Bulles, Krachs et Intermittence dans les modèles de marché multi-agents

Cablant Augustin, Tran-Thuong Tien-Thinh, Vivet Taddeo
Encadré par Mr. Maitrier

Ensae Paris

31 mars 2023

Plan de la Présentation

1. Introduction
2. Présentation du cadre
   - A - Loi de Pareto
   - B - Volatilité
3. Simulation des agents
   - Présentation de la simulation
   - Structure du code
   - Existence de 3 régimes caractéristiques en fonction de P et $\frac{g}{\lambda}$
4. Analyse et discussion de l'article
   - Les prix lorsque les agents agissent aléatoirement (fully random)
   - Ce qu'il faut retenir du cadre
   - Le régime periodique
   - L'efficience du marché
5. Conclusion
6. Annexes
7. code

Introduction

Introduction

- Simulation financière : “marchés artificiels”
- Étude d’un modèle multi-agents
- Les agents considérés effectuent des choix en fonction des stratégies proposées. Chaque agent agit indépendamment des autres agents.
- Inspiration : problème du bar “El-Farol” ou “jeux de minorités”

El Farol Restaurant and Cantina, Santa Fe

Présentation du cadre

Loi de Pareto

Soit X une variable aléatoire suivant une loi de Pareto de paramètres (x_m,k) avec $k > 0$:

$$\mathbb{P}([X > x]) = (\frac{x_m}{x})^k \text{ avec } x \geq x_m$$

$$\mathbb{E}[X] = \frac{k}{k-1} x_m$$

$$\mathbb{V}[X] = (\frac{x_m}{k-1})^2 \frac{k}{k-2}$$

La loi de Pareto est reliée à la loi exponentielle puisque :

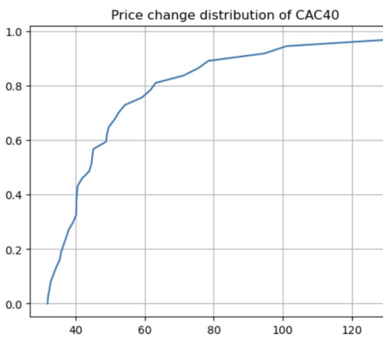
Pour $Y \approx \mathcal{E}(k)$, $f(x; k, x_m) = k \frac{x_m^k}{xk+1} = f_Y(log(\frac{x}{x_m}))$

Simulation de la loi de Pareto

Nous avons montré que les changements de prix suivent une **loi de pareto** :

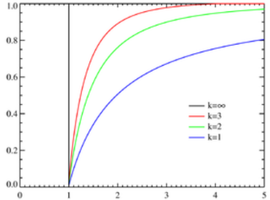
```python
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

```python
def plot_distribution(tick, label="", normalise=False):
    # CREATE TICKER INSTANCE
    ticker = yf.Ticker(tick)

    # GET TODAYS DATE AND CONVERT IT TO A STRING WITH YYYY-MM-DD FORMAT (YFINANCE EXPECTS THAT FORMAT)
    end_date = datetime.now().strftime('%Y-%m-%d')
    start_date = (datetime.now() - timedelta(50)).strftime('%Y-%m-%d')

    # GET DATAFRAME
    df = ticker.history(start=start_date,end=end_date, interval="15m")
    df['Price Change'] = abs(df['Close'].diff())

    # PLOTTING
    df['Price Change']
    x = np.array(sorted(df['Price Change'][df['Price Change']>df['Price Change'].std()*3].dropna()))
    # On ne garde que la queue de
    if normalise:
        x = 100*x/max(x)
    n = len(x)
    y = [i/n for i in range(n)]
    plt.plot(x, y, label=label)

df = plot_distribution("^FCHI")
plt.title("Price change distribution of CAC40")
plt.grid()
```

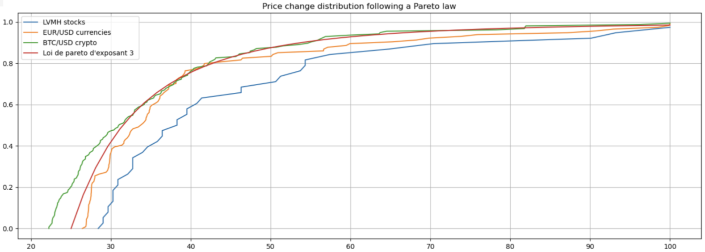


Simulation de la loi de Pareto

```python
plt.figure(figsize=(18, 6))

plot_distribution("MC.PA", label="LVMH stocks", normalise=True)
plot_distribution("EURUSD=X", label="EUR/USD currencies", normalise=True)
plot_distribution("BTC-USD", label="BTC/USD crypto", normalise=True)

xm = 25
x3 = np.linspace(xm, 100)
y3 = [1-(xm/x_)**3 for x_ in x3]
plt.plot(x3, y3, label="Loi de pareto d'exposant 3 ")

plt.title("Price change distribution following a Pareto law")
plt.grid()
plt.legend()
plt.savefig("Pareto_law.jpg")
```

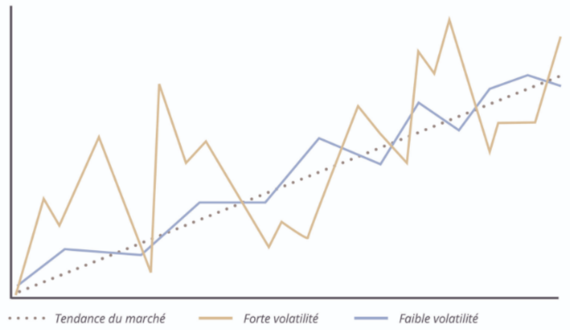


Fonction de répartition d'une loi de Pareto



Volatilité

La **volatilité** :
- Quantifie le risque d'un actif financier
- Dépend de la variabilité des prix et de l'incertitude
- On utilise l'écart-type des variations historiques de rentabilité pour la calculer.



Qu'est-ce que la volatilité ?

Simulation des agents

Cadre

Les agents peuvent prendre trois décisions :
- La décision d'acheter des actions (en convertissant les obligations en espèces)
- De vendre des actions
- D'être inactif (c'est-à-dire de conserver les obligations)

Chaque stratégie se voit attribuer un score, qui est mis à jour en fonction de ses performances. La stratégie jouée au temps t par un agent donné est celle, parmi celles qui lui sont disponibles, qui aurait été la plus **performante dans un passé récent**.

N agents qui dispose de $S - 1$ stratégies dites actives et d'une stratégie inactive. À chaque instant t, un agent $i \in 1, ..., N$ dispose de $\phi_i(t)$ d'actions ("stocks") et de $B_i(t)$ d'obligations.

La richesse total de l'agent i : $B_i(t) + \phi_i(t)X(t)$.

Les dynamiques du modèle entre les instants t et $t + 1$:

i) Information :
Tous les agents disposent de la même information $\mathcal{I}_t$ donnée par les m derniers pas de l'historique de la série temporelle (m pour mémoire). Cette information est qualitative et dépend **uniquement** du signe des précédents changements :

$\mathcal{I}_t = \mathcal{X}(t - m), ..., \mathcal{X}(t - 1)$

$\mathcal{X}(t) = \text{sign}[log(\frac{X(t)}{X(t-1)}) - \rho]$ où ρ désigne de ce que rapporte l'obligation sans risque.

Les dynamiques du modèle entre les instants t et $t+1$:

ii) Stratégies :
Chaque agent i se voit attribué un nombre S de stratégies fixées. Ces stratégies convertissent l'information $\mathcal{I}_t$ en une décision $\epsilon_i(\mathcal{I}_t) = \pm 1, 0$ (respectivement acheter, vendre ou rester inactif).
Toutefois, on peut donner un biais à ce choix aléatoire, qui résulterait de stratégies de mode ou d'anticonformisme, en définissant la **magnétisation**, $M \in [-1, 1]$ de la chaîne $\mathcal{X}(1), \mathcal{X}(2), ..., \mathcal{X}(m)$ défini comme :

$$M = \frac{1}{m} \sum_{j=1}^{m} \mathcal{X}(j).$$

Dès lors, l'agent choisit la décision correspondant $\epsilon = 1$ avec une probabilité de $\dfrac{1+PM}{2}$ et $\epsilon = -1$ avec une probabilité $\dfrac{1-PM}{2}$ où $P \in [-1, 1]$ représente la **polarisation** des stratégies.

Les dynamiques du modèle entre les instants t et $t+1$:

iii) Décisions :
Sachant les stratégies de l'agent i et l'information $\mathcal{I}_t$ dont il dipose, nous pouvons trouver la décision $\epsilon_i(t)$ de chaque agent. En fonction de cette valeur, l'agent achète ou vend une certaine quantité $q_i(t)$ qui est proportionnelle à ce qu'il possède. En d'autres termes :
$q_i(t) = g\dfrac{B_i(t)}{X(t)}$ pour $\epsilon_i(t) = 1$ puisque dans ce cas, l'agent i est invité à acquérir des obligations.
$q_i(t) = -g\phi_i(t)$ pour $\epsilon_i(t) = -1$ puisque dans ce cas, l'agent est invité à vendre des actions.
$q_i(t) = 0$ pour $\epsilon_i(t) = 0$.

Formation du prix et mécanisme de compensation des prix

Pour étudier les variations de prix on introduit le déséquilibre total :

$Q(t) = \frac{1}{\Phi}\sum_{i=1}^{N} q_i(t) = Q^+(t) - Q^-(t)$ où Φ désigne le nombre total d'actions en circulation (on le suppose constant), Q^+ est la fraction des ordres d'achats et Q^- la fraction des ordres de ventes.

Actualisation des scores

Chaque agent attribue des scores à ses stratégies pour mesurer leurs performances et utilise à l'instant t la meilleure stratégie, c'est-à-dire celle qui a obtenu les meilleurs scores.

Structure du code

I/ Initialisation

1) N agents, S stratégies

2) P polarisation des stratégies (1 : Trend following, -1 : Mean reverting) $\frac{g}{\lambda}$

(g : la fraction du portefeuille à investir, λ *: « rigidité » du marché)*

II/ Simulation

1) Choix de la meilleure stratégie pour chaque agent (basé sur les prix passés)

2) Ordre d'achat ou de vente en fonction de la meilleure stratégie pour chaque agent

3) Calcul des quantités effectivement à échanger

4) Calcul du nouveau prix

III/ Analyse

1) Afficher les prix à partir de la 3000 *è itération pour atteindre un régime permanent indépendant des prix initiaux*

Initialisation

```python
class simulation:
    """
    Cette classe permet de créer la simulation décrite dans l'article :
    BUBBLES, CRASHES AND INTERMITTENCY IN AGENT BASED MARKET MODELS
    de Irene Giardina et Jean-Philippe Bouchaud
    décrit en 2018
    """
    # Nous déclarons ci-dessous les constantes de la classe
    S = 3 # Le nombre de stratégie (sachant que 1 stratégie est l'inactivité)
    m = 5 # La quantité de mémoire des agents
    g = 0.005 # La fraction d'action acheté ou vendu
    f = 0.05 # Le niveau de confiance
    N = 1_001 # Le nombre d'agent
    beta = 1-1e-2 # La mémoire du score

    def __init__(self, P, g_sur_l=0.1, alpha=1-1e-4, rho=0.02, phi=3_003, pi=0):
        self.l = self.g / g_sur_l # Lambda
        self.P = P #Polarisation
        self.alpha = alpha
        self.phi = phi # Le nombre total d'action en circulation (Quelle valeur faut-il mettre ?)
        self.rho = rho # taux d'intérêt (Quelle valeur faut il mettre ? Regarder la croissance de la courbe Figur
        self.pi = pi
```

Création des stratégies

Au lieu d'encoder l'information avec une liste de +1 (hausse) et −1 (baisse), on le fait avec des 1 et des 0 c'est-à-dire que nous prenons la représentation binaire des nombres entre 0 et 2^m.

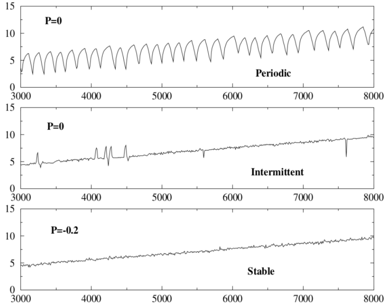
```
def creation_strategie(self, m):
    # Il y a 2^m I_t (information) possible
    # L'information est une suite de m élément 1 ou -1
    # Je vais donc représenter I_t est donc une suite de binaire 1:1 et 0:-1
    # Pour simplifier la manipulation de I_t je vais prendre sa représentation décimale
    # I_t est donc à valeure dans [0, 2^m-1]

    # Choisissons les possibilités acheteuses, d'après la variable self.P
    # P=1 tendance à Trend Following
    # P=-1 tendance à Mean Reverting
    liste_acheter = []
    for info in range(2**m):
        nb_positif = (bin(info)[2:]).count("1")
        nb_negatif = self.m - nb_positif
        M = (nb_positif-nb_negatif)/self.m
        if np.random.random() < (1+self.P*M)/2:
            liste_acheter.append(info)

    # Créons la fonction epsilon_i(I_t)
    def epsilon_i(I_t):
        if I_t in liste_acheter:
            return 1
        else:
            return -1
    return epsilon_i
```

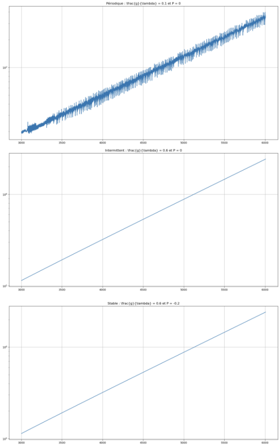
Calcul du score

On effectue une mise à jour des scores des stratégies actives proportionnellement au profit relatif, corrigé par le taux d'intérêt :
$S_i^{\alpha}(t+1) = (1-\beta)S_i^{\alpha}(t) + \beta\epsilon_i^{\alpha}(\mathcal{I}_{t-1})[r(t)-\rho], \alpha \in \ 1, ..., S-1\ .$

```
def calcul_score(self):
    info = self.information(t=-1)
    r = np.log(self.X[-1]/self.X[-2])
    for i in range(self.N): # Calcul des scores de l'agent i
        for alpha in range(self.S-1): # Calcul des scores de la stratégies
            arg1 = (1-self.beta)*self.score[i][alpha]
            arg2 = self.beta*self.epsilon[i][alpha](info)*(r-self.rho)
            self.score[i][alpha] = arg1 + arg2
```

Régimes caractéristiques





Pourquoi n'avons nous pas les mêmes résultats ?

Pourquoi n'avons nous pas les mêmes résultats ?

Dans le modèle développé par les auteurs, les paramètres sont :

- $S = 3$, $N = 1001$
- $m = 5$
- $g = 0.005$
- $f = 0.05$
- $1 - \beta = 10^{-2}$ et $1 - \alpha = 10^{-4}$

Or, nous n'avons pas accès à la valeur de certaines variables :

- ρ : Le "risk free rate", c'est le gain sur un investissement dit à "zéro risque", il est aussi appelé "interest rate" (taux d'intérêt).

Théoriquement nous avons voulu le fixer à 2% (comme le taux directeur de la BCE)

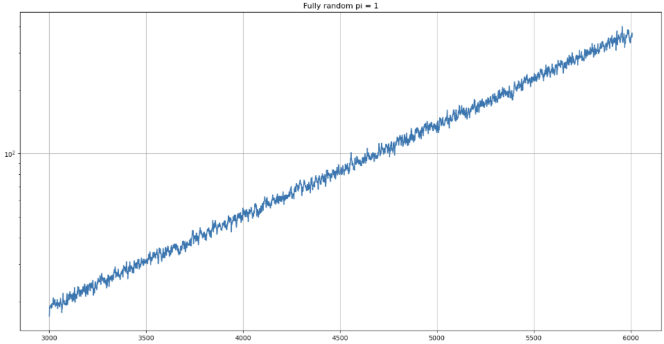
Mais il se trouve que les graphiques de fully random nous donnent $1e-3$

- ϕ : Le nombre total d'action en circulation

Nous avons utilisé le graphique fully random pour adapter la quantité de ϕ.

Analyse et discussion de l'article

Les prix lorsque les agents agissent aléatoirement (fully random)



On obtient une courbe des prix ressemblant à celui de l'article en échelle logarithmique pour l'ordonnée.

Les prix lorsque les agents agissent aléatoirement (fully random)

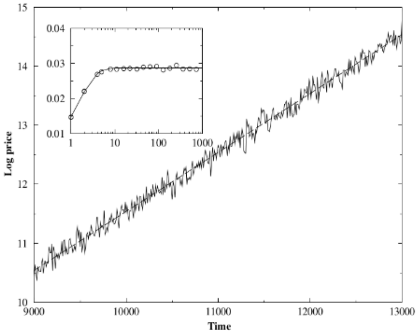
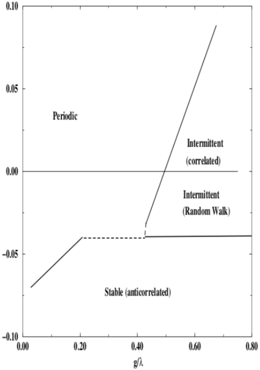


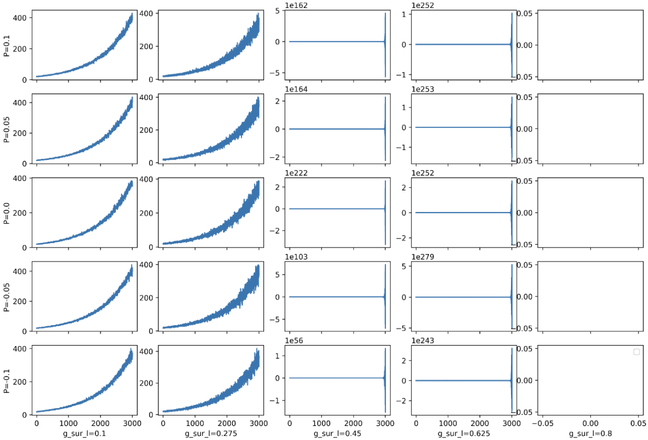
Figure 3: Behaviour of the price as a function of time for purely random strategies. Inset: Variogram of the price fluctuations, and Ornstein-Uhlenbeck fit.

Ce qu'il faut retenir du cadre

Les paramètres vraiment importants sont : $\frac{g}{\lambda}$ et la polarisation P, qui déterminent les changements de prix.
Les autres paramètres influencent les résultats quantitatifs, mais pas les caractéristiques qualitatives, ce qui se traduit par l'apparition de trois régimes qualitativement différents :
- Un "régime oscillatoire" : $\frac{g}{\lambda} <= 0.4$, $P >= 0$.
- Un "régime turbulent" $\frac{g}{\lambda} >= 0.4$, $P >= -|P_0|$
- Un "régime stable" qui se produit si la polarisation P est suffisamment négative (prédominance des stratégies contrariantes).

Les 3 phases





La démarcation Périodique/Intermittent est plus visible que celle Périodique/Stable

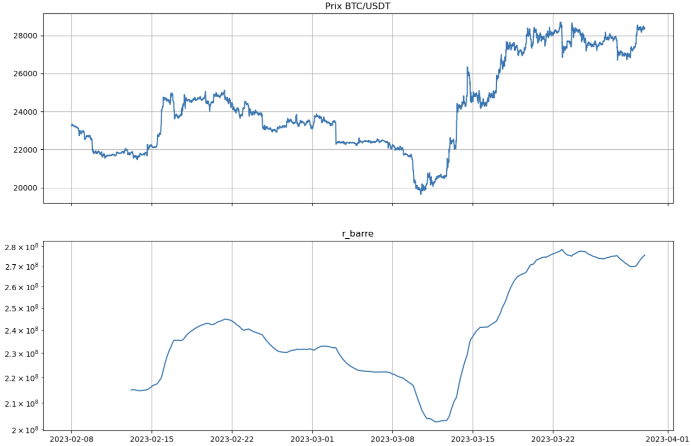
Le régime périodique

1. La durée des périodes est inversement proportionnelle à $\frac{g}{\lambda}$
- La durée des périodes est croissante avec λ
- La durée des périodes est décroissante avec g
2. Ce sont les fondamentalistes qui commencent
- Les bulles
- Les "anti-bulles"
3. Ce sont ceux avec des stratégies gagnantes qui :
- Renforcent les bulles
- Renforcent les "anti-bulles"

Le régime périodique

4. Indication de la présence de bulle :
- $r < \rho$: début de formation de la bulle, le prix augmente mais reste inférieur à sa valeur fondamentale.
- $r > \rho$: saturation de la bulle, les fondamentalistes agissent à contre courant, r se rapproche de ρ.
- $r > \rho$: fin du krach, les fondamentalistes recommencent à acheter.

Le régime périodique



L'efficience du marché

- Modèle avec beaucoup de paramètres, mais il y a deux paramètres qui sont vraiment importants : $\frac{g}{\lambda}$ et P.

-Pour des faibles valeurs de $\frac{g}{\lambda}$: obtention de bulles très longues, les agents vont augmenter leur investissement car le risque est faible $\rightarrow$ augmentation de g.
En même temps, le taux d'exécution diminue $\rightarrow$ évolution du marché pour rendre plus petit, ainsi le marché deviendra plus liquide.
-Au bout d'un certain temps $\frac{g}{\lambda}$ cesse de croître car le marché est devenu très volatil, les agents ralentissent leurs investissements.

L'efficience du marché

-Si les agents sont peu contrariants : très peu de fluctuations, peu d'écart entre le prix et sa valeur fondamentale.
-La psychologie humaine montre que les agents ont généralement une stratégie mimétique, c'est-à-dire qu'ils suivent la tendance. C'est cela qui maintient le marché réel dans le régime intermittent.

Conclusion

Conclusion

L'article utilise une simulation fondée sur le modèle **Minority Game** et **Santa Fe artificial market**.

Le modèle prend en compte beaucoup de paramètres, mais seul la variable P (polarisation) et le rapport $\frac{g}{\lambda}$ sont analysés dans cet article.

Cela amène **3 régimes** possibles :

- Périodique
- Intermittent
- Stable

Ce qui est critiquable sur la simulation est le fait que tous les agents ont un horizon de temps égal et synchronisé.

La simplification bond et stock est également critiquable.

Références

Inductive reasoning and bounded rationality : the el farol problem.

On the minority game : analytical and numerical studies.

Irene Giardina, J.-P. B. (2018).
Bubbles, crashes and intermittency in agent based market models.
pages 1–39.

Scipy.
Loi de pareto, python.

Wikipedia.
Loi de pareto.

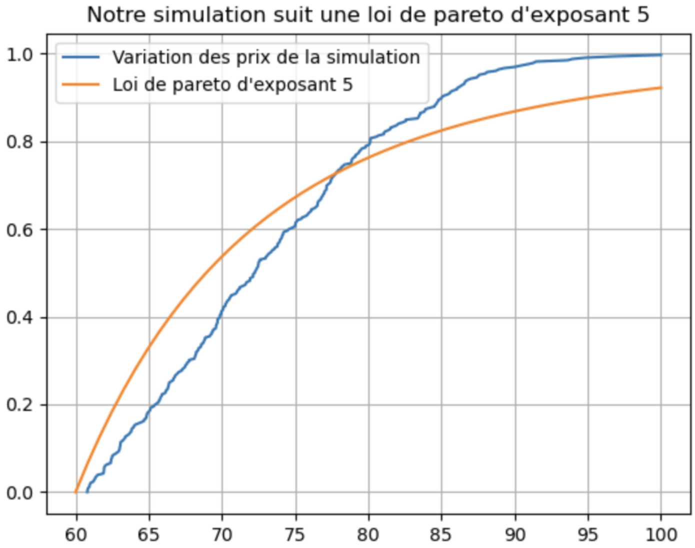
Wikipedia.
Système multi-agents.

Wikipedia.
Volatilité.

Yfinance.
Yfinance, python.

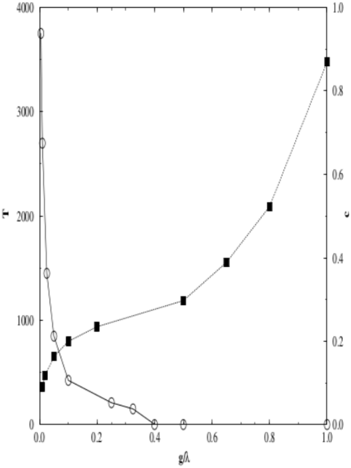
Annexes

Variation des prix de la simulation

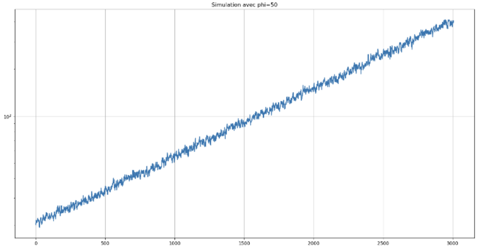


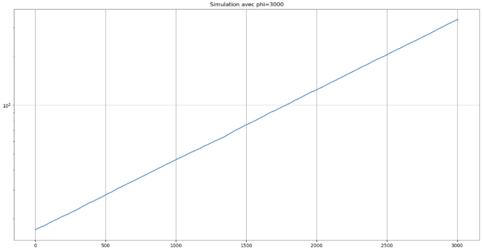
La période est inversement proportionnelle à $\frac{g}{\lambda}$

Période T des oscillations (échelle de gauche), fraction c des commandes satisfaites (échelle de droite) comme fonction de $\frac{g}{\lambda}$.



Impacts de la quantité d'actions ou "stocks" (ϕ)





Code

```python
#### IMPORTATIONS #####
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

```python
np.random.seed(seed=2)
class simulation:
    """
    Cette classe permet de creer la simulation decrite dans l'article :
    BUBBLES, CRASHES AND INTERMITTENCY IN AGENT BASED MARKET MODELS
    de Irene Giardina et Jean-Philippe Bouchaud
     d crit en 2018
    """
    # Nous declarons ci-dessous les constantes de la classe
    S = 3 # Le nombre de strategie (sachant que 1 strategie est l'inactivite)
    m = 5 # La quantite de memoire des agents
    g = 0.005 # La fraction d'action achete ou vendu
    f = 0.05 # Le niveau de confiance
    N = 1_001 # Le nombre d'agent
    beta = 1-1e-2 # La memoire du score
```

```
    def __init__(self, P, g_sur_l=0.1, alpha=1-1e-4, rho=0.02, phi=3_003, pi=0):
        self.l = self.g / g_sur_l # Lambda
        self.P = P #Polarisation
        self.alpha = alpha # 0.04
        self.phi = phi # Le nombre total d'action en circulation (Quelle valeur faut-il mettre ?)
        self.rho = rho # 0.000054255 # 2% d'int r t (Quelle valeur faut il mettre ? Regarder la croissance de la courbe Figure 1 ?)
        self.pi = pi

        self.epsilon = [
            [self.creation_strategie(self.m) for _ in range(self.S-1)] + [lambda x: 0]
            for _ in range(self.N)
        ] # Les S strat gies pour les N agents
        self.theta = np.random.random(self.N) # Le nombre de stock
        self.B = np.random.random(self.N) # Le nombre de bond
        self.X = (np.random.random(self.m+2)/100_000 + 5).tolist() # Les prix initialis  de fa on al atoire
```

```
        self.score = [
            [0 for _ in range(self.S)]
            for _ in range(self.N)
        ] # Les scores    t
```

```
    def creation_strategie(self, m):
        # Il y a 2^m I_t (information) possible
        # L'information est une suite de m element 1 ou -1
        #Nous allons donc representer I_t est donc une suite de
 binaire 1:1 et 0:-1
        # Pour simplifier la manipulation de I_t nous allons
prendre sa representation decimale
        # I_t est donc    valeure dans [0, 2^m-1]

        # Choisissons les possibilites acheteuses, d'apres la
variable self.P
        # P=1 tendance    Trend Following
        # P=-1 tendance    Mean Reverting
        liste_acheter = []
        for info in range(2**m):
            nb_positif = (bin(info)[2:]).count("1")
            nb_negatif = self.m - nb_positif
            M = (nb_positif-nb_negatif)/self.m
            if np.random.random() < (1+self.P*M)/2:
                liste_acheter.append(info)

        # Creons la fonction epsilon_i(I_t)
```

```python
        def epsilon_i(I_t):
            if I_t in liste_acheter:
                return 1
            else:
                return -1
        return epsilon_i
```

```
    def information(self, t=0):
        I_t = 0
        for i in range(self.m):
            if (np.log(self.X[-1-i+t]/self.X[-2-i+t]) - self.rho) > 0:
                I_t += 2**i
        return I_t
```

```
    def calcul_score(self):
        info = self.information(t=-1)
        r = np.log(self.X[-1]/self.X[-2])
        for i in range(self.N): # Calcul des scores de l'agent i
            for alpha in range(self.S-1): # Calcul des scores de la strat gies alpha
                arg1 = (1-self.beta)*self.score[i][alpha]
                arg2 = self.beta*self.epsilon[i][alpha](info)*(r-self.rho)
                self.score[i][alpha] = arg1 + arg2
```

```python
    def r_barre(self):
        r_barre = 0
        t = len(self.X)
        for t_ in range(t-1):
            r_barre += (self.alpha**(t-t_-1))*np.log(self.X[t_+1]/self.X[t_])
        r_barre = r_barre / (1-self.alpha)
        pf = min(
            1,
            self.f * abs(r_barre-self.rho) / self.rho
        )
        return r_barre, pf
```

```
    def simulation(self):
        # Calcul du score
        self.calcul_score()
        alpha_star = np.argmax(self.score, axis=1) # Les meilleurs strat gies
        info = self.information()
        r_barre, pf = self.r_barre()
        choix = []
        # Calcul du choix des agents
        for i in range(self.N):
            random = np.random.random()
            if random < self.pi: # L'agent fait un truc random
                choix.append(np.random.randint(3)-1)
            elif random < (1-self.pi)*pf: # Probas fondamentaliste
                if r_barre > self.rho:
                    choix.append(-1)
                else:
                    choix.append(1)
            else:
                action = self.epsilon[i][alpha_star[i]](info)
                choix.append(action)
```

```
        # Calcul des quantites par agents
        quantite = []
        for i in range(self.N):
            if choix[i] == 1:
                quantite.append(
                    self.g*self.B[i]/self.X[-1]
                )
            elif choix[i] == 0:
                quantite.append(0)
            else:
                quantite.append(
                    -self.g*self.theta[i]
                )
        # Calcul des quantites globale
        q_plus = 0
        q_moins = 0
        for q in quantite:
            if q > 0:
                q_plus += q
            elif q < 0:
                q_moins -= q
        q_plus = q_plus/self.phi
        q_moins = q_moins/self.phi
```

```
        q = q_plus-q_moins

        self.X.append((q/self.l + 1)*self.X[-1]) # Mise    jour du prix
        # Calcul des volont s d' change
        q_tilde = q_plus * self.X[-2] / self.X[-1]
        if q_moins < q_tilde:
            phi_plus = q_moins/q_tilde
        else:
            phi_plus = 1
        if q_tilde < q_moins:
            phi_moins = q_tilde/q_moins
        else:
            phi_moins = 1
        # phi_plus = min(1, q_moins/q_tilde)
        # phi_moins = min(1, q_tilde/q_moins)
        # Calcul de l' change   r el
        delta_theta = []
        for q in quantite:
            if q > 0:
                delta_theta.append(q*phi_plus)
            elif q < 0:
                delta_theta.append(q*phi_moins)
```

```
            else:
                delta_theta.append(0)

        for i in range(self.N):
            self.theta[i] = self.theta[i] + delta_theta[i]
            self.B[i] = self.B[i]*(1+self.rho) - delta_theta[i]
    *self.X[-1]
```

```python
if __name__ == "__main__":
    liste_simulation = [
        ({"P":0, "phi":50, "alpha":1e-2, "rho":1e-3, "pi":1}, "Fully random"),
        ({"P":0, "g_sur_l":0.1, "phi":50, "alpha":1e-2, "rho":1e-3, "pi":0}, "P riodique"),
        ({"P":0, "g_sur_l":0.6, "phi":50, "alpha":1e-2, "rho":1e-3, "pi":0}, "Intermittent"),
        ({"P":-0.2, "g_sur_l":0.6, "phi":50, "alpha":1e-2, "rho":1e-3, "pi":0}, "Stable"),
    ]
    for kwargs, title in liste_simulation:
        sim = simulation(**kwargs)
        plt.figure(figsize=(18, 9))
        for _ in range(6000):
            sim.simulation()
        plt.plot(range(3_000, 3_000 + len(sim.X[3000:])), sim.X[3000:])
        plt.yscale("log")
        if title == "Fully random":
            plt.title(f"{title} pi = {kwargs[’pi’]}")
        else:
```

```
        plt.title(f"{title} : " + r"\frac{g}{\lambda}" + f" = {kwargs['g_sur_l']} et P = {kwargs['P']}")
    plt.grid()
    plt.savefig(f"simulation-log-{title}.png")
```