# 人工市場 U-Mart における
# Random 型マシンエージェントの介入による
# 価格の安定と平準化

科目名：演習Ⅱ
指導教官：有賀裕二
学籍番号：10C2127009L
氏名：松岡　賢

# 内容

## 用語解説と注意

- 「ut」
  U-Mart 時間（U-martTime） 。U-Mart における取引時間の最小単位。

- 「○○型エージェント」
  本文中での売買の判断や注文価格および数量の決定方法が基本的に同じで、参照する価格が異なるエージェントを便宜上まとめてこう呼ぶ。
  あくまで便宜上のものであるため注意。
  （例）RandomStrategy、SRandomStrategy、MarketRandomStrategy
  →Random 型エージェント

- Random、Trend など途中で切れたマシンエージェント名
  文章の長さなどの関係で、一度使ったマシンエージェント名は Strategy を省略することがある。

# 序論

2012 年 11 月 25 日に行われた中央大学と近畿大学の合同 U-Mart ゼミナールにおいて、私は普段のゼミで学んだ平準化というものに興味を持ち、「Random 型エージェントの持つ平準化機能について」というテーマで研究、発表を行った。

この時は時間的な余裕の無さなどから思うように研究が進められず、実験の不備や分析の不足なども見られた。

そのため今回も引き続き平準化について、U-Mart 上において平準化がどのように作用しているのか、Random 型エージェントを中心としたマシンエージェント構成の変化がどう影響するのかなどに注目して実験と分析を行った。

## 平準化機能

平準化とは正確には価格平準化作用といい、以下のようなものであるとされる。

> 取引所機能の一つ。価格が需給を調整するとともに、また先物取引においても他市場に比べて割高（安）な市場には売り（買い）が入って地域間の価格がならされる働き（地域的平準化）があり、そしてまた先物取引には、将来の価格変動のもととなる諸材料が前もって徐々に織り込まれて価格が形成されるので大変動を緩衝する働き（時間的平準化）もある。

要するに、先物取引において価格が大きく上がれば高く売ろうとする人が増え価格は下がっていく。また大きく下がれば安く買おうとする人が増え、価格は上がっていく。こうして価格の大きな変動が抑えられ安定することが平準化である。

合同ゼミでは Random 型エージェント 3 種の参加による平準化に特に注目した。これは Random エージェントが売り買いを無作為に行うため、注文価格が市場価格とかけ離れる可能性が低く、大きな利益も損失も出さず安定する。理論上取引期間が長くなるほど売り買いがバラけるため安定度も上がり、長期的に市場平準化に貢献できると予測したためである。

なお、この「取引が進むほど損も得も大きくならず安定する。」という機能についても「Random 型エージェント自身への平準化機能」と呼んで調査した。

# 実験準備

### 昨年の実験結果

昨年は前に挙げた 2 種類の平準化機能について、Random 型エージェント 3 種類を対象にその有無を調べた。
その内容と結果を簡単にまとめておく。なお結果は去年の時点での分析内容である。

① 自身への平準化機能について
Random 型エージェント以外のエージェントを一通り参加させ、そこに RandomStrategy、SRandomStrategy、MarketRandomStrategy を一種類ずつ、計 3 パターンのエージェント構成で取引を行わせ、これら Random 型エージェント 3 種のそれぞれの損益の推移をグラフ化して比較した。
エージェントの参加比率はランダム型：その他＝１（３体）：６（９種類×２体）とした。

結果：MarketRandomStrategy 以外は損益が比較的安定しており、自身への平準化機能が認められた。MarketRandomStrategy は成り行き注文のため他 2 種類と違い価格決定の基準になる価格がなく、他のエージェントの注文に左右されてしまったと思われる。

② 市場平準化機能について
TrendStrategy のみ参加している市場に RandomStrategy、SRandomStrategy、MarketRandomStrategy を一種類ずつ参加させ、市場価格の推移を見た。そのままでは一方的に価格が上昇してしまう状態で、Random 型エージェントがどう影響するかを調べた。
エージェントの参加比率はランダム型：TrendStrategy＝５：３とした。

結果：RandomStrategy 以外は導入により先物価格と現物価格の乖離が正常な状態に近づいた。RandomStrategy は暴騰した先物を価格決定の基準にしてしまったため、かえって価格の乖離を促進してしまう結果となったと思われる。

### 分析のミス

先ほどの実験②の分析は間違っており、実際には先物上昇時に Trend が買い注文を出し続け、Random がそれを受け入れることで更に先物が暴騰していた。つまり Trend と Random という組み合わせが原因だった。

このように、エージェントの組み合わせの影響を考慮していないという大きなミスがあった。SRandom や MarketRandom についても、他のエージェントと組ませた場合結果が変わる可能性がある。

そのため今回は Random に限らず、複数のエージェントを組み合わせての調査を中心に行った。

## 実験の目的

先ほど「価格の大きな変動が抑えられ安定することが平準化である」と説明した。
今回の実験ではこの平準化を

- 修正･･･大きな価格差が解消されるか
- 安定･･･価格の上下移動が抑えられ安定しているか

に分けて主に分析したいと思う。

先ほど説明したように、価格平準化がなされれば価格は安定する。しかし今回の実験は人間による取引が一切絡まない。このためニュースなどのデータにない外的要因に影響されたり、価格の大きな変化があったためにそれまでと全く異なる注文の仕方に変えたりといったことは起こらない。
つまりイレギュラーな価格の加速が起こりくいため、一時的に価格が荒れたとしても、人間による取引に比べその後は安定しやすい。

そのため「乖離が起きていない平時の価格変動」についても注意をすることにする。仮に特定のエージェントを追加したことで価格の乖離が解消されやすくなったとしても、平時の価格が激しく上下していては安定しているとはいえないからである。

また合同ゼミでの価格平準化についての実験が、マシンエージェントが 2 種類しか参加しないものであったこと、そのため結果に狂いが生じた可能性があることは前に述べた。

その反省を踏まえ、加えて取引主体が多様であるほうが実際の市場や人間が参加するネットワーク実験に少しでも近くなるため、今回の実験では、複数種類のエージェントが参加する状態について調べる。

## 実験用エージェント紹介

今回の実験では一時的な価格の乖離を再現する。

そのために、TrendStrategy を改変して一時的な価格の乖離を起こすためのエージェントを製作した。2 体で一組のエージェントとなっている。

① JumpStrategy_A

注文方法　ポジションを保持していない場合、
または売りポジションの場合に指値買い注文を 700 個出す
買いポジションを保持している場合、売り注文を出す

注文数量　買いの時　700
売りの時　1※

注文価格　買いの時　TrendStrategy に同じ
売りの時　成行

※本来は「買いポジションを保持していたら取引しない」だったが、
一度だけ全エージェントが取引停止してしまったことがあり、
ここを売りに変えたところ正常になったため残してある。

② JumpStrategy_B

注文方法　成行売りのみ
ただし売りポジションの総数が 2500 を超えたら取引しない

注文数量　TrendStrategy と同じ

注文価格　成行

一時的な市場の荒れを作るためのエージェントとなっている。

JumpStrategy_A が大量の買い注文を出し、JumpStrategy_B がそれを受け入れる事で一気に先物価格を引き上げる。

MarketRandomStrategy は参加させないため、成行優先原則に則り Jump_B が優先的にこの買い注文を受け入れる。その後 Jump_B は取引を停止するため市場に干渉しなくなり、Jump_A は売り注文を出すものの数が少ない上に他にも売りを出すエージェントが複数参加しているため、市場に与える影響は割合的には少ない。

なお、①と②を合わせて「Jump エージェント」と呼ぶこととする。

**現物基準と先物基準**

U-Mart のマシンエージェントは注文決定の基準にどの価格を用いるかで

A)　先物価格のみを基準に用いるマシンエージェント

B)　現物価格のみを基準に用いるマシンエージェント

の２種類に大別でき、さらに先物と現物の両方を利用する SFSpreadStrategy と、成行注文に特化した MarketRandomStrategy が加わる。

今回の実験では合同ゼミでの研究の延長でもあるため、A、B の中でも RandomStrategy と SRandomStrategy の調査を中心とする。その他のエージェントは比較や、あるいは必要に応じて個別に調査する。

※MarketRandomStrategy は追加エージェントとの兼ね合いから除外。

# 実験とまとめ

### 実験の基本設定

以下の設定を条件１とし、基本の設定とする。
ここから実験ごとに条件を変えていく。

- エージェント構成

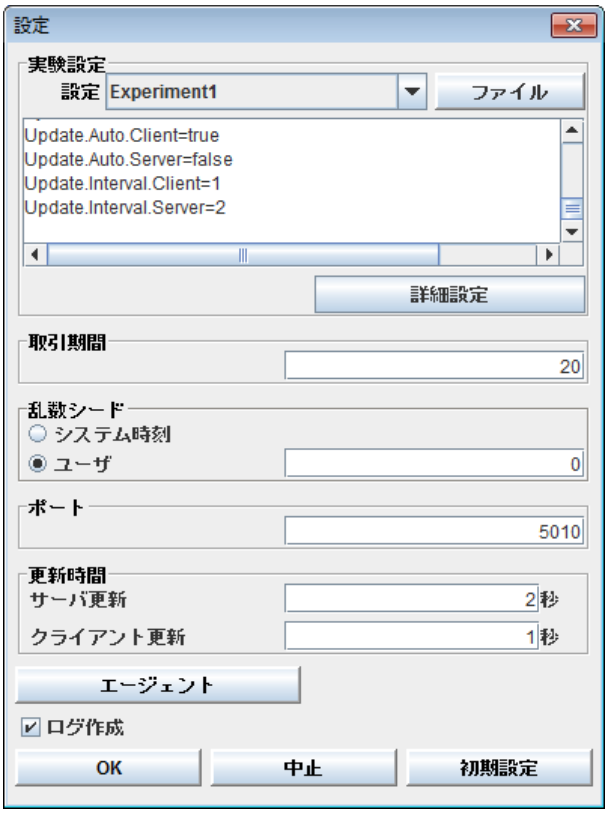
| | | |
|---|---|---|
| 1. | TrendStrategy | 1 体 |
| 2. | AntiTrendStrategy | 3 体 |
| 3. | RandomStrategy | 1 体 |
| 4. | SRandomStrategy | 3 体 |
| 5. | RsiStrategy | 1 体 |
| 6. | SRsiStrategy | 3 体 |
| 7. | MovingAverageStrategy | 1 体 |
| 8. | SMovingAverageStrategy | 3 体 |
| 9. | SFSpreadStrategy | 2 体 |
| 10. | DayTradeStrategy | 2 体 |
| 11. | JumpStrategy_A | 1 体 |
| 12. | JumpStrategy_B | 1 体 |

先物参照型：現物参照型が１：３で、そこに SFSpread と DayTrade を加え、更に Jump エージェントを足したエージェント構成だと考えるといい。

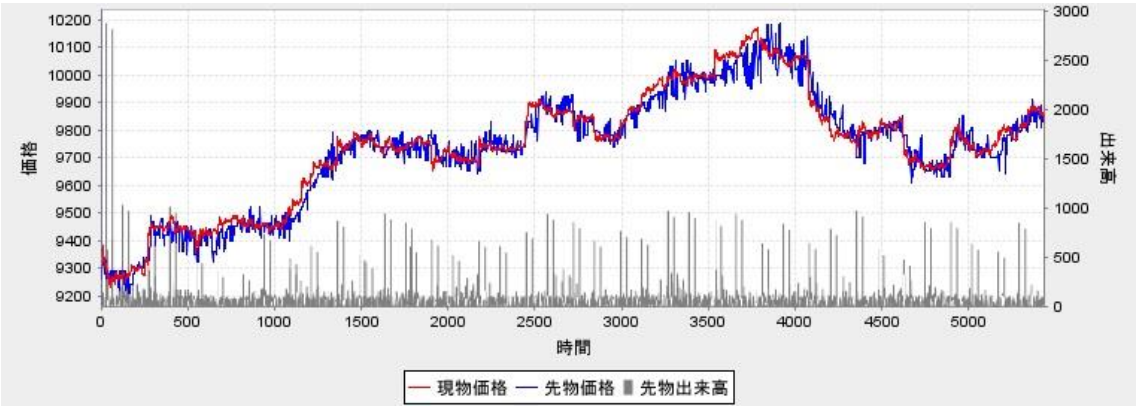
- 初期所持金額
  基本的に¥$10\times10^{8}$
  JumpStrategy_A および JumpStrategy_B は¥$10\times10^{11}$
  （取引が極端で非常に破産しやすいため）

- 取引期間　20 日

- その他の設定は全てデフォルト（以下の通り）




参考として、条件 1 から Jump エージェントを除いた状態でのグラフ（異常な価格変動のない状態のグラフ）は以下のようになる。



## RandomStrategy の調査

RandomStrategy は先物を価格決定の基準として、売り買いをランダムに注文を出す。破産しない限りどんな条件下でも売り、買いどちらも行う可能性があり、そのため他のエージェントの注文をスムーズに約定させるのに使える。

先ほども述べたが、先の合同ゼミでの実験では、暴騰した先物を基準として売買を繰り返すために価格の乖離を推し進めてしまったと結論付けた。

しかし逆に言えば、RandomStrategy はどれほど価格が乖離しても安定して取引ができるエージェントであるということでもある。また同じ RandomStrategy 間でも売り買いがバラけるため、Random 単独でも市場を安定させられる可能性がある。

そのため今回は RandomStrategy が市場平準化機能を発揮できるエージェントの組み合わせを調査する。

**実験方法**

1. 条件 1 に RandomStrategy と SFSpreadStrategy を 3 体になるように加える
2. 条件 1 に RandomStrategy と RsiStrategy を 3 体になるように加える

以上 2 つの条件で実験を行う。また 1.と 2.について Random を 6 体にした場合、および RandomStrategy を 1 体のまま追加せずに実験を行った場合とも比較を行う。

SFSpreadStrategy は先物が現物を上回る傾向にあれば売り、下回る傾向にあれば買う。

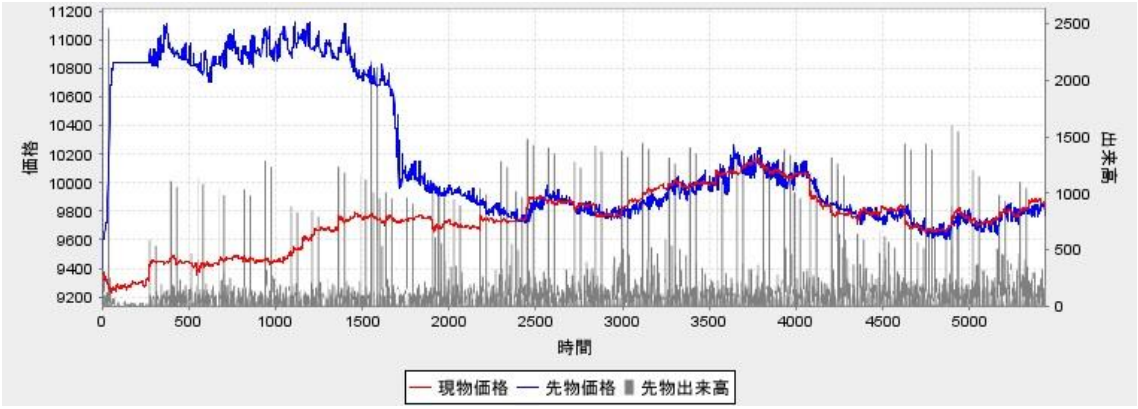
RsiStrategy は Rsi 値（一定期間内で先物が直前の ut よりも上がった割合）を算出し、一定以上なら売りを出す。また一定以下であれば買いを出す。デフォルトでは 10ut ごとに Rsi 値を測定し、0.7 以上なら売り、0.3 以下なら買う。

どちらも先物価格を価格決定の基準として用いており（ただし SFSpread は先物と現物の平均を基準とする）、現在の先物に反発するように注文をする。このため Random エージェントによって活発に取引がなされることで、価格の乖離の解消につながると予想する。

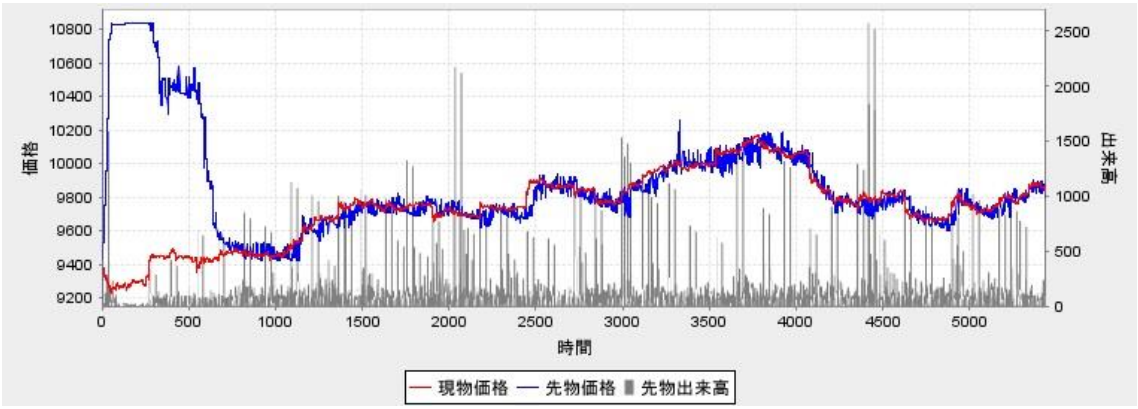
結果

① **Random＋Rsi の場合**

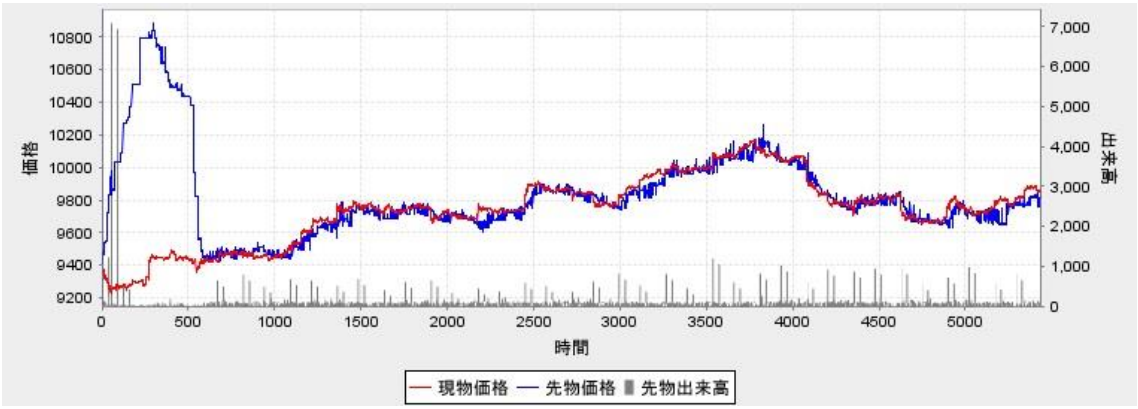
A1（Random6 体、Rsi3 体）




A2（Random3 体、Rsi3 体）




A3（Random1 体、Rsi3 体）



それぞれ A1 は最初に提示した Random6 体という条件でのグラフ、A3 は Random が多すぎて、Random 同士での取引が過熱したことで乖離の修正に時間がかかってしまった判断し、Random を 3 体に減らした場合のグラフ、A3 は Random が 1 体（追加しない場合）のグラフである。

先物と現物の差が正常になるまで最も早いのが A3、遅いのが A1 となった。もう書いてしまったが、価格が上昇した先で Random 同士での取引が過熱するようになる。Random は売り買いをランダムに決めるために注文が偏らず、小刻みに先物が上下するようになったことで戻りにくくなったと思われる。

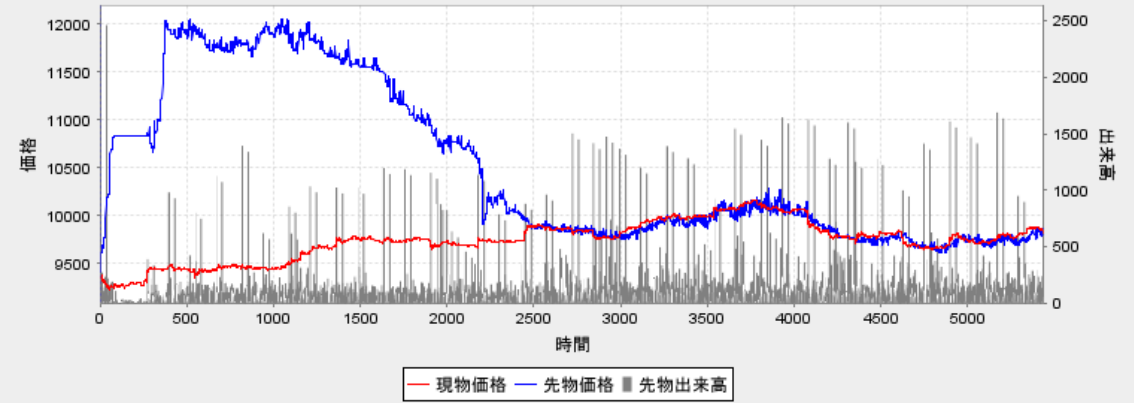
一方で先物の暴騰時の最高値は A1 が最も大きい。これは価格上昇によって Trend が出した大量の買い注文を RandomStrategy が引き受けてしまうためである。常に先物価格を元に売りも買いも出せるため、TrendStrategy などの異常な注文には弱い。これは合同ゼミでも実証された結果である。

また先物の下降スピード自体はどの状態でも非常に早く、現実の市場ならそのまま現物を下回って大きな谷を形成してもおかしくはない。

よって「Random の参加によって乖離が解消された」ということは無く、場合によってはむしろ先物の暴走を促してしまう。なので、当初想定していた Random による平準化機能は確認できたとはいえない。

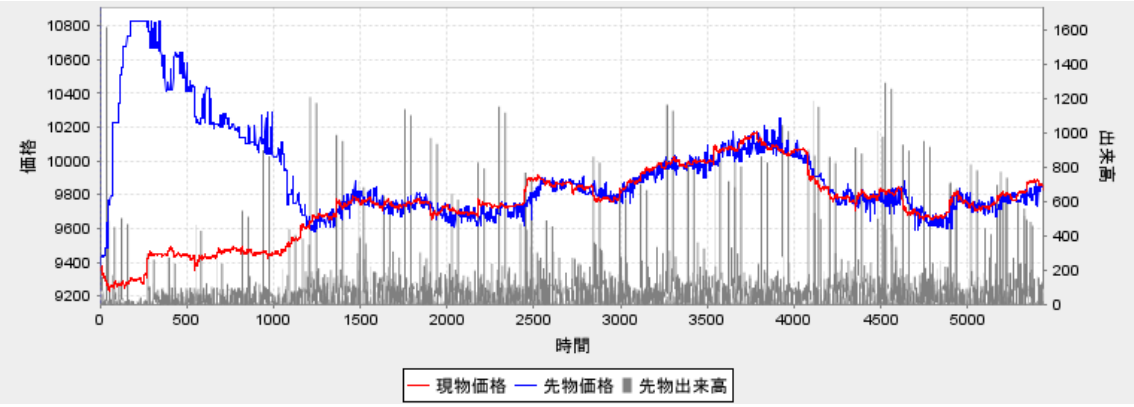
② **Random＋SFSpread の場合**

B1（Random6 体、SFSpread3 体）

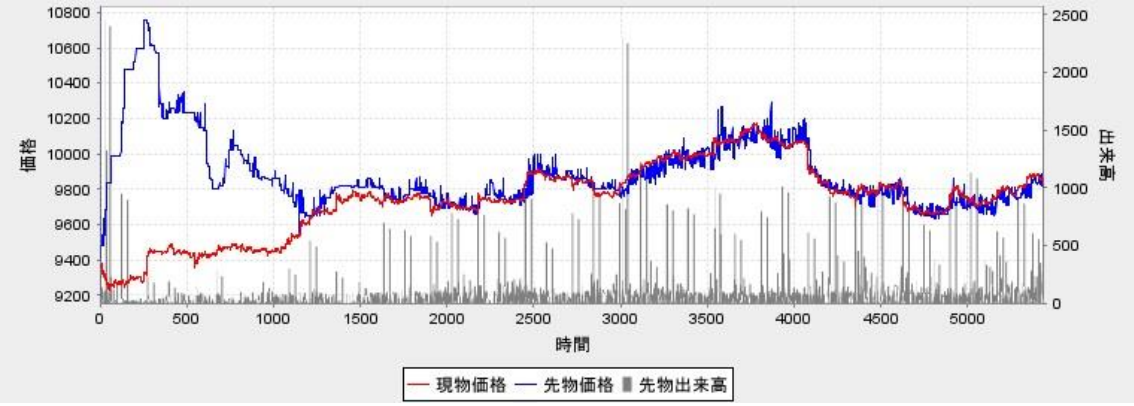



B2（Random2 体、SFSpread2 体）

※3 体ずつだとおかしな動きをしたので 2 体ずつに変更




B3（Random1 体、SFSpread3 体）




SFSpread の場合、Random6 体追加時には回復までが長引く代わりに下降が緩やかになるという点では Rsi と同じであった。ただし B2 と B3 を比べても先物の下降速度にあまり変化は無かった。

その代わり下降中のグラフを比べると、B3 にあった一時的な急落や急上昇が無くなり、平均化された印象である。また A1 や A2 のグラフや B1 や B2 のグラフを比べた場合も、B のグラフの方がグラフの傾斜がなだらかになっている。

これは SFSpread が先物>現物なら常に売り注文を出すため、安定して売り注文が約定を続けるためと推測される

このように Random と他のエージェントを組み合わせることで、「Random が参加することで乖離の解消までの時間が延びる」反面、「値動きが緩やかになり市場が急激に荒れにくくする」事ができ、乖離の解消と市場の安定化を両立できる＝市場平準化機能を有すると言える。

### まとめ

RandomStrategy の数がある程度増えると RandomStrategy 同士の取引が活発になり、売り買いのバランスよく注文が約定するため、急な価格変動が起きにくくなる。

ただし RandomStrategy には他のエージェントの影響を受けやすい側面があり、場合によっては RandomStrategy がいることで価格の暴走を推し進めてしまう。このため RandomStrategy 単体で市場平準化機能に期待するのは難しい。

ただし価格変動を完全に停止させることは難しく、乖離の解消に効果のあるエージェントと合わせることで、緩やかに乖離を解消でき、かつ急激な価格変動が起きにくく安定しやすい先物の状態を作ることができる。特に SFSpreadStrategy と合わせると、先物は滑らかなグラフを描く。

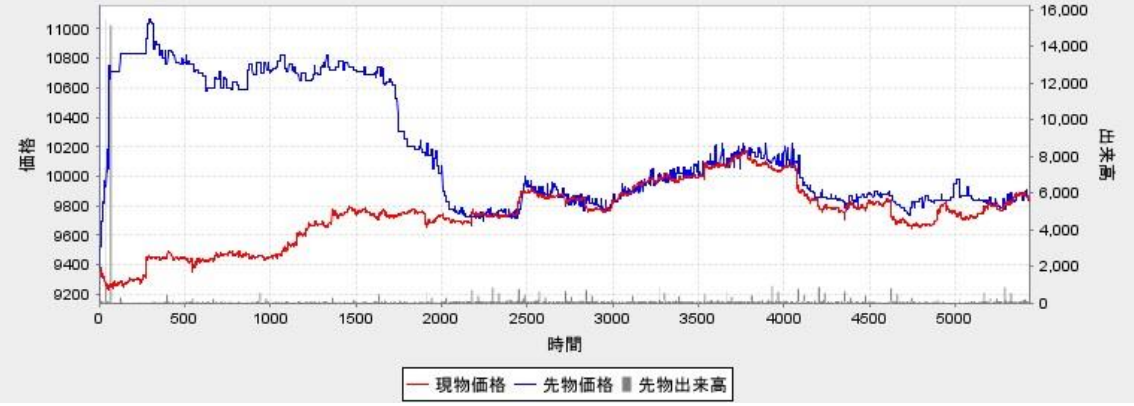
この場合、市場平準化機能が認められると言っていい。

順張り型エージェントの影響

① TrendStrategy の場合

以下、実験中に偶然気づいた内容になる。
以下に３種類のグラフを載せる

C1




C１のグラフは条件１の設定から Jump_B を間違えて削除した状態で実験を行い、結果早い段階で TrendStrategy が破産してしまった時のグラフである。

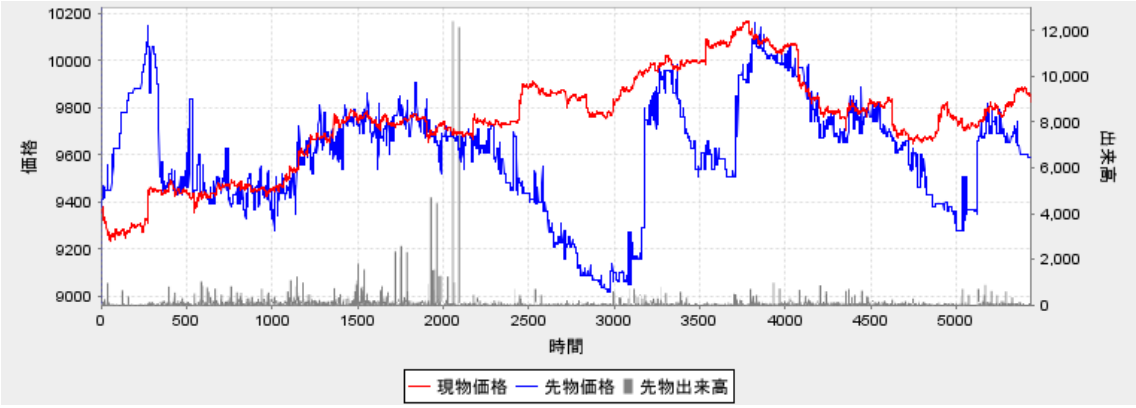
基本的に価格の上下の幅は少ないが、最初に先物価格が急騰してから落ち着くまで時間がかかっているが、かわりに上昇時と比べると緩やかに下がっている。

C2


C2 のグラフは上記 C1 の実験と同じ条件のまま TrendStrategy の初期所持金額を¥10×$10^{11}$ に増やし、TrendStaregy を破産させないようにした状態でのグラフである。

C1 のグラフに比べて急騰した価格が急激に下落し、更に平時の先物価格の値動きもいくらか激しくなっている。

C3




C3 のグラフは同じく C1 の構成から TrendStrategy を 3 体に増やし、内 1 体が最後まで破産しなかった時のグラフである。

最初の先物急騰からの回復は C2 と同様に急激である。さらにその後の平時の先物価格変動幅が大きく、特に ut2000 前後で Trend2 体が破産する前は、B のグラフと比べてもさらに大きくなっている。

また後半の先物の荒れは急落→急上昇という流れの繰り返しだが、特に急上昇時は最初の急落と同様、非常に急激に価格が上がっている。グラフで見てもほぼ垂直になっている。

このように、TrendStrategy がいると価格移動が明らかに早くなり、また平時の価格も乱れやすくなっていた。

Trend は順張り型であり、かつ直近と直近より 1ut 前というごく短期かつ近い時間の価格の変化を見ている。そのため何度か続けて同じ方向に価格が動いただけで同種の注文が大量に出て値動きが激しくなり、また一瞬の価格の上下でも売り注文を出すために平時の価格が乱高下したと考えられる。

よって価格平準化という面から見ると、TrendStrategy は平準化を阻害するエージェントであると言える。

**まとめ**

TrendStrategy の参加によって価格の乱高下を引き起こしやすくなるリスクは確実に大きくなっており、また平時でも TrendStrategy の数が多くなるほど平時の先物の上下幅は大きくなった。このことから TrendStrategy が持つ機能は、価格平準化とは真逆であると言っていい。

### ② MovingAverageStrategy の場合

前項までの実験で、TrendStrategy には平準化とは逆の、価格変動を激しくする機能が働くことがわかった。

その機能が順張りでの取引によるものという側面もあったため、同じく順張り型の MovingAverageStrategy についても調査を行う。

こちらは先物の短期移動平均と中期移動平均が重なった時に、短期移動平均が上向きなら買い、下向きなら売るエージェントである。

ただし順張りには変わりないが、取引数量は Trend に比べ大きく減る。そのため Trend ほど大きく市場は荒れないと思われる。

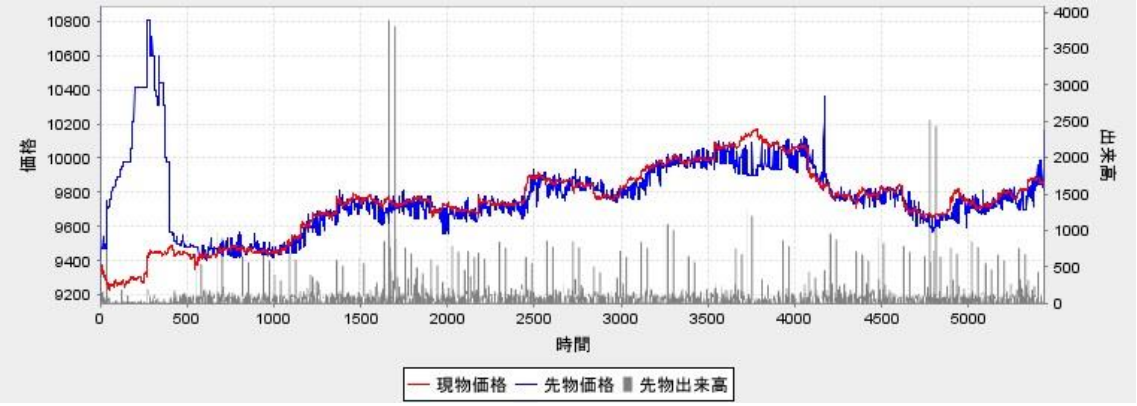
**実験方法**

条件 1 に MovingAverageStrategy を 2 体足し、3 体の MovingAverageStrategy を参加させて実験を行う。その後も更に MovingAverageStrategy を足して実験を行うということを繰り返す。

**実験結果**

参考：C2 のグラフ


D1（MovingAverageStrategy3 体）



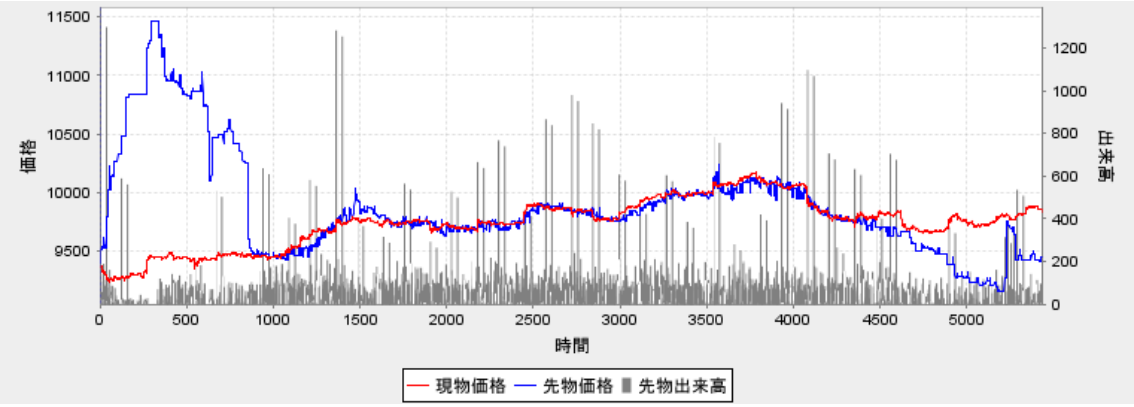

3 体追加の時点では追加前よりも価格も下落が早くなっており、平時の取引も活発で TrendStrategy に近いグラフになっていた。

ただしここからある程度 MovingAverageStrategy の参加数を増やしても、ある程度 TrendStrategy に近い形のグラフを作るものの、Trend のように価格の移動が激しくなることはなかった。以下は MovingAverageStrategy20 体の時のグラフ D2 である。

D2（MovingAverageStrategy20 体）


Trend3 体の時ほどでは無いが先物が何度か大きく上昇しては下がっており、通常より激しい値動きとなっている。これだけ見ると MovingAverage の追加によって価格の乱高下が起こったように見えるが、TrendStrategy を除外して実験すると以下のグラフのように乱高下がなくなり、これが TrendStrategy によるものとわかる。

D2’（MovingAverageStrategy20 体、Trend 除外）




こちらは終わり際で比較的大きな価格の下落とそこからの回復が起こっている。MovingAverageStrategy は Trend とは違い、一時的に先物が上昇しても短期平均が下降傾向にあると売り注文を出す。そのために売り注文があふれ、現物が下降から持ち直しても先物がそれについていかず乖離してしまったことが原因である。

ただし、TrendStrategy の様に平時の先物の乱高下を起こすことはなかった。

**まとめ**

MovingAverageStrategy にも、TrendStrategy と似たような機能がある。また TrendStrategy と比べ必要以上に市場を荒らさない。ただし TrendStrategy ほどその効果は強くはない。

## AntiTrendStrategy の調査

TrendStrategy が市場に与える影響が非常に大きなものであったため、逆張り版 TrendStrategy である AntiTrendStrategy についても調査してみる。

逆張りとは準張りとは逆に、価格が上昇傾向にあれば売り、下降傾向にあれば買いで投資すること。価格が底値や天井から反発することを見越しての取引手法で、AntiTrendStrategy、RsiStrategy などがこの方法で取引をする。

これらは言ってみれば現在のトレンドに反発するエージェントであり、一方的な価格の上昇や下落時に歯止めをかけ、価格の変動を緩やかにする効果を発揮する可能性がある。その効果の有無を確かめる。

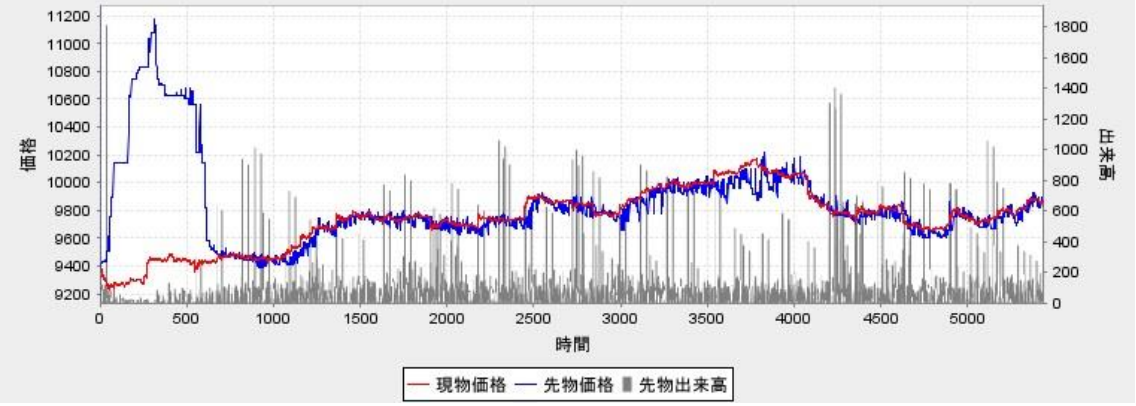
### 実験方法

条件 1 に AntiTrendStrategy を 1 体ずつ追加しては実験を行い、グラフを比較する。

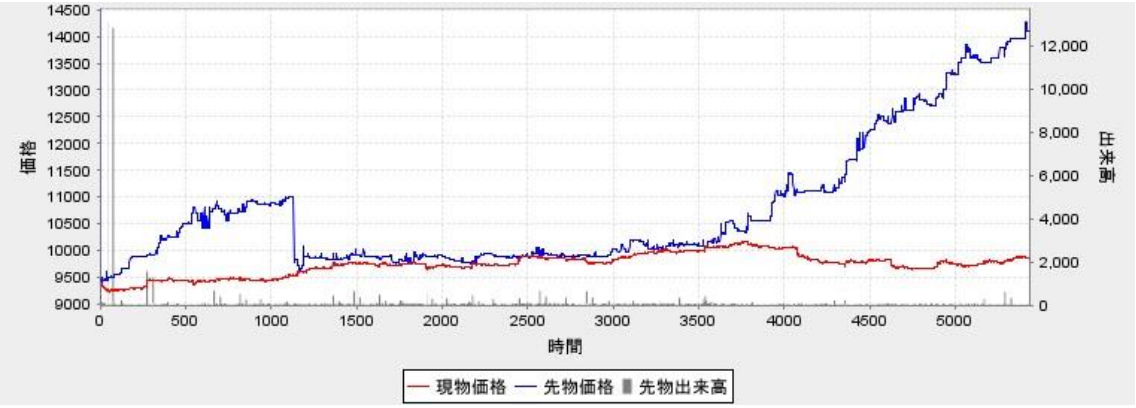
同時に TrendStrategy となんらかの関係性があるかどうかも調べるため、TrendStrategy の所持金額を¥$10\times10^{12}$ に増やして破産を防ぐ。

**結果**

E1（AntiTrend1 体追加）




E2(AntiTrend2 体追加)




E3（AntiTrend3 体追加）


AntTrend が増えるほど価格の上昇傾向が強くなり、また全体的に上昇、下降が共に緩やかになっている。

この理由として、以下の様に予測できる。

Trend と AntiTrend はどちらも先物価格を基準として、お互いに逆の注文を出している。そのためこの 2 種類のエージェントは注文価格やエージェント一体あたり注文数量が近く、そのため Trend と AntiTrend の間で約定する確立が高い。

すると数の多いエージェントが出した注文は必然的に余る。今回は AntiTrend が多いので、先物が上昇すれば売りが、下降すれば買いが余る。このため、

1. Jump エージェントによって先物が急上昇する
↓
2. それにより Trend が買い注文を、AntiTrend が売り注文を出す
AntiTrend の方が多いので売りが多く余る
↓
3. 先物が下がりだす
↓
4. AntiTrend が買いを出し、先ほどの余った売り注文との間で一部の買いが成立
今度は買いが余る
↓
5. 先物の下降がストップ、しばらくして上がりはじめる
↓
6. 再び AntiTrend が売りを出して先物の上昇を妨害し、
市場に再び売りが余る

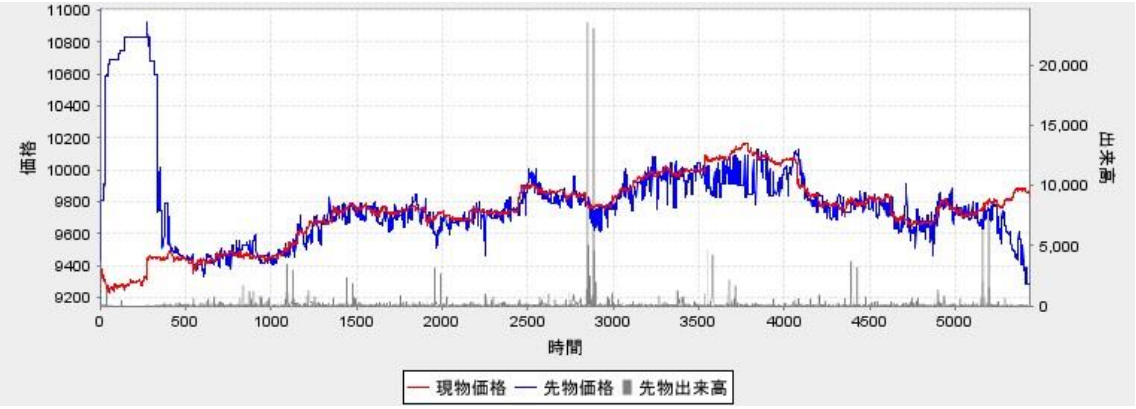
という循環が生まれ、先物が上がるにしろ下がるにしろじわじわと動くようになったと思われる。

また先物が下がっても先物と現物が近くなると、今度は現物を参照するエージェントによって余った買い注文が約定され、先物の下降にブレーキがかかる。このため先物と現物の間に一定以上の差が常にできてしまい、先物が下がりきらないままにまた上がってしまったと思われる。

AntiTrend が増えすぎなければ、この作用は先物の無駄な乱高下を抑えるという意味でプラスに働く。例えば C2 と E1 のグラフは AntiTrend の数以外に違いは無いが、平時の先物の触れ幅が F1 のほうがやや緩やかになっている。

また、試しに AntiTrend を 2 体追加時に Trend も 2 体足し、注文の余りが少なくなるようにした場合、グラフは以下のようになる。

E4（Trend および AntiTrend2 体追加）




参考：C2 のグラフ


グラフは Trend 追加時のものに近く、平時の先物が細かく上下し、変動幅が大きくなっている。どうやら先物が上昇する場合、Trend は AntiTrend よりもかなり影響が強いようだ。

**まとめ**

AntiTrendStrategy が増えていくと値動きが緩やかになり、また増えすぎると先物価格と現物価格の間に一定の差が生じる。このため急激な価格の乖離に対応できなくなる。

特に TrendStrategy が健在の市場で AntiTrendStrategy が増えすぎると、価格差を広げる力が増す一方で AntiTrendStrategy により急な先物価格の上下が抑制されてしまい、長期的な価格の乖離を引き起こす。

ただし適量での参加であれば、平時の価格の乱高下の幅を抑える働きがある。

## SRandomStrategy の調査

Random の実験で認められた平準化機能が、SRandom でも発揮されるかを確かめる。

先の合同ゼミでの研究では、SRandom は価格の修正に効果があると結論を出した。しかしその時の TrendStrategy と SRandomStrategy という組み合わせは、今考えれば問題である。

先物だけが暴騰した場合、先物を注文価格の基準にする Trend から見て、SRandom の注文価格はとても安くなる。そのため先物価格に比べて非常に安い売りが約定するようになり、価格が下がったという可能性もある。その確認も兼ねて実験する。

### 実験方法

1. 条件 1 に SRandomStrategy と SRsiStrategy が 6 体になるように足す
2. 条件 1 に SRandomStrategy と SFSpreadStrategy が 6 体になるように足す。

以上 2 つの条件で実験を行う。また 1.と 2.について SRandom を 12 体にした場合、および SRandom を 3 体のまま追加せずに実験を行った場合とも比較を行う。

SRsiStrategy は現物を基準とする RsiStrategy で、Random と Rsi の組み合わせが一定の平準化効果を発揮したため実験に採用した。

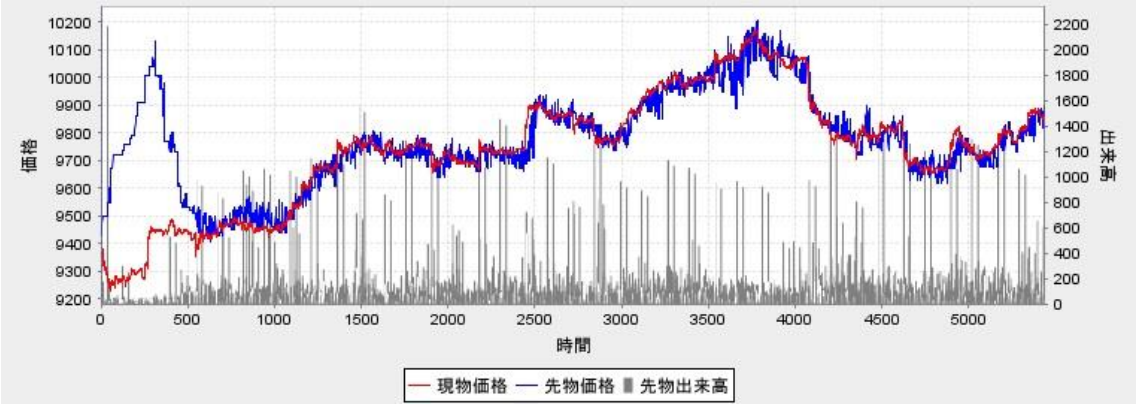
普通の Rsi にしなかったのは、価格乖離時に SRandom との注文価格が大幅に食い違う可能性があったため。

SFSpread は先物と現物の平均で価格を決めるため、Rsi ほど SRandom との相性は悪くないと判断し、Random の実験に引き続き採用した。

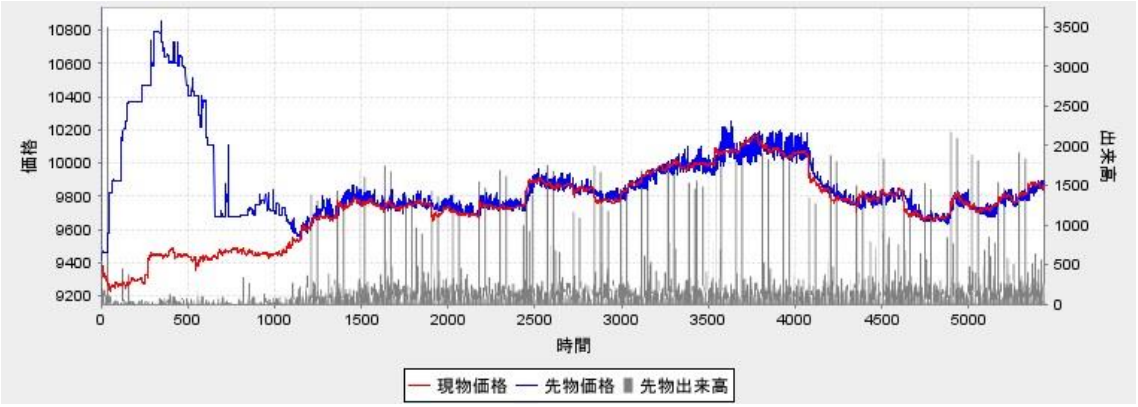
**結果**

以下、Rsi と SRandom を組み合わせた時のグラフである。

F1（SRandom6 体、SRsi6 体）

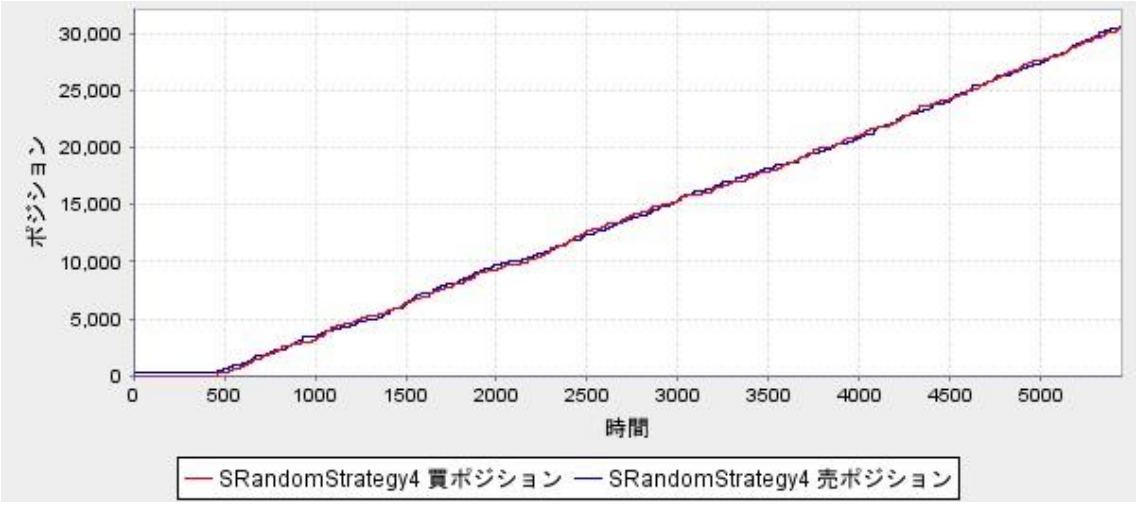



F2（SRandom12 体、SRsi6 体）

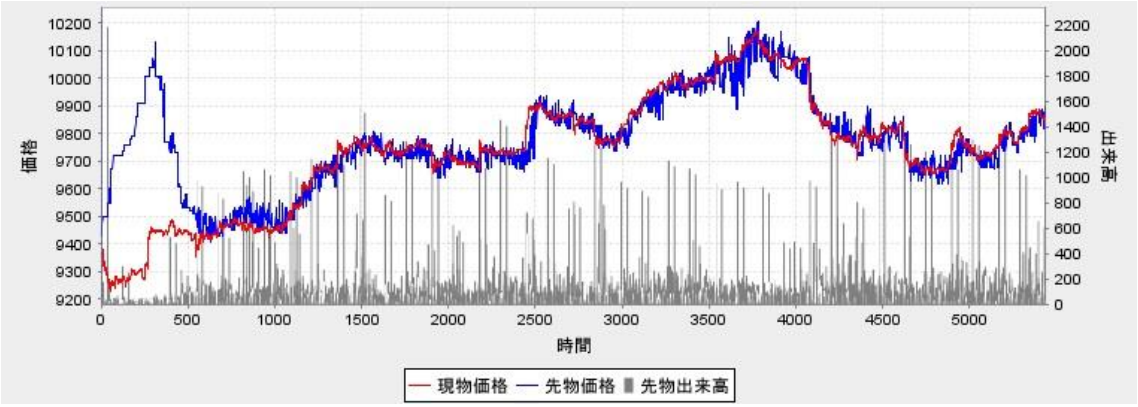



一見 SRandom が増えたことでやや乖離の解消までが早くなったように見えなくもないが、実際はそうではない。例として F1 の時の SRandom のポジションの推移を見ると、次のようになっている。

F3（F1 の実験での SRandomStrategy4 のポジションの推移）




参考：F1




グラフを見比べると、先物と価格が近づくまでほとんど取引が行われておらず、ポジションが変化していない。

つまり、乖離が解消されるまでろくに取引ができていなかったことになる。そのため価格の修正に役立ったとはいえない。これはやはり、先物を価格決定基準にするエージェントと注文価格の差がつき過ぎたことが原因と思われる。

もう少し詳しく言うと、先物価格が上がったことで先物価格のエージェントを注文価格決定の基準とするエージェントは、注文価格が大きく上がる。

この状態で SRandom が買い注文を出した場合、先物基準のエージェントからすると非常に安く売ることになり、条件が悪いため全く約定しない。

一方売り注文を出した場合、先物基準のエージェントからすると非常に安く買える条件のいい注文となり、高確率で約定する。こうして売りばかり約定することで SRandom のポジションがすぐに限界に達し、硬直したと思われる。

その証拠に、先物と現物が一度近づくと途端に約定しはじめ、活発に取引をする。
ただし売り買いのバランスが良いために、価格の上下の移動こそ激しいものの極端な高値や安値にならず、最後まで安定している。

※SFSpread もほぼ同じような結果になったので省略する

### まとめ

SRandomStrategy は先物価格を注文価格の決定に用いるエージェントが他に参加している場合、それらのエージェントとの注文価格の差が開きすぎてしまい、極端に注文が偏ってポジション数が限界に達し硬直する。

一方で平時の先物と現物が近い状態では活発に取引を行い、先物価格の上下が行き過ぎることを防止する。

このため価格の修正にはあまり役立たないが、安定化には役立つエージェントであると言える。

## 終わりに

今回の論文は単に興味があったり合同ゼミでのやり残しを清算したいという意図もあったりしたのだが、

それ以上に実際の先物取引の経験もプログラミング等の知識も全くない自分が、自分にできる簡単な実験やマシンエージェントの改変だけでどこまで論文をかけるかという思いがあった。

そのため実験は単純なものばかりになったし、改変エージェントもほぼコピー&ペーストと一部の数値の変更など、ごく簡単な改変しかされていない。そのため当初想定していた実験ができなくなったり、ミスが多発して一から書き直すことになったりと苦労した。それでも最後にはこうしてなんとか論文として形にできて、ひとまずは納得している。

私の経験とこの論文が、今後私と同じように技能も知識も経験もないゼミ生が U-Mart についての論文を製作する上で、少しでも助けになってもらえれば幸いである。

最後に、2 年間御指導いただいた有賀先生、および共に学んだゼミ生の皆に感謝の意を示し、本論文を締めくくることとする。

参考文献（Web を含む）

塩沢由典、中島義裕、松井啓之、小山友介、谷口和久、橋本文彦
『人工市場で学ぶマーケットメカニズム - U-Mart 経済学編 - 』共立出版（2006）

喜多一、森直樹、小野功、佐藤浩
『人工市場で学ぶマーケットメカニズム - U-Mart 工学編 - 』共立出版（2009）

阿部寛之
『人工市場 U-Mart における市場取引～ヒューマンエージェント戦略とマシンエージェントの実装～』（2005）
http://c-faculty.chuo-u.ac.jp/~aruka/sotsuron/sotsuron0503/index0503.html

『大阪堂島商品取引所』
http://ode.or.jp/annai/kinou.html

日本ユニコム株式会社
『商品先物取引一般用語集【へ】』
http://www.unicom.co.jp/commodity/glossary/ha/he.html

『商品先物取引ポータル　CX PORTAL』
http://www.cx-portal.com/yougo/232.html

『ADVFN』
http://jp.advfn.com/Help/heijunka-1366.html

## 付録：改変マシンエージェントソースコード

### JumpStrategy_A

```
package strategy;

import java.util.StringTokenizer;

import server.UMartTime;
import strategy.data.Order;
import strategy.data.UDayFuturePricesAndVolume;
import strategy.data.UDaySpotPricesAndVolume;
import strategy.data.UExecution;
import strategy.data.UOrderPriceAndVolume;
import strategy.data.UServerInfo;
import strategy.data.UTimeAndSales;
import strategy.data.UUnContractOrder;

public class JumpStrategy_A extends UStrategy {

  /** Default value of width of price on order */
  public static final int DEFAULT_WIDTH_OF_PRICE = 5;

  /** Default value of maximum order quantity */
  public static final int DEFAULT_MAX_QUANT = 10;

  /** Default value of minimum order quantity */
  public static final int DEFAULT_MIN_QUANT = 1;

  /** Default valued of maximum (long/short) position */
  public static final int DEFAULT_MAX_POSITION = 700;

  /** Default value of price used for the case where no price information from the market is valid */
  public static final int DEFAULT_NOMINAL_PRICE = 500;

  /** Width of the price on order */
  private int fWidthOfPrice = DEFAULT_WIDTH_OF_PRICE;
```

```
  /** Maximum order quantity */
  private int fMaxQuant = DEFAULT_MAX_QUANT;

  /** Minimum order quantity */
  private int fMinQuant = DEFAULT_MIN_QUANT;

  /** Maximum (long/short) position */
  private int fMaxPosition = DEFAULT_MAX_POSITION;

  /** Price used for the case where no price information from the market is valid */
  private int fNominalPrice = DEFAULT_NOMINAL_PRICE;

  /**
   * ポジションが 0 以下の時のみ一度に 700 個の成行買い注文を出す
   * ポジションが 1 以上のときは売り注文を出す
   * ※ポジションが整数の場合ロング、負数の場合ショート
   * ポジションの限界を無視
   *
   * @param param
   */
  public JumpStrategy_A(int param) {
          super(param);
  }

  public void action(int[] spotPrices,int[] futurePrices,int[] futureVolume,int position,long cash,
                      UTimeAndSales[] timeAndSale,UDayFuturePricesAndVolume[] dayFuturePAV,
                      UDaySpotPricesAndVolume[] daySpotPAV,UMartTime
umartTime,UUnContractOrder[] unContractOrders,
                      UExecution[] execution,UServerInfo serverInfo,UOrderPriceAndVolume[]
buyOrderList,
                      UOrderPriceAndVolume[] sellOrderList) {
    Order order =
getOrder(spotPrices,futurePrices,position,cash,serverInfo,buyOrderList,sellOrderList);
    orderRequest(order);
  }
```

```
  public Order getOrder(int[] spotPrices,int[] futurePrices,int pos,long money,UServerInfo serverInfo,
                        UOrderPriceAndVolume[] buyOrderList,UOrderPriceAndVolume[]
sellOrderList) {
    int bitAndAskedPriceUnit = (int) serverInfo.getBitAndAskedPricesUnit();
    Order order = new Order();

    int price1 = futurePrices[0]; //直近の約定価格
    int price2 = futurePrices[1]; //直近の一つ前の約定価格

    if(price1 > 0 && price2 > 0) {
      if(pos < 1) {
        order.buysell = Order.BUY; //ポジション 0 以下
      } else if(pos > 0) {
        order.buysell = Order.SELL; //ポジション 1 以上
      }
    }

        /**ポジション調整*/
      if(order.buysell == Order.BUY) {
        if(pos > 0) {
          order.buysell = Order.NONE;
          return order;
        }
      } else if(order.buysell == Order.SELL) {
        if(pos < 1) {
          order.buysell = Order.NONE;
          return order;
        }
      }


    int prevPrice = getLatestPrice(futurePrices);
    if(prevPrice == -1) {
      prevPrice = getLatestPrice(spotPrices); // use spot price instead
    }
```

```
    if(prevPrice == -1) {
      prevPrice = fNominalPrice; // use nominal value instead
    }
    if(order.buysell == order.SELL) {
      for(int i = 0;i < buyOrderList.length;i++) {
        long buyVol = buyOrderList[i].getVolume();
        long price = buyOrderList[i].getPrice();
        if(buyVol > 0 && price > 0) {
          order.price = (int) price; //気配からなので端数処理はいらない
          order.quant = 1;//super.getRandomInteger((int)buyVol) + fMinQuant;
          /**  買いの時は一気に 700 の注文を出す  */
          return order;
        }
      }
    } else if(order.buysell == order.BUY) {
      for(int i = 0;i < sellOrderList.length;i++) {
        long sellVol = sellOrderList[i].getVolume();
        long price = sellOrderList[i].getPrice();
        if(sellVol > 0 && price > 0) {
          order.price = 0;//(int) price; //気配からなので端数処理はいらない
          order.quant = 700;//super.getRandomInteger((int)sellVol) + fMinQuant;
          /**  売りの時は 1 の注文を出す  */
          return order;
        }
      }
    }
    order.buysell = Order.NONE;
    return order;
  }

  /* (non-Javadoc)
   * @see strategy.UBaseStrategy#setParameters(java.lang.String[])
   */
  public void setParameters(String[] args) {
    super.setParameters(args);
    for(int i = 0;i < args.length;++i) {
```

```
            StringTokenizer st = new StringTokenizer(args[i],"= ");
            String key = st.nextToken();
            String value = st.nextToken();
            // System.err.println(key + "=" + value);
            if(key.equals("WidthOfPrice")) {
               fWidthOfPrice = Integer.parseInt(value);
            } else if(key.equals("MinQuant")) {
               fMinQuant = Integer.parseInt(value);
            } else if(key.equals("MaxQuant")) {
               fMaxQuant = Integer.parseInt(value);
            } else if(key.equals("MaxPosition")) {
               fMaxPosition = Integer.parseInt(value);
            } else {
               System.err.println("Unknown parameter:" + key + " in RandomStrategy.setParameters");
               System.exit(5);
            }
         }
      }

   }
```

JumpStrategy_B

```
package strategy;

import java.util.StringTokenizer;

import server.UMartTime;
import strategy.data.Order;
import strategy.data.UDayFuturePricesAndVolume;
import strategy.data.UDaySpotPricesAndVolume;
import strategy.data.UExecution;
import strategy.data.UOrderPriceAndVolume;
import strategy.data.UServerInfo;
import strategy.data.UTimeAndSales;
import strategy.data.UUnContractOrder;

public class JumpStrategy_B extends UStrategy {

   /** Default value of width of price on order */
   public static final int DEFAULT_WIDTH_OF_PRICE = 5;

   /** Default value of maximum order quantity */
   public static final int DEFAULT_MAX_QUANT = 10;

   /** Default value of minimum order quantity */
   public static final int DEFAULT_MIN_QUANT = 1;

   /** Default valued of maximum (long/short) position */
   public static final int DEFAULT_MAX_POSITION = 300;

   /** Default value of price used for the case where no price information from the market is valid */
   public static final int DEFAULT_NOMINAL_PRICE = 500;

   /** Width of the price on order */
   private int fWidthOfPrice = DEFAULT_WIDTH_OF_PRICE;

   /** Maximum order quantity */
```

```
  private int fMaxQuant = DEFAULT_MAX_QUANT;

  /** Minimum order quantity */
  private int fMinQuant = DEFAULT_MIN_QUANT;

  /** Maximum (long/short) position */
  private int fMaxPosition = DEFAULT_MAX_POSITION;

  /** Price used for the case where no price information from the market is valid */
  private int fNominalPrice = DEFAULT_NOMINAL_PRICE;

  /**
   * 売りポジションが 2500 以下の時に成行売り注文を出す（売りのポジションは負数で表される）
   * 注文数量の決定方法は TrendStrategy に準ずる
   * 買い注文と売りポジションが 2500 を超えた状態での売り注文は無効
   * ポジションの限界を無視
   *
   * @param param
   */
  public JumpStrategy_B(int param) {
          super(param);
  }

  public void action(int[] spotPrices,int[] futurePrices,int[] futureVolume,int position,long cash,
                      UTimeAndSales[] timeAndSale,UDayFuturePricesAndVolume[] dayFuturePAV,
                      UDaySpotPricesAndVolume[] daySpotPAV,UMartTime
umartTime,UUnContractOrder[] unContractOrders,
                      UExecution[] execution,UServerInfo serverInfo,UOrderPriceAndVolume[]
buyOrderList,
                      UOrderPriceAndVolume[] sellOrderList) {
    Order order =
getOrder(spotPrices,futurePrices,position,cash,serverInfo,buyOrderList,sellOrderList);
    orderRequest(order);
  }

  public Order getOrder(int[] spotPrices,int[] futurePrices,int pos,long money,UServerInfo serverInfo,
```

```
                                UOrderPriceAndVolume[] buyOrderList,UOrderPriceAndVolume[]
sellOrderList) {
    int bitAndAskedPriceUnit = (int) serverInfo.getBitAndAskedPricesUnit();
    Order order = new Order();

    int price1 = futurePrices[0]; //直近の約定価格
    int price2 = futurePrices[1]; //直近の一つ前の約定価格

    if(price1 > 0 && price2 > 0) {
        if(pos > -2500) {
          order.buysell = Order.SELL; //ポジション 0 以下
        } else if(pos < -2501) {
          order.buysell = Order.NONE; //ポジション 1 以上
        }
      }

         /**ポジション調整*/
         if(order.buysell == Order.SELL) {
            if(pos < 0) {
               order.buysell = Order.NONE;
               return order;
            }
         } else if(order.buysell == Order.BUY) {
               order.buysell = Order.NONE;
               return order;
            }



         int prevPrice = getLatestPrice(futurePrices);
         if(prevPrice == -1) {
            prevPrice = getLatestPrice(spotPrices); // use spot price instead
         }
         if(prevPrice == -1) {
            prevPrice = fNominalPrice; // use nominal value instead
         }
```

```
        if(order.buysell == order.SELL) {
          for(int i = 0;i < buyOrderList.length;i++) {
            long buyVol = buyOrderList[i].getVolume();
            long price = buyOrderList[i].getPrice();
            if(buyVol > 0 && price > 0) {
              order.price = 0;//(int) price; //気配からなので端数処理はいらない
              order.quant = super.getRandomInteger((int)buyVol) + fMinQuant;
              return order;
            }
          }
        } else if(order.buysell == order.BUY) {
          for(int i = 0;i < sellOrderList.length;i++) {
            long sellVol = sellOrderList[i].getVolume();
            long price = sellOrderList[i].getPrice();
            if(sellVol > 0 && price > 0) {
              order.price = (int) price; //気配からなので端数処理はいらない
              order.quant = super.getRandomInteger((int)sellVol) + fMinQuant;
              return order;
            }
          }
        }
        order.buysell = Order.NONE;
        return order;
                }

/* (non-Javadoc)
 * @see strategy.UBaseStrategy#setParameters(java.lang.String[])
 */
public void setParameters(String[] args) {
  super.setParameters(args);
  for(int i = 0;i < args.length;++i) {
    StringTokenizer st = new StringTokenizer(args[i],"= ");
    String key = st.nextToken();
    String value = st.nextToken();
    // System.err.println(key + "=" + value);
    if(key.equals("WidthOfPrice")) {
```

```
            fWidthOfPrice = Integer.parseInt(value);
        } else if(key.equals("MinQuant")) {
            fMinQuant = Integer.parseInt(value);
        } else if(key.equals("MaxQuant")) {
            fMaxQuant = Integer.parseInt(value);
        } else if(key.equals("MaxPosition")) {
            fMaxPosition = Integer.parseInt(value);
        } else {
            System.err.println("Unknown parameter:" + key + " in RandomStrategy.setParameters");
            System.exit(5);
        }
      }
    }

}
```