慶應義塾大学学術情報リポジトリ
Keio Associated Repository of Academic resouces

Title	エージェントベースモデルによる観光業の政策評価に関する探索的研究
Sub Title	
Author	中村, 淳海(Nakamura, Atsumi) 高橋, 大志(Takahashi, Hiroshi)
Publisher	慶應義塾大学大学院経営管理研究科
Publication year	2020
Jtitle	
JaLC DOI	
Abstract	
Notes	修士学位論文. 2020年度経営学 第3736号
Genre	Thesis or Dissertation
URL	https://koara.lib.keio.ac.jp/xoonips/modules/xoonips/detail.php?koara_id=KO40003001-00002020-3736

慶應義塾大学学術情報リポジトリ(KOARA)に掲載されているコンテンツの著作権は、それぞれの著作者、学会または出版社/発行者に帰属し、その権利は著作権法によって保護されています。引用にあたっては、著作権法を遵守してご利用ください。

The copyrights of content available on the KeiO Associated Repository of Academic resources (KOARA) belong to the respective authors, academic societies, or publishers/issuers, and these rights are protected by the Japanese Copyright Act. When quoting the content, please follow the Japanese copyright act.

慶應義塾大学大学院経営管理研究科修士課程

学位論文（　2020　年度）

論文題名

エージェントベースモデルによる観光業の政策評価に関する探索的研究

指導教員	高橋　大志　　教授
副指導教員	林　高樹　　教授
副指導教員	山尾　佐智子　准教授

氏　名	中村　淳海

論文要旨

所属ゼミ	高橋研究会	氏名	中村　淳海

（論文題名）
エージェントベースモデルによる観光業の政策評価に関する探索的研究

（内容の要旨）

はじめに

本研究では、ビジネスホテルにおける価格戦略の有効性について検討を行う。分析を通じ、ビジネスホテルの賃貸期間と売り上げへのインパクト、政策シミュレーションを導入することでどのように変化するのかを明らかにすることを試みる。本研究では価格戦略を政策とする。本研究では、エージェントベースシミュレーションを通じ、借手エージェントと貸手エージェントのマッチングを行い、需要と供給のバランスの結果について分析したのち、価格戦略を考慮した分析を行う。先行研究で価格を考慮したフレームワークを作成したものを、個々の戦略から政策効果を測る。ボトムアップで政策評価をし、有効なツールであることを探索的に検討する。

先行研究

先行研究は、シミュレーションを活用し Airbnb の成長性を分析したものである。分析の結果、(1)短期賃貸の価格は外因的であり, 期間と空室数は相互に関係している可能性があると仮定している。(2)賃貸市場においても，他の市場と同様に，市場のメカニズムを需要と供給でモデル化することができる。(3)Airbnb 規制の政策措置を加味したシミュレーションを実行したことが解明できた。これらの先行研究の貢献は、Airbnb のようなシェアリングエコノミーと観光地の住宅市場の発展を結びつけたことである。

分析方法

本分析では、全体の挙動をエージェントの挙動からシミュレートする。需要数や属性パラメータに基づき動くエージェントが、供給側の希望に最も近い部屋にマッチングする事によって生じるシミュレーションを行う。マッチングするエージェントがない場合は機会損失としてカウントする。期間は 360 日に設定する。借手エージェントは長期短期含め 9,000 人からスタートし、貸手エージェント数は 3,600 室からスタートする。状況シナリオとしては、短期物件の割合による入居数の変化を検討する。施策シナリオとして、価格格戦略の有無とする。分析には NTT データシステム S-quattro Simulation System を使用した。

分析結果

本分析の貸手の物件の割合は短期 60%、長期 40%とした。また、本分析の価格戦略については、短期物件の場合は 10 日、長期物件は 30 日を超えると値下げが行われるものとして分析を行った。まずは、1 日 250 件~450 件の需要数でマッチングを実行し空室率を測った。価格戦略がない場合の結果、価格戦略がある場合の結果より、短期の空室率が相対的に低く入居率が高いことを確認できた。次に、供給数が大幅に増加した場合（1 日 3,600 件~9000 件）のシミュレーションを実行した。前者と比較すると、相対的に空室率が増加していることを確認できた。分析の結果、(1)短期賃貸は先に埋まること、(2)価格戦略に効用があること、(3)価格のボリュームゾーンは市場の平均値に近いことを見出した。

まとめ

本研究では、価格を考慮したフレームワークを作成し個々の戦略から政策効果を測った。具体的には、貸手エージェントと借手エージェントの需要と供給レベルを作成し、ホテルの入居率に関する基礎的な分析と価格弾力性の検討を実施した。今後の課題として、部屋の質などの指標を追加すること、価格弾力性の更なる検討と最適価格を求めることが挙げられる。これらは、先行研究に基づく仮説とモデルが有用であること、そして価格戦略を導入した独自性に貢献すると考える。

目次

第１章　序論 2

第２章　関連研究 2
第１節　分析モデル 5

第３章　シミュレーションとエージェントベースモデリング 6

第４章　賃貸市場モデル 7
第１節　モデルの概略 8
第２節　借手エージェント 9
第３節　貸手エージェント 10

第５章　分析結果 11
第１節　システム構築 11
第２節　シミュレーション分析 12
第３節　基本ケース 12
第４節　供給数が大幅に増加した場合 13
第５節　空室数の変化 14

第６章　価格の弾力性 17

第７章　まとめ 18

参考文献 19

謝辞 20

Appendix 20

第１章　序論

オリンピック招致のプレゼンテーションで世界にアピールした日本の「おもてなし」は，今やわが国のサービス産業の高い品質を象徴する代名詞となった。このアピールが功を奏してか，日本は 2020 年の東京オリンピックの招致に成功し，これを機に外国人観光客の増大を期待してわが国のサービス産業，わけても宿泊，飲食等に代表されるホスピタリティ関連の業種に属する企業は久々に活気づいた。訪日外国人旅行者は 2010 年代から急増し，2018 年には 3,000 万人を超えた（Japan National Tourism Organization [JNTO], 2019b）。しかし、2020 年 2 月頃より発生したコロナ・ショックにより未曾有の経済低迷に直面している。2020 年度の観光白書によると、日本の観光需要は２８兆円あります。その内のインバウンドは 4.8 兆円の消費額となっている。インバウンドの旅行者数は３千万人、日本人は２千万人海外旅行へ行っており、つまり昨年度より３千万人減少、２千万人回帰するという予測になっている。観光市場においては、インバウンドがなくなった場合の需要は約７%程度であり、日本国内の需要でどの程度戻せるかということが議論されてきた。2020 年 4~5 月で 9 割減収し、コロナが 2～3 年続けば観光業界は縮小する見立てがされている。

本研究の目的は、観光業における民泊とビジネスホテルが与える影響を比較し、政策評価を行うことである。先行研究のモデルの再現性を踏まえて、ビジネスホテルの賃貸期間と売り上げへのインパクト、地域別の課税や政策シミュレーションを導入することでどのように変化するのかを検討する。本研究では、エージェントベースシミュレーションを通じ、借手エージェントと貸手エージェントのマッチングを行い、需要と供給のバランスの結果について分析したのち、価格戦略を考慮した分析を行う。先行研究で価格を考慮したフレームワークを作成したものを、個々の戦略から政策効果を測る。ボトムアップで政策評価をし、有効なツールであることを検証する。使用システムは S4（NTT データ実装シミュレーションシステム）とする。経営学的知見としては、シミュレーションを通じて価格戦略などの仮説を考察し、(1)短期賃貸は先に埋まること、(2)価格戦略に効用があること、(3)価格のボリュームゾーンは市場の平均値に近いことを見出した。

第２章　関連研究

先行研究は、シミュレーションを活用し Airbnb の成長性を分析したものである。分析の結果、(1)短期賃貸の価格は外因的であり, 期間と空室数は相互に関係している可能性があると仮定している。(2)賃貸市場においても，他の市場と同様に，市場のメカニズムを需要と供給でモデル化することができる。(3)Airbnb 規制の政策措置を加味したシミュレーションを実行したことが解明できた。これらの先行研究の貢献は、Airbnb のようなシェアリングエコノミーと観光地の住宅市場の発展を結びつけたことである。

Airbnb（エアビーアンドビー）は、宿泊施設・民宿を貸し出す人向けのウェブサイトである。世界 192 カ国の 33,000 の都市で 80 万以上の宿を提供している。Vinogradov et al,（2019）の研究によると、政策介入ありのシナリオと政策介入なしのシナリオを比較することで、Airbnb の成長ダイナミクスをどのように予測できるかという問題を取り上げている。この研究は、シェアリングエコノミーと観光に関連する他の現象の両方のモデリングに ABM（Agent Based Model）アプローチが適用可能であ

るという示唆を支持している。ABM はシミュレーションの一手法である。そして、論文は以下の点で既存の文献に貢献している。第一に、オンライン CtoC 宿泊施設とその地域の住宅市場への影響に関する実証的文献を修正している（Barron, Kung, & Proserpio, 2018; Guttentag, 2015; Oskam & Boswijk, 2016)。それは、観光セクターの規制されていない市場セグメントで議論され、実施されている政策介入の理論的根拠に基づいた説明（Edelman & Geradin, 2015; Gurran & Phibbs, 2017）と、短期・長期賃貸市場のモデルを提供することによって議論される。したがって、Airbnb のビジネスモデルとその持続可能性に関する研究が現在までに十分に発展していないという事実に対応している。第二に、ABM を技術的にも概念的にも豊富な手法として利用し、観光業の将来の発展をシミュレーションしている。したがって、Vinogradov は Nicholls (2017)が提唱した「より広い観光客に ABM を紹介する」というアジェンダを取り上げている。全体として、本論文は、ロサンゼルスのように Airbnb を規制するための政策措置を議論している都市、またはベルリンのように最近導入した都市に関する最近の議論を拡張している。Vinogradov モデルでは、賃貸市場においても，他の市場と同様に，市場のメカニズムを需要と供給でモデル化することができる。そして、短期賃貸の価格は外因的であり，期間と空室数は相互に関係している可能性があると仮定している。

【Vinogradov モデル】

$$S=f(Q-R-c)・Ha \quad (1)$$

$$L=H-f(Q-R-c)・Ha \quad (2)$$

(1)と(2)を下記に代入すると(3)となる。

$$R = r(L) \text{where} \frac{\partial g}{\partial d} < 0$$

$$Ha=g(d)$$

$$\text{where} \frac{\partial g}{\partial d} > 0$$

（1）短期賃貸ユニット=f(賃貸率－長期賃貸住宅の賃貸率－短期・長期の共通要素）×不在地主が所有する住宅数

（2）長期賃貸ユニット＝f(住宅の固定在庫－短期賃貸ユニット)と定義される。

S＋L（短期＋長期）＝H（住宅の固定在庫)、仮に Q（賃貸率）－c(共通要素)＞R(長期賃貸の賃貸率)であれば、空家を開放する。例えば、次の数式の通りである。

$$S=f(Q-R-c)・g(d) \quad (3)$$

$$L=H-f(Q-R-c)・g(d) \quad (4)$$

$$\frac{\partial S}{\partial Q} = g(d)\frac{\partial f}{\partial Q} \tag{5}$$

$$\frac{\partial S}{\partial d} = f(Q - R - c) \cdot \frac{\partial g}{\partial d} = \frac{\partial g}{\partial d} \cdot \frac{S}{g(d)} \tag{6}$$

Airbnb のような短期賃貸は、平均的には賃貸期間が短いとはいえ、住宅の賃貸市場として捉えられるかもしれない。したがって、これらのユニットの市場は、一般的に賃貸期間が長い従来の賃貸市場との間には一定の違いはあるものの、賃貸市場とみなすことができる。また、短期賃貸は、ホテルやペンションなどの伝統的な宿泊サービス提供者の代替としても機能しており、現地在住者向けの賃貸ではなく、観光市場に該当するのが一般的である。（Lee, 2016）しかし、Airbnb を通じた短期賃貸は賃貸市場の発展につながっているため、ホテル市場への影響よりも、Airbnb の賃貸が現地の短期賃貸市場にどのような影響を与えているかに注目している。

賃貸市場においても, 他の市場と同様に, 市場のメカニズムを需要と供給でモデル化することができる。これは，短期・長期の賃貸市場においても同様である。本稿では，短期・長期のセグメントを含めた賃貸市場の供給側に焦点を当てている。特に，一方のセグメントにおける供給の変化が他方のセグメントにどのような影響を与えうるかに注目している。例えば、家主が長期賃貸市場での賃貸から短期賃貸市場での賃貸に変更することを決めた場合などが考えられる。需要側では、観光の発生を考慮して、短期賃貸はホテル宿泊の代替手段として捉えられており、供給側では、民間短期賃貸は長期賃貸の代替手段として捉えられている。（Lee, 2016）

賃貸住宅のストックは短期的には固定されているが, 賃貸物件への投資に対する期待収益率の変化に応じて時間の経過とともに変化する可能性があるというストックフローモデル（Gabriel & Nothaft, 2001）は，賃貸市場を分析するための出発点として機能する可能性がある。このことは，期待収益率の変化が長期的に賃貸住宅のストックを変化させ，それによって賃貸住宅の総ストックと，短期・長期賃貸に割り当てられた賃貸住宅のシェアの両方に影響を与える可能性があることを示唆している。また、需要や供給が過剰になると、賃貸住宅の市場価格が調整され、その結果、家計の空き家を貸し出すかどうかの判断に影響を与える可能性がある。しかし、短期賃貸市場の住宅ストックは、家主が短期賃貸か長期賃貸かを判断することで、かなりのスピードで変化する可能性がある。期待収益率は価格に依存しており、これもまた需要過多や供給過多によって決まるため、空室率や空室期間にも影響を与える。(Gabriel & Nothaft, 2001)

また、短期と長期の賃料のセグメントでは期待収益率が異なる場合があり、あるセグメントの空室を保有することで家主に影響を与え、別のセグメントにシフトさせる可能性がある。また、期待収益率は、それ自体が、賃貸住宅を持つ世帯が期間ごとにどちらのセグメントを選択するかを決定し、短期的にも 2 つのセグメントの賃貸ストックに影響を与える可能性がある。したがって、1 つのセグメントではなく 2 つのセグメントで運営することで、各セグメント内の賃貸ユニットのストックがよりボラティリティになる可能性があるため、Airbnb の市場参入はより大きなボラティリティにつながる可能

性がある。Barron（2018）は、短期賃貸市場で自分の住宅ユニットを供給することを選択する大家の選択を、外因的に与えられた短期住宅の賃貸料率-短期市場での賃貸にかかるすべての追加コストを差し引いたものが、長期住宅賃貸セグメントの賃貸料率よりも大きいかどうかによって決定されるものとしてモデル化している。

Vinogradov(2019)は、概念的な考え方、ABM 法の背後にある理論、実証的な検証とその意味合いを中心に、世界各地の地域レベルでのシミュレーションを行った。シミュレーションでは、長期的な市場発展の安定性を生み出すことができるため、どのような施策をとるべきかを示唆するとともに、成長している市場セグメントを不安定化させないためには、どのような施策を避けるべきかを示している。さらに、シミュレーションされた政策介入の結果は、短期・長期賃貸市場の理論的な経済モデルによっても示唆され、シミュレーションされた ABM モデルが理論モデルの知見を検証していることを示唆している。したがって、先行研究の理論的貢献は、Airbnb のようなシェアリングエコノミーと観光地の住宅市場の発展を結びつけることである。

第１節 分析モデル

Gabriel &Nothaft 数式モデル(2001)は賃貸市場の空室率を表すために下記の数式を作成した。

数式における *Mi* は, ある期間, 例えば１年の間に i 番目のアパートが空室となった月数を表し, *Vi* は, その年に第１番目の賃貸住宅が空室となった回数を表すとすると，年間を通して入居しているアパートの場合は、*Mi* と *Vi* はゼロになる。H は，その年の各月の賃貸住宅ストック数である。その年の市場空室率は(1)となる。

アパートが１年の間に何度か空室になっている率が高いのを表すには、(2)であり空室期間の平均は月単位である。(3)または、空室期間を１年の端数で表す。(4)は明確に、空室率は発生率と平均期間の績である。(5) の導出では，簡単のために，期間中の賃貸住宅ストックが一定であると仮定しているが，賃貸住宅ストックからの追加や削除を組み込むために分析を拡張することは容易である。例えば，Hi を，1 年のうちに i 番目の住戸が賃貸住宅に入居している月数であると定義すると，この分析は，賃貸住宅ストックからの追加または削除を行うことが容易になる。例えば、賃貸住宅ストックの一定割合が平均１ヶ月間空室となっている場合や、その割合の４分の１が空室となっている期間が平均４ヶ月間となっている場合には、等価空室率が計算される。入居中の物件または賃貸用の空室物件のいずれかである場合には、$\bar{H}$は $\Sigma iHi/12$ であり，下記の分解は(6)のように書き換えることができる。

$$\frac{\Sigma i\, Mi}{12 * H'} \qquad \frac{\text{月} \times \text{空室}}{12 \times \text{賃貸在庫}} \tag{1}$$

$$\frac{\Sigma i\, Vi}{H} \qquad \frac{\text{その年の i 番目の賃貸単位によって入力された空室}}{\text{賃貸在庫}} \tag{2}$$

$$\frac{\Sigma i\, Mi}{\Sigma i\, Vi} \qquad \frac{\text{月}}{\text{対象の賃貸在庫}} \tag{3}$$

$$\frac{\Sigma i\, Mi}{12 * \Sigma i\, Vi} \qquad \frac{\text{月} \times \text{空室}}{12 \times \text{対象の賃貸在庫の合計}} \tag{4}$$

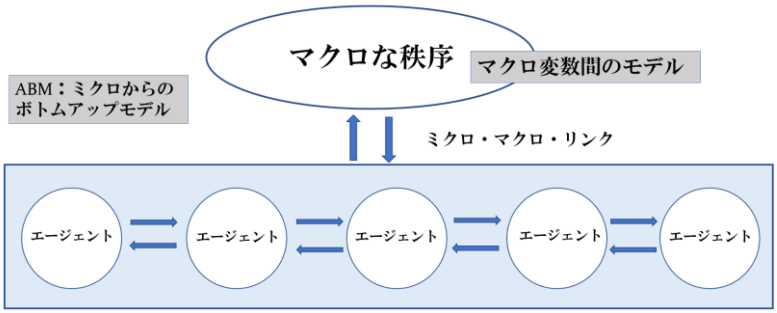
$$\frac{\Sigma i\, Mi}{12 * H} = \frac{\Sigma i\, Vi}{H} * \frac{\Sigma i\, Mi}{12 * \Sigma i\, Vi} \tag{5}$$

$$\frac{\Sigma i\, Mi}{12 * \bar{H}} = \frac{\Sigma i\, Vi}{\bar{H}} * \frac{\Sigma i\, Mi}{12 * \Sigma i\, Vi} \quad \cdot \tag{6}$$

第３章　シミュレーションとエージェントベースモデリング

シミュレーションは現実のシステムをモデル化（模擬）して、モデルを実行することで、 その振る舞いを分析・予測する問題解決手法である。その中でも、エージェントシュミレーションは、全体の挙動をエージェントの挙動からシミュレートする 。また一定のルールに従い自律的に行動するエージェントが、相互に作用し合いながら行動する事によって生じる現象のシミュレーションなど、多数のエージェントからなる仮想的な世の中を構築する。

社会シミュレーション（遠藤・寺野・佐藤・栗原・倉橋・田中, 2017）によると、エージェントベースモデリングでは、「エージェント」と呼ぶ内部状態と意思決定・問題解決能力、ならびに通信機能を備えた主体が必要である。そしてこのような相互作用に基づく創発的な現象やシナリオを分析しようとする。その特徴は、①ミクロ的な観点においてエージェントが個別の内部状態を持ち、自律的に行動・適応し、情報交換と問題解決に携わる点、②その結果として対象システムのマクロ的な性質が創発する点、③エージェントとエージェントを囲む環境とがミクロ・マクロ・リンクを形成し、互いに影響を及ぼしあいながら、システムの状態が変化していく点にある。図１に示すような、エージェント間のミクロレベルの相互作用で創発するマクロな現象が生じ、それがトップダウンにエージェントに影響を与えることになる。



出典　寺野（2017:21）表 2-1

図１　エージェントベースモデリングの一般的な構造

例えば、不動産賃貸を行う人々をエージェントとみなそう。すると、個々の取引というミクロな行動によって、不動産市場の価格の変化というマクロな現象が創発することになる。さらに、不動産市場の価格というマクロな情報が、個々のエージェントの意思決定に影響を与える。寺野ら（2017）によると、このようにミクロな現象とマクロな現象が絡み合って非常に複雑な動きをするのが現実の社会である。このような絡み合いをミクロ・マクロ・リンクという。このような現象は、非線形的であり、スケールフリー・ネットワークにみられるような「べき乗則」も、またカオス的な動きも発見することができる。解析的な方法、あるいは従来の数値的なシミュレーションでは得られない結果が導かれることも多い。

第４章　賃貸市場モデル

本研究では、不動産賃貸市場が直面する課題にエージェントベースモデリングを通じ取り組む。図２は、コンピュータ内に構築した賃貸市場の実行画面の一部を示したものである。賃貸市場には実行主体として借手エージェントと貸手エージェントの２種類が存在する。モデル構築の際に、日本における民営の賃貸物件に先行研究の理論が当てはまるか検証するため、自社データによって構築するものとする。なお、本研究では、NTT データ数理システム S-quattro Simulation System を用い分析を行った。

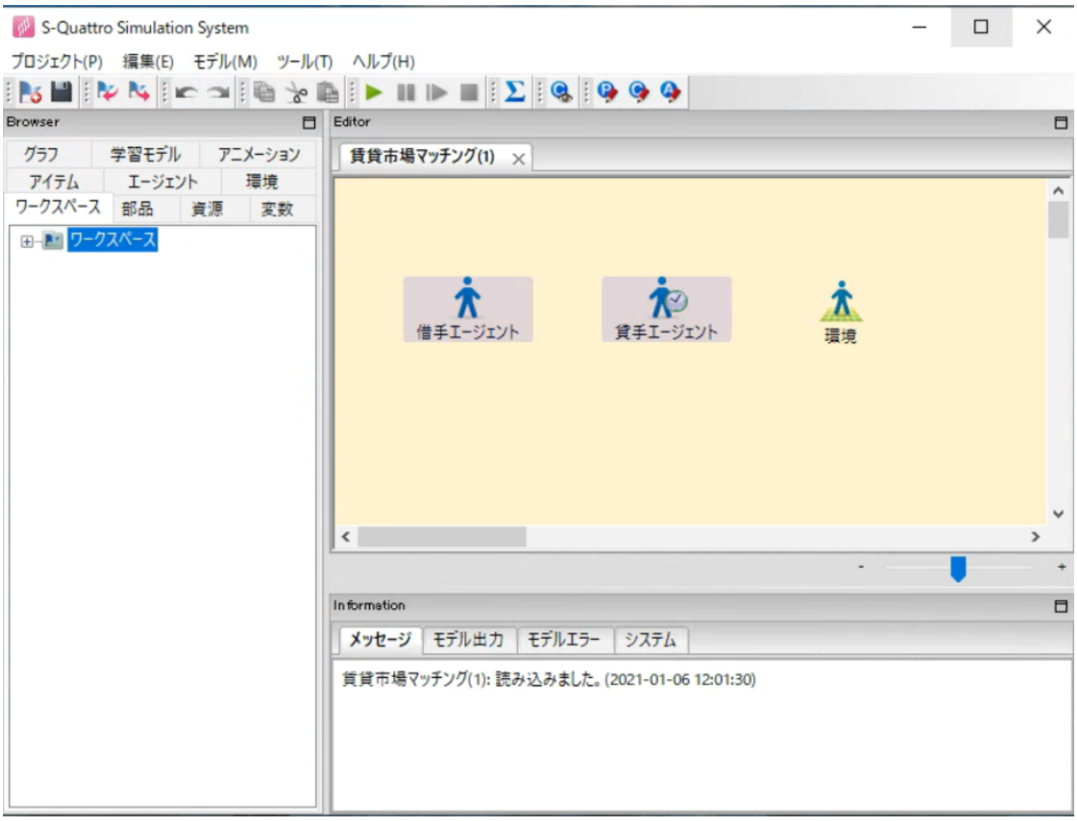


図２　分析実行画面

第１節　モデルの概略

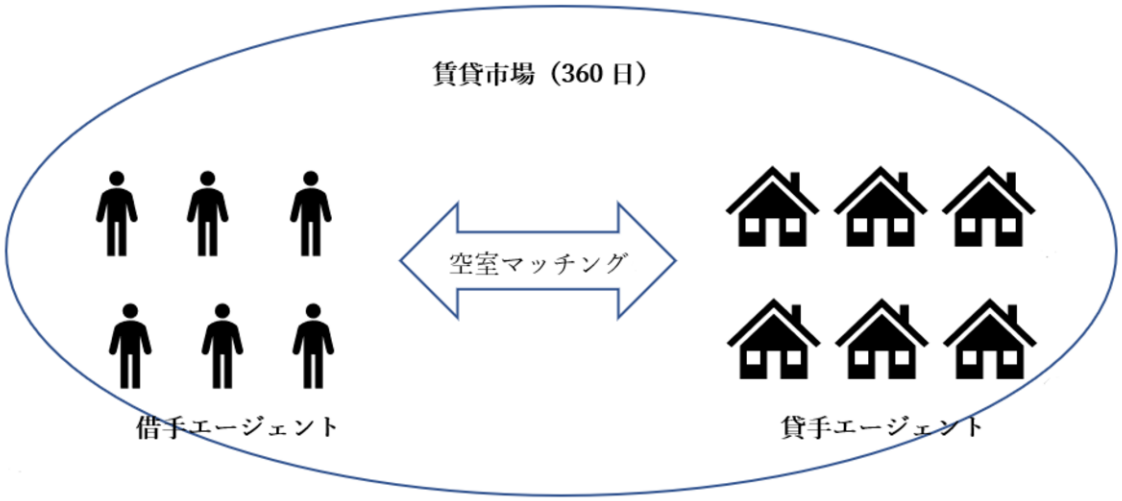


図３　賃貸市場モデル

賃貸市場の期間は 360 日に設定する。借手エージェントは長期短期含め 9,000 人からスタートし、貸手エージェント数は3600 室からスタートする。状況シナリオとしては、 短期物件の割合による空室数の変化を検討する。施策シナリオとして、価格戦略の有無とする。想定する価格戦略として、１）動的（振れ幅大）、２）動的(振れ幅小)、３）静的、４）最低価格の４つである。借手エージェントと

貸手エージェントとのマッチング設定は次の通りである。まず、需要（借手エージェント）が発生する。需要は属性(サラリーマンや国内旅行者など)、滞在日数（長期あるいは短期)、レベル(1~6)を持っている。
次に貸手エージェントとのマッチングを行う。借手エージェントは、空室と滞在日数がマッチする、レベルが最も近いエージェントを探す。マッチしたら、貸し手エージェントの空室を OFF にする。マッチする貸し手エージェントがない場合は機会損失としてカウントするというものである。

第２節　借手エージェント

表１は借手エージェントの種類と設定を示している。賃貸市場において、滞在日数とマッチした物件を探す人を借手エージェントと置く。サラリーマン、国内旅行客、カップル、インバウンドの４種類の借手エージェントを設定する。エージェント数は 9,000 人（360 日×25 人）とし、シミュレーションは､1 月 1 日から始まり､借手の 1/360 日が 1 日ごとに借手市場に投入され､賃貸探しを開始する。マッチする貸し手エージェントがない場合は機会損失としてカウントする。客層は支払能力を表す値として 1～６のレベルを有する。各エージェントによってレベルの幅が異なるが、各レベルの人数は正規分布によって割り当てる。全体の需要に対する割合もエージェントによって異なるが、合計を 100%にするよう設定した。また、滞在日数に関しても短期と長期の合計が 100%、滞在日数の詳細についても短期/長期別にそれぞれ 100%となっている。これらの分布は全て自社ビジネスホテルの過去のデータより算出したものである。表３の左側より、行動目的、内部データ、データを表記している。借手エージェントの目的は滞在日数と予算にマッチした宿泊を目指すことである。そのため、１回目に完全マッチングしなかった場合は、レベルを－１、－２して宿泊先を探す。データの項目欄にあるサラリーマンの割合 40%、国内旅行客の割合 30%、カップルの割合 10%、インバウンドの割合 20%と滞在日数の欄はサラリーマン短期 90% 長期 10%、国内旅行客　短期 80% 長期 20%、カップル 短期 90% 長期 10%、インバウンド 短期 60% 長期 40%は、全て合計 100%となるよう配分されている。

表１　借手エージェントの種類と設定

行動目的	滞在日数とマッチした物件を探す ・安価な賃料の宿泊を目指す
内部データ	・レベル（客層） サラリーマン　2～6 国内旅行客　　3～5 カップル　　2～5 インバウンド　1～3 ・滞在日数(短期、長期)

データ	・需要数(360 日で発生するエージェント数、キャンセル率も考慮した数) ・全体の需要に対する割合 サラリーマンの割合 40% 国内旅行客の割合　30% カップルの割合　10% インバウンドの割合 20% ・滞在日数 サラリーマン 短期 90% 長期 10% 国内旅行客　短期 80% 長期 20% カップル　短期 90% 長期 10% インバウンド 短期 60% 長期 40% 短期、長期別の滞在日数の詳細 短期: 1 日 40%, 2 日 20%, 3～5 日 10%, 6,7 日 5% 長期: 30 日 25%, 60 日 25%, 90 日 25%, 180 日 25%

Note:表中において、左段から行動目的、内部データ、データの詳細を示している。レベル（客層)、全体の需要に対する割合、滞在日数、短期・長期別の滞在日数の詳細は各々レベル分けや合計 100%となるよう配分されている。

第３節　貸手エージェント

賃貸市場において､必要な空室を確保する人のことを貸手エージェントと置く｡賃料のレベルは 4000 円～9000 円の 6 段階に分ける。価格戦略は、ミニマム空室期間を切ったら価格を下げる。下げ幅は－１～２である。ミニマム空室期間は短期（1 日～9 日)、長期（30 日、60 日、90 日、180 日）ともに 50%ずつに設定している。物件の割合は短期 60%、長期 40%とする。次に、想定するホテルの価格戦略を設定する。これは、ホテルの平均価格を指標として価格設定戦略の分析を行った結果である。オリンピック開催やライブなど周辺で大きなイベントがあるかで価格を変更するボリュームゾーンが 4 つに分けられる。最安値は最低価格より低くなることはないため、振れ幅が大きい。また、最高値は安くすることはあれ高くはならないため振れ幅が小さい。想定するホテルの戦略としては、１．動的（振れ幅大）4000 円～5000 円とする。これは、イベント有無で大きく価格の変更があるものである。２．動的（振れ幅小）8000 円～9000 円とする。イベント有無で価格を変更するものである。３．静的は 6000 円～7000 円とし、価格を初期設定から変更しないものを示す。最低価格 4000 円は常にその期で販売される最安価格で販売するものがある。

表 2　貸手エージェントの種類と設定

行動目的	・必要な空室を確保する

内部データ	・レベル（賃料）１～６ ・賃貸期間(短期、長期) ・価格戦略（ミニマム空室期間）を切ったら価格(レベル)を下げる．下げ幅(1～2) ・課税有 10%/ 課税無 0%(課税ありはレベル 1 増)
データ	・部屋数(エージェント数) ・レベル別の部屋の割合 レベル 1　30% 4000 円 レベル 2　20% 5000 円 レベル 3　15% 6000 円 レベル 4　15% 7000 円 レベル 5　10% 8000 円 レベル 6　10% 9000 円 ・賃貸期間別ミニマム空室期間 短期　1 日～9 日　50% 長期　30 日、60 日、90 日、180 日　50% 全て同じ割合 ・物件の割合 短期 60%　長期 40%

Note:表中において、左段から行動目的、内部データ、データの詳細を示している。レベル（賃料）、賃貸期間、価格戦略、レベル別の部屋の割合、賃貸期間別ミニマム空室期間、物件の割合の詳細は各々合計 100%となるよう配分されている。

第５章　分析結果

第１節　システム構築

本研究では、空室時点をベースにする時点空室率を採用した。空室率=空室数÷全室数(賃貸することが可能な物件戸数)×100 でその時点での全戸数に対する空室数の割合を指す。シンプルに空室率を出したいときに使用する計算方法である。空室数とは、計算したその時点で入居者がいない部屋数のことである。それを実際に賃貸可能な物件戸数（その時点で空室のものとして賃貸可能な物件数の合計）で割って 100 をかけることにより空室率が算出される。戸数ベースでの計算式に関して、部屋数は比較的簡単に把握することができる点に加え、計算のしやすさが最大のメリットである。
その反面、部屋のサイズなどの反映はされないので、部屋の間取り、スペースの大きさがまちまちの物件では空室率に大きなブレが生じる可能性がある。戸数ベースの空室率計算は、賃貸アパート、マンションなどでよく使われる。Gabriel &Nothaft 数式モデル(2001)では、年間のアパート空室率を示しているが、今回はビジネスホテルの賃貸市場ということで独自にパラメータを作成した。

【作成パラメータ】

$$\frac{短期空室率}{短期部屋数}+\frac{長期空室率}{長期部屋数}=\frac{短期空室率+長期空室率}{空室合計}\quad 11$$

分母を整えトータルが 100 になるよう設定した。短期空室率÷短期部屋数で短期の空室率を求め、長期空室率÷長期部屋数で長期の空室率を求めた。足し算をするとトータルの空室率が算出される。

第 2 節　シミュレーション分析

本分析では、シナリオごとにモデルを構築し、シミュレーションを行う。表 3 は部屋の空室率を示したものである。表 3 は 1 日の需要件数が 250 件～450 件の需要数でマッチングを実行し空室率を示したものである。借手エージェント数は N=9000, N=108000, N=126000, N=144000 となっている。これを 360 日で割ると 1 日当たりの需要件数が算出される。また、貸手エージェント数は N=3600 に設定する。表 3 より、短期の空室率が相対的に高いことを確認できる。なお、ここでの価格戦略の有無による有意差は約 0.001～0.010 程度見られた。

第 3 節　基本ケース

図 4 の左段から大きく、需要件数、価格戦略がない場合の結果、価格戦略がある場合の結果を示している。各結果はそれぞれ、需要件数が 250 件のとき短期空室率は 0.6 を示しており、これは 6 割の空室率で 4 割の入居率を示す。一方で、需要件数が 250 件のとき空室数（長期）0.7 を示しており、これは 7 割の空室率で入居率は 3 割ということを示す。需要件数が増加していくにつれて、短期空室率が下がるという結果はモデルの有効性を表していると思われる。

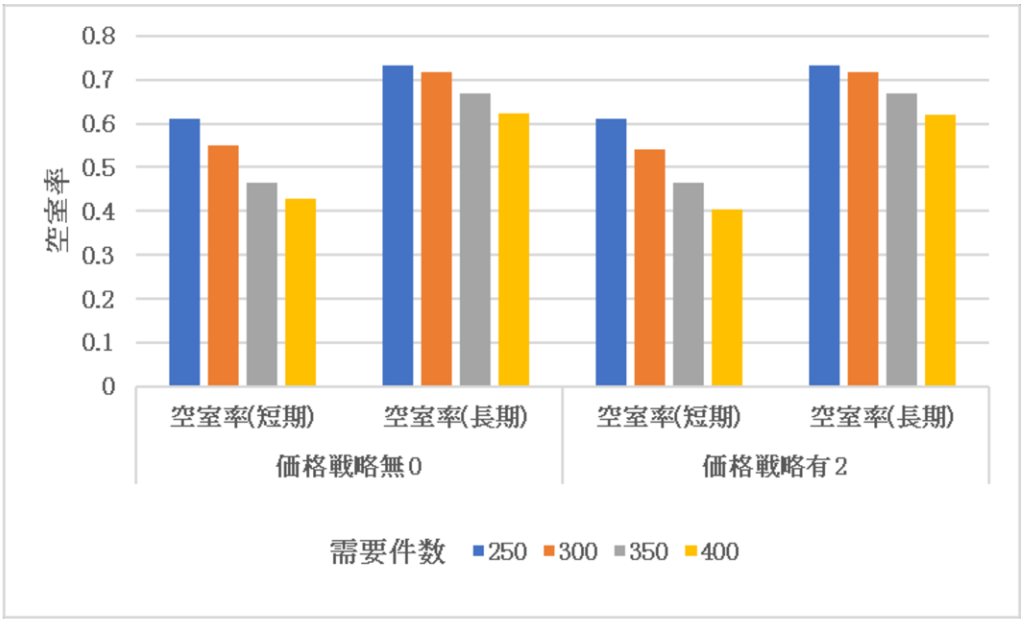


図 4　部屋の空室率（価格戦略の有無比較）

Note:図中において、左段から大きく価格戦略 0 のとき、右段は価格戦略 2 のときの結果を示している。各結果はそれぞれ、各需要数に対する短期・長期空室率を示しており、短期の方が空室率が減少している。

第 4 節　供給数が大幅に増加した場合

図 5 は、供給数が大幅に増加した場合（貸手エージェント数 1 日 3600 件~9000 件）、価格戦略 0~2 の空室率を示したものである。表 3 と比較すると、相対的に空室率が増加していることを確認できる。借手エージェント数は N=9000, N=108000, N=126000, N=144000 とし、借手エージェントの数を N=3600, N=5400, N=7200, N=9000 に設定した。なお、ここでは僅かながら価格戦略の有意差はあるものの、0.001 程度であった。そのため、価格戦略による大きな動作は確認できなかった。表 4 の左段から大きく、需要件数、価格戦略がない場合の結果、価格戦略がある場合の結果を示している。各結果はそれぞれ、需要件数が 250 件のとき空室率（短期）は 0.6 を示しており、これは 6 割の空室率で 4 割の入居率を示す。一方で、需要件数が 250 件のとき空室数（長期）0.7 を示しており、これは 7 割の空室率で入居率は 3 割ということを示す。需要件数が増加していくにつれて、空室率（短期）が下がっているという結果はモデルの有効性を表していると思われる。

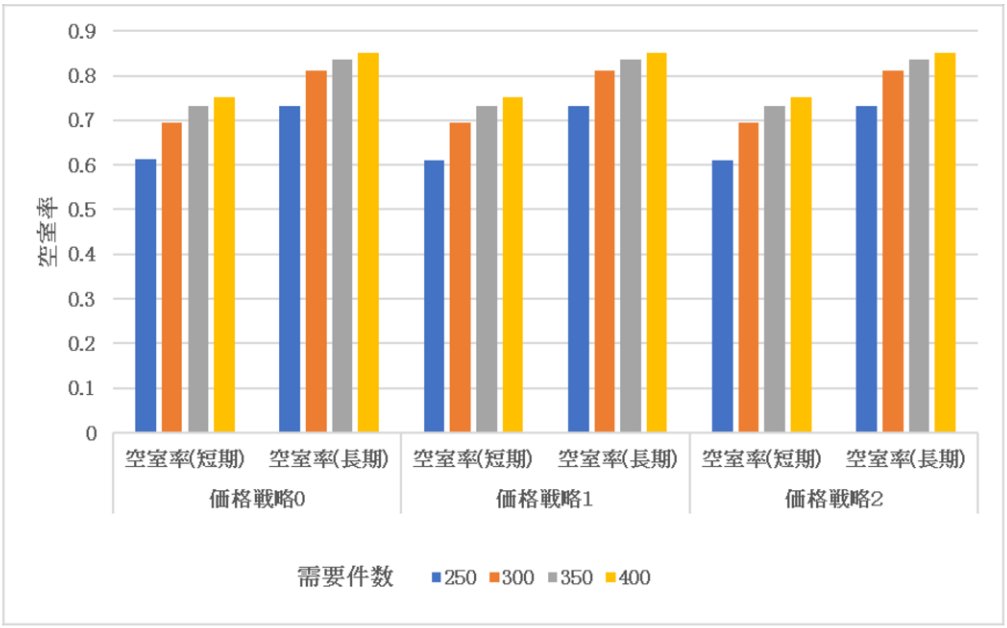


図 5 部屋の空室率（需給拡大）

Note:図中において、左段から大きく価格戦略 0 のとき、右段は価格戦略 2 のときの結果を示している。各結果はそれぞれ、各需要数に対する短期・長期空室率を示している。

従って短期物件は空室率が低いという仮説の探索的研究を試みた。短期の空室率が相対的に低く、入居率が高い結果となった理由として、完全マッチングしなかったエージェントが希望の部屋よりもレベルを１，２つ下げることにより入居率が上がるからである。つまり 1 回目のマッチングで殆どのエージェントがマッチすること、シミュレーション設定の割合に忠実に当てはまったことである。また、マッチする貸手エージェントがない場合は機会損失としてカウントするため、次のチャンスがなく棄却される。そして、需要数と供給数のバランスが一番大事であり、需要に対して供給が少なすぎると空室率が上昇する。そのため、需要数は 1 日 250 件から増加させ、供給数は 3600 で変えずに推移した方がエージェントの挙動が分かり易い。

第５節　空室数の変化

本分析では、シミュレーションを通じて空室数の変化とマッチングに要する日数の解明を試みた。分析の結果、(1)価格戦略無よりも価格戦略有の方が価格のレベルに均等にマッチングしたこと、(2)価格帯は市場の平均に近いこと、(3) 価格変動数は 33 日程度でマッチングを完了したなどのことを見出した。第 4 節までは 1 回のみの完全マッチングであったが、本分析は全てのエージェントがマッチングするまで挙動を行った。これらは、短期賃貸が比較的マッチングしやすいという仮説に貢献するものである。下記に本分析の過程とルールについて説明する。

ステップ１　すべてのエージェントは最も望ましい部屋に応募する。
「（１回目のマッチング）のテーブル」を参照して、自分のレベルに応じた部屋を確率的に選択する。例えば、レベル 1 の人は、4000 円を 23%の確率、5000 円を 20%の確率、6000 円を 5%の確立で選択する。
ステップ２　供給側はエージェントを選択しない設定、部屋が空いていれば貸す。
ステップ３　ステップ 1 で選択した部屋が空いていなければ、「ステップ２（望ましいレベルのマッチング)」のテーブルを参照して、確率的に選択する。但し、ステップ 1 で選択した部屋以外を選択するようにする。
ステップ４　部屋が空いていなければ、「ステップ３（２回目のマッチング、要望に近いものにマッチング)」のテーブルを参照して、確率的に選択する。
※ステップ 3 で選択した部屋以外を選択するようにする。選択した部屋が空いていなければ、ステップ 3 に戻る。

表 3 ステップ１（１回目のマッチング）

		1	2	3	4	5	6
レベル 1	4000 円	23%	34%	3%	10%	15%	20%
レベル 2	5000 円	20%	16%	15%	10%	15%	20%
レベル 3	6000 円	5%	15%	40%	15%	20%	15%
レベル 4	7000 円	11%	4%	30%	15%	20%	15%
レベル 5	8000 円	18%	3%	3%	20%	10%	10%
レベル 6	9000 円	23%	28%	9%	20%	10%	10%

Note:表中において、左段から大きく価格レベル 1~6 のとき、右段はエージェントが 1~6 のときの結果を示している。

表 4 ステップ 2（望ましいレベルのマッチング）

		1	2	3	4	5	6
レベル 1	4000 円	20%	15%	30%	34%	10%	3%
レベル 2	5000 円	20%	15%	20%	16%	10%	15%
レベル 3	6000 円	15%	20%	5%	15%	15%	40%
レベル 4	7000 円	15%	20%	10%	4%	15%	30%
レベル 5	8000 円	10%	10%	15%	3%	20%	3%
レベル 6	9000 円	10%	10%	20%	28%	20%	9%

表 5 ステップ 3（2 回目のマッチング、要望に近いものにマッチング）

		1	2	3	4	5	6
レベル 1	4000 円	25%	25%	28%	19%	25%	15%
レベル 2	5000 円	16%	6%	11%	8%	22%	18%
レベル 3	6000 円	19%	27%	9%	24%	10%	21%
レベル 4	7000 円	17%	14%	23%	16%	17%	18%
レベル 5	8000 円	10%	21%	16%	24%	5%	15%
レベル 6	9000 円	13%	7%	13%	9%	21%	13%

図 6 では上記のシミュレーションを 100 回実行した際の結果を示している。貸手エージェント数は 3600 件に設定し、1~6 の価格レベルがどの程度マッチングしたかをバブルチャートで示した。レベル 1~6 は 4000 円から 9000 円の価格レンジである。各々の円は上から下へ空室数が減少している。円の大きさは空室数の割合を示している。価格戦略無の場合は、4~6 に空室が集中している。一方で、図 7 の価格戦略有では、1~3 に空室が集中している結果となった。戦略有の方が全体的に均等な挙動が確認された。これは、レベル 6 でマッチングしなかったエージェントに対して理想に近い部屋にマッチ

ングするよう価格を－2 した結果、レベル 1~3 に集中したものと考えられる。

2014 年から毎年実施されているＡカードホテルシステムによるアンケート調査によると、東京と大阪の宿泊代のボリュームゾーンは５年連続で 9,000 円～1 万円で、全体の 25～26％で推移。次に多い価格帯が 7,000 円～8,000 円で、同 21～22％を占める。一方で東京・大阪以外では、7,000 円～8,000 円が 25～27％で推移。地方都市での出張による宿泊代が、東京・大阪に比べ、約 2,000 円安くなっている。この次に多い価格帯は 9,000 円～1 万円で同 19～20％となる。
東京・大阪以外では、2018 年に限って 6,000 円～7,000 円の価格帯が、前年より 3 ポイント増の 21％となり、2 番目に多い価格帯になっている。5,000 円～6,000 円の価格帯も 3 ポイント増の 11％を占める。そのため、7000 円～9000 円のボリュームゾーンに人が集中しやすいと考えられる。

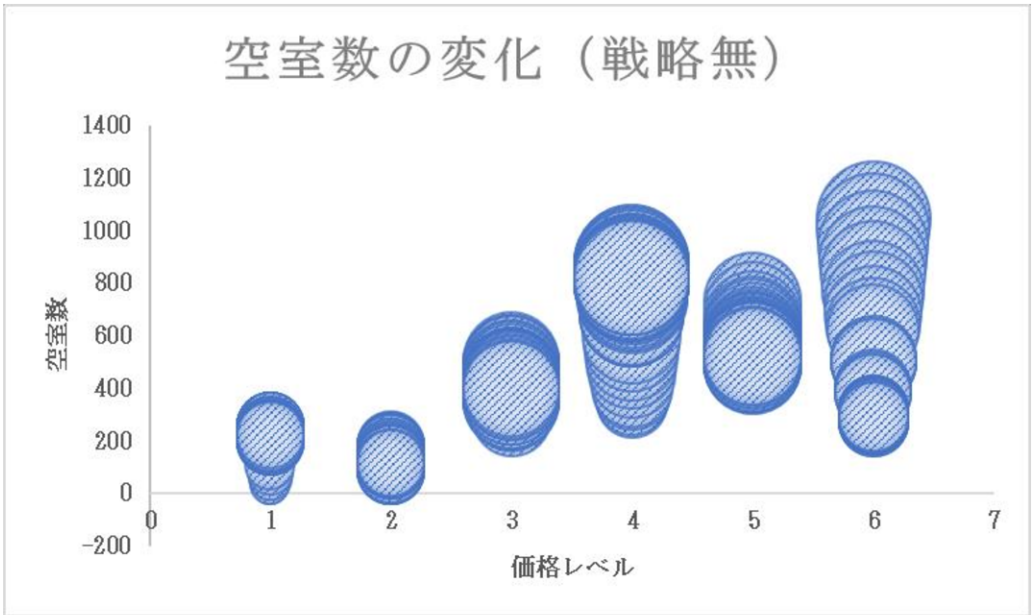


図 6　空室数の変化（価格戦略無）

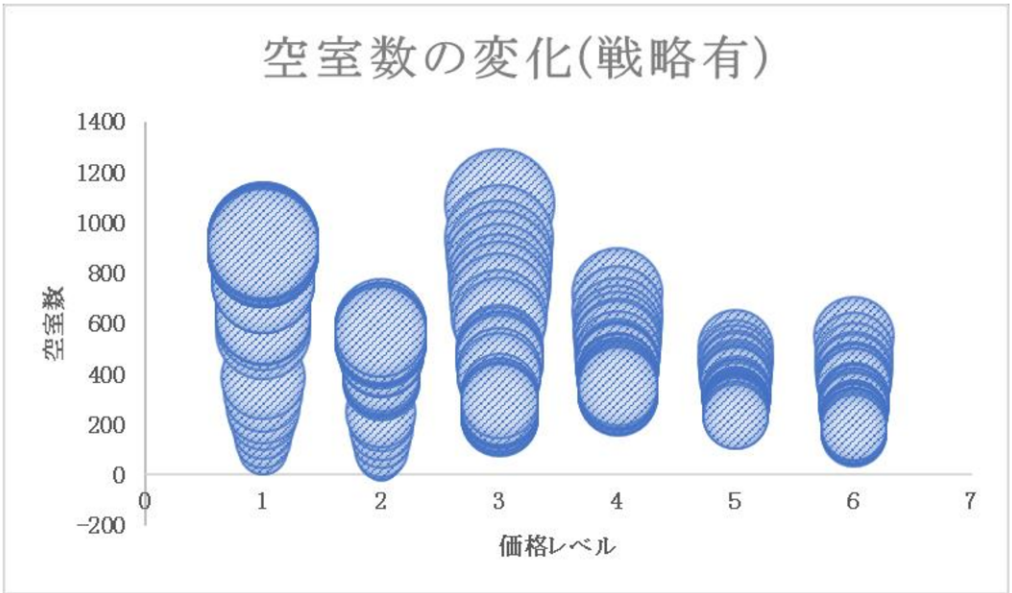


図 7　空室数の変化（価格戦略有）

図 8 では価格変動数の推移を示した。33 日程度までは 3500 の価格変動数が見られたが、33 日以降は停止している。これは、全エージェントがマッチングするまで 33 日程度要することを表している。短期の空室率が相対的に高い結果となった理由として、貸手エージェントが全てマッチングするまで挙動を行うよう変更した点が考えられる。

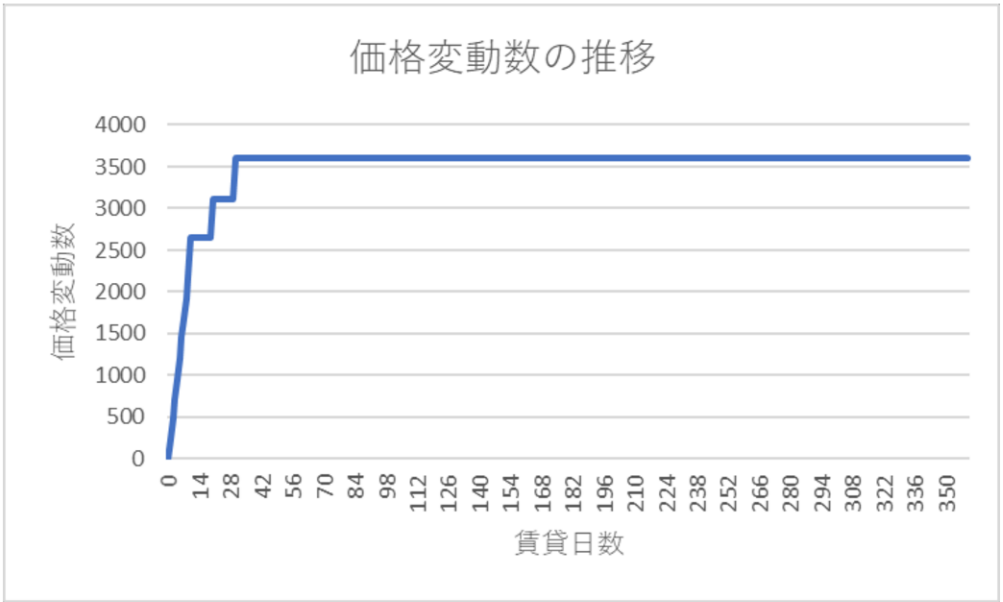


図８　価格変動数の推移

第 6 章　価格の弾力性

価格の弾力性とは、価格の変動によって、ある製品の需要や供給が変化する度合いを示す数値のことを指す。本分析ではホテルの価格変動によって需要や供給が変化するか検討した。分析結果は、価格の中間点で需要の価格弾力性を計算すると、非弾力的ということを見出した。価格帯によって需要が変化したり、最適価格を求める詳細な分析は今後の課題である。まずは比較対象の 2 点の中間を取る中間点の計算方法を試した。最低価格の 4000 円と最高価格の 9000 円の価格の中間点は 6500 円である。変化量をこの中間点で割ると、変化率は(9000 円-4000 円)/6500 円×100=76.92%となる。同様に需要量の変化率を求める。需要数の中間点は(250+400)/2=325 となる。変化量をこの中間点の 325 で割ると(400-250)/325×100=46.15%となる。次に、需要の価格弾力性の計算式を(－需要の変化率%)/(価格の変化率%)とする。上記で求めた需要の変化率と価格の変化率を当てはめると、-46.15%/76.92%=-0.59 となる。一般的には、需要の価格弾力性は 1 未満の場合は非弾力的と呼ぶ。そのため、中間点の計算式で求めた結果は価格の中間点は非弾力的となった。これは、ホテルの部屋は非弾力な財であり、いくら価格が上昇しても部屋という区画を増やすことはできないため、任意に供給量を増加することができない可能性がある。需要を考える際に期間を長く取るほど、需要の価格弾力性は大きくなる傾向にある。それに対して、少しの期間を分析対象にすると、比較的非弾力的になる傾向がある。なお、価格帯によって需要が変化したり、最適価格を求める詳細な分析は今後の課題である。

第7章　まとめ

本研究では、ビジネスホテルの価格戦略に関する研究のため、エージェントベースモデルの構築を試みた。具体的には、貸手エージェントと借手エージェントの需要と供給レベルを作成し、ホテルの入居率に関する基礎的な分析を実施した。分析の結果、(1)短期賃貸は先に埋まること、(2)価格戦略に効用があること、(3)価格のボリュームゾーンは市場の平均値に近いことを見出した。これらは、先行研究に基づく仮説とモデルが有用であること、そして価格戦略を導入した独自性に貢献すると考える。また、本研究ではボトムアップで政策評価をし、モデルが有効なツールであることを探索的に検討したことも貢献と考える。今後の課題として、部屋の質などの指標を追加すること、価格弾力性の更なる検討と最適価格を求めることが挙げられる。

参考文献

Japan National Tourism Organization. (2019b). Hounichi gaikyakusuu no suii. Japan National Tourism Organization. Retrieved from https://statistics.jnto.go.jp/graph/#graph--inbound--travelers--transition (October 10, 2019).（日本政府 観光局（2019b）.「訪日外客数の推移」『日本政府観光局』）

Evgueni Vinogradov, Birgit Leick, Bjørnar Karlsen Kivedal,
An agent-based modelling approach to housing market regulations and Airbnb-induced tourism, Tourism Management, Volume 77, 2020, 104004, ISSN 0261-5177, https://doi.org/10.1016/j.tourman.2019.104004.
http://www.sciencedirect.com/science/article/pii/S026151771930202X

Barron, K., Kung, E., & Proserpio, D. (2018). The sharing economy and housing affordability: Evidence from Airbnb. Available at: SSRN: https://ssrn.com/abstract ¼3006832 https://doi.org/10.2139/ssrn.3006832. Botsman, R., & Rogers, R. (2010). What's mine is yours. London: Collins.

Deng, Y., Gabriel, S. A., & Nothaft, F. E. (2003). Duration of residence in the rental housing market. The Journal of Real Estate Finance and Economics, 26(2–3), 267–285. https://doi.org/10.1023/A:1022987010545.

Nicholls, S., Amelung, B., & Student, J. (2017). Agent-based modeling: A powerful tool for tourism researchers. Journal of Travel Research, 56(1), 3–15. https://doi.org/ 10.1177/0047287515620490.

Edelman, B. G., & Geradin, D. (2015). Efficiencies and regulatory shortcuts: How should we regulate companies like Airbnb and Uber. Stanford Technology Law Review,19,293–328.https://law.stanford.edu/publications/efficiencies-and-regulatory-shortcuts-how-should-we-regulate-companies-like-airbnb-and-uber/. (Accessed 29 May 2019).

Gurran, N., & Phibbs, P. (2017). When tourists move in: How should urban planners respond to Airbnb? Journal of the American Planning Association, 83(1), 80–92. https://doi.org/10.1080/01944363.2016.1249011.

Pizzitutti, F., Mena, C. F., & Walsh, S. J. (2014). Modelling tourism in the galapagos islands: An agent-based model approach. The Journal of Artificial Societies and Social Simulation, 17(1), 23–33. https://doi.org/10.18564/jasss.2389.

Malerba, F., Nelson, R. R., Orsenigo, L., & Winter, S. G. (1999). History friendly models of

industry evolution: The computer industry. Industrial and Corporate Change, 8(1), 3–41. https://doi.org/10.1093/icc/8.1.3.

遠藤薫, 寺野隆雄, 佐藤彰洋, 栗原聡, 倉橋節也, 田中覚. 社会シミュレーション： 世界を「見える化」する /. 東京: 東京電機大学出版局, 2017. Print.

山影進, and 服部正太. コンピュータのなかの人工社会：マルチエージェントシミュレーションモデルと複雑系 /. 東京: 構造計画研究所, 2002. Print.

謝辞

本論文の執筆に当たり、多くの方々にご支援いただきました。本研究のためにご協力いただきました NTT データシステム嶋田様を始めとする研究者の皆様に心から感謝いたします。献身的にサポート頂いたことに報いるように将来の研究や仕事に活かします。
中間発表および修士論文発表会では、高橋大志教授、林高樹教授、山尾佐智子准教授より貴重なご指導とご助言賜りましたことを感謝申し上げます。厳しいご意見も研究の発展のために必要であることを感じ、改善に向けて努めてまいりました。本論文は決して完全なものではありませんが、読者の方が何かしらの implication を得ることができるよう切に願います。
主指導教員である高橋教授には、書生の意向も汲み取って頂きながら研究の方向性から論文執筆まで多くのご指導いただきましたことを心より感謝申し上げます。また、所属する高橋ゼミの皆様には互いに切磋琢磨しながら 1 年間を過ごせたことをお礼申し上げます。今後のご活躍を心より祈念するとともに、ゼミの益々の発展を期待しております。最後に、混迷の時期にも院生生活と IP 留学を温かく励ましてくれた家族と親友たちの愛情と支えに感謝申し上げます。

Appendix

A.1　プログラムコード

```
# -*- Mode: Python; coding: utf-8-dos; -*-

import sys; sys.path.insert(0, u"C:¥¥Program Files¥¥Mathematical Systems Inc¥¥S-Quattro Simulation System V6¥¥lib3")
from psim import *
from graph import graphsendclient as graphclient
import datetime
import hashlib
graphclient.GraphServer.createServer(uri='http://127.0.0.1:59721')

initialize()
```

```
setGlobalSeed(0)

# カスタムコード
# なし

# シミュレーションパラメータの定義
class SimulationParam:
    def __init__(self,
                 inputDir = u"input¥¥default",
                 outputDir = u"output¥¥default"):
        pass

# シミュレーターの定義
class Simulator (SimulatorBase):
    def __init__(self,
                 param = SimulationParam(),
                 inputDir = u"input¥¥default",
                 outputDir = u"output¥¥default",
                 showGraph = True,
                 start = lambda self: None,
                 warm = lambda self: 0.0,
                 until = lambda self: 360.0,
                 odesolver = lambda self: dopri54Solver(dt = 0.1, rtol = 0.001, atol = 1e-06),
                 dataDir = u"data"):
        SimulatorBase.__init__(self, param, inputDir, outputDir, showGraph, start, warm, until,
odesolver, dataDir)

        # 資源の定義

        # オブジェクトの生成

        # リンクの生成
        # 環境 の定義

        class Environment (EnvironmentBase):
            def __init__(self, **keys):
                EnvironmentBase.__init__(self, u"環境", **keys)
            def initAfter(self, **keys):
```

```
            """環境生成後に呼ばれる。"""
            pass
        def initAttribute(self, agent):
            """新規のエージェントが作成されたら、必ずこのメソッドが呼ばれる。

              エージェントは必ずひとつの環境に属す。

              各エージェントに環境に固有の属性を設定する。"""
            pass
        def setPosition(self, agent, pos):
            """エージェント agent の環境に固有の位置属性を更新する。

              エージェントの環境上の位置属性が変わったら、必ずこのメソッドが呼ばれる。

              pos は、環境上の位置属性(座標やノード番号など)であり、そのフォーマットは、環境ごとに異なる。"""
            pass
        def getPosition(self, agent):
            """エージェント agent の環境に固有の位置属性を返す。

              環境上の位置属性(座標やノード番号など)であり、そのフォーマットは、環境ごとに異なる。"""
            pass
        def getRandomPosition(self):
            """環境内のランダムな位置属性を返す。

              環境上の位置属性(座標やノード番号など)であり、そのフォーマットは、環境ごとに異なる。"""
            pass
        def draw(self, panel):
            """この環境を panel 上に描画する。"""
            pass
        def drawAgents(self, panel, agents):
            """この環境上で、エージェントのリスト agents を panel 上に描画する。"""
            pass
        def custom(self):
            pass

    self.eEnvironment = Environment(simulator = self)
```

```
        self.addEnv(self.eEnvironment)
        # 借手エージェント の定義

        class AsynchronousAgentSet (AsynchronousAgentSetBase):
            def _defAgent(self):
                class _Agent (AsynchronousAgentBase):
                    def initAfter(self, *args, **keys):
                        """エージェントが作成され、environmentBase.initAttribute の呼び出し後に、このメソッドが呼ばれる。"""
                        # 属性の設定
                        # エージェントの割合
                        # salary 40%
                        # traveler 30%
                        # couple 10%
                        # inbound 20%
                        self.type = next(empiricalDistribution(weights = [[40,"salary"],[30,"traveler"],[10,"couple"],[20,"inbound"]]))

                        # レベルの設定
                        # salary 2~6
                        # traveler 3~5
                        # couple 2~5
                        # inbound 1~3
                        if self.type == "salary":
                            self.level = next(sample([2,3,4,5,6]))
                        elif self.type == "traveler":
                            self.level = next(sample([3,4,5]))
                        elif self.type == "couple":
                            self.level = next(sample([2,3,4,5]))
                        elif self.type == "inbound":
                            self.level = next(sample([1,2,3]))

                        # 滞在日数(短期 short, 長期 long)の設定
                        if self.type == "salary":
                            self.stayType = next(empiricalDistribution(weights = [[99,"short"],[1,"long"]]))
                        elif self.type == "traveler":
                            self.stayType = next(empiricalDistribution(weights =
```

```
[[99,"short"],[1,"long"]]))
                elif self.type == "couple":
                    self.stayType = next(empiricalDistribution(weights = [[99,"short"],[1,"long"]]))
                elif self.type == "inbound":
                    self.stayType = next(empiricalDistribution(weights = [[95,"short"],[5,"long"]]))


                # 滞在日数の設定
                # short 1 日 30%, 2 日 30%, 3~5 日 10%, 6~9 日 5%
                # long 30 日 25%, 60 日 25%, 90 日 25%, 180 日 25%
                if self.stayType == "short":
                    self.stay = next(empiricalDistribution(weights = [[30,1],
                                                                      [20,2],
                                                                      [10,3],
                                                                      [10,4],
                                                                      [10,5],
                                                                      [5,6],
                                                                      [5,7],
                                                                      [5,8],
                                                                      [5,9]
                                                                      ]))
                elif self.stayType == "long":
                    self.stay = next(empiricalDistribution(weights = [[25,30],
                                                                      [25,60],
                                                                      [25,90],
                                                                      [25,180]
                                                                      ]))


                # 宿泊先を探す(マッチング)
                # 触らなくて良い
                cond = None
                for agent in self.simulator.aSynchronousAgentSet.agents:
                    if self.stayType != agent.stayType:
                        continue
                    if agent.room == True:
                        continue
```

```
                #       if self.level == agent.level:
                        if agent.level <= self.level:
                            if self.level - agent.level <= 2:
                                if cond is None:
                                    cond = agent
                                elif self.level - cond.level > self.level - agent.level:
                                    cond = agent

                        if cond is not None:
                            cond.room = True
                            cond.stayTimer = self.stay



                def run(self):
                    """エージェントのプロセス処理"""
                    # 消滅
                    self.agentset.remove(self)

                    yield alwaysTrue()
                def custom(self):
                    pass
        return _Agent
    def initAfter(self, **keys):
        # 需要数（エージェント数）
        N = 90000


        # 以下は触らなくて良い
        # 需要が起きる間隔
        self.t = 360/N

        # 貸手エージェントを生成
        def genAgent():
            while True:
                self.generateAgents(1)
                t = next(exponentialDistribution(mean=self.t))
                yield pause(t)
```

```
            activate(genAgent)()
        def start(self):
            """非同期エージェント集合の動作を開始する。"""
            for agent in self.agents:
                activate(agent.run)()
        def view(self):
            """エージェントの描画を開始する。"""
            pass
            #interval = 1 #  表示間隔
            #screen = self.getAgentScreen(interval = interval, xlim = None, ylim = None)
            #screen.addAgentSet(self)
            #screen.start()
        def custom(self):
            pass

    self.aAsynchronousAgentSet = AsynchronousAgentSet(u"借手エージェント",
self.eEnvironment, simulator = self)
    self.addAgentSet(self.aAsynchronousAgentSet)

    def startAsynchronousAgentSet():
        self.aAsynchronousAgentSet.start()
        self.aAsynchronousAgentSet.view()

    self.procs.append(startAsynchronousAgentSet)

    # 貸手エージェント の定義

    class SynchronousAgentSet (SynchronousAgentSetBase):
        def _defAgent(self):
            class _Agent (SynchronousAgentBase):
                def initAfter(self, *args, **keys):

                    # レベルの設定
                    # エージェントの割合
                    # 1: 30%, 2: 20%, 3~4: 15%, 5:10%, 6: 10%
                    self.level = next(empiricalDistribution(weights =[[10,1],
                                                                      [10,2],
                                                                      [20,3],
```

```
                                                    [30,4],
                                                    [20,5],
                                                    [10,6]]))

                    # 物件(短期 short, 長期 long)の設定
                    # エージェントの割合
                    # short: 80%, long20%
                    self.stayType = next(empiricalDistribution(weights =
[[60,"short"],[40,"long"]]))

                    if self.stayType == "short":
                        self.agentset.nshort += 1
                    else:
                        self.agentset.nlong += 1

                    # 価格戦略(ミニマム空室期間)
                    # short: 10 日, long: 30 日
                    # これを超えると値下げ
                    if self.stayType == "short":
                        self.minimum =  next(sample([1,2,3,4,5,6,7,8,9]))
                    elif self.stayType == "long":
                        self.minimum =  next(sample([10,20,30]))



                    # 以下は触らなくていい
                    # 課税による価格アップ
                    self.level += self.agentset.priceup



                    # 宿泊エージェントがいるか
                    self.room = False

                    # 宿泊日数
                    self.stayTimer = 0

                    # 空室日数
                    self.emptydays = 0
```

```
            def step(self):
                """エージェントのステップ処理"""

                if self.stayTimer == 0 and self.room == True:
                    self.room = False
                    self.stayTimer = 0

                if self.emptydays is not None:
                    if self.emptydays >= self.minimum:
                        self.level -= self.agentset.pricedown
                        #  レベル 0 は 1 のまま
                        if self.level <= 0:
                            self.level = 1
                        else:
                            if self.agentset.pricedown > 0:
                                self.agentset.levelChange += 1
                        self.emptydays = None

                if self.room == True:
                    self.stayTimer -= 1


                # 空室日数のカウント
                if self.emptydays is not None:
                    self.emptydays += 1

                pass
            def custom(self):
                pass
        return _Agent
    def initAfter(self, **keys):
        self.interval = 1.0 #  ステップ間隔
        self.freezeAddAgent = True #  エージェントの追加を次ステップまで凍結する
        self.freezeDelAgent = True #  エージェントの削除を次ステップまで凍結する

        #  下げ幅(0 にすれば価格戦略なし)の設定
        self.pricedown = 2
```

```
        # 課税による上げ幅(0 にすれば課税なし)の設定
        self.priceup = 2

        # 貸手エージェントの数の設定
        n = 3600

        self.levelChange = 0

        # short の数
        self.nshort = 0

        # long の数
        self.nlong = 0


        # 以下は触らなくていい
        self.generateAgents(n)
        self.agentFreezeVars = [""] # エージェントの凍結する変数リスト

        # 空室数の状態
        self.monitor = Monitor(["時刻","空室数","空室割合"], ["f","i","f"], name="空室状態
(トータル)")
        self.simulator.addMonitor(self.monitor)

        self.monitor10 = Monitor(["時刻","価格変動数"], ["f","i"], name="価格変動")
        self.simulator.addMonitor(self.monitor10)

        # 空室数の状態
        self.monitor2 = Monitor(["時刻","空室数","空室割合"], ["f","i","f"], name="空室状態
(短期)")
        self.simulator.addMonitor(self.monitor2)

        # 空室数の状態
        self.monitor3 = Monitor(["時刻","空室数","空室割合"], ["f","i","f"], name="空室状態
(長期)")
        self.simulator.addMonitor(self.monitor3)

        # レベル別の空室状態
        self.monitor4 = Monitor(["時刻","レベル","空室数","部屋数","空室割合"],
```

```
["f","i","i","i","f"], name="レベル別空室状態")
                self.simulator.addMonitor(self.monitor4)

                # エージェント別レベル
                self.monitor5 = Monitor(["時刻","エージェント ID","レベル"], ["f","i","i"], name="エージェント別レベル")
                self.simulator.addMonitor(self.monitor5)

            def step(self):
                """エージェント集合のステップ処理を行う。"""
                # エージェント集合へのエージェントの追加削除を凍結
                self.freeze()
                # 全エージェントを凍結
                if len(self.agentFreezeVars) > 0:
                    for agent in self.agents:
                        agent.freeze(self.agentFreezeVars)
                # 全エージェントのステップ処理
                for agent in self.agents:
                    agent.step()
                # 全エージェントの凍結されていた変更を反映
                if len(self.agentFreezeVars) > 0:
                    for agent in self.agents:
                        agent.melt()
                # エージェント集合への凍結されていたエージェントの追加削除を反映
                self.melt()

                count = 0
                countS = 0
                countL = 0

                count_Level = {}
                count_LevelNum = {}

                for agent in self.agents:
                    if agent.level not in count_Level.keys():
                        count_Level[agent.level] = 0
                        count_LevelNum[agent.level] = 0
                    if agent.room == False:
                        count += 1
```

```
                    if agent.stayType == "short":
                        countS += 1
                    else:
                        countL += 1
                    count_Level[agent.level] += 1
                count_LevelNum[agent.level] += 1
                self.monitor5.observe(now(), agent.agentid, agent.level)

            self.monitor.observe(now(),count,count/len(self.agents))
            self.monitor2.observe(now(),countS,countS/self.nshort)
            self.monitor3.observe(now(),countL,countL/self.nlong)



            for k in count_Level.keys():
                self.monitor4.observe(now(), k, count_Level[k],
count_LevelNum[k],count_Level[k]/ count_LevelNum[k])

            self.monitor10.observe(now(),self.levelChange)

        def start(self):
            """同期エージェント集合の動作を開始する。"""
            # エージェント集合のプロセス
            def proc():
                try:
                    while True:
                        # ステップ処理
                        self.step()
                        yield pause(self.interval)
                except FinishException as e:
                    raise e
                except Exception as e:
                    fatal("[%s]エージェントエラー: エージェント集合のプロセスでエラーが
発生しました。" % self.name,
                          "error in agentset process", tb = True)
            # エージェント集合のプロセスを起動
            activate(proc)()
        def view(self):
            """エージェントの描画を開始する。"""
            pass
```

```
                #interval = 1 # 表示間隔
                #screen = self.getAgentScreen(interval = interval, xlim = None, ylim = None)
                #screen.addAgentSet(self)
                #screen.start()
            def custom(self):
                pass

        self.aSynchronousAgentSet = SynchronousAgentSet(u"貸手エージェント",
self.eEnvironment, simulator = self)
        self.addAgentSet(self.aSynchronousAgentSet)

        def startSynchronousAgentSet():
            self.aSynchronousAgentSet.start()
            self.aSynchronousAgentSet.view()

        self.procs.append(startSynchronousAgentSet)
    def run(self, **args):
        # オブジェクトの起動
        try:
            _sim_before(self)
        except:
            pass
        pass
        def startProcs():
            for proc in self.procs:
                proc()
            yield alwaysTrue()
        activate(startProcs)()
        # シミュレーション開始
        start(until = self.until, odesolver = self.odesolver, **args)
        # 結果の保存
        self.save()
        self.flush()
        try:
            _sim_after(self)
        except:
            pass
    def objective(self):
        mon = Monitor(name = u"空室状態(トータル)", basedir = self.outputDir)
```

```
        return mon[u"空室割合"][mon.nrow()-1]
    def objective2(self):
        mon = Monitor(name = u"空室状態(短期)", basedir = self.outputDir)
        return mon[u"空室割合"][mon.nrow()-1]
    def objective3(self):
        mon = Monitor(name = u"空室状態(長期)", basedir = self.outputDir)
        return mon[u"空室割合"][mon.nrow()-1]

if __name__ == "__main__":
    # 実行時の処理

    simulator = BatchSimulator(Simulator, SimulationParam,
        inputDir = u"input¥¥default",
        outputDir = u"output¥¥default",
        iters = [

        ],
        maxcount = lambda self: 100,
        outputs = [
            [u"空室率(トータル)", 'f', lambda self: self.objective()],
            [u"空室率(短期)", 'f', lambda self: self.objective2()],
            [u"空室率(長期)", 'f', lambda self: self.objective3()]
        ],
        start = lambda self: None,
        warm = lambda self: 0.0,
        until = lambda self: 360.0,
        odesolver = lambda self: dopri54Solver(dt = 0.1, rtol = 0.001, atol = 1e-06),
        keepResult = True,
        repSetting = {"minRep": 10, "maxRep": 10, "errRatio": 0.05,"confLevel": 0.95},
        seed = 0)
    simulator.run()
```