資産市場実験の内容

- n人1グループとし，仮想的な資産市場を通じて，参加者は資産を売買する。
- 具体的な手順
  - 初期保有として c EUCの現金と a 単位の資産が与えられる。
  - ダブルオークションなどの形式で取引を行う。
  - 一定時間のダブルオークションを1期間とし，T 期間繰り返す。
- 配当について
  - 各期の終わりに資産を持っていれば配当を受け取る。
  - 資産1単位当り d_t の配当を得る。
  - 配当で得た現金は，次期以降の取引に利用することができる。
- T 期終了後の資産
  - T 期終了後に保有資産は定められた一定額 p で実験者が買い取る，あるいは，価値を失う($p = 0$)

実際によく用いられる具体的設定

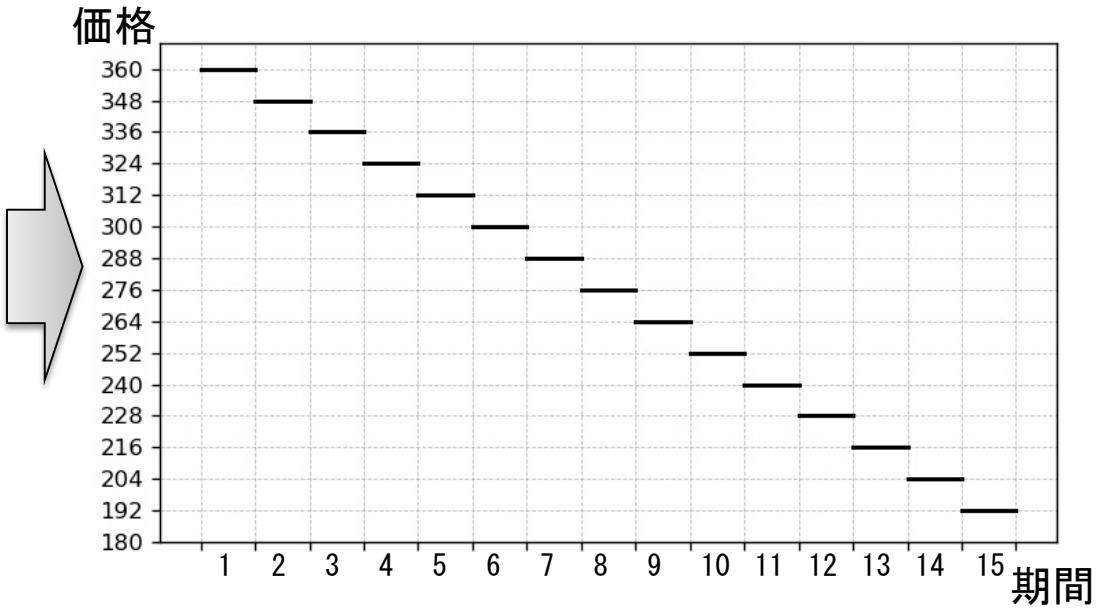
- 取引の方法
  - 連続時間ダブルオークション方式
  - コール市場方式
- 配当の方法
  - 事前に定められた一律額
  - 確率ベースでその期ごとに決まる
- その他
  - 基本的な設定では，現金を借り入れての資産の購入や，資産を借り入れての空売りなどはできない

資産の本質的価値

- 本質的価値：

$$FV_t = \sum_{k=t}^{T-t+1} E[d_k] + p$$

- 実験設定例：
  - 取引期間 $T = 15$
  - 配当 $d_t \in \{0, 6, 12, 32\}$
  - 買取価格 $p = 180$



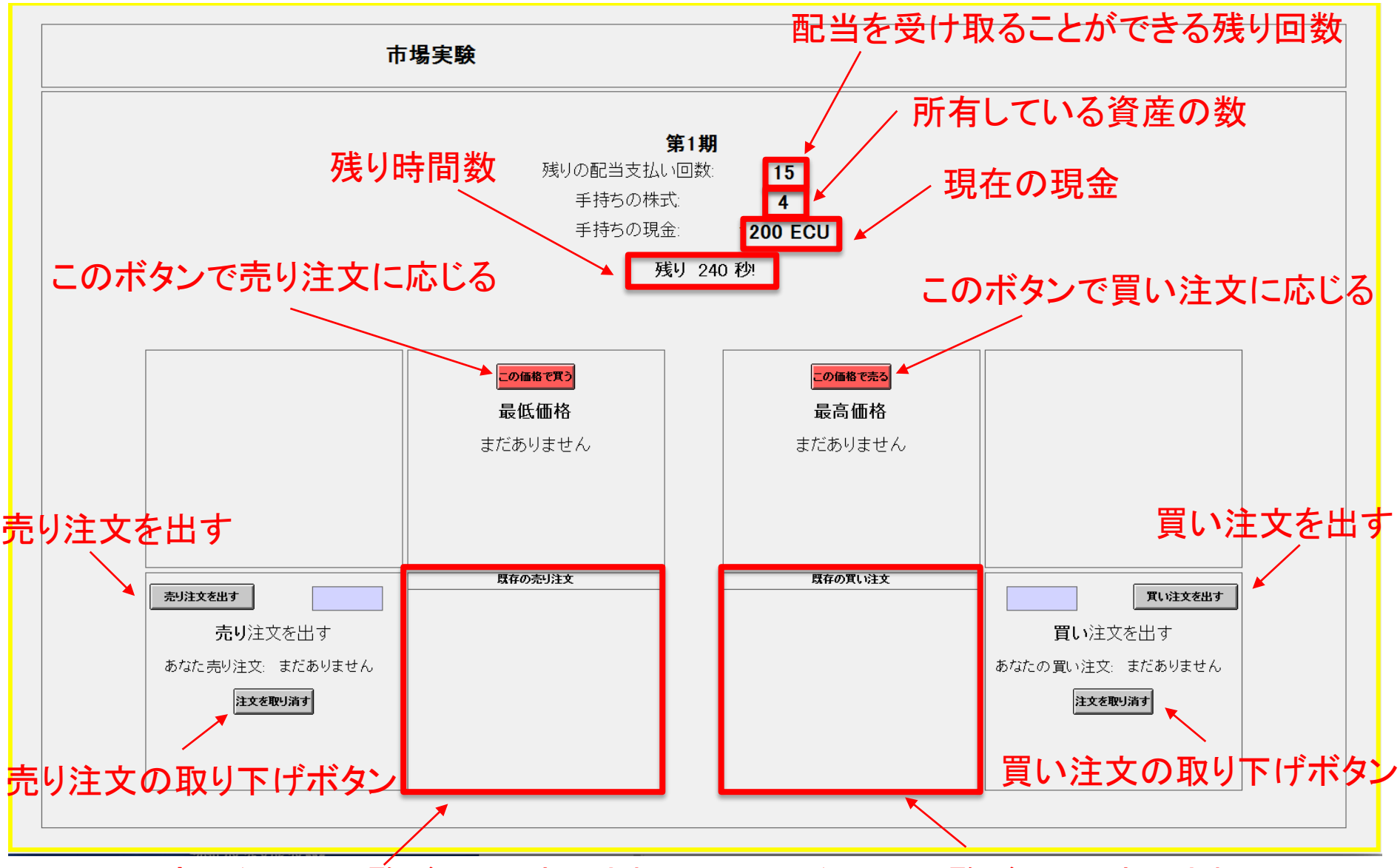
実験では本質的価値を定義できる

取引価格からの乖離を観察 → **バブルの議論が可能**

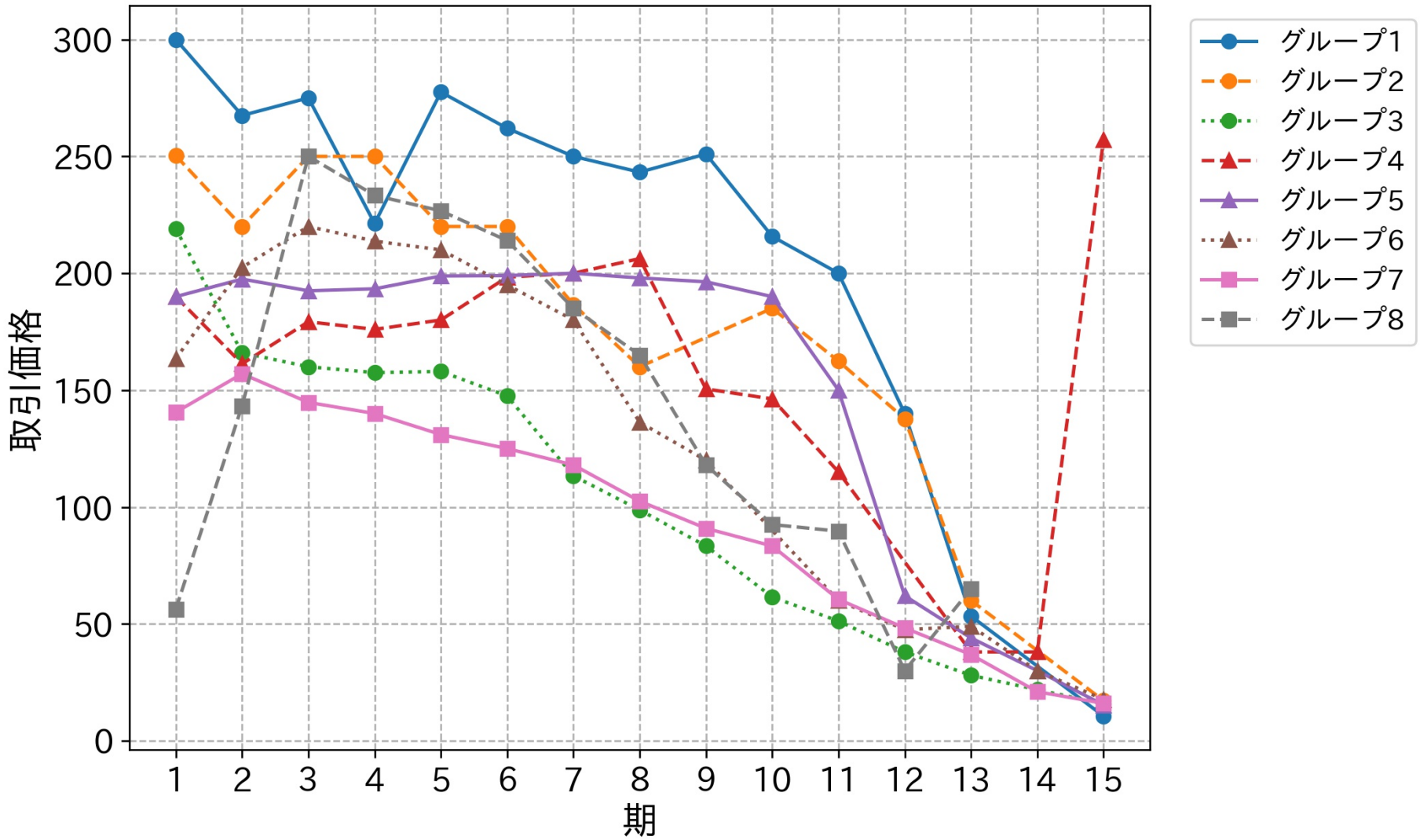
実験設定

- 6人の市場（1グループ6人）
- 初期保有
  - 各プレイヤは資産を4単位，現金を1200を持つ
- 配当の方法
  - 各期末に等確率で 0か20
- 1期間120秒の連続時間ダブルオークション方式の取引で，全部で15期間行う
- 15期間終了後は資産の価値は0

実験画面


市場実験
配当を受け取ることができる残り回数
所有している資産の数
第1期
残りの配当支払い回数:
15
手持ちの株式:
4
手持ちの現金:
200 ECU
残り時間数
現在の現金
残り 240 秒!
このボタンで売り注文に応じる
このボタンで買い注文に応じる
この価格で買う
最低価格
まだありません
この価格で売る
最高価格
まだありません
売り注文を出す
買い注文を出す
既存の売り注文
既存の買い注文
売り注文を出す
売り注文を出す
あなた売り注文: まだありません
注文を取り消す
買い注文を出す
買い注文を出す
あなたの買い注文: まだありません
注文を取り消す
売り注文の取り下げボタン
買い注文の取り下げボタン
売り注文の一覧がここに表示される
買い注文の一覧がここに表示される

実験結果


300
250
200
150
100
50
0
取引価格
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
期
グループ1
グループ2
グループ3
グループ4
グループ5
グループ6
グループ7
グループ8

Duffy and Ünver (2006) のモデル

- エージェントの行動ルール
  - 毎期，各エージェントはランダムな順番にS回行動する。
  - 一回の行動につき，売り注文か買い注文を予算制約のもとで出す。
  - 買い注文を出す確率
    - $\pi_t = \max\{0.5 - \Phi t, 0\}$
    - $\Phi \in [0, 0.5/15)$: パラメータ

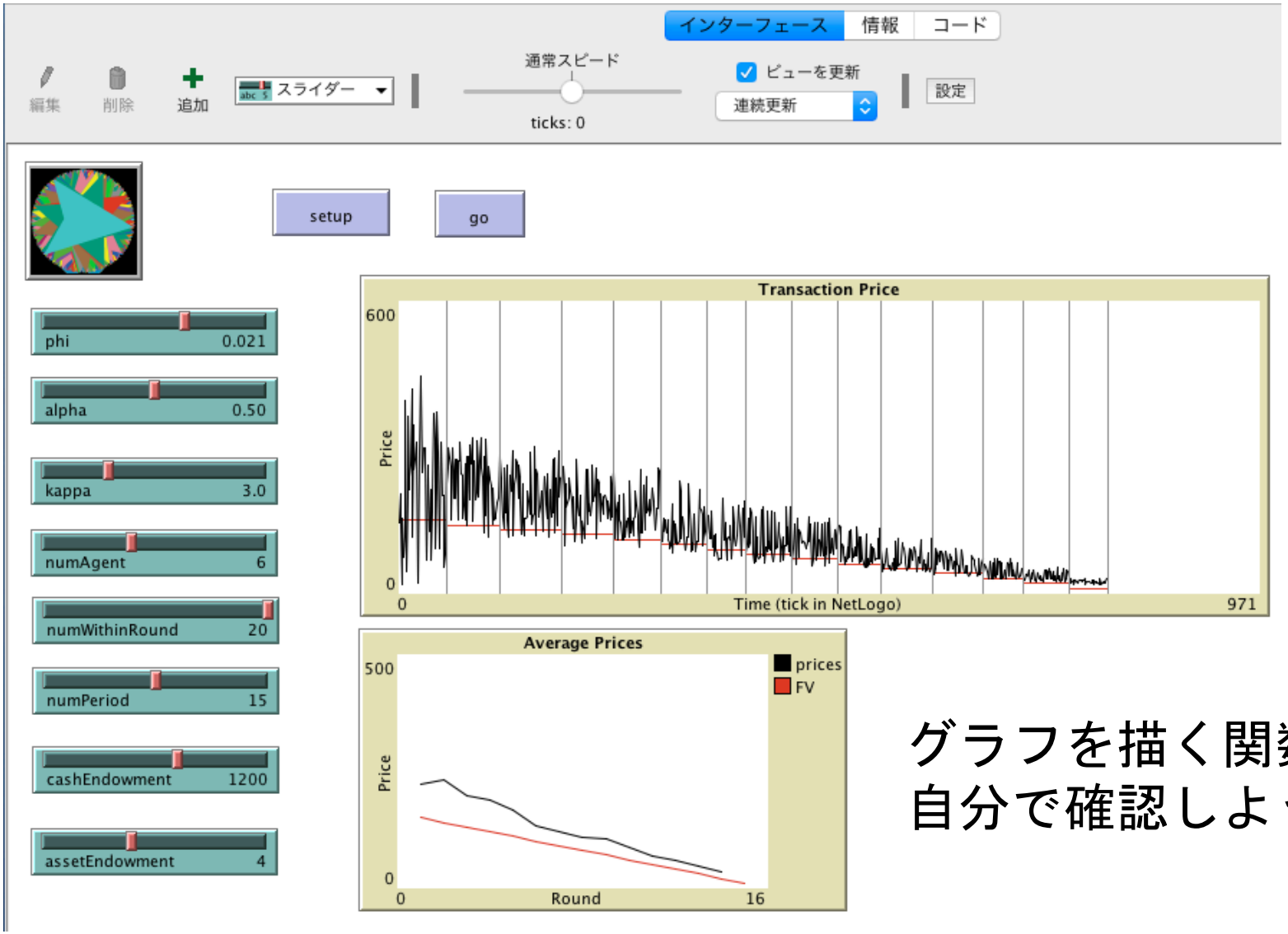
Duffy and Ünver (2006) のモデル

- エージェントの行動ルール
  - 買い注文（手元に現金があるときのみ）
    - $b_{t,s}^{i} = \min\left\{(1-\alpha)u_{t,s}^{i} + \alpha\bar{p}_{t-1}, x_{t,s}^{i}\right\}$
  - 売り注文（手元に資産があるときのみ）
    - $a_{t,s}^{i} = (1-\alpha)u_{t,s}^{i} + \alpha\bar{p}_{t-1}$

      ここで，$u_{t,s}^{i} \sim U[0, \kappa FV_t]$，$\bar{p}_{t-1}$ はt-1期の平均価格，α, κ はモデルのパラメータである。
  - ただし，t=1では，$\bar{p}_{t-1}$が定義されないので
    - $a_{t,s}^{i} = b_{t,s}^{i} = u_{t,s}^{i}$

  である。

実行例



グラフを描く関数は
自分で確認しよう。

NetLogoでの実装

- 使用したデータ構造
  - トレーダーエージェント
    - 所持する現金の額
    - 所持する証券の数
    - 買い注文提出時の提出価格
    - 売り注文提出時の提出価格
  - 買い注文表エージェントと売り注文表エージェント
    - 提出価格
    - 出したトレーダーエージェントのID
    - 取引されたかどうか
  - 各期の途中で成立した取引の価格を保存するリスト
  - 各期の平均価格を保存するリスト

グローバル変数など

- FV : 各期の本質的価値保存用のリスト
- TransactionPrices : 取引価格保存用のリスト
- AveragePrices :平均取引価格保存用のリスト
- Div : 各期の可能な配当額 [0,20]
- meanDiv : 配当額の平均
- maxOrder : κFV_t （最大注文価格）
- AveragePastPrice: p_{t-1} （前期の平均取引価格）
- probBuy : π_t （購入確率）
- period : t （期）
- randomOrder: ランダムにトレーダーエージェントを並べるためのリスト

- 制御用の変数
  - active : 現在動いているトレーダーエージェント
  - numBuy : 出された買い注文の数
  - numSell : 出された売り注文の数
  - numTrade : 成立した取引の数

３種類のエージェント

```
breed [traders trader]
breed [buyOrders border]
breed [sellOrders sorder]

traders-own [cash asset buy sell]
buyOrders-own [price agent traded]
sellOrders-own [price agent traded]
```

- breed [type name]
  - type: エージェントの種類
  - name: 1エージェントを呼び出したい際に使う名前（IDで指定）
- 注意点
  - Breedで種類分けはするが，すべてturtlesエージェント
  - nameを使って，それぞれのエージェントを呼び出す際のIDに注意する必要あり（つぎのスライド参照）

各エージェントのIDの例

- create-traders 6
- create-buyOrder 10
- create-sellOrder 10

という風に３種類のエージェントを作成すると，各エージェントのIDは

- trader 0～5
- border 6～15
- sorder 16 ～25

という具合に，IDは３種類のエージェントを通じて共通のものとなる。

エージェントのタイプで振り直されないので注意

エージェント変数の初期化

```
to setup
  clear-all
  reset-ticks

  set-parameters
  create-traders numAgent
  create-buyOrders (numAgent * numWithinRound)
  create-sellOrders (numAgent * numWithinRound)

  set numTrade 0
  set period 0
  set maxOrder ( kappa * (item period FV))
  ask traders [
    ;トレーダーエージェントが持つ各変数の初期化
    set shape "person"
    set cash cashEndowment
    set asset assetEndowment
    set buy random maxOrder
    set sell random maxOrder
  ]
  reset-orderbook
end
```

```
to reset-orderbook
;注文表エージェントが持つ
;各変数の初期化
  ask buyOrders [
    set price -1
    set agent -1
    set traded -1
  ]

  ask sellOrders [
    set price -1
    set agent -1
    set traded -1
  ]
end
```

パラメータなど

```
to set-parameters
  set Div [0 20]
  set meanDiv mean Div
  set TransactionPrices []
  set AveragePrices []
  set AveragePastPrice -1.0
  set FV n-values numPeriod [ i -> meanDiv * (numPeriod - i)]
  set probBuy 0.5
  set randomOrder n-values numAgent [i -> i]
end
```

本質的価値のリスト（FV）について：
n-values numPeriod により，0からnumPeriod-1までの数列ができる。それぞれの値を meanDiv * (numPeriod –i) に代入することで作成している。

プログラムの本体

三段階のループ

```
repeat numPeriod [ ;1期当りの動き
      repeat numWithinPeriod [ ; S回繰り返す
            repeat numAgent [
                  ; 各エージェントへ命令
                  ; 注文を提出する
                  ; 注文表を更新する
                  ; 取引をチェックする
            ]
      ]
]
```

プログラムの本体

```
to go
  ;1つめのループ
  repeat numPeriod [
    reset-orderbook
    initializePeriod
    ;2つめのループ
    repeat numWithinRound [
      set randomOrder shuffle randomOrder
      let i 0;
      ;3つめのループ
      repeat numAgent [
        set active (item i randomOrder)

            （中略）

       set i i + 1
     ]
   ]
   draw-transaction-price
   updateCash
   set period period + 1
 ]
 draw-Average-Prices
end
```

- randomOrder リストの中身を各期の中のS回の繰り返しの最初にランダムに並び替える（shuffleを使う）
- その後，randomOrder の準備に，active なトレーダーエージェントを1人づつ定義し，中略の部分のプログラムを実行

各期における初期化

```
to initializePeriod
  set maxOrder ( kappa * (item period FV))
  set probBuy max (list (0.5 - phi * period) 0)
  set numBuy count traders
  set numSell (count traders) + (count buyOrders)
  if period > 0 [
     set AveragePastPrice mean TransactionPrices
     set AveragePrices lput AveragePastPrice AveragePrices
  ]
    set TransactionPrices []
end
```

- numBuy とnumSellの値の設定の仕方に注目
- エージェント個別IDは通し番号なので，調整が必要なため

注文提出

```
ifelse (random-float 1.0 < probBuy )[
          if([cash] of trader active > 0 )[
            ask trader active [
              set-buyOrder
            ]
            ask border numBuy [
              set price [buy] of trader active
              set agent active
              set traded -1
            ]
            set numBuy numBuy + 1
            checkTradeBuy
          ]
        ]
```

買い注文の値段の設定はモデルの説明通り

ここで，出された注文を随時買い注文表エージェントに保存。エージェントIDに注意

売り注文も同様に処理

取引チェック

注文表から最低売却価格での売り注文を検索

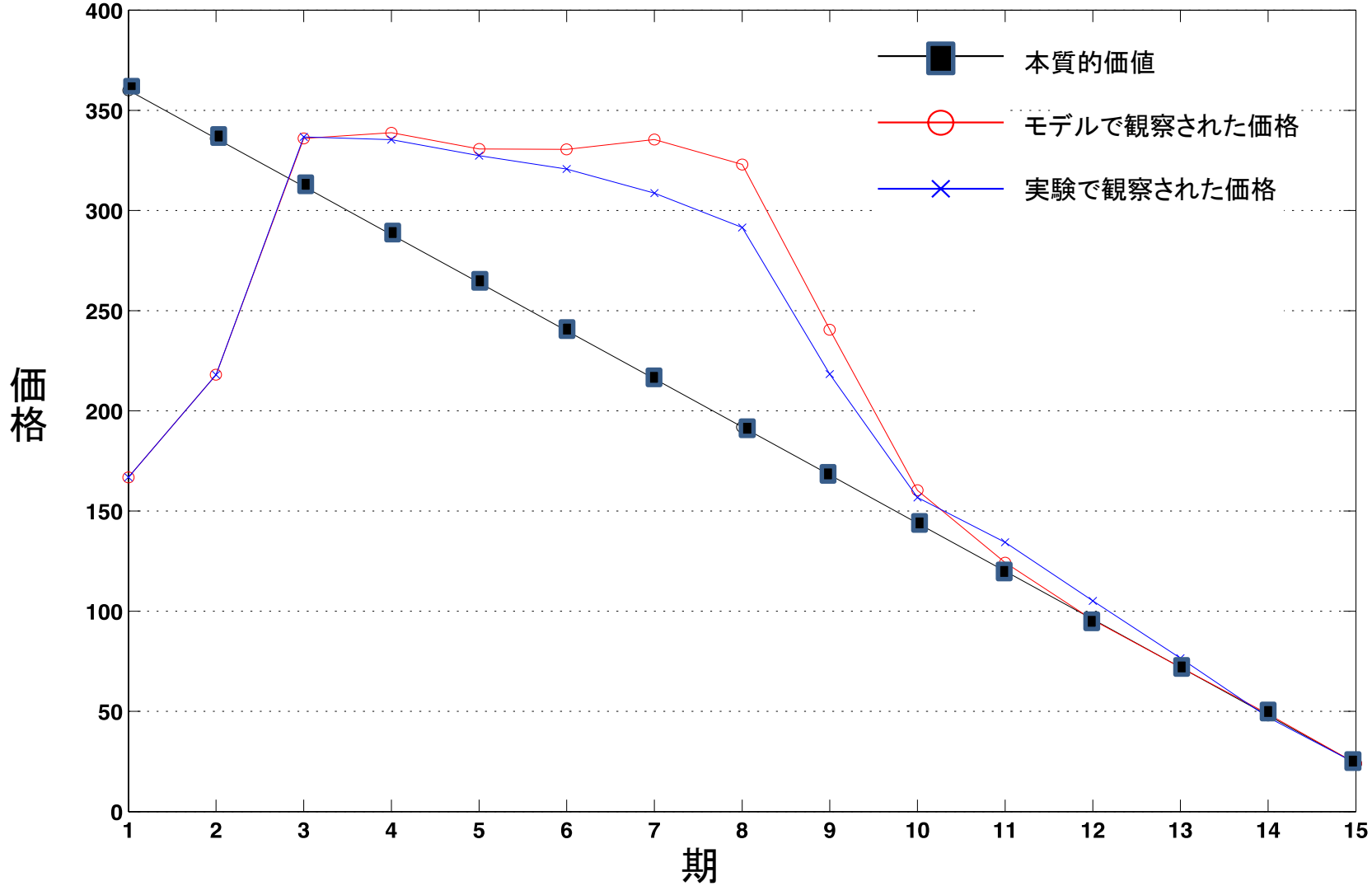
```
to checkTradeBuy
  let candidate-seller min-one-of (sellOrders with [traded = -1 and price
>= 0])[price]
  if candidate-seller != nobody [
    let potPrice ([price] of candidate-seller)
    if( [buy] of trader active >= potPrice )[ ;; trade
      set TransactionPrices lput potPrice TransactionPrices;
      ask trader active [
        set cash cash - potPrice
        set asset asset + 1
      ]
      ask trader ([agent] of candidate-seller) [
        set cash cash + potPrice
        set asset asset - 1
      ]
      ask border numBuy [
        set traded 1
      ]
      ask candidate-seller [
        set traded 1
      ]
    ]
  ]
end
```

取引可能であれば，実行し，関与しているトレーダーエージェントの現金と証券の保有額を更新

買い注文表および売り注文表エージェントの情報も更新

売り注文が出された場合も同様にチェック

実験結果とモデルシミュレーションの結果の対比



Source: Haruvy and Noussair (2006)
のモデルと実験データを基に作成