Université Nice Sophia Antipolis

POLYTECH NICE-SOPHIA

Projet de fin d'études
Simulateur de marché financier et Trading Haute Fréquence

Maxime Bonelli Youness Abdessadek
Corentin Valleroy Jérémie Montiel

Encadrant : Dr Mireille Bossy

Année scolaire 2012/2013

Résumé

Le but de ce rapport est de mettre en évidence les résultats obtenus durant le projet de fin d'études effectué dans le cadre du master IMAFA à l'école Polytech Nice-Sophia. Les travaux, encadrés par le Dr Mireille Bossy, consistaient à implémenter des stratégies de trading haute fréquence sur un simulateur de marché financier, à l'aide du langage de programmation C++.

Ce document se compose de trois parties principales. En premier lieu l'organisation du projet est présentée. Le simulateur de marché et le modèle utilisé sont ensuite exposés. Enfin, les stratégies de trading haute fréquence implémentées sont l'objet de la dernière partie.

Remerciements

Nous remercions vivement le Dr Mireille Bossy pour nous avoir proposé ce sujet qui nous a permis d'élargir nos connaissances en finance de marché. Nous la remercions également pour son encadrement et sa disponibilité tout au long du projet.

Table des matières

1 Organisation du projet

1.1 Présentation

Le cœur de ce projet de fin d'étude était d'implémenter des stratégies de trading haute fréquence (HFT, de l'anglais "high-frequency trading"), et d'analyser des statistiques des gains ou pertes observés sur le marché simulé. En effet, les transactions à haute fréquence sont une pratique relativement récente mais qui s'est très rapidement répandue sur les marchés financiers durant ces dernières années. Elles consistent à exécuter des transactions financières à très grande vitesse à l'aide d'algorithmes informatiques. Les opérations réalisées peuvent ainsi être exécutées en un temps de l'ordre de la microseconde. Le but de notre projet était de mesurer concrètement à quelle point ces stratégies de trading peuvent être rentables, et également à quels risques potentiels l'organisme les utilisant est exposé. Afin de réaliser cette tâche, il était nécessaire, d'une part de simuler l'activité pouvant être observée sur un marché financier et plus précisément sur le carnet d'ordres d'un titre, et d'autre part de définir et implémenter une ou plusieurs stratégies. Une fois ces deux étapes réalisées, des simulations ont pu être effectuées et ont donné lieu au calcul de certaines statistiques sur les résultats empiriques obtenus. Nos travaux peuvent être trouvés sur le dépôt GitHub suivant https ://github.com/Macfly/HFTSimulator.

1.2 Reprise d'un projet existant

Afin d'avoir une base fonctionnelle, et éviter de passer une grosse partie de notre temps à coder entièrement le simulateur de marché, nous avons fait le choix de reprendre le projet "Marketsimulator" réalisé par l'École centrale Paris. "Marketsimulator" est un projet open-source écrit en C++, disponible librement sur internet (http ://marketsimulator.svn.sourceforge.net). Il permet de créer un marché, avec un ou plusieurs carnets d'ordres, sur lesquels plusieurs agents interagissent. Deux types d'ordres peuvent être passés sur un carnet d'ordres : des ordres limites qui remplissent le carnet, et des ordres de marché qui le vident (cf. paragraphe 2.1). Le carnet d'ordres s'occupe, entre autres, de faire correspondre les ordres entre eux, de calculer le bid et le ask, de calculer les volumes pour chaque prix. Dans le projet de base, on retrouve deux types d'agents : un "liquidity taker" (appelé Noise Trader) , qui ne passe que des ordres de marché et un liquidity provider, qui ne passe que des ordres limites. Un graphique des quantités du carnet d'ordres ainsi que quelques statistiques sont présents dans le projet initial.

1.3 Compte-rendu d'activités

Dans le but de présenter de manière précise la façon dont s'est organisé notre projet, le lecteur peut trouver un diagramme de Gantt en annexe A, détaillant les étapes de travail et les échéances rencontrée, ainsi qu'un rapport d'activité en annexe B.

1.4 Technologies utilisées

Dans ce projet, diverses technologies ont été utilisées. Les choix de celles-ci ont été faits en se focalisant autour du langage C++ qui a servi à l'implémentation du simulateur de marché de base et du notre. En effet, le C++ est, de nos jours, le langage de programmation le plus performant en terme de rapidité d'exécution et d'efficacité. Il est aussi le plus utilisé dans les programmes de cette catégorie. Cependant, la gestion de la mémoire est assez complexe et demande beaucoup d'attention.

1.4.1 Boost

Boost est un ensemble de bibliothèques logiciels libres écrit en C++, qui vise à remplacer la bibliothèque standard de C++. Dans le cadre de ce projet, Boost nous a servi pour la partie multithreading et la partie génération de nombres pseudo-aléatoires. En effet, les lois exponentielles, uniformes et normales utilisées proviennent de cette librairie.

1.4.2 GNUPLOT

Au niveau des sorties graphiques, nous avons utilisé le programme sous licence libre GNUPLOT. Nous avons pu tracer des courbes simples comme celle du prix en fonction du temps mais aussi des graphiques

plus complexes grâce aux opérations de tracé conditionnel de GNUPLOT. Nous avons donc pu par exemple séparer distinctement les valeurs d'achat et de vente sur le marché ou bien même différencier les ordres selon l'acteur ayant soumis ceux-ci auparavant.

1.4.3 Git

Pour travailler à plusieurs sur la partie implémentation nous avons utilisé le logiciel de gestion de version décentralisé Git. Contrairement à SVN, Git ne repose pas sur un serveur centralisé. Cela permet de ne pas dépendre d'une seule machine comme point de défaillance. Les développeurs peuvent travailler sans être connectés au gestionnaire de version (un serveur web qui stocke le repository) et ils ont tous l'historique du projet en local (sur leur machine). Avec Git, il est très facile de faire des branches en local pour faire des tests, et si cela fonctionne, merger la branche de test sur la branche master. Pour envoyer une modification aux autres développeurs il faut, contrairement à SVN, faire deux commandes. Un premier, commit, qui envoie les fichiers au repository local. Une deuxième, push, qui envoie les fichiers sur le serveur pour que les autres développeurs puissent avoir les modifications. Utiliser Git sur ce projet nous a permis de nous perfectionner sur son fonctionnement et notamment de créer deux branches (la première monothread, la seconde multi-threads). Cela nous sera utile pour notre insertion professionnelle car aujourd'hui, Git à largement remplacé SVN.

1.4.4 Matlab

Afin d'analyser les statistiques des résultats empiriques obtenus, nous exportons les données générées par le programme C++ dans des fichiers puis nous importons ces derniers avec Matlab qui est un outil puissant et ergonomique pour le traitement de ce type de données. De plus, utiliser Matlab nous a permis de ne pas passer du temps à coder les fonctions utilisées telles que les estimations de densités.

2 Modèle de simulation de marché

Dans cette section nous présentons le fonctionnement général d'un carnet d'ordres et les détails techniques du projet.

2.1 Fonctionnement d'un carnet d'ordres

Pour passer un ordre de bourse sur une valeur, nous avons besoin de connaitre la composition de son carnet d'ordres. Ce dernier représente, à un instant donné, les ordres d'achat (Bid) et de vente (Ask) émis par les différents acteurs du marché qui n'ont pas encore été exécutés. Chaque ligne du carnet donne le nombre et le volume total des ordres émis à un certain prix. Dans notre modélisation nous avons deux types d'ordres : les ordres à cours limités (ou ordres limites) et les ordres de marché. Un ordre limite n'est exécuté que s'il y a une contrepartie plus intéressante ou équivalente. Son prix est fixé par l'investisseur mais sa date d'exécution est quant à elle fixée par le marché. Un ordre de marché est un ordre qui s'exécute au meilleur Ask (plus petit prix de vente) si c'est un ordre d'achat et au meilleur Bid (plus grand prix d'achat) si c'est un ordre de vente. Le carnet d'ordres ne contient donc que des ordres limites. Sur la figure 1 est présenté un aperçu d'un carnet d'ordres pouvant être trouvé sur le site web officiel de NYSE Euronext.

Order Book

Active as of 25 Feb 2013 10:08 CET

View Data | View Chart

Refresh

# ORDERS	# SHARES	BID PRICE	ASK PRICE	# SHARES	# ORDERS
3	838	29.935	29.96	1,164	4
3	366	29.93	29.965	1,990	5
2	345	29.925	29.97	681	4
5	1,033	29.92	29.975	1,598	9
4	819	29.915	29.98	1,474	8
2	1,145	29.91	29.985	2,286	9
1	261	29.905	29.99	1,230	3
3	708	29.90	29.995	2,674	4
2	1,089	29.895	30.00	1,752	4
2	895	29.885	30.005	701	3
27	7,499	Total		15,550	53

FIGURE 1 – Aperçu d'un carnet d'ordre

2.2 Acteurs du marché

Notre marché est composé de 3 types d'agents :
- le Liquidity Provider,
- le Noise Trader,
- le trader HFT.

Le Liquidity Provider (qui joue le rôle de "Market Maker") est l'acteur qui remplit le carnet d'ordre. En effet, il envoie seulement des ordres limites qui ne seront exécutés que s'il existe une contrepartie meilleure ou équivalente. Ces ordres peuvent également être annulés si l'agent envoie un ordre d'annulation.

Le Noise Trader est un agent de type "Market Taker" qui vide le carnet. Il envoie des ordres de marché de type achat ou vente. Cet acteur doit forcément acheter au prix du best ask et vendre au prix du best bid. En contrepartie il est prioritaire sur tous les autres types d'ordres ce qui assure l'achat ou la vente des titres. Il est, par conséquent, l'acteur qui fait bouger le cours du titre modélisé.

Le trader HFT effectue les mêmes types d'ordres que le Liquidity Provider. Cependant, sa stratégie d'achat ou de vente est différente. Effectivement, son but est de remarquer plus rapidement que les autres l'apparition d'un spread et d'y placer des ordres d'achat ou de vente selon la stratégie voulue (cf. paragraphe 3.1).

2.3 Modèle mathématique

Le modèle mathématique régissant le carnet d'order doit inclure la modélisation des ordres de marché (taker), la modélisation de ordres à court limite (maker) ainsi que la dynamique du prix de l'actif. On fixe un nombre M de slots de prix de part et d'autre d'un spread minimal de taille δ, les slots sont espacés de δ. On fais le choix de prendre $M = 20$ et $\delta = 1$ centime, mais ces entrées sont paramétrables dans le code.

2.3.1 Modélisation des ordres à cours limite (maker)

En toute généralité, les ordres à court limite sont caractérisés par leur sens bid ou ask (vente ou achat), leur volume, leur prix limite, leur date d'émission et leur date d'expiration. Le carnet d'ordres à une date t classe les ordres limites par prix croissants, les demandes d'achat (bid) moins chères à gauche, les demandes de ventes (ask) à droite. Dans notre modèle, les ordres limites du Liquidy Provider arrivent suivant un processus de Poisson avec un paramètre de fréquence choisi par l'utilisateur. Lorsque ce processus effectue un saut, une variable aléatoire uniforme sur $[0; 1]$ est tirée pour simuler une loi de Bernouilli à quatre réalisations (comme le résultat du lancé d'un dé à quatre faces, éventuellement pipé). Selon le résultat de ce tirage, on détermine le type et le sens de l'ordre limite :

1. achnat limite
2. vente limite
3. annulation d'achat
4. annulation de vente.

L'utilisateur peut paramétrer la probabilité d'occurrence de chacun des quatre types, correspondant à un découpage de l'intervalle $[0, 1]$.

Dans le cas où l'ordre est de type achat ou vente, il faut déterminer le prix limite auquel il va être passé. On tire alors une variable aléatoire exponentielle afin de déterminer l'écart (au minimum δ) de l'ordre limite par rapport au best bid ou best ask. Ce choix modélise la décroissance du nombre d'ordres en queues de carnet. Une fois le prix de la transaction tiré, on détermine le volume. Ce dernier va dépendre du prix simulé. On note $V_i^a, i = 1, 2 \ldots$, $V_i^b, i = 1, 2, \ldots$, les volumes associés au prix limite sur le slot i. On numérote à partir du best ask et best bid, c'est à dire que V_1^a est le volume pour un ordre limite pour le meilleur prix de vente et V_1^b est le volume pour un ordre limite pour le meilleur prix d'achat. Déterminer le volume va donc dépendre de i et signifie tirer la variable $V_i^{a,b}$ suivant :
$GrVol :=$ taille d'un grand cumul d'ordres (on utilise 100)

- pour $i = 1, \ldots, 25\%$ de M, on tire $V_i^{a,b}$ selon une loi $\mathcal{N}(70\%GrVol, \sqrt{20\%GrVol})$,
- pour $i > 25\%$ de M, $V_i^{a,b}$ est tiré selon une loi uniforme $U\left[0; 70 \times \frac{6}{i^{\frac{3}{2}}}\right]$ pour obtenir une décroissance nette des volumes en queue de carnet.

Dans le cas où l'ordre est de type annulation d'achat ou de vente, le Liquidity Provider annule les ordres passés auparavant sur les slots $i > M$ et annule également avec une (très) faible probabilité certains des ordres passés sur un slot $i \leq M$.

2.3.2 Modélisation des ordres de marché (taker)

Les ordres de marché du Noise Trader arrivent à des dates aléatoires modélisées par un autre processus de Poisson. Lorsque celui-ci saute à l'instant aléatoire τ_n, on tire U, une variable aléatoire uniforme sur

[0; 1]. Selon sa valeur, l'ordre sera de type vente ou achat (par exemple pour un marché sans tendance, si la réalisation de U est inférieure ou égale à $\alpha = 0.5$, l'ordre sera de type achat, et vente sinon). A nouveau le volume doit être déterminé. On tire ainsi ce dernier suivant la loi uniforme $U\{0, 1, \ldots, GrVol\}$. On peut faire un marché avec tendance en modifiant la valeur de α.

2.3.3 La transaction au prorata

Supposons qu'à l'instant aléatoire τ_n, un ordre de marché arrive de taille ζ_n qui a pour contreparties des V_i^a.

On pose

$$\begin{aligned} m_1 &= \zeta_n \wedge V_1^a \\ m_2 &= \zeta_n \wedge (V_1^a + V_2^a) - m_1 \\ &\ldots \\ m_M &= \zeta_n \wedge \left(\sum_{i=1}^{M} V_i^a \right) - m_{M-1} \end{aligned}$$

et le prix d'exécution au prorata de l'ordre de marché est

$$P_{\tau_n} = \frac{1}{\zeta_n} \sum_{i=1}^{M} P_i^a m_i.$$

Ainsi le prix d'exécution pour l'acteur ayant passé l'ordre de marché est une moyenne pondérée des prix des slots dans lesquels il va chercher ses contreparties.

2.4 Conception

2.4.1 Architecture de base

Le projet Marketsimulator d'ECP, bien que très performant et bien conçu, ne correspondait pas à notre modèle du marché. Marketsimulator est mono-threadé, c'est-à-dire que les actions sont effectuées par le même processus. Cela peut se traduire par le fait que les ordres sont exécutés instantanément et que les ordres sont émis et exécutés par la même personne. Cela ne correspond pas à la réalité. En effet sur les marchés réels, lorsqu'un trader émet un ordre il ne sait pas à quel moment son ordre va être exécuté. Plusieurs secondes peuvent s'écouler, le prix de l'action a le temps d'évoluer. Dans le monde réel le carnet d'ordres d'une action est accédé de manière concurrente. Pour simuler cette variabilité additionnelle, on peut utiliser des threads.

2.4.2 Architecture de programmation

Dans notre modèle, les acteurs émettent des ordres de façon concurrente, lorsqu'un ordre est exécuté, l'agent reçoit une notification. Cela peut sembler facile à implémenter mais lorsque l'on travaille avec des threads il faut prendre en compte le phénomène introduit par la concurrence.

Dans un programme non concurrent, ou séquentiel, l'ordre d'exécution des instructions du programme reste le même d'une exécution à l'autre pour les mêmes paramètres en entrée. Dans un programme concurrent, l'exécution n'est jamais la même. Comme la politique d'ordonnancement, le composant du noyau du système d'exploitation choisissant l'ordre d'exécution des processus sur les processeurs, est inconnu ou incontrôlée, on ne peut pas savoir quel thread va s'exécuter sur le processeur à un certain moment. Le nombre de chemin d'exécution d'un programme multi-threadé est beaucoup plus grand. Les interactions aléatoires entre les threads peuvent modifier le résultat du programme.

À cause de l'indéterminisme de l'exécution, l'accès à des données partagées par les entités concurrentes, ici les acteurs et le carnet d'ordre, peut conduire à des incohérences au niveau des relations liant

ces données. Pour simplifier l'écriture de programme concurrent, il existe de nombreuses libraires qui proposent des containeurs (vecteur, liste, dictionnaire) concurrent-safe, qui peuvent être accédé par plusieurs processus en même temps. Cependant, la méthode qui permet de supprimer un élément n'est pas concurrent-safe. Pour éviter les erreurs du programme lorsqu'un thread supprime un élément et un autre veut accéder à cet élément en même temps il faut utiliser des mutex.

Un mutex est une primitive de synchronisation utilisé pour éviter que des ressources partagées d'un système ne soient utilisées en même temps. Les mutex sont très simples à utiliser, lorsqu'un thread veut accéder à une variable critique, il bloque le mutex, puis lorsqu'il a fini d'utiliser la variable il le débloque. Si un deuxième thread veut accéder à la variable en même temps il ne peut pas, il doit attendre que le mutex soit libéré. Cependant les mutex ont des effets secondaires : les interblocages et les famines. Les interblocages arrivent quand un thread attend qu'un autre relâche un verrou acquis pour pouvoir progresser. Les famines arrivent lorsque qu'un thread essaye d'acquérir une ressource, mais n'est jamais ordonnancé au moment où elle est disponible.

2.4.3 Transformations apportées

La transformation de l'application mono-thread en application multithread s'est fait en trois étapes. La première a été de minimiser le nombre de ressources partagées pour éviter les problèmes mentionnés ci-dessus. Puis, optimiser les méthodes pour gagner en temps d'exécution. Et pour finir, changer l'architecture générale pour y incorporer les threads.

Dans le programme de base, la structure de données qui s'occupait de stocker les ordres était directement accessible par toutes les classes. De plus, le programme parcourait tout le carnet pour calculer les quantités pour chaque prix. Il parcourait aussi tout le carnet pour supprimer un ordre.

Après avoir implémenté les modifications, nous avons augmenté les performances d'environ 50%. Nous avons ajouté les threads. Pour que cela soit possible, nous avons dû modifier la façon dont les ordres sont envoyés au carnet. Pour éviter d'avoir des problèmes de synchronisation, nous avons utilisé une queue pour mettre les ordres qui doivent être exécuté par le carnet d'ordres. Les acteurs push les ordres dans la queue et le carnet d'ordres les pop pour les exécuter. La queue utilisée est celle fournie par Microsoft, elle est concurrent-safe et peut avoir plusieurs producteurs et un seul consommateur, ce qui correspond à notre modèle.

Une fois que les ordres sont mis dans la queue, le thread qui s'occupe du carnet d'ordres les pop pour les exécuter. Le carnet d'ordres est modélisé dans la classe OrderBook.cpp. Les ordres (modélisés dans Order.cpp) sont stockés dans des map dont les clés sont les prix auxquels les ordres ont été émis et les éléments sont les listes des ordres émis aux prix donnés par les clés :

```
std::map< int , std::list < Order > > m_bids ;
std::map< int , std::list < Order > > m_asks ;
```

Lorsque le LiquidityProvider ou le MarketMaker émettent un ordre, celui-ci est ajouté à la variable m_{bids} si c'est un ordre d'achat et m_{asks} si c'est un ordre de vente. Il est alors ajouté à la liste d'ordres au prix correspondant. Le fait d'utiliser l'objet *std : :list < Object >* fait que notre modèle est FIFO (« First In First Out ») puisque les ordres arrivant en premier sont ceux qui sont exécutés les premiers étant donné qu'il sont placés en haut de la pile. Cependant pour le MarketMaker, nous définissons un attribut $m_{activateHFTPriority}$ qui, lorsqu'il vaut true, donne la priorité aux ordres émis par le MarletMaker en les plaçant à chaque fois en hauts des listes des ordres présentent dans m_{bids} et m_{asks}. Les serveurs où sont exécutés les automates HFT sont très proche des serveurs des bourses ce qui leur permet d'avoir une latence très faible. Grâce à cela, ils peuvent déceler l'apparition d'un spread extrêmement rapidement et envoyer les ordres voulus tout aussi rapidement.

Lorsque c'est le NoiseTrader qui émet un ordre, suivant son volume et suivant si c'est un ordre d'achat ou un ordre de vente, celui-ci va être matché avec un ou plusieurs ordres limites au meilleur Bid ou au meilleur Ask, voire même avec d'autres ordres plus éloignés du spread si son volume est suffisamment important.

Lorsque le carnet fait correspondre deux ordres, il envoie une notification d'exécution aux deux agents. Pour cette méthode nous avons eu besoin d'utiliser un mutex car les deux threads accèdent à la même variable. Cette variable stocke les ordres qui ont été envoyé mais pas exécuté. Les agents ont besoin de cette variable pour avoir l'ID des ordres qu'ils peuvent annuler. Le carnet d'ordres a besoin d'accéder à cette variable pour enlever les ordres qui ont été exécuté. Il est très difficile d'avoir un container concurrent-safe lorsqu'il y a plusieurs producteurs et plusieurs consommateurs. L'utilisation d'un mutex a des impacts sur les performances car il y a des interblocages. Pour éviter cela il aurait fallu revoir une grande partie de l'architecture.

Pour analyser et comparer les résultats, nous avons ajouté de nombreuses variables. Ces variables nous permettent de savoir le nombre de transactions par seconde, le nombre de transactions total, les quantités échangées, la volatilité ainsi que la série temporelle du prix.

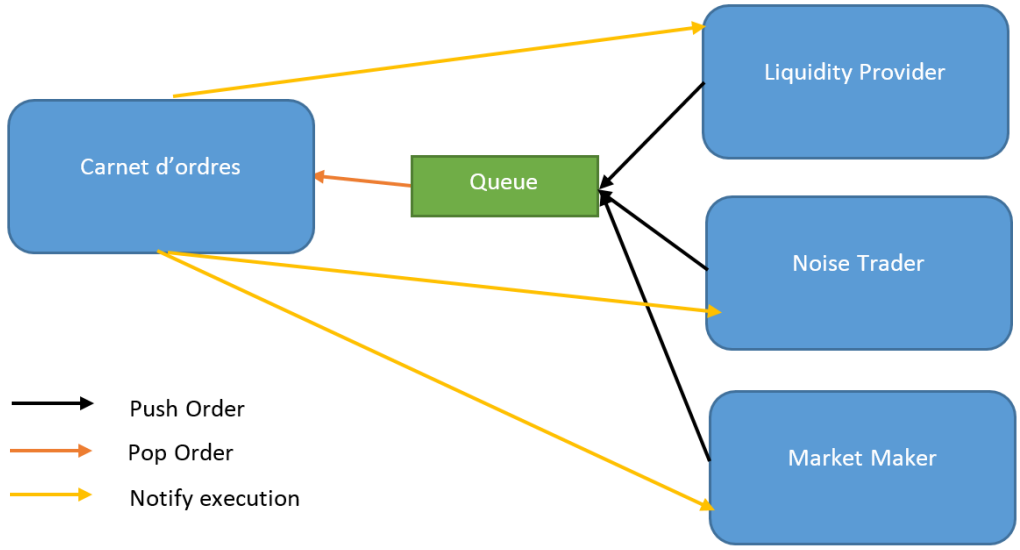

FIGURE 2 – Cheminement d'un ordre

Nous avons aussi dû améliorer le graphique représentant les quantités pour chaque slot de prix sur le marché. En effet, celui du projet repris était imprécis et contenait plusieurs erreurs d'affichage (cf. figure 3).

Ainsi, nous avons dans un premier temps réglé le problème des slots qui avaient des largeurs plus ou moins importantes. Le problème est qu'ils ne géraient pas les slots de quantité nulle. Ils prenaient alors une valeur voisine. De plus, le graphique se trouvait être difficilement lisible. Nous avons donc séparé les boites graphiques par des espaces puis différencié les ordres d'achat et de vente par des couleurs. De plus, les ordres émis par le trader HFT apparaissent dorénavant en vert pour être plus visibles. Le reste des modifications concerne le post-traitement des données envoyées à GNUplot afin que ce dernier les affiche. Nos graphiques sont du type de celui de la figure 4.

2.4.4 Performances du carnet d'ordres

Pour savoir si notre simulateur est performant et si les résultats coïncident avec la réalité, nous les avons comparés avec les données issues du site de NYSE Euronext. Ceci nous a aussi permis de

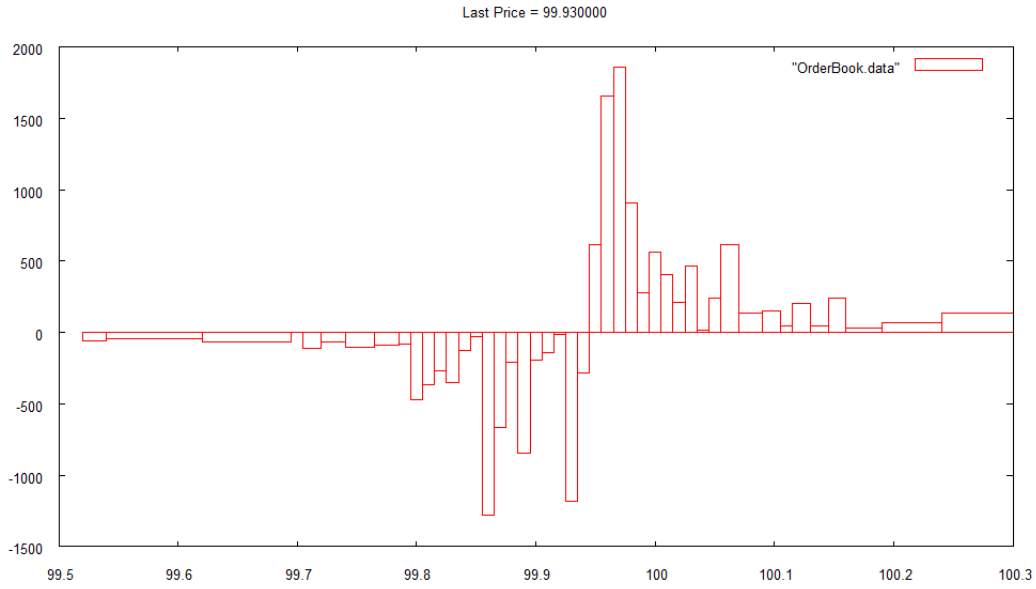

FIGURE 3 – Ancienne sortie graphique du carnet d'ordres sur le projet d'ECP.

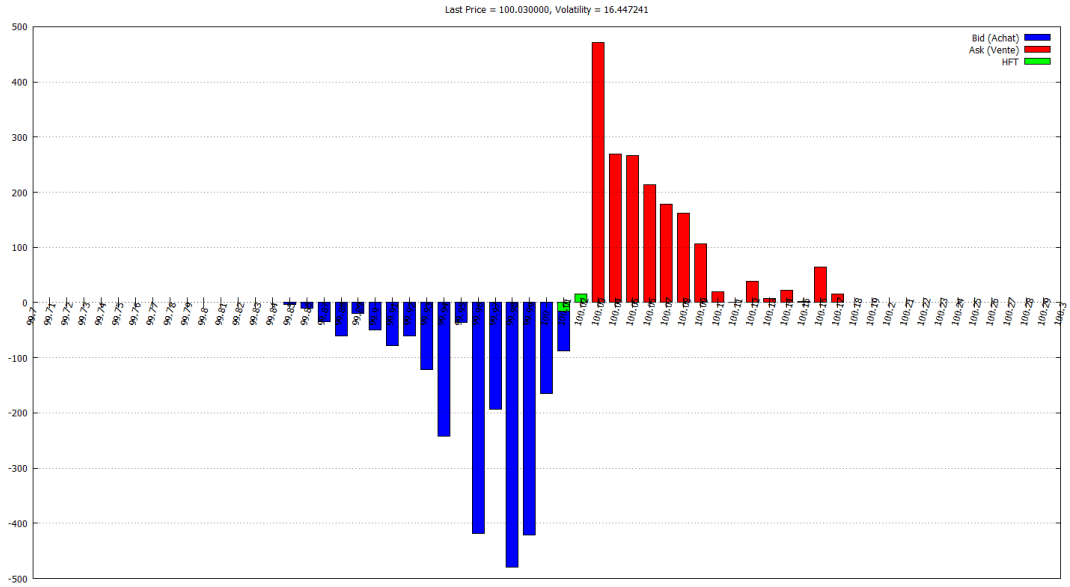

FIGURE 4 – Nouvelle sortie graphique du carnet d'ordres

trouver les bons paramètres pour faire des simulations réalistes. D'après leur site internet, le NASDAQ peut exécuter 900 ordres par seconde. En moyenne notre simulateur exécute 2000 ordres par seconde. Les cours de bourse sont très différents d'une action à une autre. De nombreux paramètres peuvent influencer l'intérêt pour une action, tel que la période de l'année, si des résultats sont publiés ou si des nouvelles sont annoncées. Il est donc très difficile d'avoir un ordre de grandeur des volumes journaliers échangés pour une action. En regardant le volume moyen des 30 derniers jours sur une dizaine d'actions, les volumes échangés correspondent à la réalité.

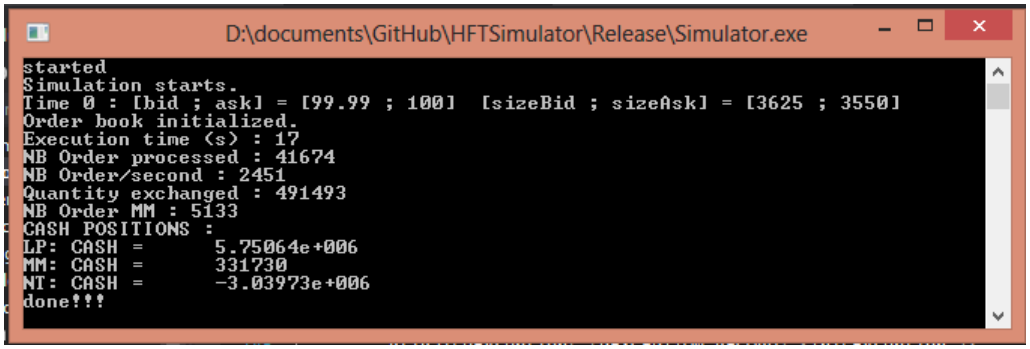

FIGURE 5 – Sortie console après une exécution complète

3 Trading haute fréquence

3.1 Concepts et exemple de stratégie simple

Le HFT consiste en l'exécution à grande vitesse de transactions financières faites par des algorithmes informatiques. Il existe différents types de stratégies HFT. Contrairement aux stratégies classiques de buy-and-hold, un investissement en HFT ne dure que quelques secondes, voire même quelques millisecondes, avec un nombre d'ordres passés et annulés pouvant atteindre les dizaines de milliers. A la clôture du marché, il n'y aucune position ouverte. Les algorithmes de HFT reposent sur la vitesse de traitement (programmation en temps réel, infrastructure à faible latence) et sur l'accès aux données du marché plus rapide que pour les autres acteurs. Il existe différentes stratégies HFT. Dans notre programme nous en implémentons deux : le market making et le market making avec influence du cours.

3.1.1 Le market making

Cette stratégie consiste à émettre simultanément des ordres limites d'achat et de vente dès que l'on détecte l'apparition d'un spread de façon à profiter de celui-ci. Etant donné que l'on se place en market maker lorsqu'on réalise cette stratégie, nous avons la possibilité d'acheter moins cher (au bid) et de revendre plus cher (au ask) l'actif et de dégager ainsi le gain du bid-ask spread. En réalité l'émission simultanée d'ordres d'achat et de vente n'est pas réalisable en programmation et un de ces deux ordres sera placé avant l'autre avec des probabilités identiques.

3.1.2 Le market making avec influence du cours

Le principe est le même que précédemment (gain du bid-ask spread), mais cette fois-ci le choix du type d'ordre à placer en premier dépend d'un paramètre, $buyFrequencyMM$, qui est la probabilité que le Market Maker émette un ordre d'achat. On peut choisir ce paramètre entre 0 et 1. Lorsqu'il tend vers 1, le Market Maker aura tendance à placer d'abord des ordres d'achat puis des ordres de vente. De cette façon il donne une tendance haussière au cours de l'actif. Et inversement lorsque l'on fait tendre le paramètre vers 0.

3.2 Résultats et statistiques

Dans cette partie nous présentons les résultats empiriques obtenus en mettant en application nos stratégies de HFT sur le simulateur. Nous avons dans chaque cas réalisé 1500 simulations correspondant chacune à une journée de trading avec obligation pour les acteurs de solder leurs positions en fin de simulation. Les statistiques présentées dans cette section portent sur la valeur des gains ou pertes (P&L) à l'issue de la journée de trading.

3.2.1 Market Making sans priorité

Dans un premier temps, nous appliquons une stratégie de Market Making sans priorité donnée aux ordres de notre trader HFT par le carnet d'ordre. Cela consiste pour ce dernier à détecter un spread

supérieur à 2 centimes et à passer quasi-simultanément un ordre d'achat et de vente dans cet écart de prix d'un volume égal à 10% du slot le plus proche. On précise que le trader HFT n'a, dans cette partie, aucune préférence pour l'achat ou la vente en premier, il n'essaye donc pas d'influencer le cours du prix mais seulement de profiter du spread pouvant apparaître. Les résultats obtenus sont les suivants :

- Moyenne du P&L du trader HFT : 6.7254e+004,
- Intervalle de confiance : [6.2116e+004 ; 7.2392e+004],
- P&L mininum : -977.19,
- P&L maximum : 602503,
- Volatilité du P&L : 1.0153e+005,
- Moyenne du nombre d'ordres passés par le trader HFT : 5.2294e+003,
- Moyenne du nombre d'ordres total : 4.1445e+004,
- Proportion d'ordres moyenne du trader HFT : 12.62 %.

On présente également la densité estimée du P&L à l'aide de la méthode des histogrammes sur la figure 6.

On remarque que la moyenne est clairement positive (de l'ordre de 10^4). De plus le P&L minimum est seulement d'environ -1000 alors que les gains peuvent atteindre plus de 600000, ce résultat met en évidence la performance de la stratégie utilisée. Concernant la densité estimée, on observe que la majorité de la masse est concentrée sur les valeurs positives mais peu élevées des P&L. Une décroissance est observée avec l'augmentation du profit.

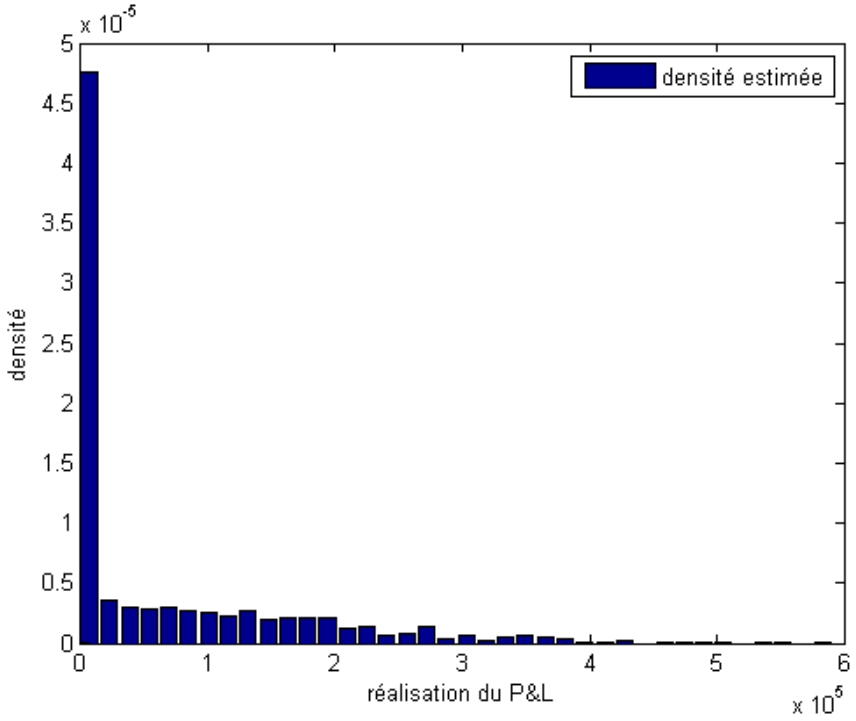

FIGURE 6 – Densité estimée du P&L

3.2.2 Market Making prioritaire

A présent, nous mettons en place une stratégie de Market Making avec ordres du trader HFT prioritaires sur le reste du marché. La stratégie reste cependant la même que celle du paragraphe précédent (sans préférence pour l'achat ou la vente en premier). Les résultats obtenus sont les suivants :

- Moyenne du P&L du trader HFT : 6.6410e+004,

– Intervalle de confiance : [6.0344e+004 ; 7.2476e+004],
– P&L mininum : -527.83,
– P&L maximum : 564648,
– Volatilité du P&L : 9.7875e+004,
– Moyenne du nombre d'ordres passés par le trader HFT : 5.3020e+003,
– Moyenne du nombre d'ordres total : 4.1455e+004,
– Proportion d'ordres moyenne du trader HFT : 12.79 %.

On présente à nouveau la densité estimée du P&L à l'aide de la méthode des histogrammes sur la figure 7.

Les résultats sont très semblables à ceux du paragraphe précédent. Cela peut s'expliquer par le fait que même lorsque le trader HFT n'est pas volontairement considéré comme prioritaire par le carnet d'ordre, il est souvent le premier à passer des ordres sur les slots de prix sur lesquels il agit (nouveau bid et nouveau ask). Ainsi il a de grandes chances d'être le premier acteur à être exécuté.

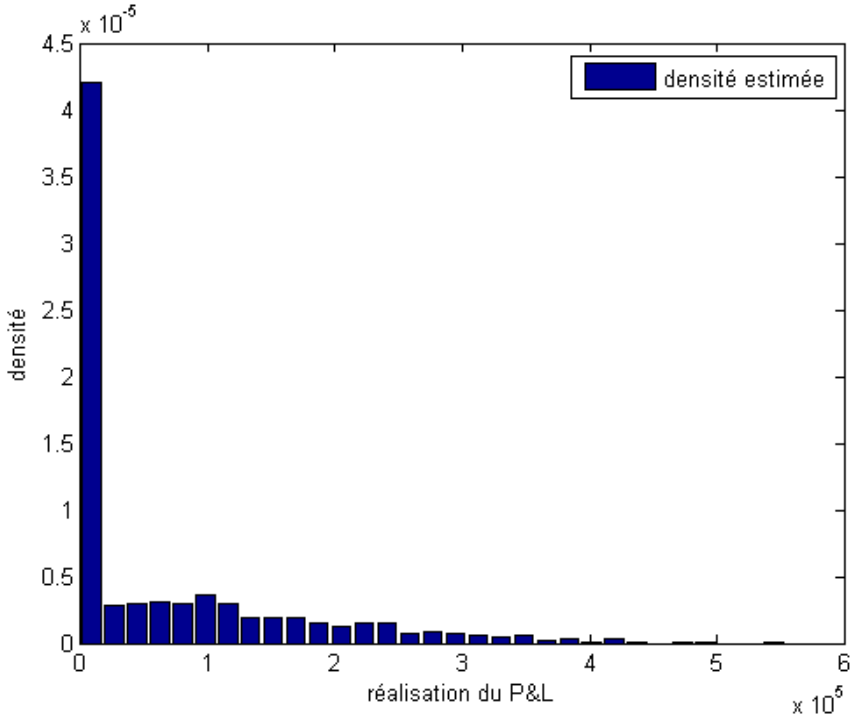

FIGURE 7 – Densité estimée du P&L

3.2.3 Market Making prioritaire avec volume 5 %

A nouveau, nous réalisons une stratégie de Market Making avec ordres du trader HFT prioritaires sur le reste du marché. La stratégie reste la même que celle du paragraphe précédent hormis le fait que cette fois-ci les volumes des ordres passés lors de l'apparition d'un spread suffisamment large sont de l'ordre de 5% de celui du slot de prix le plus proche. Les résultats obtenus sont les suivants :
– Moyenne du P&L du trader HFT : 3.3895e+004,
– Intervalle de confiance : [3.0639e+004 ; 3.7152e+004],
– P&L mininum : -243.63,
– P&L maximum : 303715,
– Volatilité du P&L : 5.2539e+004,
– Moyenne du nombre d'ordres passés par le trader HFT : 5.2144e+003,
– Moyenne du nombre d'ordres total : 4.1778e+004,

– Proportion d'ordres moyenne du trader HFT : 12.48 %.

La densité estimée du P&L est représentée sur la figure 8.

On remarque une dispersion plus faible des P&L (écart-type inférieur aux cas précédents), ainsi les gains et pertes extrêmes sont moins importants.

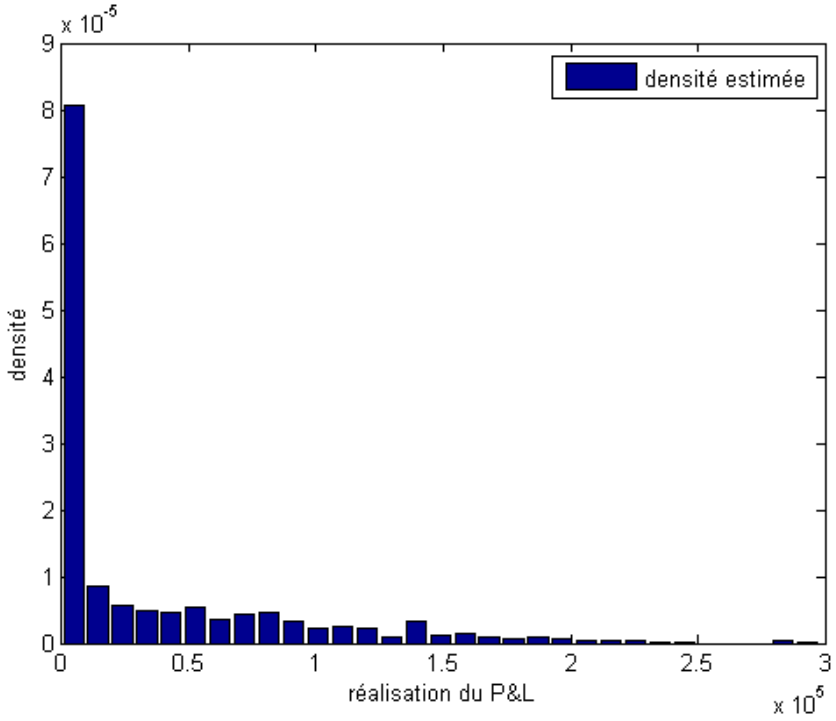

FIGURE 8 – Densité estimée du P&L

3.2.4 Market Making prioritaire avec volume 1 %

Nous mettons en place la même stratégie de Market Making que celle du paragraphe précédent mais les volumes des ordres passés lors de l'apparition d'un spread sont de l'ordre de 1% de celui du slot de prix le plus proche. Les résultats obtenus sont les suivants :

- Moyenne du P&L du trader HFT : 1.4470e+004,
- Intervalle de confiance : [1.3398e+004 ; 1.5541e+004],
- P&L mininum : -149.87,
- P&L maximum : 121025,
- Volatilité du P&L : 2.1173e+004,
- Moyenne du nombre d'ordres passés par le trader HFT : 5.1819e+003,
- Moyenne du nombre d'ordres total : 4.1956e+004,
- Proportion d'ordres moyenne du trader HFT : 12.35 %.

La densité estimée du P&L est représentée sur la figure 9.

On observe le même comportement que précédemment de manière plus marquée avec cette fois-ci un maximum de l'ordre de 120000 et un minimum de -150. L'écart-type est à nouveau plus faible. Cela est en accord avec l'intuition qui est que lorsque le trader émet des ordres avec des volumes moins importants, les risques sont plus modérés et les gains par conséquent plus faibles. On peut ainsi considérer que la meilleure stratégie pour le trader HFT est de cumuler au maximum les petits gains sur chaque journée de trading sans prendre forcément de grands risques.

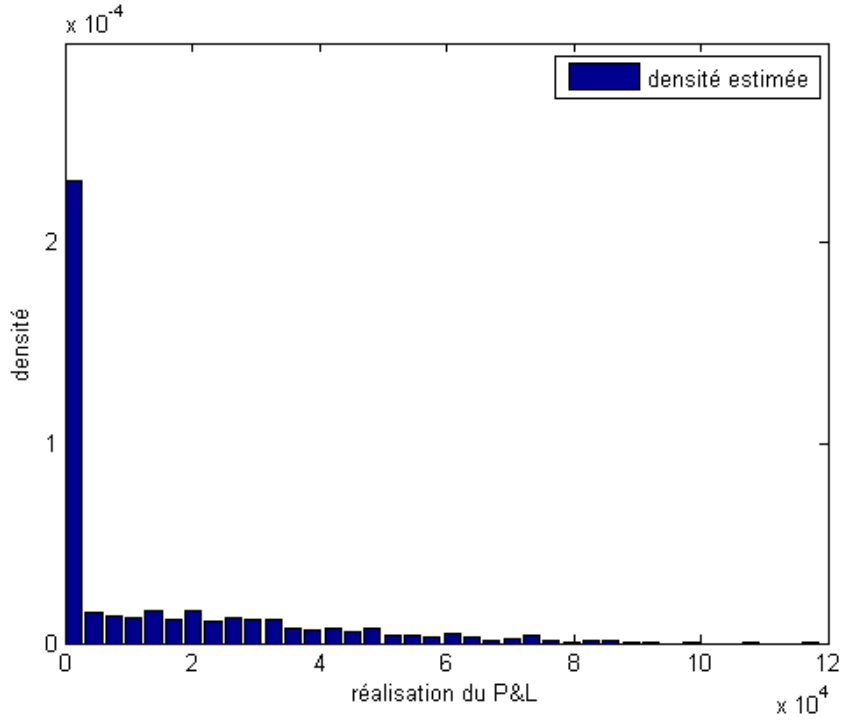

FIGURE 9 – Densité estimée du P&L

3.2.5 Market Making avec influence du cours

Dans cette dernière configuration, nous appliquons une stratégie de Market Making (toujours prioritaire) visant à influencer le cours de l'actif. Cela consiste toujours pour le trader HFT à détecter un spread supérieur à 2 centimes et à passer quasi-simultanément un ordre d'achat et de vente dans cet écart de prix (avec un volume égal à 10% du slot le plus proche). Cependant cette fois-ci le trader HFT a une plus grande probabilité de passer en premier un ordre d'achat (0.6) que de passer un ordre de vente (0.4). Cela a pour effet de combler le spread en faisant monter le prix du sous-jacent. Nous devons garder à l'esprit que ce type de stratégie peut être utile si le trader HFT agit pour le compte d'un organisme ayant tout intérêt à ce que le cours de l'action monte (par exemple une banque ayant vendu des options barrières). Les résultats obtenus sont les suivants :

- Moyenne du P&L du trader HFT : 2.6034e+005,
- Intervalle de confiance : [2.5208e+005 ; 2.6859e+005],
- P&L mininum : -270.17,
- P&L maximum : 843857,
- Volatilité du P&L : 1.6310e+005,
- Moyenne du nombre d'ordres passés par le trader HFT : 5.1578e+003,
- Moyenne du nombre d'ordres total : 4.1358e+004,
- Proportion d'ordres moyenne du trader HFT : 12.47 %.

On présente la densité estimée du P&L sur la figure 10.

On note que la masse de la densité estimée est beaucoup plus étalée sur les valeurs positives du P&L comme en témoigne la valeur de l'écart-type. En effet, l'influence du prix de l'actif à la hausse a une conséquence positive sur les gains du trader. Le maximum atteint allant même jusqu'à plus de 800000.

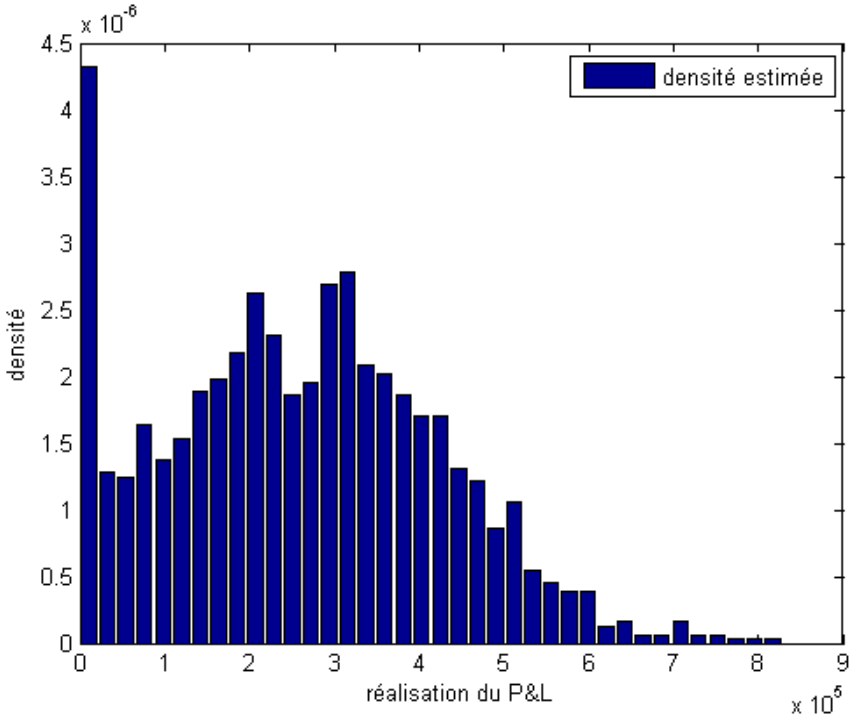

FIGURE 10 – Densité estimée du P&L

3.2.6 Comparaison des densités des différents P&L

Afin de pouvoir effectuer une comparaison des différentes densités des P&L nous superposons les estimations réalisées à l'aide de la méthode du noyau sur la figure 11. Le résultat confirme nos observations précédentes. En effet, les densités dans les cas où le trader HFT est prioritaire et non prioritaire (pour un même volume d'ordre de 10%) sont quasiment identiques. En revanche, on observe bien le fait que lorsque le volume diminue, la densité du P&L est plus concentrée au niveau des valeurs faibles et moins étalée. Pour la stratégie biaisée, la densité est de nature différente que pour les autres cas. Elle est plus étalée sur les valeurs importantes du P&L, et on n'observe pas de pic important.

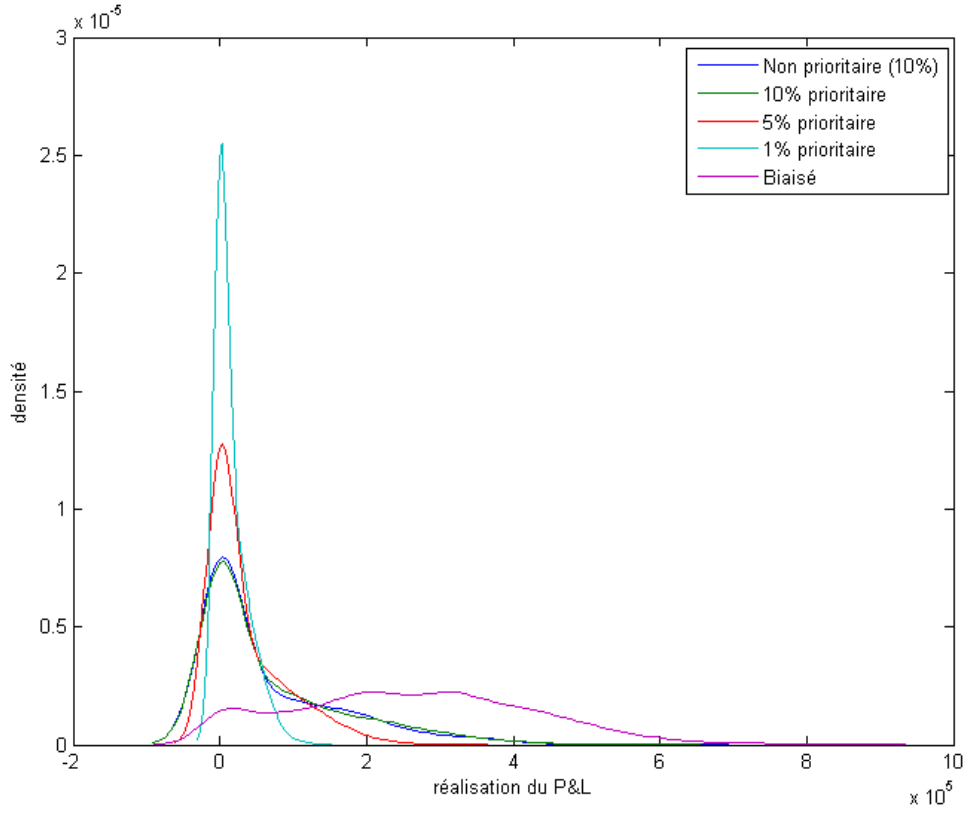

FIGURE 11 – Superposition des estimation de densité des P&L par méthode du noyau

3.3 Trajectoires des prix

Nous présentons dans cette partie quelques trajectoires de prix générées par notre simulateur.

3.3.1 Trajectoire sans trader HFT

Sur la figure 12 on peut observer la trajectoire d'un prix sans la présence de l'agent HFT sur le marché. Aucune tendance d'évolution du marché n'est mis en évidence, le prix final est quant à lui relativement proche de la valeur de départ.

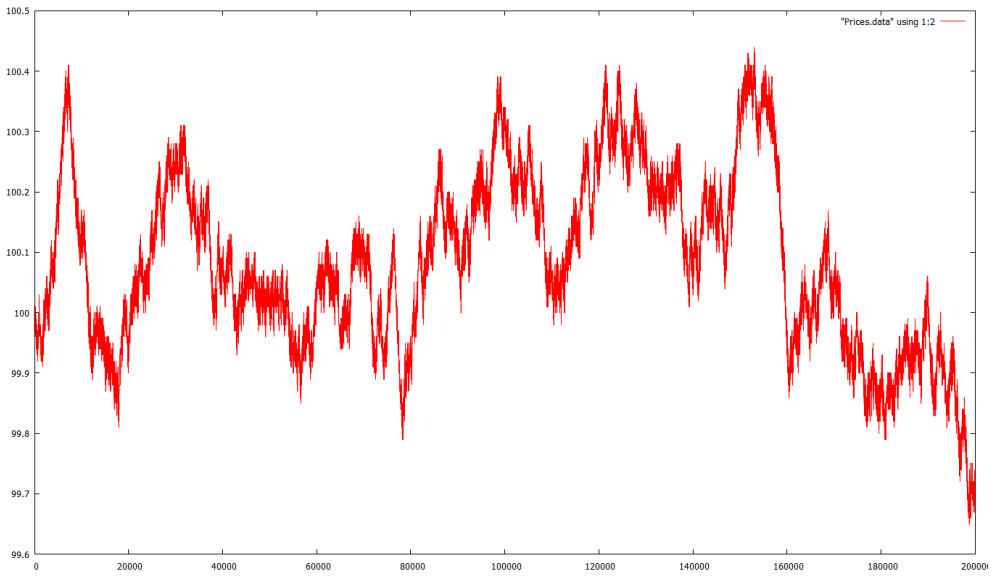

FIGURE 12 – Trajectoire du prix sans trader HFT

3.3.2 Trajectoire avec trader HFT et volume 10 %

Nous introduisons à présent le trader HFT adoptant la stratégie du paragraphe 3.2.2. Une trajectoire simulée est visible sur la figure 13. On remarque que le prix semble être moins volatile. Nous pouvons penser que le fait que le trader HFT s'intercalle à chaque fois dans les slots libres constituant le spread est à l'origine de ce phénomène. De cette façon, le prix ne bouge que faiblement entre chaque actions.

3.3.3 Trajectoire avec trader HFT et volume 1 %

La trajectoire présente sur la figure 14 correspond à un prix simulé avec présence du trader HFT adoptant la stratégie du paragraphe 3.2.4. On remarque la valeur du prix final est beaucoup moins éloignée du prix initial que dans la cas où le volume des ordres était de 10%.

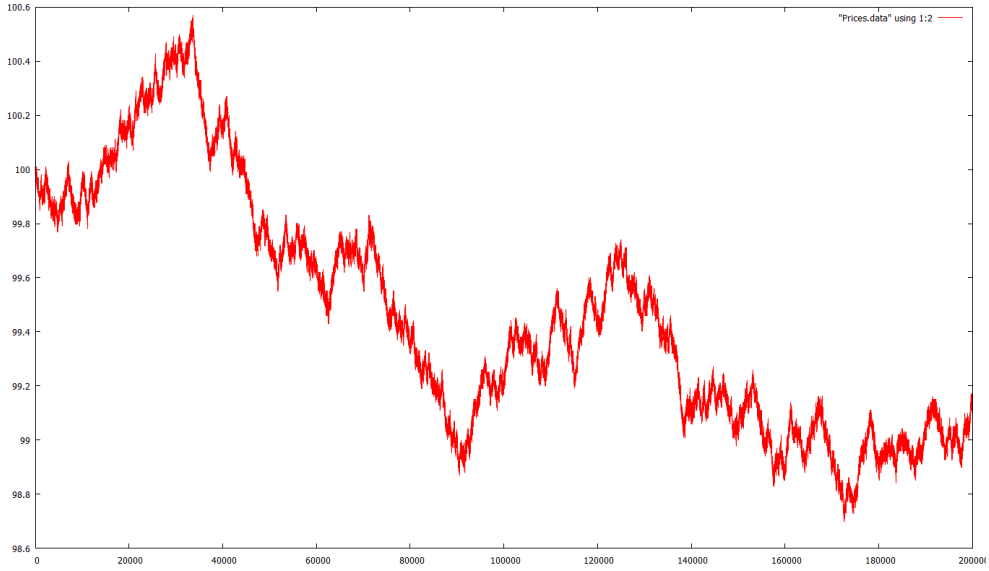

FIGURE 13 – Trajectoire du prix avec trader HFT (10%)

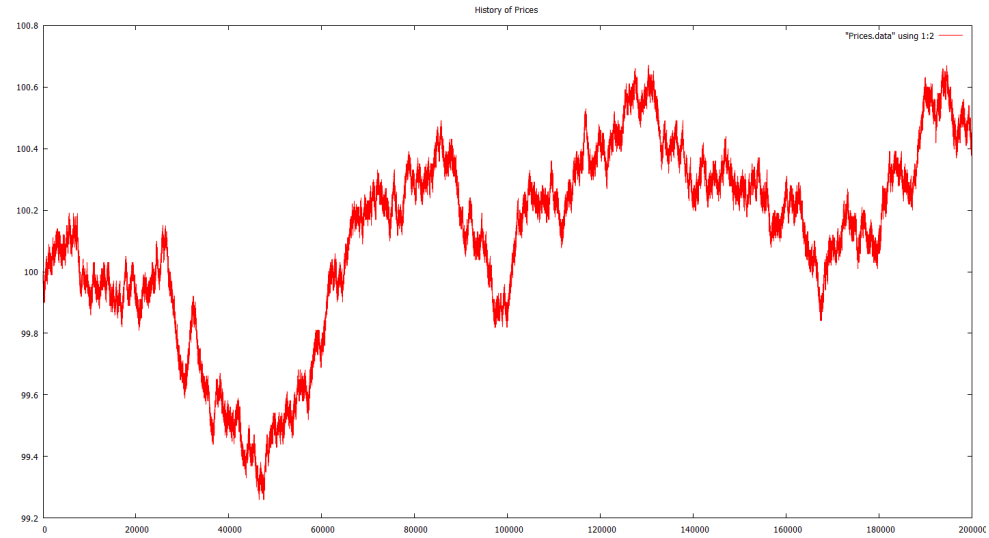

FIGURE 14 – Trajectoire du prix avec trader HFT (1%)

3.3.4 Trajectoire avec trader HFT et influence du cours à la hausse

La dernière trajectoire présentée sur la figure 15 est obtenue avec un trader HFT manipulant le cours de l'actif à la hausse (cf. paragraphe 3.2.5). On voit que la stratégie influence clairement le prix, en effet le prix final est d'environ 103 pour un prix initial de 100.

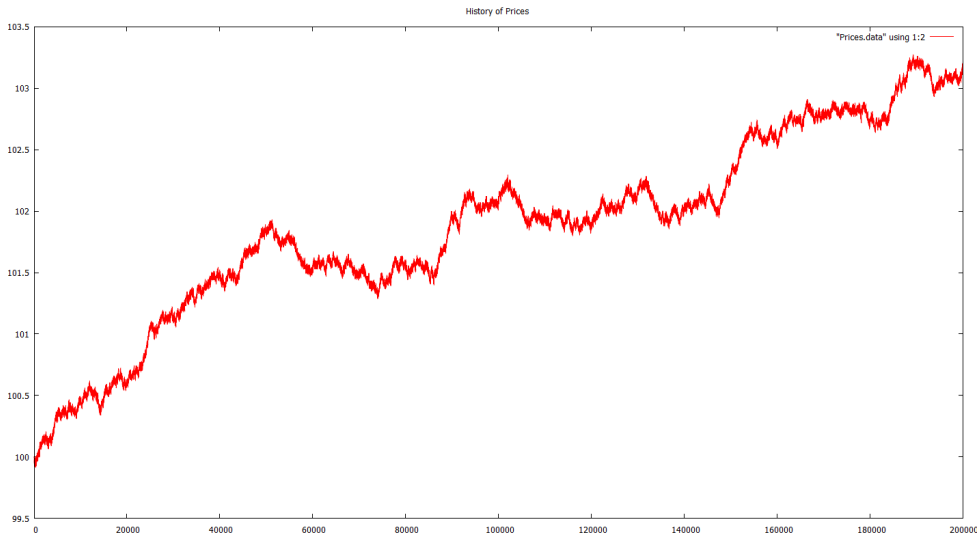

FIGURE 15 – Trajectoire du prix avec trader HFT manipulant à la hausse

Conclusion

Ce projet de fin d'étude constituait une véritable application des notions et des outils mathématiques, informatiques et financiers étudiés au cours de cette dernière année d'étude. Le cœur du projet était d'implémenter certaines stratégies de HFT et d'étudier les résultats qu'elles peuvent fournir. Cela nous a permis d'une part d'améliorer nos connaissances en culture et mathématiques financières, mais également de progresser face aux problèmes de programmation rencontrés. En effet, la première phase de notre projet a consisté à se documenter sur le fonctionnement d'un carnet d'ordres et sur les stratégies HFT. La deuxième phase a nécessité une mobilisation de l'ensemble de notre groupe sur l'implémentation en C++, ce qui nous a notamment amené à utiliser pleinement un gestionnaire de version. Enfin, l'analyse de nos résultats et le calcul de statistiques ont occupé la dernière partie de nos travaux. Ces derniers ont mis en évidence la performance de nos stratégies malgré leur simplicité. Cependant il est impératif de garder à l'esprit que les résultats fournis sont issus d'un simulateur de marché et ce dernier ne prend pas forcément en compte tout les évènements qui peuvent se produire sur les marchés financiers réels. Au niveau des perspectives, il serait, dans un premier temps, intéressant de tester nos stratégies avec un marché réagissant instantanément aux ordres passés par notre agent HFT. De plus, expérimenter nos stratégies en présence de deux traders HFT sur le marché pourrait également être un sujet de réflexion. Enfin, implémenter de nouvelles stratégies plus complexes pourrait s'avérer être une suite logique à notre projet.

Annexes

Les informations comprises dans cette partie ont pour but de préciser la façon dont s'est organisé le projet.

A Diagramme de Gantt

Le lecteur trouvera sur la figure 16 le diagramme de Gantt de notre projet.

B Rapport d'activités

Période du 18/12/2012 au 08/01/2013
- Youness Abdessadek : Recherche,
- Maxime Bonelli : Recherche,
- Jérémie Montiel : Recherche,
- Corentin Valleroy : Recherche.

Réunion le 19/12/2012 avec Dr Mireille Bossy et Mme Anne-Marie Hugues

Semaine du 09/01/2013 au 16/01/2013
- Youness Abdessadek : Amélioration du plot,
- Maxime Bonelli : Mise en place du multi-threading,
- Jérémie Montiel : Amélioration du plot,
- Corentin Valleroy : Mise en place du multi-threading.

Semaine du 17/01/2013 au 24/01/2013
- Youness Abdessadek : Adaptation du nouveau modèle mathématique,
- Maxime Bonelli : Adaptation du nouveau modèle mathématique,
- Jérémie Montiel : Mise en place du multi-threading,
- Corentin Valleroy : Mise en place du multi-threading.

Soutenance intermédiaire le 17/01/2013 avec Mme Anne-Marie Hugues

Semaine du 25/01/2013 au 01/02/2013
- Youness Abdessadek : Mise en place du multi-threading,
- Maxime Bonelli : Mise en place du multi-threading,
- Jérémie Montiel : Mise en place du multi-threading,
- Corentin Valleroy : Mise en place du multi-threading.

Réunion le 30/01/2013 avec Dr Mireille Bossy

Semaine du 02/02/2013 au 09/02/2013
- Youness Abdessadek : Implémentation stratégie HFT,
- Maxime Bonelli : Implémentation stratégie HFT,
- Jérémie Montiel : Implémentation stratégie HFT,
- Corentin Valleroy : Implémentation stratégie HFT.

Réunion le 08/02/2013 avec Dr Mireille Bossy

Semaine du 10/02/2013 au 17/02/2013
- Youness Abdessadek : Simulations de Monte-Carlo,
- Maxime Bonelli : Simulations de Monte-Carlo,
- Jérémie Montiel : Simulations de Monte-Carlo,
- Corentin Valleroy : Implémentation stratégie HFT.

Semaine du 18/02/2013 au 25/02/2013

- Youness Abdessadek : Statistiques Matlab,
- Maxime Bonelli : Statistiques Matlab,
- Jérémie Montiel : Rapport et présentation,
- Corentin Valleroy : Rapport et présentation.

Réunion le 20/02/2013 avec Dr Mireille Bossy

Semaine du 25/02/2013 au 28/02/2013

- Youness Abdessadek : Rapport et présentation,
- Maxime Bonelli : Rapport et présentation,
- Jérémie Montiel : Rapport et présentation,
- Corentin Valleroy : Rapport et présentation.

Réunion le 26/02/2013 avec Dr Mireille Bossy

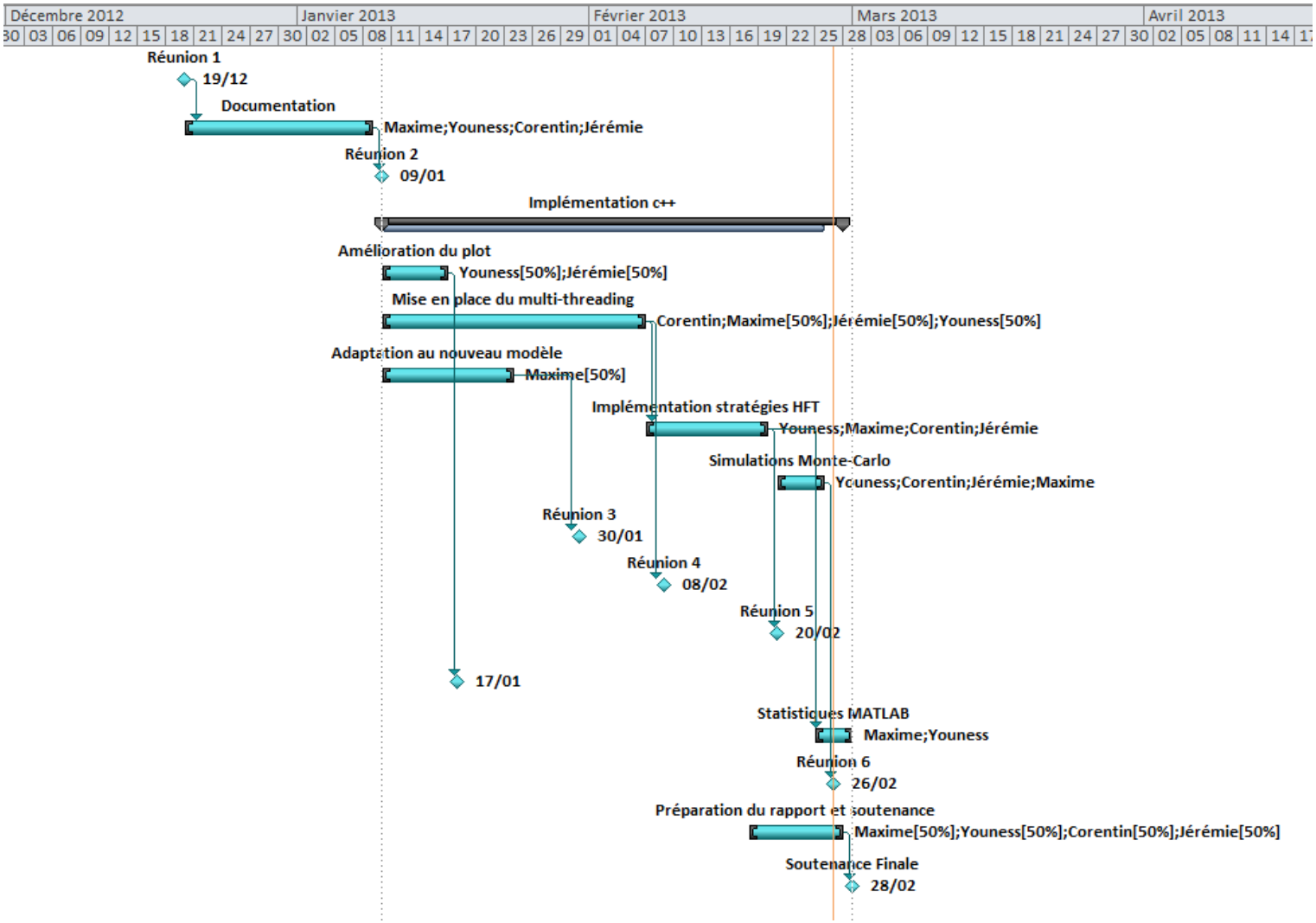

FIGURE 16 – Diagramme de Gantt

Références

[1] Damien Lamberton, Bernard Lapeyre, *Introduction au Calcul Stochastique Appliqué à la Finance*, Ellipses, 3ème édtion, 2012.

[2] Carl Graham , Denis Talay, *Simulation stochastique et méthodes de Monte-Carlo*, Ecole polytechnique, 2011.

[3] Rapport de stage, Ecole centrale Paris encadré par F.Abergel, *Simulation des marchés financiers par modèle multi-agents*, 2012.

[4] Fabien Guibaud, Huyên Pham *Optimal high-frequency trading in a pro-rata microstructure with predictive information*, 2012.

[5] T. Preis, S. Golke, W. Paul, J. Schneider, *Multi-agent–based Order Book Model of financial markets*, 2006.

[6] Frantisek Slanina, *Critical comparison of several order-book models for stock-market fluctuations*, 2008.

[7] Encyclopédie libre, *Wikipedia.org*.