# モデルの概要

- 2章の実験1と同じで，12エージェント（買い手6，売り手6）からなる市場
- 買い手
  - それぞれ異なる財の価値が与えられる
  - 利得 = 財の価値 − 取引価格
- 売り手
  - それぞれ異なる仕入れ価格が与えられる
  - 利得 = 取引価格 − 仕入れ価格
- 取引ルール
  - 買い手と売り手の双方が自由に注文を出すことができ，また，市場に出ている注文を自由に受け入れることができる。
- その他の仮定
  - 各自が取引できるのは1つの財のみ
  - 取引が成立しなかった場合は利得は0

# ゼロ知能エージェント (Gode and Sunder, 1993)

エージェントの行動：

まったく知能を持たず，ランダムな価格で注文を行う単純なエージェント

- ✓ 実際の人間の振舞いを記述するアプローチとは正反対
- ✓ KISS原理の考え方に近い

- このような単純なエージェントによって，市場がどのように形成されるかを調べた。
- ゼロ知能エージェントであっても，市場効率性の観点で，**被験者実験と同等のパフォーマンスを実現する**ことが明らかになった。

# エージェントの行動ルール

- 売り手，買い手ともに基本的に同じルール
- 2種類のタイプ
  - 「価格制約なし」と「価格制約あり」
- 価格制約なし（ZI-U）
  - [0,100]からランダムで注文価格を選ぶ。
- 価格制約あり（ZI-C）
  - 買い手の場合：
    [0,財の価値]からランダムで注文価格を選ぶ。
  - 売り手の場合：
    [仕入れ値,100]からランダムで注文価格を選ぶ。

# 1. パラメータの準備

- 冒頭で変数を定義

```
globals [Values Costs MaxPrice TransactionPrices RoundLine CS PS TS
AveragePrice PricesAtTheRound]
turtles-own [trader-type value profit offer-price traded]
```

- パラメータに関する2つのプロシージャを定義

```
to set-parameters
    set MaxPrice 100
    set Values [45 40 35 30 25 20]
    set Costs [3 8 13 18 23 28]
    set TransactionPrices []
    set RoundLine []
    set CS 0
    set PS 0
    set TS 0
    set AveragePrice 0
    set PricesAtTheRound []
end
```

```
to initialize
   ask turtles [
       set offer-price -1
       set traded false
       set profit 0
       set CS 0
       set PS 0
       set TS 0
       set PricesAtTheRound []
   ]
end
```

# 2. エージェントの作成

```
to setup
    clear-all
    set-parameters
    foreach Values [
        x -> create-turtles 1 [
        set trader-type "BUYER"
        set value x
        ]
    ]
    foreach Costs [
        x -> create-turtles 1 [
        set trader-type ”SELLER"
        set value x
        ]
    ]
    ask turtles [
         set profit 0
         set offer-price -1
         set traded false
    ]
    draw-demand-supply-functions
    reset-ticks
end
```

買い手エージェントの作成

売り手エージェントの作成

各エージェントのパラメータの初期化

# 3. 行動ルールの記述

```
to-report agent-decision
    let price -1
    if (Agent-model = "ZI-U")[
        set price random (MaxPrice + 1)
    ]
    if (Agent-model = "ZI-C")[

    ]
    report price
end
```

価格制約ありの場合のエージェントの行動を
自分で書いてみよう

# 4. エージェント間の取引の記述

```
to go-one-round
  initialize
  repeat 20 [
    ask turtles with [traded = false] [
      if (trader-type = "BUYER") [
        set offer-price agent-decision
                let candidate-seller min-one-of (other turtles with [trader-type =
                "SELLER" and offer-price >= 0 and traded = false])[offer-price]
        if (candidate-seller != nobody and traded = false) [
                    if (offer-price >= [offer-price] of candidate-seller )[
                        set TransactionPrices lput ([offer-price] of candidate-seller)
                        TransactionPrices
                        set PricesAtTheRound lput ([offer-price] of candidate-seller)
                        PricesAtTheRound
                        set profit value - [offer-price] of candidate-seller
                        set traded true
```

```
                    ask candidate-seller [
                        set profit offer-price - value
                        set traded true
                    ]
                ]
            ]
        ]
            if (trader-type = "SELLER") [

        ]
    ]
  ]
  calc-at-round-end
  draw-transaction-price
  draw-round-end-line
  tick
end
```

売り手の場合の処理を
自分で書いてみよう

# その他の必要なプロシージャ

```
to calc-at-round-end
    set AveragePrice mean PricesAtTheRound
    ask turtles [
        if (trader-type = "BUYER") [
            set CS CS + profit
        ]
        if (trader-type = "SELLER") [
            set PS PS + profit
        ]
    ]
    set TS CS + PS
end
```

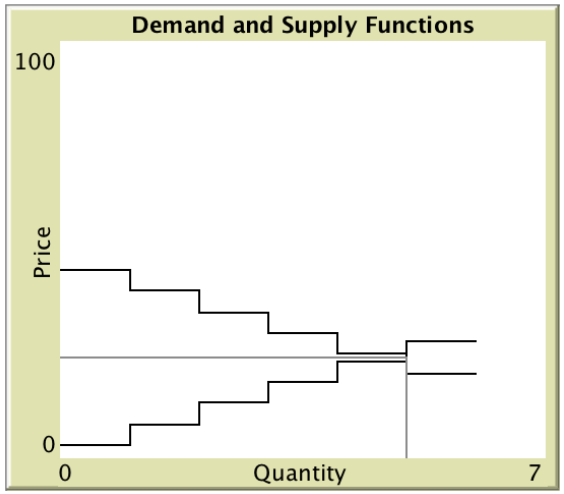
```
to draw-transaction-price
    set-current-plot "Transaction Price”
    clear-plot
    create-temporary-plot-pen "transaction
    price"
    set-plot-pen-color black
    let x 0
    foreach TransactionPrices [
        p -> plotxy x p
        set x x + 1
    ]
    create-temporary-plot-pen "equilibrium
    line”
    set-plot-pen-color gray
    plotxy 0 24
    plotxy plot-x-max 24
end
```

```
to draw-demand-supply-functions
    set-current-plot "Demand and Supply
    Functions"
    clear-plot  let x 0
    create-temporary-plot-pen "Supply”
    foreach Values [
        v -> plotxy x v
        plotxy x + 1 v
        set x x + 1
    ]
    set x 0
    create-temporary-plot-pen "Demand"
    foreach Costs [
        c -> plotxy x c
        plotxy x + 1 c
        set x x + 1
    ]
    create-temporary-plot-pen "equilibrium"
    set-plot-pen-color gray
    plotxy 0 24
    plotxy 5 24
    plotxy 5 0
end
```

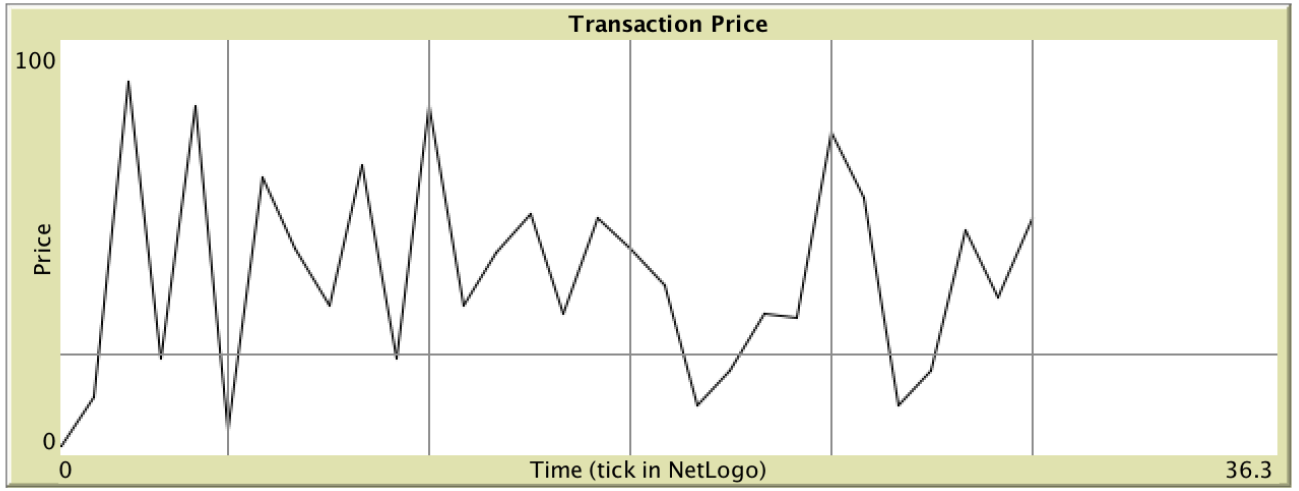
```
to draw-round-end-line
    set RoundLine lput (length
    TransactionPrices - 1) RoundLine
    set-current-plot "Transaction Price"
    foreach RoundLine [
        x ->
        create-temporary-plot-pen word "line:"
        x
        set-plot-pen-color gray
        plotxy x 0
        plotxy x plot-y-max
    ]
end
```

# プログラムを実行してみよう

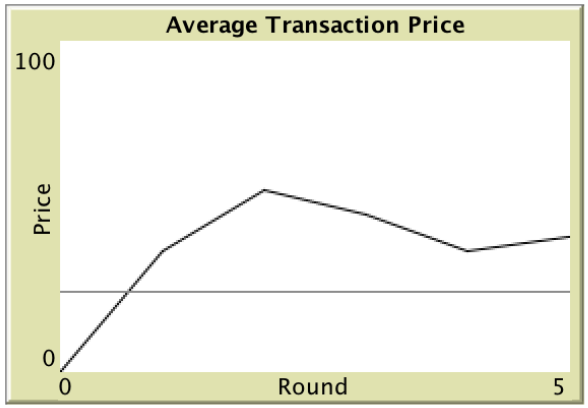
1ラウンド



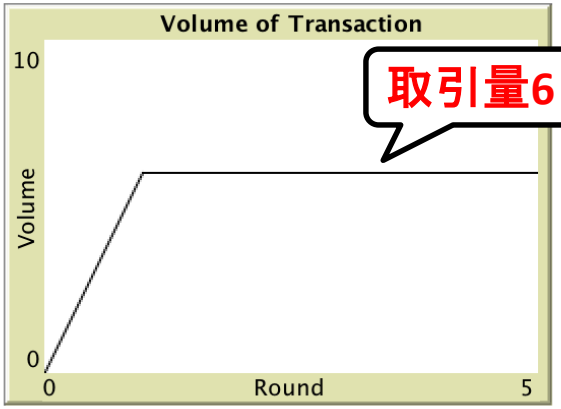

(a)　需要関数と供給関数



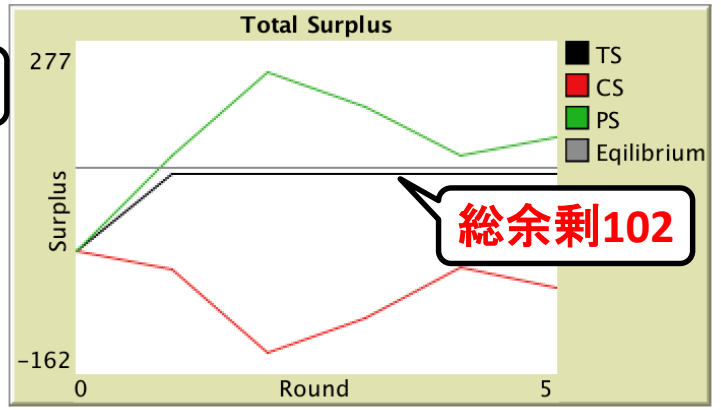

(b)　取引価格




(c)　平均取引価格




(d)　取引量

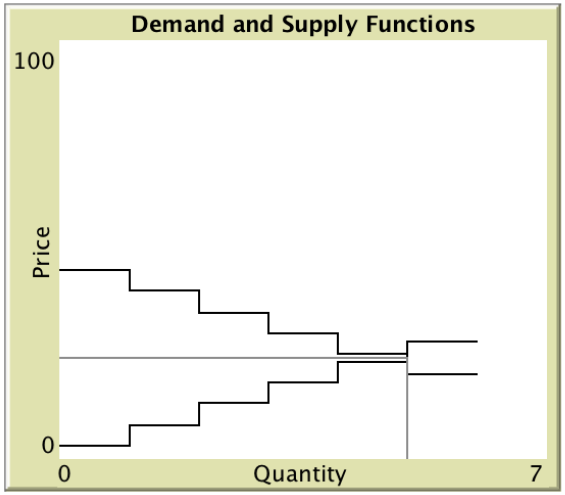



(e)　余剰

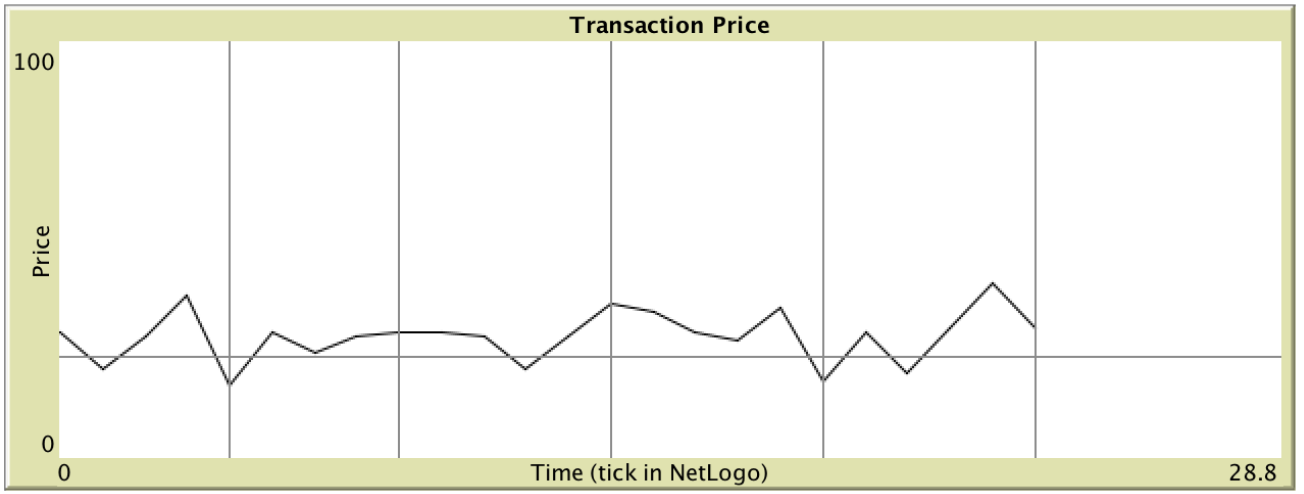
Fig.　価格制約なし(ZI-U)エージェントの結果の例

# プログラムを実行してみよう

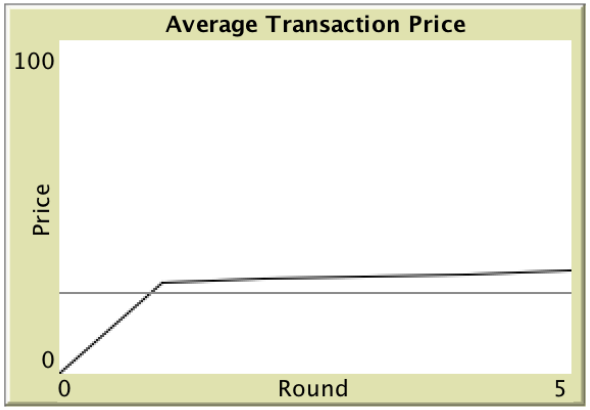
1ラウンド



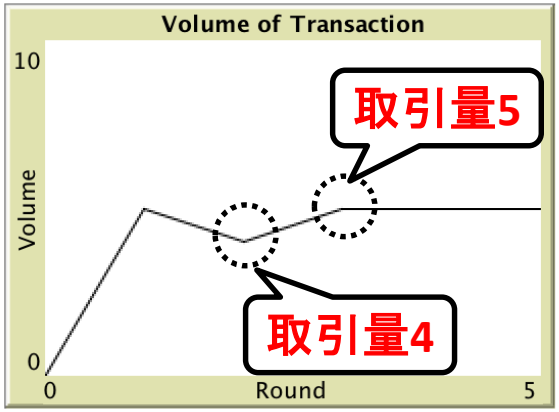

(a)　需要関数と供給関数



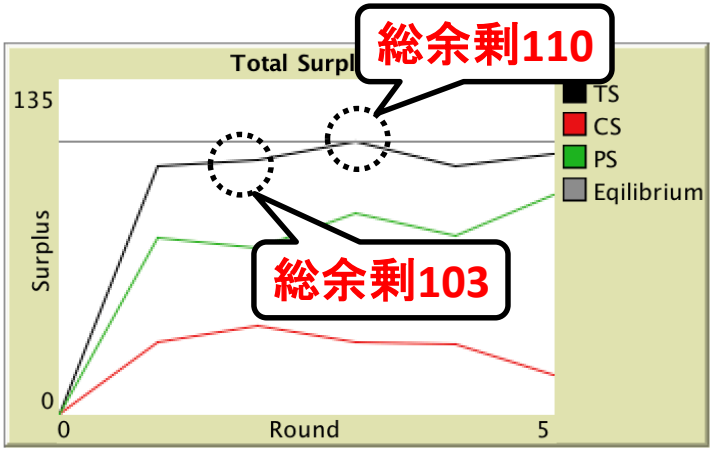

(b)　取引価格




(c)　平均取引価格




(d)　取引量




(e)　余剰

Fig.　価格制約あり(ZI-C)エージェントの結果の例

# ディスカッション

シミュレーションの結果について考察してみよう。

- 「制約なし(ZI-U)」と「制約あり(ZU-C)」の違いについて
  - 取引価格や価格が違う理由はなぜか？
  - 総余剰の違いはなぜ生じるか？

- Gode and Sunder(1993)の結果と一部が異なる理由について考えよう。

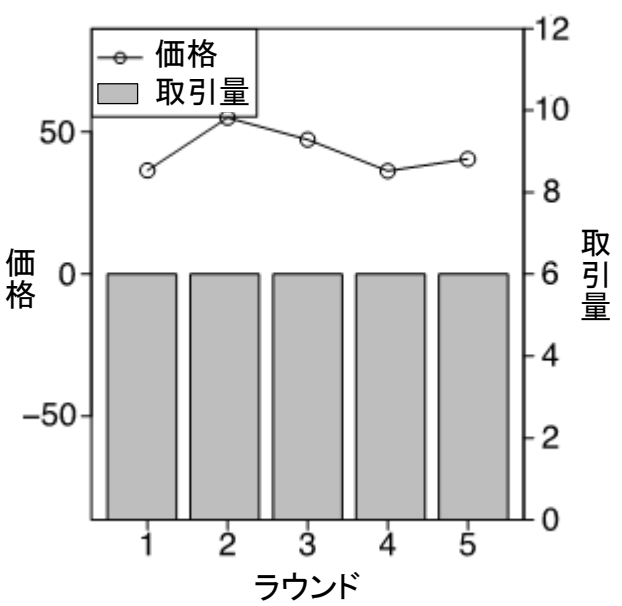
  [ヒント]
  - 12エージェントのうち，どのエージェントが取引しているのだろうか？
  - 今回，単純化のため過去の注文は消える(offer-priceを上書き)が，本来の設定ではそのまま残る。これは，どういう影響を与えているか？

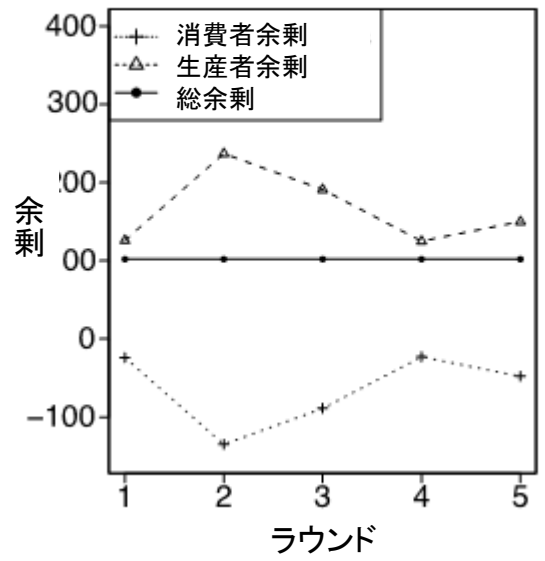
Table 先行研究の結果例

| エージェントの種類 | 市場効率性※ |
|---|---|
| ZI-U | 90.0 |
| ZI-C | 99.9 |

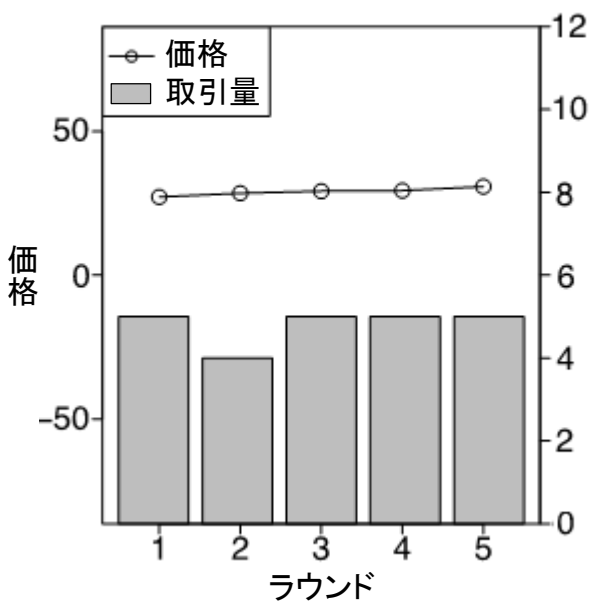
※均衡時の総余剰を100としている。また，Market1という設定下での結果である。

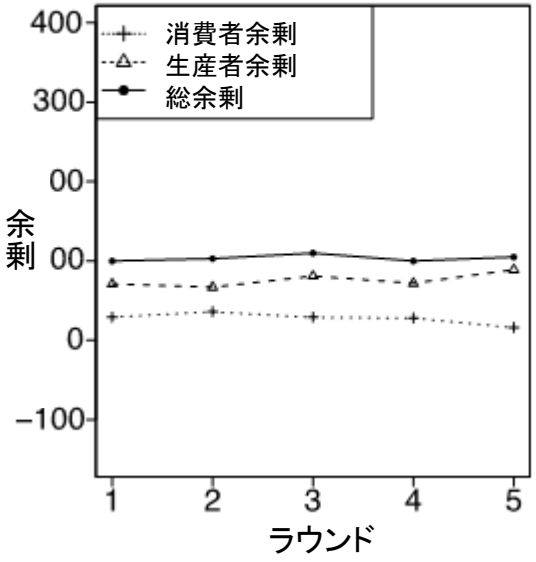
# 経済実験の結果との比較 1



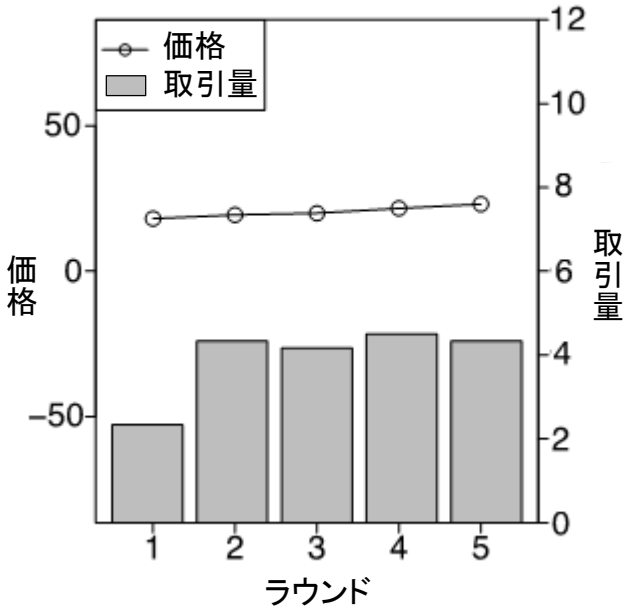



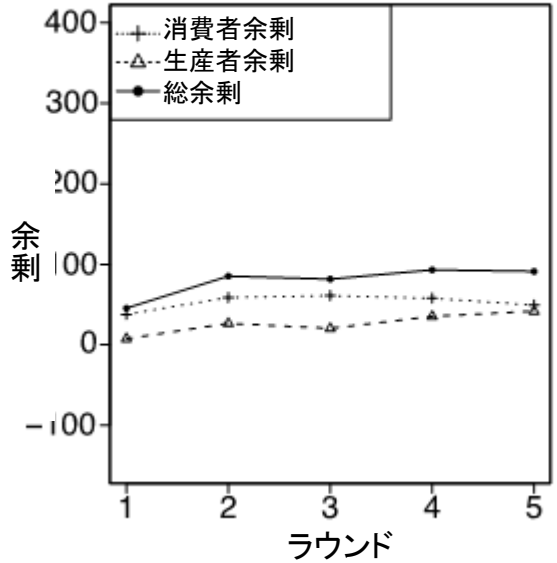

(a)　ZI-U




(b)　ZI-C




(c)　被験者

# 経済実験の結果との比較 2

Table 5ラウンドの平均値の比較

| エージェントの種類 | 市場効率性 |
|---|---|
| ZI-U | 92.7 |
| ZI-C | 94.2 |
| 被験者 | 72.2 |

Table Gode and Sunder(1993)の結果

| | Market 1 | Market 2 | Market 3 | Market 4 | Market 5 |
|---|---|---|---|---|---|
| ZI-U | 90.0 | 90.0 | 76.7 | 48.8 | 86.0 |
| ZI-C | 99.9 | 99.2 | 99.0 | 98.2 | 97.1 |
| 被験者 | 99.7 | 99.1 | 100.0 | 99.1 | 90.2 |

※Market1～5という異なる設定を用いており，需要・供給関数の形がそれぞれ異なる。

ゼロ知能エージェントが高い市場効率性を実現

⇩

被験者の知能は必ずしも必要ではない

⇩

ダブルオークションという市場メカニズムが高い効率性をもたらす