Le génie pour l'industrie

RAPPORT TECHNIQUE
PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
DANS LE CADRE DU COURS GTI792 PROJET DE FIN D'ÉTUDES EN GÉNIE DES TI

PROJET D'ENRICHISSEMENT DES SIGNAUX POUR AUTOMATES DE TRADING

NICOLAS HUBERT
HUBN30099004

DÉPARTEMENT DE GÉNIE LOGICIEL ET DES TI

Professeur-superviseur

ALAIN APRIL

MONTRÉAL, 14 DÉCEMBRE 2016
AUTOMNE 2016

REMERCIEMENTS

Ce projet a été rendu possible par la supervision du professeur Alain April, du département de Génie Logiciel à l'École et des TI's de Technologie Supérieure, qui a assuré un suivi constant du projet et a répondu rapidement à toutes les questions qui ont pu être posées dans le cadre du projet.

Je remercie aussi Thomas Maketa, étudiant au doctorat en Génie Logiciel à l'École de Technologie Supérieure, qui a effectué un transfert de ses connaissances du domaine financier et des automates de transactions, afin de concevoir ce premier prototype sur des bases solides.

PROJET D'ENRICHISSEMENT DES SIGNAUX POUR AUTOMATES DE TRADING

NICOLAS HUBERT
HUBN30099004

RÉSUMÉ

Ce rapport technique présente des travaux qui visent à résoudre un problème d'analyse et de gestion des transactions d'actions à la bourse. La complexité des marchés et la rapidité requise pour effectuer des transactions incitent les experts en finance à utiliser des outils automatisés pour les assister. L'objectif du projet consistait à créer une plateforme de «Backtesting» en utilisant des données boursières historiques afin de créer et tester des stratégies de transactions fondées sur des situations réelles. Le rapport s'adresse aux experts en génie logiciel ayant un intérêt pour les automates de transactions boursières, et désirant utiliser et modifier ces technologies.

Actuellement, certaines plateformes de «Backtesting» qui utilisent des données historiques existent mais elles n'incluent pas toutes les types d'instruments financiers afin de simuler des stratégies de placements. Ce projet vise la modification d'une plateforme libre existante de «Backtesting» afin de permettre la prise en compte des options et de leurs expirations, ainsi que d'assurer la compatibilité avec Interactive Broker (IB), pour effectuer des essais sur des marchés réels.

L'objectif de ce projet est de créer un prototype logiciel pouvant répondre à ces 2 besoins. Pour ce faire, le logiciel libre PyAlgoTrade, permettant la gestion de transactions basées sur des données historiques (c.-à-d. le «Backtesting»), a été modifié pour prendre en compte de nouveau type de transactions et leur expiration. Le résultat est un prototype logiciel permettant de simuler des stratégies de transactions et d'exprimer leur comportement à l'aide de statistiques complètes.

TABLE DES MATIÈRES

LISTE DES FIGURES

Page

INTRODUCTION

Aujourd'hui, les automates de transactions boursières sont omniprésents dans les marchés boursiers. Pour tirer leur ficelle du jeu, les différents acteurs du domaine, comme les fonds de placement les fonds de pension, ou tout simplement les courtiers en bourse à Wall Street utilisent ces automates pour leur rapidité de calcul et l'efficacité de leurs algorithmes.

Ces algorithmes doivent cependant faire leurs preuves avant d'entrer en fonctions sur les marchés, car un algorithme mal conçu peut faire perdre beaucoup d'argent à celui qui le met en œuvre. Afin de tester ces algorithmes, et de valider leurs performances, des tests doivent être préalablement faits. Ces tests reposent sur des plateformes de simulation basées sur des données historiques. Ces simulations permettent de déterminer, à l'aide du cours du marché passé, si ces stratégies peuvent donner un rendement positif ou si des modifications sont requises pour assurer un meilleur résultat.

L'objectif de ce projet est de concevoir et implémenter un tel système, qui permet de tester des stratégies de transactions boursières en se basant sur des évènements passés. La première étape du projet est d'analyser les solutions existantes, et de sélectionner celle qui est la mieux adaptée pour accomplir la tâche. La deuxième étape est de déterminer les modifications requises pour atteindre les objectifs du projet. Ces modifications devront être conçues et implémentées, puis testées pour s'assurer de leur bon fonctionnement. Les résultats de ces tests seront ensuite revus et analysés afin de valider si l'objectif est bien atteint

DÉTERMINATION DE L'OBJECTIF DU PROJET

CHAPITRE 1

1.1 Contexte et problématique

L'utilisation de la technologie est incontournable dans le domaine financier. Des algorithmes de calculs effectuent déjà des millions de transactions à haute fréquence à la bourse. Les récentes avancées dans le domaine de l'apprentissage machine et la facilité d'utilisation des grappes de calculs permettent aussi d'utiliser la technologie pour améliorer les décisions prises par ces différents algorithmes. Ce projet de fin d'études vise à concevoir une plateforme pouvant être utilisée par des algorithmes d'apprentissage machine pour tester des stratégies de transactions évoluant dans le temps selon l'état des marchés financiers et les résultats des calculs précédents.

1.2 Objectifs

Afin d'atteindre les objectifs de ce projet, une plateforme de test par évènements, basée sur des données historiques, doit être conçue et expérimentée. Des stratégies simples de transactions boursières seront testées afin d'évaluer les possibilités d'amélioration du projet dans le futur. Afin de répondre à tous ces objectifs, et ainsi avoir un premier prototype fonctionnel, la plateforme doit premièrement lire et indexer les données contenues dans un fichier texte (format .csv). Ces données décrivent les évènements de base qui dicteront le fonctionnement de la plateforme. Deuxièmement, la plateforme doit permettre la création de stratégies de transactions basées sur les évènements. Conséquemment, des données statistiques doivent être générées, lors de l'exécution de ces stratégies, afin d'en évaluer l'efficacité et pouvoir évaluer les différents résultats obtenus. Dans l'optique de la conception d'une plateforme complète, mais qui ne peut être réalisée dans le cadre de ce premier projet, la plateforme devrait aussi permettre à un usager d'utiliser une interface graphique pour interagir avec les différentes fonctionnalités. Ultimement, elle devra aussi permettre

l'interaction avec un « live broker », qui permettrait de tester les stratégies en temps réel sur les marchés. La conception et la réalisation de cette première version de la plateforme ne sera pas faite à partir de zéro. Deux logiciels libres existants pourraient être utilisés pour supporter les fonctions voulues. Ces deux options sont analysées en profondeurs afin de déterminer quelle est la meilleure base, en se fiant à des critères tels que : la facilité de modification, la complexité, la stabilité, et la possibilité d'expansion de chaque logiciel. Les deux solutions disponibles se ressemblent, elles sont codées en python et utilisent sensiblement les mêmes librairies externes. Elles utilisent aussi la méthode « par évènement » pour donner le tempo aux opérations devant être réalisées. Suite à l'analyse détaillée, leur implémentation s'est révélée être très différente.

MYALGOSYSTEM

CHAPITRE 2

2.1 Introduction au logiciel MyAlgoSystem

Ce logiciel a été créé par Thomas Maketa, et est basé sur l'implémentation décrite dans le livre [successful algorithmic trading, Michael L. Halls-Moore]. À la base de cette implémentation se trouvent différents objets qui interagissent entre eux pour faire la simulation des transactions basées sur des données historiques. Pour utiliser ce programme, l'API de transaction IbPy (Voir ANNEXE A, SCRUM de la semaine du 21 septembre) doit être installé. Cette librairie permet la communication entre le programme et <<Interactive Broker>>, qui est un courtier en ligne de transactions boursières. L'API ne sera pas utilisé dans le cas des tests historiques, mais la structure des valeurs dont il a besoin en entrée donne une ligne directrice à respecter pour la construction des ordres.

2.1.1 Description de la source

Les classes suivantes sont la base du fonctionnement de MyAlgoTrade. Leur description permet une meilleure compréhension du système.

Events: l'objet « évent » est l'objet de base de la simulation, et est celui qui correspondra à chaque "évènement" [(bars)] du marché. Les différents types d'évènements sont générés par les autres objets de l'implémentation.

Event Queue: Implémentation d'une queue Python. C'est le fil conducteur de la solution. Les évènements sont créés et gérés dans cette Queue. C'est ce qui permet à la l'ordonnancement des évènements et leur traitement par les bonnes classes. Toutes les autres classes se servent de la Queue pour aller chercher les évènements existants et y transmettre les nouveaux évènements.

DataHandler: Une classe abstraite, qui peut être héritée et modifiée pour un fonctionnement avec des données historiques, comme dans le cas présent, ou pour des données en temps réel, pour expérimenter les stratégies sur le marché réel. Cette classe génère les « Market Event » premier type de « évent » contenu dans la queue. Dans notre cas, ils seront générés par chaque ligne de notre fichier de données historiques. Les transactions simulées par la suite se baseront sur ces « events ».

Strategy: Une classe Abstratire, qui peut être héritée pour créer différents types de stratégie. Elle utilise les « Market Event » générés par le « DataHandler », et créés des « SignalEvent » associés. Les <<signals event >> sont des suggestions d'achat ou de vente, et son généré en analysant l'état actuel du marché, en prenant compte les évènements passés et actuels. C'est le cœur des décisions prises par la solution, et c'est ce qui lui donne toute sa valeur.

Portfolio : La classe Portfolio fait la gestion de notre portefeuille, et génère des « OrdersEvents » à partir des « SignalEvent » générés précédemment par la stratégie. Les « OrdersEvents » sont des ordres d'achats et de vente qui seront envoyés au courtier pour être exécutés. La prise en compte des transactions passées et la gestion du risque peuvent aussi être incluses dans cette classe.

ExecutionHandler : Cette classe fait la simulation d'une connexion avec un courtier. Dans le cas de traitement historique, le courtier doit être simulé pour donner des résultats plus près de la réalité. L'« ExecutionHandler » utilise les « SignalEvent » créés par le portfolio pour créer les « FillEvent ». Les « FillEvent » décrivent les transactions du courtier en tant que tel.

2.1.2 Modifications déjà faites

Dans la solution MyAlgoSystem, la gestion des options et de l'échange de devises ont déjà été ajoutés aux fonctions de base décrites dans le livre. La gestion de la connexion avec l'IB (c.-à-d. l'interactive Broker) est aussi déjà implémentée. La solution est cependant axée sur la gestion des transactions en live, et certaines fonctionnalités pour la gestion des données

historiques sont manquantes. (Voir ANNEXE A, Scrum du 4 octobre 2016). La Figure 1 représente l'implémentation actuelle de la solution, ainsi que les classes devant être ajoutées pour atteindre les objectifs du projet.

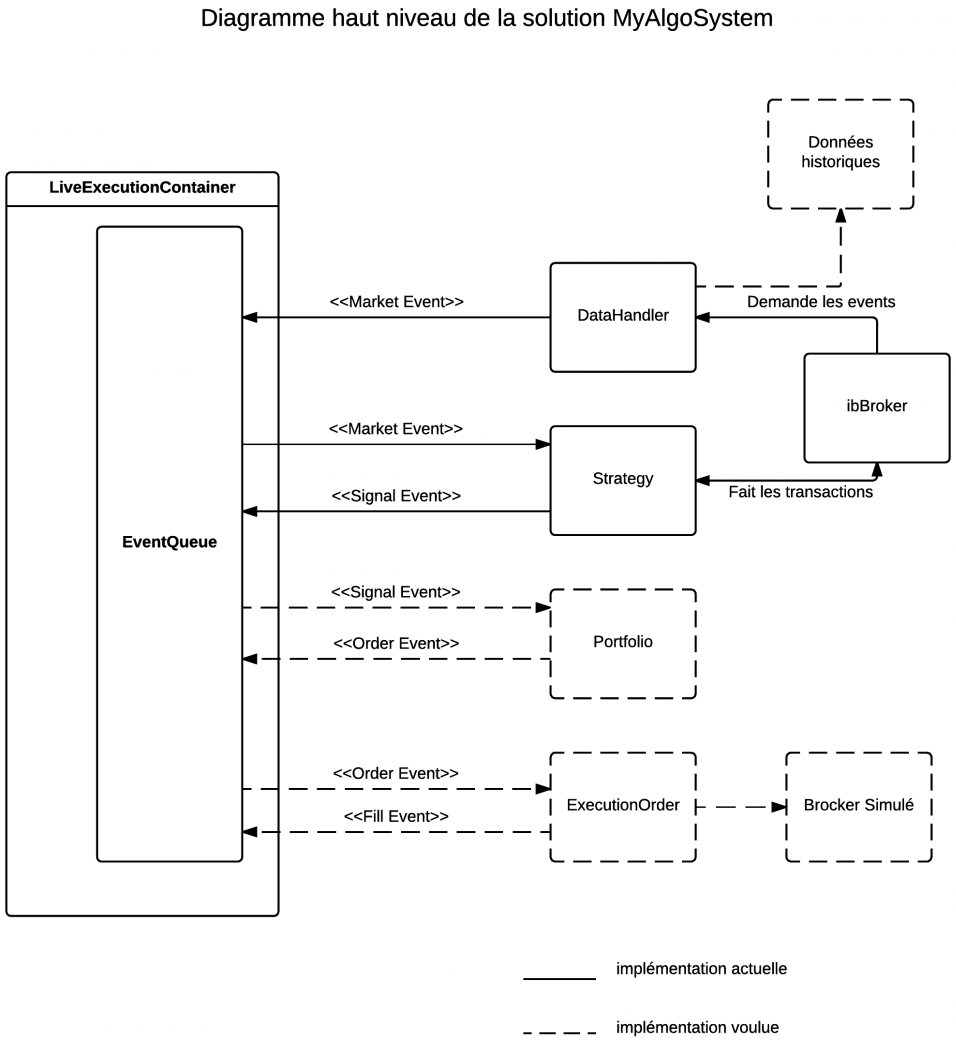


Figure 1 Diagramme au niveau de la solution MyAlgoSystem

2.1.3 Avantages

Le code simple de cette solution permet une compréhension rapide du système et de ses composantes. De plus, divers types de transactions sont déjà supportés par l'application, entre autres les options et les devises, qui sont déjà utilisées avec les fonctions de transactions

en temps réel sur les marchés boursiers. Des statistiques de base sont aussi déjà comptabilisées, ce qui permet déjà de faire des comparaisons entre les résultats de différentes stratégies. La fonction de transaction sur les marchés en temps réel est aussi implémentée, ce qui permettrait d'utiliser les stratégies dans un contexte réel sans modifications supplémentaires.

2.1.4 Inconvénients

Les inconvénients de cette solution sont la complexité relative de l'évolution du système, l'implémentation d'une simulation de « broker » nécessaire, ainsi que la gestion des fichiers de données historiques. Ces modifications doivent aussi être faites en respectant les fonctionnalités actuelles de traitement de données « live », qui est l'objectif principal de l'expérimentation actuelle.

2.2 Introduction au logiciel PyAlgoTrade

La solution PyAlgoTrade est une solution développée depuis 2011 par Gabriel Becedillas. Elle est disponible publiquement sous la licence Apache Version 2.0. Plusieurs parallèles peuvent être faits entre cette solution et MyAlgoSystem puisque le fonctionnement global est sensiblement le même. C'est dans l'implémentation que ces deux solutions divergent. Alors que MyAlgoSystem utilise une stratégie en série, avec l'utilisation de la queue, la solution PyAlgoTrade utilise plutôt une stratégie en parallèle. En utilisant le patron des observateurs, pour chaque nouvel évènement, les observateurs sont notifiés et accomplissent leurs tâches (c.-à-d. des calculs de statistiques, création d'ordres et exécutions d'ordres).

2.2.1 Description de la source

Bar: L'objet « Bar » s'apparente au « MarketEvent » de la solution précédente, et contient les données du marché en un temps donné.

Feed et BarFeed : Ces deux classes font la gestion des données historique. Le « Feed » est bâti à partir du fichier de données historiques fourni, c'est lui qui extrait les données du fichier et les prépare pour utilisation. Le « Barfeed » utilise ces données pour générer les « Bars », qui seront utilisés dans le reste de l'implémentation.

Strategy : La classe « Strategy » est le cœur du fonctionnement de PyAlgoTrade. La fonction « Onbars » contenue dans la stratégie est exécutée à chaque évènement, et c'est elle qui contient toute la logique de la stratégie. C'est aussi lors de l'exécution de cette méthode que tous les observateurs sont notifiés et que les différentes actions sont exécutées.

Broker : La classe « Broker » contient les méthodes de créations et de gestion des ordres d'achats et de ventes. Il est contenu dans la stratégie et sert à exécuter les actions à prendre suite à l'analyse des données courantes et passées.

Stratanalyzer : Classe qui contient les outils de mesures pour faire des statistiques sur la simulation en cours (c.-à-d. Sharpe ratio, drawdown duration, returns, etc.). Ces statistiques sont calculées à chaque « évent », puisqu'ils sont notifiés grâce au patron « observateur ».

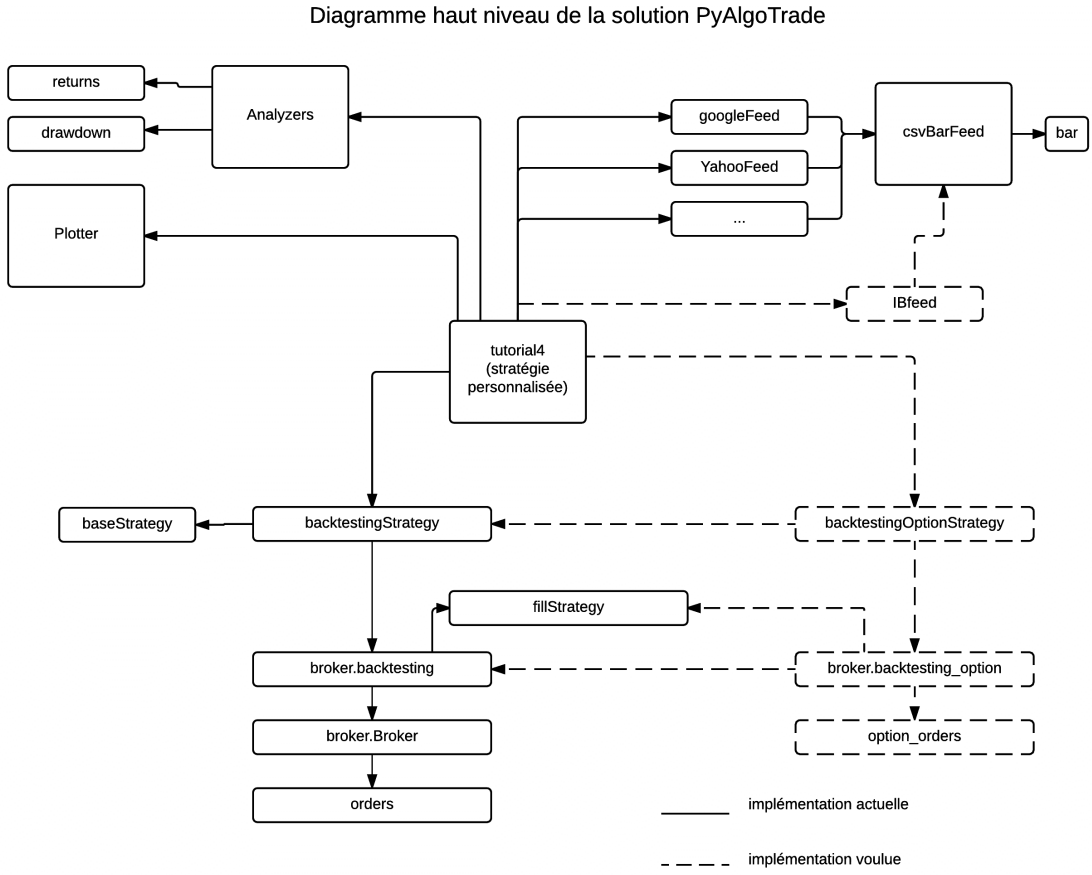


Figure 2 Diagramme haut niveau de la solution PyAlgoTrade

2.2.2 Avantages

La présence de plusieurs modules de comptabilisation de statistiques, un module qui permet les transactions avec l'IB déjà disponible. La simulation du « Broker » est déjà implémentée et complète, avec la gestion du « slippage » et des taux de taxation paramétrable.

L'ajout de nouveaux modules peut se faire sans modifier le code existant. Ceci en utilisant le principe d'héritage déjà grandement utilisé et en utilisant le patron Observateur, qui permet d'être notifié en même temps que les autres fonctions de l'arrivée d'un nouvel évènement.

L'adaptation du « feed » de données est facile à faire, par la présence de plusieurs exemples

déjà fonctionnels. Beaucoup de documentation est disponible (c.-à-d. à haut niveau), avec des exemples de stratégies et leurs résultats.

2.2.3 Inconvénients

Le code de ce logiciel est plutôt complexe. Il faut un moment d'adaptation (c.-à-d. un effort assez grand) pour le comprendre et devenir efficace à le modifier. Les fonctions de gestion des options et des devises étrangères ne sont pas implémentées.

DESCRIPTION DE LA SOLUTION CHOISIE

CHAPITRE 3

3.1 Modifications requises

Pour atteindre les objectifs du projet, plusieurs modifications sont nécessaires. La première modification consiste en l'ajout de la gestion des options dans le logiciel existant. Il a été déterminé, en premier lieu, que la duplication du « flow » de la gestion des actions permettrait l'ajout de la gestion des options et ses paramètres (c.-à-d. date d'expiration, call/put et prix). Des classes ont donc été conçues et ajoutées pour dupliquer les actions des actions (c.-à-d. création des ordres d'achat/ventes (stop, limit, stoplimit) et annulations). Ces modifications ont été effectuées dans la deuxième moitié du mois d'octobre (voir ANNEXE A, semaines du 17, 24, 31 octobre)

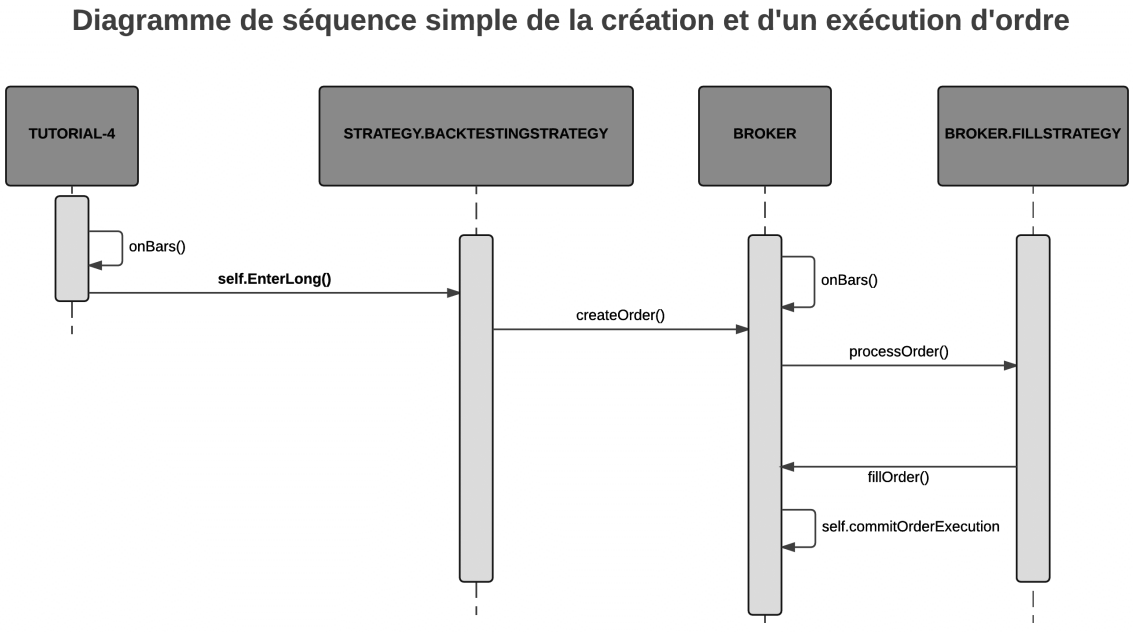


Figure 3 Diagramme de séquence simple de la création et de l'exécution d'ordre

Dans un premier temps, ces nouvelles classes et leurs méthodes ont été ajoutées dans le code actuel de Pyalgotrade. Elles ont par la suite été transférées dans de nouvelles classes, créées en héritant des classes actuelles. Cette décision de conception permet une compatibilité avec le code existant sans en modifier le contenu. Conséquemment, il est possible de garantir au moins les fonctionnalités de base, peu importe les modifications apportées. En utilisant le patron observateur, déjà bien implémenté dans le code présent, l'utilisation des modules de statistiques est aussi assurée pour les nouvelles classes incorporées à PyAlgoTade.

Par la suite, des changements ont été apportés au niveau de la classe « backtesting.py ». Les classes « OptionOrder », « OptionLimitOrder », « OptionStopOrder » et « OptionStopLimitOrder » ont été conçues et ajoutées, en se basant sur des classes existantes telles que : « MarketOrder », « LimitOrder », « StopOrder » et « StopLimit Order ». Ces nouvelles classes permettent, à l'aide de nouveaux paramètres des options (c.-à-d. right, strike et expiry), de faire la gestion des ordres d'achat des options.

Pour utiliser ces classes, de nouvelles méthodes de création des ordres d'achat ont dû être conçues et ajoutées au « backtesting broker ». Ces méthodes servent à créer les ordres d'achat, et de les envoyer au broker simulé.

En plus de ces ajouts recréant la structure des transactions pour les options, l'ajout de la gestion de l'expiration des options a été nécessaire afin de créer un comportement représentatif de la réalité lors des simulations. La rencontre de cette date d'expiration signifie que les options ne sont plus valides. Conséquemment, leur valeur tombe à zéro et les ordres d'achat associés doivent être annulés. En premier lieu, cet ajout a été fait dans le « broker » de la stratégie, dans la méthode « getEquityWithBar ». Cette méthode est appelée à chaque évènement, ce qui la rend idéale pour une validation de temps.

Diagramme haut niveau de la solution PyAlgoTrade

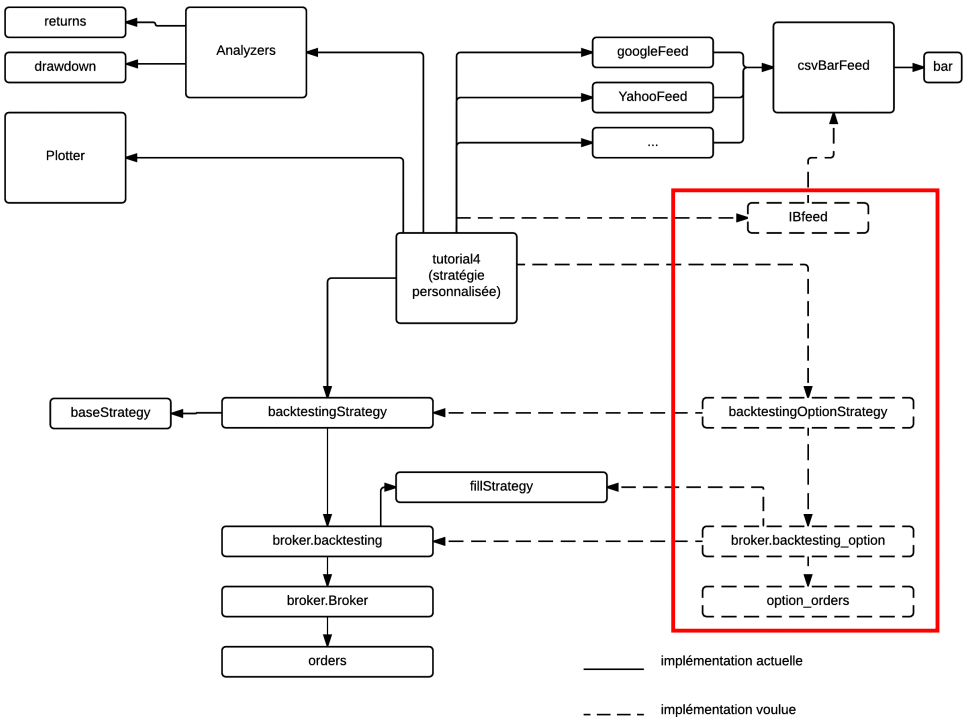


Figure 4 Ajout des nouvelles classes de PyAlgoTrade

CONCEPTION ET IMPLÉMENTATION DE LA SOLUTION CHOISIE

CHAPITRE 4

4.1 Nouveaux objets « OptionOrders »

La conception et l'ajout des nouveaux objets pour la gestion des options seront faits dans le « broker ». Ceci évite de modifier le code existant. Un nouveau « broker » nommé « backtesting_option » est conçu à cet effet. Pour conserver le même fonctionnement que les ordres de base, toute la structure a été copiée puis modifiée.

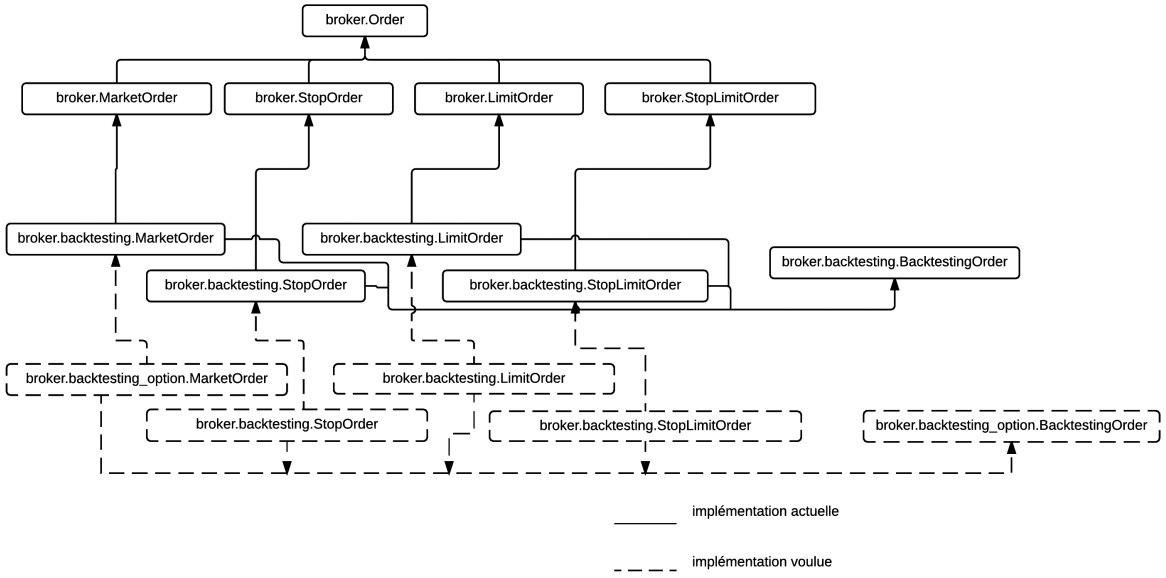


Figure 5 Diagramme de classe des « Orders »

Les objets « OptionOrder », « OptionLimitOrder », « OptionStopOrder» et « OptionStopLimitOrder » ont été conçus et ajoutés, en se basant sur les classes déjà existantes « MarketOrder », « LimitOrder », « StopOrder » et « StopLimit Order ». Vu la ressemblance entre les différents objets, seuls la conception et l'ajout de « OptionOrder » seront décrits en détail.

Dans le « Market Order », les paramètres qui sont initialisés sont : « action » (c.-à-d. achat ou vente), « instrument » (c.-à-d. l'instrument de référence, qui correspond au code de la compagnie), « quantity » (c.-à-d. le nombre d'actions de l'ordre), « onClose » (qui prends la valeur vraie ou fausse, pour savoir si l'ordre doit être exécuté à la fermeture des marchés), et « instrumentTraits ». Dans « OptionOrder », il est nécessaire d'ajouter les paramètres « right » (« put » contrat de vente ou « call » contrat d'achat), « strike » (c.-à-d. le prix futur estimé, et auquel l'option devrait être exécutée), et « expiry » (qui représente la date d'expiration de l'option)

```
class MarketOrder(broker.MarketOrder, BacktestingOrder):
    def __init__(self, action, instrument, quantity, onClose, instrumentTraits):
        super(MarketOrder, self).__init__(action, instrument, quantity, onClose,
instrumentTraits)

    def process(self, broker_, bar_):
        return broker_.getFillStrategy().fillMarketOrder(broker_, self, bar_)


class OptionOrder(broker.OptionOrder, BacktestingOrder):
    def __init__(self, action, instrument, quantity, right, strike, expiry,
onClose, instrumentTraits):
        super(OptionOrder, self).__init__(action, instrument, quantity, right,
strike, expiry, onClose, instrumentTraits)

    def process(self, broker_, bar_):
        return broker_.getFillStrategy().fillOptionOrder(broker_, self, bar_)
```

4.2 Modifications du « backtesting broker »

Pour faire la gestion de ces nouveaux objets, des méthodes ont dû être conçues et implémentées. Dans l'exemple de « OptionOrder », la méthode « createOptionOrder », est le constructeur de l'ordre d'option qui gère les nouveaux paramètres.

```
def createOptionOrder(self, action, instrument, quantity, right, strike, expiry,
onClose=False):
    # In order to properly support market-on-close with intraday feeds I'd need to
know about different
    # exchange/market trading hours and support specifying routing an order to a
specific exchange/market.
    # Even if I had all this in place it would be a problem while paper-trading
with a live feed since
    # I can't tell if the next bar will be the last bar of the market session or
not.
    if onClose is True and self.__barFeed.isIntraday():
        raise Exception("Market-on-close not supported with intraday feeds")
```

```
    return OptionOrder(action, instrument, quantity, right, strike, expiry,
onClose, self.getInstrumentTraits(instrument))
```

Grâce à l'héritage, les autres méthodes de gestion des options, comme la cancellation des ordres (cancelOrder(self, order)), qui sont indépendants des nouveaux paramètres, vont toujours être fonctionnelles, et n'ont donc pas besoin d'être modifiées.

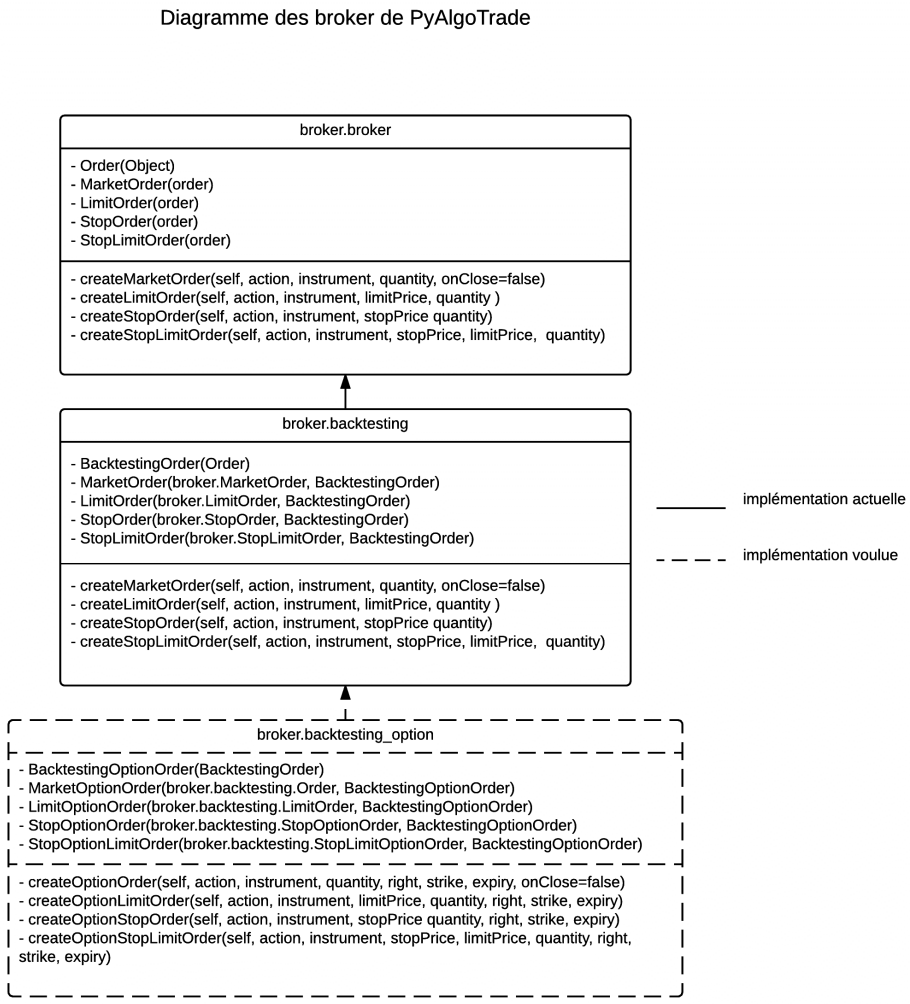


Figure 6 Diagramme des Broker de la Solution PyAlgoTrade

4.3 Gestion de l'expiration

La gestion de l'expiration est primordiale dans le programme, puisque lorsqu'une option rencontre cette date d'expiration, sa valeur devient nulle, ce qui peut avoir un grand impact sur la performance d'une stratégie. L'expiration doit être implémentée dans les nouvelles classes de gestion des options. Les détails du déroulement de l'implémentation sont contenus en annexe (ANNEXE A, Scrum du lundi 31 octobre)

4.3.1 Première itération

Pour faire la gestion de l'expiration des options, la première approche de conception a été d'en faire la gestion dans la méthode « __getEquityWithBar » du « broker ». Cette méthode est appelée à chaque évènement et fait la mise à jour de la valeur des positions en fonctions des prix actuels de l'évènement.

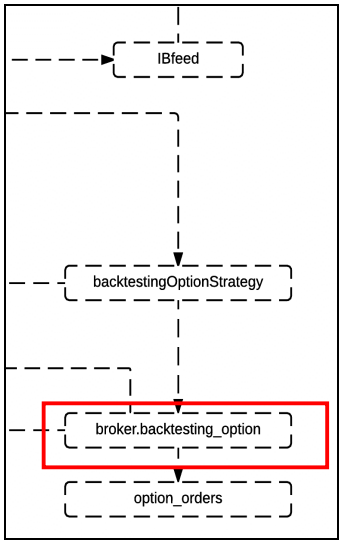


Figure 7 Ajout de l'expiration dans le broker

Le pseudocode suivant décrit la première conception implémentée, avec son contexte initial :

Si l'évènement n'est pas nul
Pour chaque paire (instrument : shares)
Début du code ajouté########
si l'instrument est une option
si la date d'expiration est plus petite que la date actuelle
affichage du message « position expirée »
mettre le nombre de parts à 0
pour chaque ordre contenant l'instrument
dé enregistrer l'ordre
Fin du code ajouté########
Mise à jour de la valeur des « shares »
Retour de la nouvelle valeur

Code ajouté dans le <<Broker>>

```
##### on détecte vaguement si le format est pour une option
 #                if len(instrument) > 8:
 #                    time = self._getBar(bars, instrument).getDateTime()
 #                    year = self._getBar(bars, instrument).getDateTime().year
 #                    month = self._getBar(bars, instrument).getDateTime().month
 #                    day = self._getBar(bars, instrument).getDateTime().day
                     ####### Si la date de la bar est égale ou supérieure à la date
d'expiration, on annule les parts et empêche les ordres sur cet instrument
#                    currentdatetime= datetime.strptime(instrument[-8:], '%Y%m%d')
#                    if datetime.strptime(instrument[-8:], '%Y%m%d') <=
datetime(year, month, day):
#                        self.__logger.debug("POSITION EST EXPIREE")
#                        shares = 0
#                        instument=None
#                        for order in self.getActiveOrders(instrument):
 #                           self._unregisterOrder(order)
```

4.3.2 Deuxième itération

La deuxième itération de conception a été de déplacer cette fonction dans la stratégie, dans une méthode « isExpired » qui est appelée à chaque évènement (c.-à-d. lors de l'appel de la méthode « onBars ()»). La fonction est la même, mais le fait d'être dans la stratégie évite de faire des modifications dans le code source de l'application.

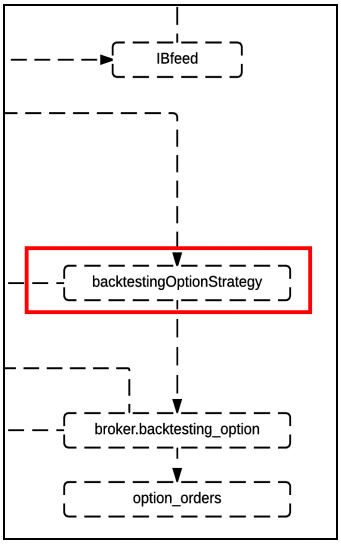


Figure 8 Transfert de la gestion de l'expiration dans la stratégie

Code ajouté à la stratégie dans le cadre de la deuxième implémentation :

```
def isExpired(self, bar, position):
        ret=False
        time = position.getExpiryDate()
        year = position.getExpiryDate().year
        month = position.getExpiryDate().month
        day = position.getExpiryDate().day
        ####### Si la date de la bar est égale ou supérieure a la date
d'expiration, on annule les part et empêche les order sur cet instrument
#         currentdatetime= datetime.strptime(instrument[-8:], '%Y%m%d')
        if datetime.datetime.strptime(self.__instrument2[-8:], '%Y%m%d') <=
datetime(year, month, day):
            self.__logger.debug("POSITION EST EXPIREE")
            ret=True
        return ret
```

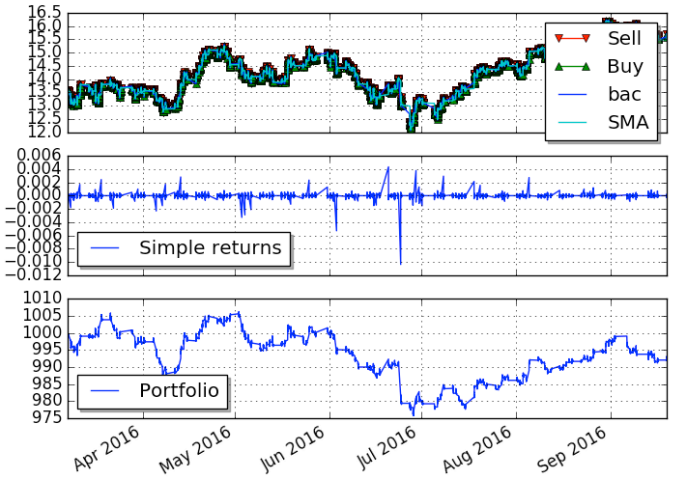
TEST ET ANALYSE DES RÉSULTATS DE LA SOLUTION

CHAPITRE 5

Différents scénarios de tests ont été réalisés afin de s'assurer que le traitement des options est fonctionnel dans l'application. Ces tests ont tous comme base le fichier tutorial4.py de PyAlgoTrade. Ce fichier contient une stratégie simple de transaction basée sur une moyenne mobile, qui permet de faire des tests simples, mais qui donnent une bonne idée du fonctionnement de la stratégie et de ses composants. Pour avoir une représentation visuelle du déroulement de la simulation, les méthodes graphiques contenues dans le tutorial5.py ont été ajoutées dans le fichier.

5.1 Scénario de test des options

Le premier scénario de test réalisé est un test sans gestion d'expiration. Le but premier a été de déterminer si le comportement des nouvelles classes et méthodes dédiées aux options avait le même comportement que les classes originales. L'hypothèse est que le comportement serait le même. Le test consiste en l'utilisation de la stratégie sur une longue période de temps, pour s'assurer que les achats et ventes d'options s'équilibrent. Dans les différents graphiques de la figure 8, on peut voir la variation du prix de l'option dans le temps, avec un indicateur pour chaque action d'achat ou de vente d'une option, le retour d'argent (ou perte), pour chaque transaction et le suivi de la valeur du portefeuille dans le temps.



Valeur du symbole dans le temps

Valeur des retours à chaque instant

Valeur du portfolio

Figure 9 Représentation graphique du test de comportement des options en simulation

Suite aux résultats obtenus, on peut conclure que le comportement de la simulation avec l'utilisation de la nouvelle structure est semblable au comportement original, et donc que les valeurs ont du sens et que ce premier prototype semble fonctionnel.

5.2 Scénario de test de la gestion de l'expiration

Dans le test ci-dessous, une simulation sur trois jours a été exécutée avec la date d'expiration étant la deuxième journée. On peut voir (à la Figure 9) que des transactions ont été effectuées lors de la première journée, mais que par la suite la gestion de l'expiration a empêché la création de nouveaux ordres. Dans ce cas-ci, puisque la date d'expiration était la deuxième journée, les transactions auraient dû s'effectuer dans la journée et cesser à 16h30, mais c'est une erreur de compréhension. Le tout sera corrigé lors de la prochaine mise à jour. On voit aussi la diminution de la valeur du portfolio suite à l'expiration des options.

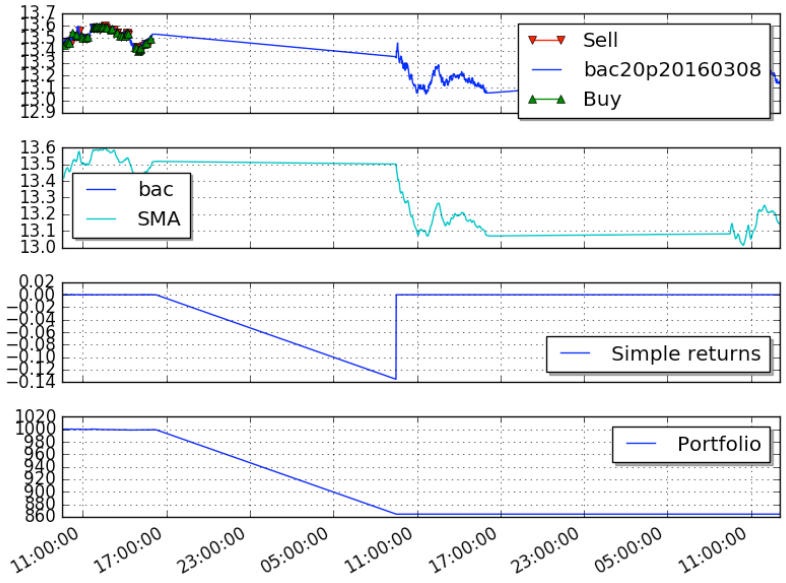


Figure 10 Représentation graphique du test de gestion de l'expiration des options

5.3 Analyse des résultats

Les résultats des différents scénarios de tests sont en ligne avec le comportement attendus lors de la conception de la solution. Les tests sur de longues périodes ont permis de démontrer que les transactions suivent la stratégie de la même manière que les transactions d'ordres initiaux. Les tests sur la gestion de l'expiration permettent aussi de déterminer que les ordres sont bien arrêtés aux moments du, et que la valeur du portefeuille est directement influencée par l'expiration des options.

Certains inconvénients sont présents dans la conception actuelle du programme. Au niveau de la gestion de l'expiration, la première implémentation était déficiente puisque l'inclusion du code de gestion était présente dans une méthode qui faisait le traitement de la valeur du portefeuille, changeant donc la fonction première de cette fonction. Dans la deuxième implémentation, la gestion de l'expiration se fait directement dans la stratégie, ce qui n'est pas l'idéal puisque la gestion devient plus accessible, et pourrait facilement être manipulée, ce qui n'est pas désirable dans le sens ou c'est une fonction essentielle au traitement des options. Pour corriger ces défauts, une séparation des fonctions de la gestion de l'expiration

est nécessaire. Ces modifications ne sont pas implémentées dans le programme actuel, mais sont préférables pour une meilleure gestion de l'expiration, sans dénaturer le code déjà présent.

Premièrement, la gestion de la mise à zéro devrait être déplacée dans le broker, à l'emplacement de la première implémentation de la gestion de l'expiration. La gestion de la valeur du portefeuille est la fonction première de la méthode «__getEquityWithBar », donc il est pertinent d'inclure la gestion de la mise à zéro de la valeur des options dans cette méthode.

Deuxièmement, la gestion des ordres d'achat et de vente après l'expiration devrait à déplacée dans la stratégie et dans la « fillStrategy ». Ces classes font déjà la gestion des ordres, donc une simple condition d'expiration avant la création (dans la stratégie) où l'exécution (dans la « fillStrategy ») pourrait contrôler de manière adéquate la gestion des ordres une fois la date d'expiration rencontrée.

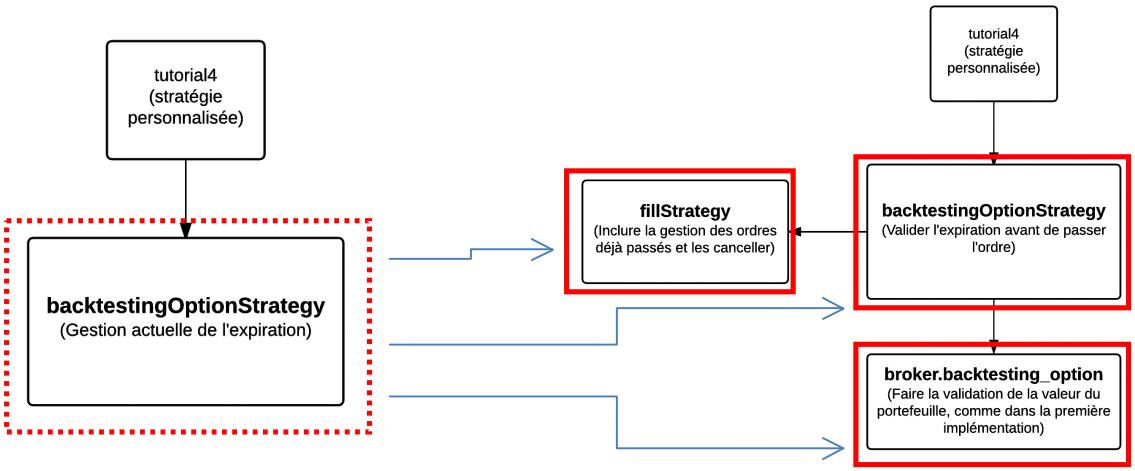


Figure 11 Déplacement des fonctions de la gestion de l'expiration

CONCLUSION

Les stratégies de transactions boursières doivent être validées avant d'être mises en œuvre sur les marchés dans un contexte temps réel. Une stratégie mal adaptée au contexte actuel ou contenant des erreurs peut entrainer des pertes importantes pour celui qui l'utilise. Afin de tester ces stratégies, des plateformes de tests sur des données historiques (c.-à-d. des logiciels de « Backtesting ») sont utilisés. Pour être efficaces, ces logiciels doivent être flexibles, robustes et adaptables pour évoluer avec les nouveaux services financiers.

Ce projet visait à concevoir un premier prototype logiciel qui permet de tester des stratégies sur les options. Ce type d'instrument financier peut maintenant faire l'objet d'analyse de stratégies avant d'être utilisés dans un contexte réel. Ce premier résultat permet la gestion de stratégies diverses d'options, sur des données historiques, et faisant la gestion de plusieurs types de transactions qui permettent une adaptation aux différents contextes du marché.

LISTE DE RÉFÉRENCES

Becedillas, Gabriel. 2016. PyAlgoTrade. Version 0.18. Logiciel. Github. Lieu indéterminé.

Maketa, Thomas. 2016. MyAlgoSystem. Logiciel. Github . Montréal.

Lampe, Bryce. 2016. IbPy. Logiciel. Github. San Francisco.

BIBLIOGRAPHIE

Giridhar, Chetan. 2016. *Learning Python Design Patterns: Leverage the power of Python design patterns to solve real-world problems in software architecture and design.* 2e éd. UK : Packt Publishing Ltd. 311p.

L. Halls-Moore, Michael . 2015. *Successful algorithmic trading*. 1re éd. 199 p.

ANNEXE I
SCRUM RÉDIGÉS DURANT LE PROJET

Scrum du mercredi, le 21 septembre 2016

Rencontre:

- Lieu : Bibliothèque ETS, 13h
- Personnes présentes:
    - Thomas Maketa, ÉTS
    - Nicolas Hubert, ÉTS
- Rédacteur du scrum: Nicolas Hubert

Ce qui a été fait:

- Apprentissage à partir du livre de référence
    - Intro au “Algorithmic trading”
    - Intro au “Backtesting”
        - sources d’erreurs
        - Différence avec le “live”

- Installation de l’environnement et des différents outils
    - Installation du package anaconda, qui contient la plupart des librairies scientifiques qui seront utilisées dans le cadre du projet.
    - Installation de IbPy, qui est un API pour utiliser les fonctionnalités de IB en Python. L’API ne sera pas utilisé dans le cadre de “backtesting” puisque la source des données sera des fichiers textes et que le portfolio sera simulé à même la code python.

- Durant la rencontre, nous avons passé en revue le code de base contenu dans le livre, en expérimentant son fonctionnement et en révisant le output.

Éléments bloquants:

Plan d'action:

- Nicolas:
    - Appliquer le contenu du livre dans la section de l’implémentation du code python
    - Si possible commencer une stratégie simple d’achat, en utilisant le format de données de Thomas.
- La prochaine rencontre est prévue pour le **26 ou 28 septembre prochain**.

Scrum du mercredi 28 septembre 2016

Ce qui a été fait:

- Nicolas:
  - Lecture des chapitres abordant le sujet de l'implémentation d'un "Event-Driven Trading Engine"
  - Implémentation du code contenu dans le livre pour compréhension. (chapitre 15)
  - Regroupement de tous les fichiers nécessaires dans le dossier partagé sur Google

Éléments bloquants:

- Nicolas:
  - Sans être bloquantes, la compréhension des différentes classes et leurs interconnexions ralentissent l'avancement

Plan d'action:

- Nicolas:
  - création d'une première stratégie d'achat et de vente sur la base du code contenu dans le livre pour comprendre la structure et l'utiliser

Scrum du mardi 4 Octobre 2016

Ce qui a été fait:

Suite à la rencontre du samedi 1 octobre:

- Nicolas:
  - introduction au travail d'assurance qualité et de test qui sera fait par Charly
  - Implémentation de la première stratégie de test sur la base du code du livre
    - Stratégie simple d'achat au temps 0, et suivi de la valeur du stock pendant une période de temps déterminé. Affichage des statistiques liées à la stratégie
  - Questionnement en groupe sur la stratégie à utiliser pour l'exécution du backtesting.
    - Choix 1: En reprenant le code de Thomas fonctionnant en "live trading"
    - Choix 2: En partant du code de "pyalgotrade" qui est une plateforme existante de backtesting de stratégie, mais qui ne prend pas en compte l'utilisation de transaction d'option ou de devises.

Éléments bloquants:

- Nicolas:
  - Il faut clarifier s'entendre sur la vision à adopter en groupe, pour être sur que nous ne travaillons pas trop sur des solutions qui ne sont pas pertinentes dans le cadre de la portée du projet.

Plan d'action:

- Nicolas et Charly:
  - Évaluer l'implémentation en partant du code de Thomas:

    Avantage: Le code est déjà capable de faire le "trading" live avec IB broker. Le code prend en compte. Le code gère déjà les transactions d'options et de devises.

    Désavantages: Il faut coder un broker et simuler ses actions. Il faut implémenter toutes les fonctions de backtesting et ajouter la gestion des fichiers historiques (.csv par exemple)
  - Évaluer l'implémentation de Pyalgotrade

    Avantages: Toutes les fonctions de backtesting sont déjà en place, il est possible d'utiliser du code déjà fait pour aller en trade "live" sur IB broker.

    Désavantages: Il faut ajouter la gestion des autres types de transactions (options, devises)

Scrum du samedi 8 Octobre 2016

Ce qui a été fait:

Suite à la rencontre du samedi 1 octobre:

- Nicolas et Charly
    - Analyse des deux solutions proposées au scrum précédent :
    - sélection de la solution Pyaglotrade pour la suite de l'implémentation.

Éléments bloquants:

Plan d'action:

- Nicolas et Charly:
    - Pour le 14 octobre, avoir terminé l'implémentation des différents types de contrats (devises, options) dans la plateforme pyalgotrade.
    - Faire un compte rendu lors de la rencontre du 12 octobre pour avoir un état de l'avancement.

Scrum du lundi 17 Octobre 2016

Ce qui a été fait:

Suite à la rencontre du samedi 1 octobre:

- Nicolas et Charly
    - Création de diagrammes de séquences pour documenter le “flow” de la création des ordres.
    - Création d’une structure parallèle de gestion des options basée sur la gestion actuelle des stock orders de pyalgo trade.
    - Documentation des modifications dans le notebook jupyter pour références futures
    - Début des tests du nouveau “flow” de gestion des options dans pyalgotrade.
        - On réussit à “fill” des orders, mais non à les exécuter sur le marché. Il faudra faire des tests pour voir qu’est-ce qui est problématique.

Éléments bloquants:

Plan d'action:

- Nicolas et Charly:
    - Pour le samedi le 22 octobre, avoir continué les tests sur la gestion des options et avoir un exemple fonctionnel, du moins la base (fill, execute orders).
    - Bien documenter quelles classes / méthodes sont utilisées dans chaque étape du traitement des actions(création / submitted / filled / exited)
    - Pour compléter le “flow” il faudra ajouter la gestion des fichiers de prix des options. Pour se faire, il faudra retrouver les fichiers de par le strike, right et expiry. Dans (Myalgo) la gestion se fait en se basant sur le format du paramètre passé à la fonction getInsturment().
    - Modifier le code pour suivre à tout moment le PnL des options

Scrum du lundi 24 octobre 2016

Ce qui a été fait:

- Nicolas et Charly
    - Les tests des conditions de sorties ont été réalisés, sans succès. Les ordres des options sont traités et filled, mais le cycle de l'exit order ne se complète pas.
    - Les diagrammes de séquences de la création des positions sont en annexes.
    - La documentation de "qui fait quoi" est mise à jour pour ce qui est des "entryorders". Il reste à compléter la documentation pour les exit orders.
    - La gestion des fichiers de prix des options se fera à l'aide de feeds basés sur le nom (strike:right:expiry)

Éléments bloquants:

- Il faut comprendre que se passe-t-il une fois qu'une option est exécutée et complétée
    - le "flow" est différent que lors de la création des entryorders, il faut bien le documenter avec d'entreprendre le traitement des exit orders pour les options.

Plan d'action:

- Nicolas et Charly:
    - Avoir un test fonctionnel pour le 30 octobre, sans nécessairement avoir toutes les fonctionnalités opérationnelles. Pouvoir au moins faire un cycle entryorder/exitorder.
        - Il reste à compléter les "exitorders"
    - Bien documenter quelles classes / méthodes sont utilisées dans chaque étape du traitement des actions pour les exitorders (création / submitted / filled)
    - Il faudra mettre à jour la documentation du notebook jupyter pour que les modifications soient documentées en détail pour la réalisation de l'interface graphique.

Scrum du lundi 31 octobre 2016

Ce qui a été fait:

- Nicolas et Charly
    - La gestion du temps d'expiration est implémentée et semble fonctionner selon les premiers tests.
        - La gestion de l'expiration se fait dans le broker de la stratégie. La méthode "__getEquityWithBars" calcule la valeur du portfolio à chaque évènement. On utilise le fait que la méthode utilise l'objet "bars" pour aller chercher la date de l'évènement actuel pour la comparer avec la date d'expiration qui est contenue dans le "Instrument". Puisque les Instruments des options contiennent la date d'expiration dans le nom du fichier et le nom d'instrument qu'on génère au début.

-En premier lieu on va chercher la valeur de la date de l'évènement actuel.

```
year = self._getBar(bars, instrument).getDateTime().year
month = self._getBar(bars, instrument).getDateTime().month
day = self._getBar(bars, instrument).getDateTime().day
```

-Par la suite, on compare la date avec un "parse" du nom de l'instrument.

```
if datetime.strptime(instrument[-8:], '%Y%m%d') <= datetime(year, month, day):
```

-Si les deux dates correspondent, ont met le nombre de shares à "0" et on "unregister" tous les orders associés à cet instrument, qui est expiré.

```
self.__logger.debug("POSITION EST EXPIREE")
    shares = 0
    for order in self.getActiveOrders(instrument):
        self._unregisterOrder(order)<
```

La valeur est donc prise en compte lors de l'actualisation de la valeur du portfolio.

- Pour faire un suivi visuel des résultats, la création des graphiques du tutorial-5 a été ajoutée à nos tests pour avoir une meilleure idée des résultats. On peut entre autres voir le résultat de l'annulation des shares sur la valeur du portefeuille. Les tests faits en ce moment sont des tests sur différentes périodes de temps pour valider que l'expiration est fonctionnelle.
- Pour le moment nous pouvons déterminer qu'avec une stratégie de base sma, le backtesting est fonctionnel.

Éléments bloquants:

- Il faut définir clairement les prochaines étapes et les tests à effectuer pour ne pas perdre notre temps. La réalisation des premiers tests a été retardée par des problèmes de communications et objectif flou au départ. Par exemple, en ayant plus d'information et le contexte de livraison, l'implémentation de la gestion de l'expiration est le seul élément essentiel pour la poursuite du backtesting. La gestion des "options orders" sera surtout nécessaire lors de l'utilisation de IB trade.
- Il me manque le feedback de Charly pour savoir s'il y aurait des éléments bloquants de son côté, le document sera mis à jour pour les inclure.

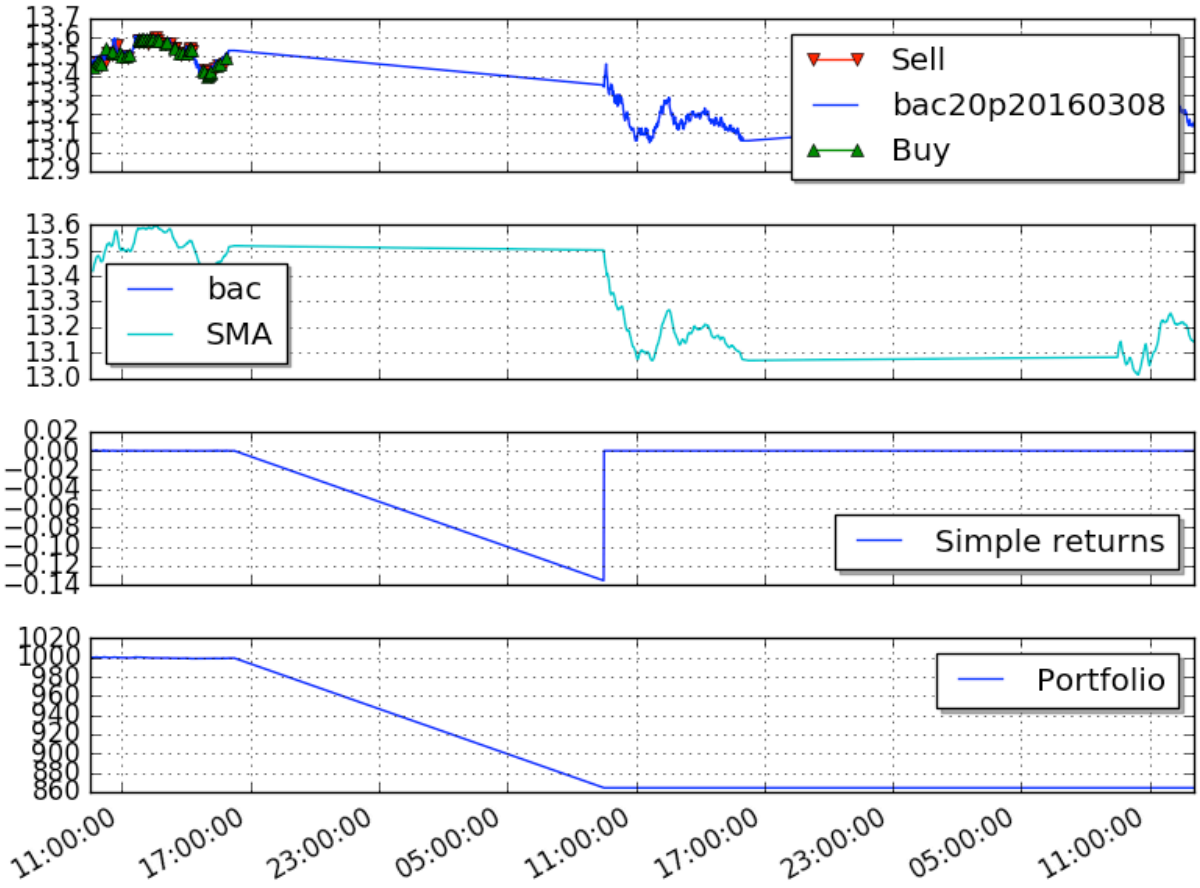
Plan d'action:

- Nicolas et Charly:
  - Il faut documenter les modifications dans le notebook jupyter pour avoir un suivi des changements et avec un exemple des résultats.
  - Il faut faire des tests approfondis, entre autres pour valider la fonction de validation lorsque le fichier de données est terminé. Pour le moment tout semble fonctionnel, mais dans des circonstances qui ne correspondent pas à la réalité. Nous pourrons nous servir du livre de référence que Thomas a fourni pour structurer les tests. Il est aussi possible de commencer.
  - L'implémentation des stratégies peut commencer, elles pourront nous aider pour ressortir les bug et corriger l'implémentation en cas de besoin. Ça pourra faire partie de la série de tests à faire.

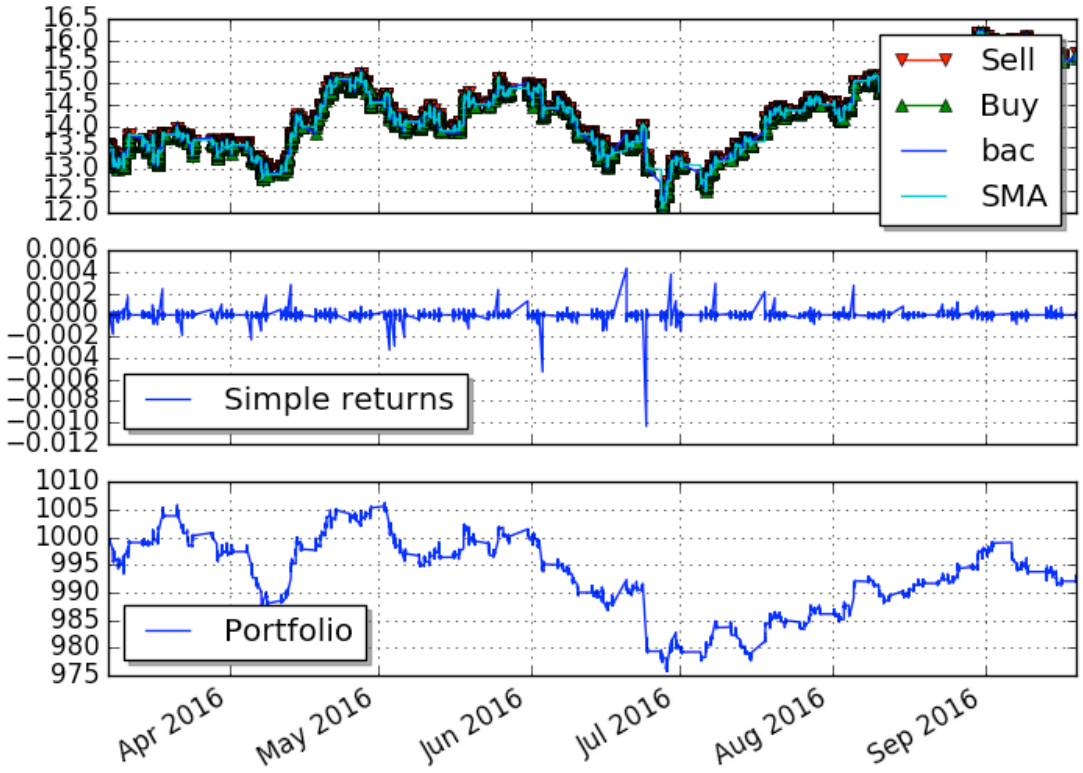
Scrum du lundi 7 novembre 2016

Ce qui a été fait:

- Nicolas
  - Différents scénarios de tests ont été réalisés pour s'assurer que les options sont bien intégrées dans l'application. Des tests sur différentes périodes et avec différentes dates d'expirations ont été réalisés. Dans le test ci-dessous, une simulation sur trois jours a été roulée avec la date d'expiration étant la deuxième journée. On peut voir que des transactions ont été effectuées lors de la première journée, mais que par la suite la gestion de l'expiration a empêché la création de nouveaux ordres. Dans ce cas-ci, puisque la date d'expiration était la deuxième journée, les transactions auraient dû s'effectuer dans la journée et cesser à 16h30, mais c'est une erreur de compréhension. Le tout sera corrigé lors de la prochaine mise à jour. On voit aussi la diminution de la valeur du portfolio suite à l'expiration des options.



- Pour s'assurer que les options ont le même fonctionnement que les stocks, des tests sur de plus longues périodes ont été faits. On peut voir sur le graphique suivant l'évolution de la valeur du portefeuille et des options, ainsi que les transactions effectuées.



- Une stratégie de base impliquant les stocks et les options, mais des erreurs de conditions empêche la simulation de se compléter. Cette stratégie sera corrigée sous peu. Lorsqu'elle sera corrigée, les stratégies impliquant les stocks et les options pourront être créées.

Éléments bloquants:

Plan d'actions:

- Nicolas
  - Corriger la stratégie utilisant les stocks et les options.
  - Créer une nouvelle backtesting strategy en utilisant l'héritage de "backtesting.py".
  - Modifier le code qui permet la gestion de l'expiration. Plutôt que de l'inclure dans le code existant, de nouvelles méthodes seront créées avec le patron observer, pour analyser à chaque évent de manière indépendante.
  - Mettre à jour la gestion de l'expiration en modifiant le format de l'instrument et en ajoutant les méthodes qui permettront de faire le parsing des informations. Le format sera le suivant : s_BAC ([stock]_[code]), o_BAC_12700_c_20160125 ([option])[code]_[strike price]_[put or call]_[expiry])
  - Le but des modifications est de limiter les modifications sur le code original. Optimalement, après les modifications, une version "vierge" de pyalgotrade sera utilisée, puis les code indépendant ajoutées, pour obtenir les mêmes fonctionnalités.
  - Tester une stratégie utilisant les options et les stocks
  - Documenter les modifications et commencer le rapport.

Scrum du lundi 14 novembre 2016

Ce qui a été fait:

- Nicolas
  Suite au “plan d’action” de la semaine précédente, les changements sont plutôt dans la structure du code:
    - La stratégie sera corrigée après les changements dans le code de pyalgotrade.
    - Un nouveau broker et une nouvelle stratégie de backtesting ont été créés en utilisant l’héritage pour éviter d’avoir des modifications dans le code original, tout en ayant les fonctionnalités ajoutées. Le but final est de seulement ajouter ces fichiers à une installation fraiche de pyalgotrade pour avoir les nouvelles fonctionnalités.
    - La gestion de l’expiration a été intégrée à la nouvelle stratégie de backtesting, mais comporte encore quelques bogues. Pour que la nouvelle stratégie fonctionne, le broker a été modifié pour ajouter des fonctions, entre autres pour annuler des actions qui sont encore “actives”, ce qui était impossible auparavant.
    - La rédaction du rapport final a été commencée.

Éléments bloquants:

Plan d'actions:

- Nicolas
    - Régler les problèmes de la gestion de l’expiration des options dans la nouvelle stratégie de backtesting
    - Mettre à jour la gestion de l’expiration en modifiant le format de l’instrument et en ajoutant les méthodes qui permettront de faire le parsing des informations. Le format sera le suivant : s_BAC ([stock]_[code]), o_BAC_12700_c_20160125 ([option])[code]_[strike price]_[put or call]_[expiry])
    - Faire le squelette du rapport final, en définissant les différentes sections et en commençant une ébauche de texte pour chacune, qui sera la base du texte final. Il s’agira du “plan détaillé” du rapport final.
    - Tester une stratégie utilisant les options et les stocks
    - Documenter les modifications.

Scrum du lundi 21 novembre 2016

Ce qui a été fait:

- Nicolas
  Suite au “plan d’action” de la semaine précédente, et vue, la date de la présentation arrivant à grands pas, les efforts ont surtout été mis sur la rédaction du rapport. Environ 20 % du rapport est terminé. (jusqu’à la conception de la solution et la documentation des changements effectués dans le code).

Éléments bloquants:

Plan d'actions:

- Nicolas
  - pour s’assurer d’être prêt à temps, la rédaction du rapport et la préparation de la présentation sera une priorité. Les points suivants seront adressés si le temps le permet.

  - Régler les problèmes de la gestion de l’expiration des options dans la nouvelle stratégie de backtesting
  - Mettre à jour la gestion de l’expiration en modifiant le format de l’instrument et en ajoutant les méthodes qui permettront de faire le parsing des informations. Le format sera le suivant : s_BAC ([stock]_[code]), o_BAC_12700_c_20160125 ([option])[code]_[strike price]_[put or call]_[expiry])
  - Tester une stratégie utilisant les options et les stocks