. Par conséquent, parmi les algorithmes qui se terminent vite pour le modèle ARCH, veuillez trouver ci-dessous ceux dont le temps de calcul n'est pas accessible mais aussi les temps de calculs des algorithmes dont l'information est à disposition avec l'erreur quadratique cumulée respective.

Temps de calcul ($computingTime à 0 par défaut) non disponible :

Nom de l'algo	Erreur Quadratique cumulée des paramètres estimés
nlopt_gn_orig_direct_l	0.2
nlopt_gd_stogo_rand	0.2
nlopt_gd_stogo	0.2
nlopt_gn_esch	0.2
nlopt_num_algorithms	0.2
nlopt_gn_isres	0.2
nlopt_auglag	0.2
nlopt_auglag_eq	0.2
nlopt_g_mlsl	0.2

L'erreur quadratique est très grande et est identique pour tous ces algorithmes, ainsi, nous nous demandons s'ils ont été implémentés réellement.
Les algorithmes dont le temps de calcul est disponible sont les suivants. Voici un tableau résumant le temps de calcul moyen en fonction de l'algorithme et l'erreur quadratique associée.

Nom de l'algo	Temps en ms	EQ cumulée des params estimés
nlopt_ld_lbfgs_nocedal	50	0.0032
nlopt_ld_lbfgs	62	0.2
nlopt_ln_praxis	47	0.0032
nlopt_ld_var1	47	0.0032
nlopt_ld_var2	39	0.0032
nlopt_ld_tnewton	60	0.0032
nlopt_ld_tnewton_restart	35	0.0032
nlopt_ld_tnewton_precond	59	0.0032
nlopt_ln_neldermead	48	0.2
nlopt_ld_auglag_eq	10	0.2
nlopt_g_mlsl_lds	78	0.0032

Les algorithmes qui convergent pour le modèle ARCH avec un temps de calcul raisonnable sont au nombre de 8.

Les deux meilleurs algorithmes retenus (avec pour critère le rapport précision / temps) pour le modèle ARCH sont les suivants :

- **nlopt_ld_var2**
- **nlopt_ld_tnewton_restart**

Il conviendrait de réaliser la même démarche pour l'ensemble des modèles afin de savoir quels sont ceux qui convergent le plus rapidement selon le modèle utilisé.

Nous avons préférer nous concentrer sur les algorithmes de GSL afin de pouvoir avoir une première partie très complète sur les algorithmes de GSL, raison pour laquelle les algorithmes de NLOpt n'ont été testés que sur un modèle ARCH.
Neanmoins, il convient de souligner que la démarche reste exactement le même.

4 Relevé d'erreurs sur les modèles de type Arch

Nous avons aussi réalisé un relevé d'erreurs sur les différents modèles possibles. Tous ces tests présentant des erreurs sont dans le dossier `testWithError`.

Tarch

À chaque fit, on obtient l'erreur suivante, provenant sûrement du C++ :

```
Res1 <- RegArchFit(model=mod, Yt=ZZ1\$Yt,initPoint = modInitPoint)
# !!!! ERROR : CreateOneRealCondVar: unknown conditional var type !!!!
```

Aparch

À chaque fit, il existe une division par 0.

```
Res1 <- RegArchFit(model=mod, Yt=ZZ1\$Yt,initPoint = modInitPoint)
# !!!! ERROR : division by zero !!!!
```

FiGarch

À chaque simulation, RStudio plante.

```
ZZ1 <- RegArchSim(nSimul = 1000, model=mod)
# !!!! ERROR : Session R aborted every time we try a simulation !!!
```

UGarch

À chaque simulation, il existe un mauvais indexage.

```
ZZ1 <- RegArchSim(nSimul = 1000, model=mod)
# !!!! ERROR : bad index !!!!
```

5 Automatisation des tests

Principe

Le but de l'automatisation des tests était de lancer un script de fit pour les différents modèles (qui fonctionnent) pour tous les algorithmes disponibles afin d'accélérer notre processus d'analyse de résultats des modèles. Il s'agit du script `automationTestAlgoAndModels.R` dans le dossier `automation`. Le but en sortie était d'obtenir pour un nombre de simulation fixé une moyenne des erreurs quadratiques sur les paramètres, du temps d'exécution et du ratio de convergence des algorithmes.

Entrées

- `mod`i pour i allant de 1 à 3 : il s'agit des modèles Arch, Garch et EGarch. Ces modèles seront stockés dans la liste : `ListModels`.
- `mod`i`InitPoint` pour i allant de 1 à 3 : il s'agit des points de départ pour la partie RegArchFit. Ces modèles seront stockés dans la liste : `ListModelsInitPoint`.
- `Param` : matrice qui stocke les paramètres des modèles afin de pouvoir calculer l'erreur quadratique pour chaque modèle.
- `ListAlgosGSL` : Liste des 8 algorithmes GSL utilisés.

Sorties

- `m1`, `m2`, `m3` : matrice des erreurs quadratiques sur les paramètres pour les 3 modèles de tests. Ces 3 matrices seront stockées dans une liste de matrice : `ListMatrixError`.

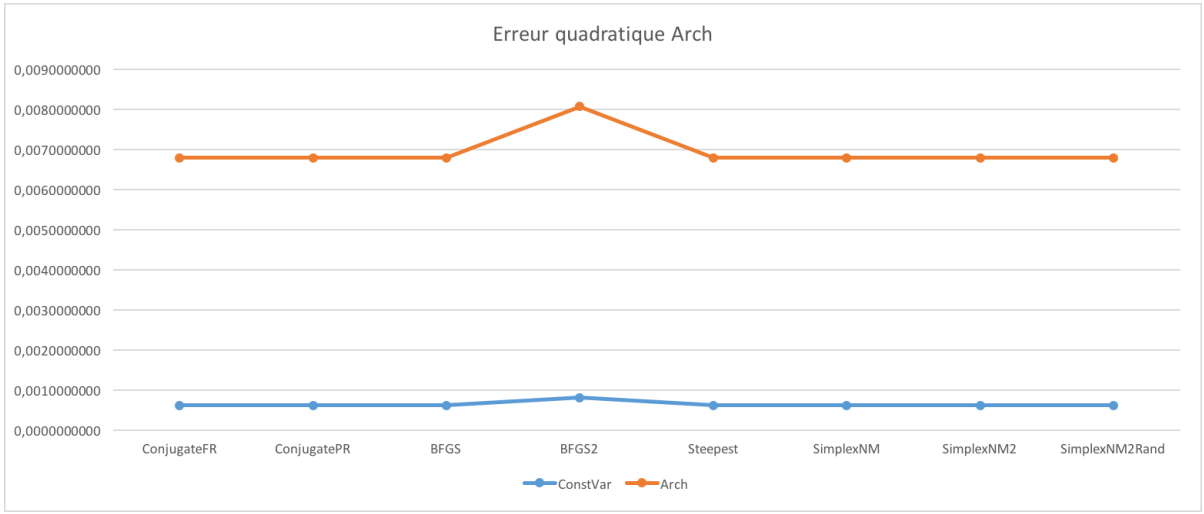

Figure 11 : Erreur quadratique sur les paramètres du modèle Arch

L'algorithme qui présente la plus grande erreur quadratique sur les paramètres du modèle Arch est l'algorithme **BFGS2**

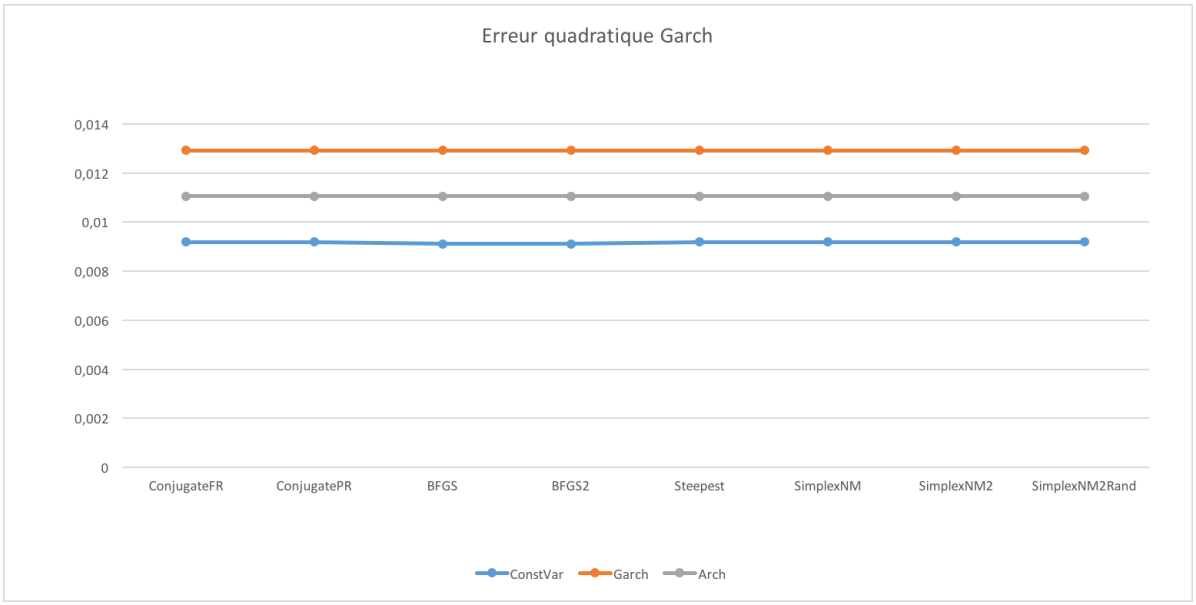

Figure 12 : Erreur quadratique sur les paramètres du modèle Garch

Nous remarquons donc que tous les algorithmes se valent pour le modèle Garch.

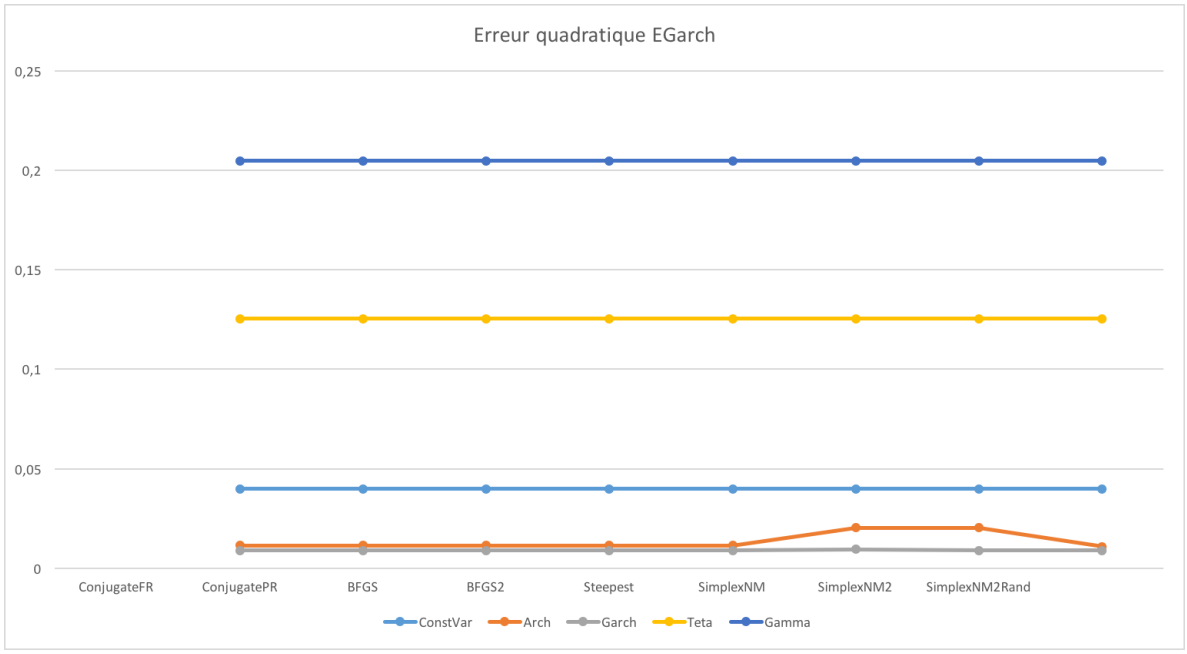

Figure 13 : Erreur quadratique sur les paramètres du modèle EGarch

Nous remarquons donc que sur le paramètre Arch du modèle EGarch, les algorithmes **SimplexNM2** et **SimplexNM2Rand** présentent une erreur quadratique plus grande.

- `computingTime` : Matrice de résultats des temps d'exécution des algorithmes pour chaque modèle.

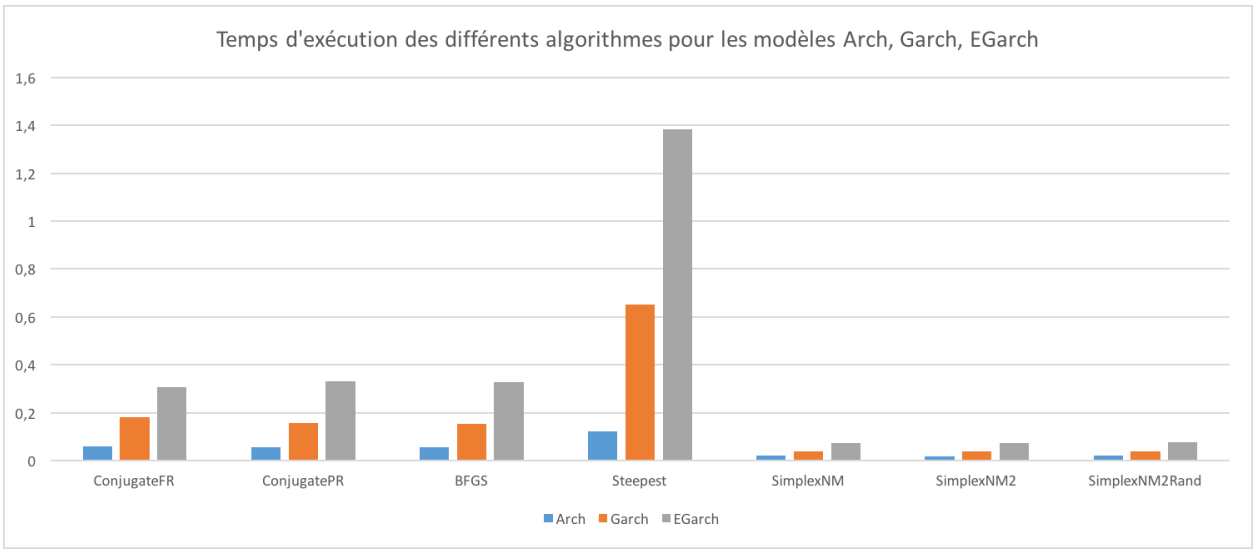

Figure 14 : Temps d'exécution des algorithmes pour chaque modèle

Nous remarquons donc que l'algorithme le plus long est le **Steepest**. Notons que nous n'avons pas inclus les résultats de l'algorithme **BFGS2** car il présentait un temps d'exécution 10 fois plus grand pour le modèle Arch et faussait donc un peu nos résultats. Nous pensons qu'il s'agit

du fait que l'algorithme converge assez mal, comme nous le verrons dans le point suivant. Voici les résultats obtenus pour l'algorithme **BFGS2** :

Modèle	Temps d'exécution (s)
Arch	10.266
Garch	0.525
EGarch	0.259

- `ratioConv` : Matrice de résultats des ratio de convergence des algorithmes pour chaque modèle.

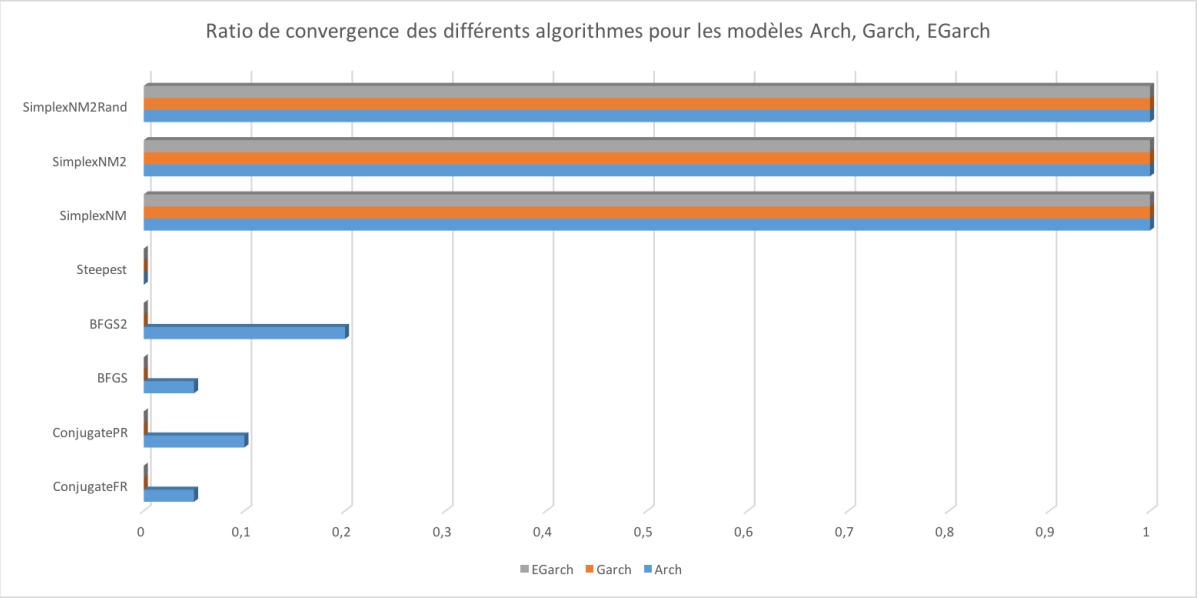

Figure 15 : Ratio de convergence des algorithmes pour chaque modèle

Nous remarquons donc que nous avons bien les algorithmes de Simplex qui convergent, alors que la convergence des autres algorithmes est assez médiocre.