2017 年度修士論文

# 交通行動推定のためのアクティビティモデルと異種交通データからなる状態空間モデルの平滑化

Smoothing of State Space Model composed of Activity-based Model and Various Transportation Data for the Estimation of Travel Behavior

37-166046 福冨 義章

主査： 布施 孝志 教授

副査： 関本 義秀 准教授

| 署名 | 日付 | 印 |
|---|---|---|
| | | |
| | | |

東京大学大学院 工学系研究科 社会基盤学専攻

2018 年 1 月 23 日

# 論文要旨

都市部における社会基盤の設計・計画のため，総合的かつ詳細な交通行動の把握は，今後更に重要性を増していくと考えられる．交通行動把握のための観測として従来からパーソントリップ調査があるが，調査間隔の長さ等の欠点がある．近年では，モバイル空間統計などの高頻度な観測データが出現している他，改正個人情報保護法の施行など，事業者が保有するデータの公共利用に向けた動きも進んでいる．一方で，シミュレーションモデルの開発も行われており，個人の生活行動を考慮することで，より合理的な交通行動の予測を目指すアクティビティモデルが用いられている．また，これらの観測データの補完やシミュレーションの精度改善のため，データ同化による交通行動推定が行われ，アクティビティモデルに対してもデータ同化が有効であることが既に示されている．しかし，アクティビティモデルの特徴である時空間プリズム制約に反する同化が行われることや，使用する観測データが 1 種類のみであることなどの課題があった．そこで本研究は，これらの課題を解決するため，時系列推移を考慮した平滑化手法であるパーティクルスムーザを用いて，アクティビティモデルと異種交通データの統合を行うことで，交通行動の推定を行うことを目的とする．

本研究では，システムモデルにはアクティビティモデルの PCATS を採用する．観測データとしてはモバイル空間統計によるゾーン人口と，交通 IC カードデータによるゾーン別発着別トリップ人数の 2 種類を想定し，これらの同時観測も考慮に入れた上で，パーティクルスムーザで平滑化を行う．状態空間モデルの状態ベクトルは，対象となる全個人の，活動開始・終了時刻，活動場所，活動後の移動開始・終了時刻などで構成される生活行動スケジュールとする．パーティクルスムーザとは，非線形・非 Gauss の状態空間モデルによるデータ同化に用いられるパーティクルフィルタを，平滑化のために発展させた手法であり，前の時刻における粒子の情報を保存しておく拡大状態ベクトルを考えることで実現される．この粒子情報の保存が，PCATS による遷移可能性を保証することになるため，時空間プリズム制約に反する可能性なく観測データを同化することができる．複数種類の観測データを同化する場合でも，時空間プリズム制約は自動的に満たされることになるため，その利用が容易である．

提案手法を，2008 年実施の第 5 回東京都市圏パーソントリップ調査データを全数の真値と見做した，仮想的な東京 23 区に対して適用した．まず，東京 23 区への PCATS 適用にあたって必要となる地域属性データ・交通ネットワーク属性データを整備した上で，真値たる PT 調査データから抽出した疑似 PT 調査データ，モバイル空間統計を想定したゾーン人口データ，疑似 IC カードデータを作成した．次に，これらのデータを用いて提案手法を実行し，真 PT 調査データの再現を試みた．そして，ゾーン人口とゾーン別発着別トリップ人数を指標とした，推定値による真値の再現率で，適用結果を評価した．その結果，従来手法と比較して，時空間プリズム制約に反しないより合理的な推定が行えること，複数種類の観測データの使用がゾーン人口の再現率向上に有効であることを確認した．

今後は，実際の都市圏への手法適用に向けて，データのスパース性への対応や適用対象地域の検討を行うことが，課題として挙げられる．

# 目次

# 図目次

# 表目次

# 1　序論

## 1.1　背景

都市部における社会基盤の設計・計画に際し，個人の交通行動や空間分布を把握することは非常に重要である．現在得られている交通行動に関する情報の例としては，公共交通機関の利用状況や道路交通量等があり，それらをもとに都市交通の現状を捉えることができる．その結果は，交通需要に見合った供給が行われているかといった現況分析や，将来の需要予測に利用することができる．近年では，人々のライフスタイルの変化や環境意識の高まりなどもあり，単に移動することだけでなく，その際の安全性・利便性・快適性を含めた総合的な評価・予測と，それに合わせた施策が求められている．鉄道会社による運行情報アプリの提供や，Web サイト上での交通情報の提供が進んできているのも，そのような総合的な評価への要請の高まりを示している．今後は，利用交通機関や事業者の枠を越え，総合的かつ詳細に交通行動を把握することの重要性が更に増していくと考えられる．

交通行動の全容を正確に知ることはその膨大さもあって不可能であるとは言え，従前よりそのための観測は続けられてきた．その一つが，三大都市圏をはじめとする各地で実施されているパーソントリップ調査である．パーソントリップ調査では，ある一日の個人の行動に対して，各移動の発着地・時刻・目的・交通手段等の交通データに，年代・性別・職業・居住地・世帯人数等の個人属性データが紐づけされて収集される．東京都市圏では，1968 年（昭和 43 年）に初めて行われて以降，2008 年（平成 20 年）の第 5 回まで 10 年ごとに行われている．これらのデータは都市圏における種々の設計・計画に利用されている一方で，大規模調査であるが故の費用，労力，結果が出るまでの時間の長さ，そしてそれらに起因する調査間隔の長さといった問題も抱えている．

近年では，情報技術の発展から，携帯電話や交通系 IC カードなどからも交通行動を観測することが可能となってきており，かつては存在しなかった観測データが出現している．これらの新しい観測データの特徴の一つとして，従来のパーソントリップ調査に比べて，高頻度なデータ取得が可能となっている点が挙げられる．例えば，NTTdocomo のモバイル空間統計では，携帯電話の基地局通信に基づく位置情報を集計し，1 時間ごとのメッシュ人口データを提供している．ここに含まれる個人属性データは年代・性別・居住地のみであるためその点ではパーソントリップ調査には劣るが，その取得頻度の高さを活用した事例が見られる．一例として柏市における都市計画への利用を目指した清家（2013） [17] があり，時間帯別人口動態と土地利用計画の整合性や，拠点振興のためのイベント時の来街状況などを把握する手法としての有用性が期待されている．他にも，Suica や PASMO 等の交通系 IC カードの普及により，都市部の内々交通における公共交通機関利用状況に関しては，各事業者がほぼ非集計のデータを保有していると思われる．法の壁もあり現時点でのこれらの非集計データの入手可能性はまだ高くはない．しかし，公共交通オープンデータ研究会（2015 年より後身の公共交通オープンデータ協議会に名称変更）の設立（2013 年）や，「匿名加工情報」として加工された個人情報の流通・利活用を促進することを盛り込んだ改正個人情報保護法の施行（2017 年）など，事業者が保有する交通データの公共利用に向けた動きは確実に進んでいる．

交通行動の観測手法に進展があった一方で，観測頻度の低さを補ったり，将来の状況を予測したりする目的で，シミュレーションモデルの開発も行われてきた．最も標準的で広く適用されている交通需要予測モデルとして，第二次世界大戦直後にアメリカで開発が始められた四段階推定法がある．しかし，北村（1996） [21] によれば，この方法には幾つかの欠点がある．詳しくは 2.1 で述べるが，例えば，四段階の各モデルが，行動論

的因果関係ではなく，データに内在する統計的相関に基づいて構築されているため，パラメータが時系列間，地域間で，安定している保証が無いという点や，各段階が独立であるため，旅行時間が未定のまま目的地が選択されてしまうといったことが起こり得る点，トリップを単位としているため，トリップ同士の相互作用を考慮できない点などである．なお，トリップとは人が一定の目的を持って出発地から到着地へと移動する単位であり，途中で交通手段を乗り換えたとしても 1 トリップと数える．

上に挙げたような四段階推定法の課題を受け，1970 年代に非集計モデルが提案され，モデルは四段階推定法のようなトリップベースから，ツアーベースを経て，アクティビティベースへと変遷した．ツアーとは自宅→職場→自宅のような連続的なトリップの連なりであり，自宅と職場などの主活動先を主ツアー，主ツアーの途中での寄り道を中間ストップ，主活動からのツアーをサブツアー，帰宅後のツアーを 2 次ツアーなどと定義する．これにより，トリップ同士の関連性を考慮した推定を可能としている．アクティビティベースモデルは，「交通需要は生活行動の派生的な需要である」という点に着目したアプローチであり [16]，就業・就学といった個人の生活行動や，時空間プリズム制約を考慮することで，より合理的な交通行動予測を目指すものである．但し，アクティビティベースモデルでは，パーソントリップ調査等から作成した個人の生活行動データを入力として必要とする．このため，パーソントリップ調査の観測頻度の低さの影響を受け，社会基盤の新規整備やライフスタイルの変化を反映したシミュレーションを行うことは困難である．

そこで，シミュレーションがより実態を反映したものとなるように，実際の交通行動を観測したデータによってシミュレーションを補正することが考えられている．このような，シミュレーションと実際の観測データを統合する手法として，データ同化がある．これは，状態推移を表すシミュレーションであるシステムモデルと，状態量から観測量を抽出する観測モデルからなる状態空間モデルの枠組みを用いてシミュレーションに観測データを統合するもので，シミュレーションの精度・性能を観測データによって改善する，観測データの不足をシミュレーションで補うといった特徴を持つ．データ同化による状態推定の手法の一つである Kalman(1960) [4] が提案した Kalman フィルタは，線形・Gauss 型の状態空間モデルにしか適用できなかった．しかし，非線形・非 Gauss 型にも適用できるパーティクルフィルタが Kitagawa(1993, 1996) [5] [6]，Gordon *et al.*(1993)[2] で提案されて以降は多くの分野に応用され，主に気象学・海洋学分野で発展してきた．

交通分野でも 2010 年代には中村（2013） [19]，淺原（2016） [12] など，パーティクルフィルタを用いた種々の研究が見られるようになった．これらの研究では，シミュレーションによる推定値を観測値で補正することでより高精度な推定が可能となったり，過去のパーソントリップ調査に基づくシミュレーションモデルと最新の観測データを統合して最新の個人位置分布を再現することに成功したりしている．このことから，交通行動の現況把握や将来予測においても，データ同化手法は有効であると言える．アクティビティモデルを用いたデータ同化の既往研究としては，原田（2017） [23] があり，この研究では，千代田・中央・港の都心三区を対象として，アクティビティモデルの PCATS によるシミュレーションと，モバイル空間統計の観測データの統合が，パーティクルフィルタを用いて行われた．その結果，アクティビティモデルに対してもデータ同化手法が一定程度有効であることが示された．

しかしながら，既往研究には大きく分けて 2 つの課題がある．1 つ目は，データ同化のシミュレーションモデルとしてアクティビティモデルを用いると，非現実的な人の動きが推定される点である．観測データを同化する段階で，アクティビティモデルの特徴である時空間プリズム制約に反する動きが推定される可能性があるという問題が，原田（2017）において挙げられている．この場合，原田（2017）ではこの個人へは観測データの同化を行わずにシミュレーションの結果のみを用いて次期へ進むとしているため，観測データによるシミュレーションの補完が行われない場合がある等の課題が残されている．

既往研究の 2 つ目の課題は，利用している観測データの種類の数である．シミュレーションをより実態を反

映したものとするには，複数種類の観測データを統合することが考えられる．しかし，現代では多種多様な観測データが存在するにもかかわらず，複数種類の交通データを統合している既往手法は少ない．また，アクティビティモデルに対して複数種類の観測データを統合する研究は前例が無く，この場合にはそれぞれについて時空間プリズム制約との整合性を考慮する必要があり，困難が伴うことが予想される．

そこで本研究では，これらの課題を克服するため，パーティクルフィルタを平滑化のために発展させた手法であるパーティクルスムーザを用いて，アクティビティモデルに観測した複数種類の交通データを統合することを目指す．平滑化とはシミュレーションによる推定時よりも未来の観測データを利用する統合手法であり，このため，統合に際して時系列の推移を考慮することになる．したがって，時空間プリズム制約を考慮したままで観測データの統合を行うことができ，アクティビティモデルへの観測データ統合や，複数種類の観測データの利用も，容易に行うことができる．

## 1.2 目的

以上の背景から，本研究の目的を，アクティビティモデルへ異種交通データを統合する状態空間モデルの平滑化による交通行動の推定，と設定する．リアルタイム推定を必要としない点を考慮し，各推定時点よりも未来の観測データも用いてフィルタリングを行う．また，既往研究においてはフィルタリングの対象外とされた移動中の個人に対してもフィルタリングを行う他，フィルタリング後の個人位置がプリズム制約外となることも無いため，全個人がフィルタリング対象となる．更に，モバイル空間統計以外の観測データを含め，複数種類の観測データが同時に観測された場合についても適用が可能となるような手法を開発する．この提案手法を，平成 20 年東京都市圏パーソントリップ調査データと，それを全数の真値と見做した疑似都市圏に対して適用する．

## 1.3 論文構成

本論文の構成は以下の通りである．まず第 1 章では，研究の背景と目的について述べた．次の第 2 章で，これまでに提案されてきた交通シミュレーションモデルと，現在存在する交通関連のデータを整理し，それらを統合する手法についての既往研究を紹介する．続いて，第 3 章ではデータ同化の理論について説明する．更に，第 4 章で，シミュレーションと交通データの平滑化による統合手法を提案する．提案した手法を第 5 章で適用し，結果と考察を示す．最後に，第 6 章で成果と今後の課題について述べる．

# 2　既往研究と交通データの整理

本章ではまず，交通需要を予測・再現するためにこれまでに提案されてきたシミュレーションモデルと，現在存在している交通に関連するデータをまとめる．次に，それらを統合した既往研究を紹介する．

## 2.1　交通シミュレーションモデル

これまで，都市の発展に伴い，政策・計画に関する意思決定をはじめとする様々な目的で，交通需要を予測・再現するための種々の手法が提案されてきた．本節ではこれまでに提案されてきた手法をその歴史的背景を交えながら幾つか紹介し，本研究で用いるアクティビティモデルと時空間プリズム制約について説明する．

### 2.1.1　四段階推定法

四段階推定法は，交通需要の推計法として最も標準的に用いられているものである．その利点・欠点を述べるにあたり，手法開発の歴史的背景を簡単に述べる．第二次世界大戦直後のアメリカでその開発が始められ，その後連邦交通局のもとで体系化された．この時代はモータリゼーション・郊外へのスプロールが急激に進行した時代であり，当時の交通計画の主眼は，急増する通勤交通需要を，道路を中心とした交通施設拡充により満たすことにあった．このため，1970 年代以降に交通計画の主眼が各種交通機関の総合的なマネジメントへと移っていくと，四段階推定法はそのニーズに対応しきれなくなってきた．更に，当時の統計・演算能力の限界もあり，可能な限り集約したデータ，単純化された関数形のモデル，少ない変数からなるモデルが要求され，四段階推定法はそれに応えたものとなっている [21].

この手法では，対象地域をゾーンに分け，ゾーン間のトリップを単位として交通需要を推定する．トリップとは，人が一定の目的を持って出発地から到着地へと移動する単位であり，途中で交通手段を乗り換えたとしても 1 トリップと数える．予測モデルは，

**生成**　トリップの総量の予測
**発生・集中**　各ゾーンを出発地・到着地とするトリップ数の予測
**分布**　各 OD（出発地・到着地の組合せ）ごとのトリップ数の予測
**分担**　各 OD のトリップが，利用する交通機関ごとのトリップ数の予測
**配分**　各 OD・交通機関のトリップが，通過する経路ごとのトリップ数の予測

の各段階に分かれており，順に推定が行われる．上に挙げたのはトリップ・インターチェンジモデルと呼ばれ，他には分布推定と分担推定の順が逆になるトリップ・エンドモデルがある．なお，予測モデルが 5 段階あるにもかかわらず「四段階推定法」と呼ばれる理由には諸説あり，生成交通量を発生・集中量の総量であるとして段階に数えないからという説，手法の開発時点では自動車交通のみを扱っており分担の段階が存在しなかったからという説などがある．

この手法の利点は，簡素でわかりやすい点であり，それ故に最も標準的に用いられることとなったと言える．開発時の要請に応えた結果，当時の統計手法，計算機性能，データ管理能力の範囲内で実用的な需要予測を可能にしたという点で，交通計画史上の意義は大きい．一方で，以下のような欠点が指摘されてきた．第一に，データの非効率性である．データをゾーン単位に集約することで情報量が切り捨てられている点が，1960 年代から批判されてきた [8]．次に，行動論的基盤の欠落である．各モデルは行動論的因果関係ではなく，データ

に内在する統計的相関に基づいて構築されているため，仮にモデルがデータによく適合していても，パラメータが時系列間，地域間で，安定している保証が無い [21]．そして最後に挙げられるのが，構造的問題である．

この構造的問題については，より詳しく説明する．まず，トリップ単位の推定であるため，トリップ間の相互作用を考慮できないことが挙げられる．例えば，仕事帰りに買い物に行く，という行動では，実際には買い物場所が帰宅経路の近傍にあることが殆どであろう．しかし，四段階推定法では，その 2 つのトリップを独立に扱うことになる．次の問題として，各段階が独立だという点がある．例えば，分布と分担が独立であるため，トリップ・インターチェンジモデルでは，交通機関を選択する前，即ち旅行時間が未定のまま，目的地が選択される．トリップ・エンドモデルでも，目的地が未定のまま，利用交通機関が選択される．これは現実では考えにくい．更に，生成交通量が最初に決まってしまうため，政策の変化や交通容量増大に伴う誘発需要を考慮できない点や，1 日の中での時間の概念が存在しない点も課題として挙げられる．

### 2.1.2　非集計アプローチ

上述したような四段階推定法の問題の解消のため，個人の意思決定の構造を捉えることで，より高精度でより普遍的なモデルの構築を目指して，1970 年代には非集計モデルによるアプローチが提案された．非集計モデルとは，原則として個人単位で計測された説明変数に基づいて組み立てられるモデルである．ロジット・モデルに代表される非集計選択モデルは離散選択問題一般に適用可能な汎用性を持ち，段階的選択のみならず同時選択構造も採り得る柔軟性を持つ．また，モデル推定に際してデータの持つ情報量が切り捨てられることが少なく効率が高いため，比較的小さなサンプルでの推定が可能である．このため，モデルの開発・更新に必要な調査費用も安価となり，特定の政策評価のためのモデル開発が費用的に可能となったのである [21]．

しかしながら，非集計選択モデルにも課題は存在する．まず，説明変数の測定の困難性が挙げられる．個人属性や選好などの変数は調査によって測定することができるが，交通ネットワークや土地利用情報については正確な非集計値で測定することは事実上困難である．他にも，選択肢集合の同定，観測されていない個人間の差異の取り扱い，誤差項の相関の処理などの課題があり，研究が進められている [21]．そして，非集計モデルの多くにおいても四段階推定法と同様に問題となっているのが，トリップを解析単位としているトリップベースモデルであることである．非集計モデルは四段階推定法に比べて制度を高め，政策対応性を広めた一方で，四段階推定法の代替物として認識され開発されることが多かったが故に，トリップ間の相互作用を考慮できないという課題を残してしまった．

### 2.1.3　アクティビティモデル

交通行動が個人の理性的判断の結果としてもたらされるという観点で見れば，個別のトリップを独立な単位と見做すことは不適切であり，個人の具体的な活動とそれが実行される時空間を推定することで，その派生需要として交通需要を捉える方が合理的である．こうして提案されたのがアクティビティベースという考え方であり，それに基づいた生活行動・交通行動のモデル（アクティビティモデル）が多数提案されている．例として，Adler and Ben-Akiva(1979) [1] などの研究は，離散選択モデルの枠組みで生活・交通行動パターンをモデル化した．また，一日を通じての活動時間やトリップ数等の指標と，個人属性や交通環境との因果関係を構造方程式の枠組みでモデル化した研究として，Robinson *et al.*(1992) [9] などがある．Spernak(1992) [11] などの研究では，活動の実行を時間資源の配分問題として捉えた行動モデルが提案されている．このようにアクティビティモデルの開発が進む中で，マイクロシミュレーションの枠組みで，時間軸上での個人の生活行動の再現が試みられてきた．その一例が Kitamura *et al.*(1995) [7] による AMOS である．

ここで，個人が交通行動を行う際には，種々の制約条件がある．その中でも，Hägerstrand(1970) [3] が個

人の生活行動を時空間座標上の軌跡として表現する手法を提案した際に考えられた時空間プリズム制約は，特に重要である．このような交通行動に際する制約条件の存在を前提に開発されたアクティビティモデルに，藤井ら（1997）[16] による PCATS がある．PCATS では，活動可能な時空間領域，即ち，各移動先において活動に割り当てることができる時間に関する制約（時空間プリズム制約）を考慮してシミュレーションを行う．個人の交通・生活行動を考えるに際して，この時空間プリズム制約が特に重要な制約であることは，西井・近藤（1989）[13] などでも示されている．本研究においても，この制約の重要性を鑑み，アクティビティモデルとして PCATS を採用する．

なお，近年ではアクティビティモデルから更に発展して，交通行動を物理的な移動という側面のみから捉えてきた従来の交通行動に関する研究を極めて一面的だったと反省し，心理学的・社会学的・生理学的・経済学的側面からも交通行動を捉えるべきだという主張が，藤井（2011）[15] によって為されるなど，交通行動・交通計画に対する考え方はより深まってきている．

## 2.2　PCATS の概要

PCATS(Prism-Constrained Activity-Travel Simulator) は，藤井ら（1997）[16] によって開発された，時空間プリズム制約を考慮した上で，個人の生活行動に関する意思決定を時間軸上で逐次再現し，それに伴う生活行動の軌跡を生成する生活行動マイクロシミュレーションモデルである．このモデルにより，通勤時間や勤務時間等の勤務条件の変化，交通速度改善などの生活環境・交通環境の変化に伴う，個人の活動の実行・非実行，活動の順序，場所，内容，利用交通機関の変化を予測することが可能とされている．

### 2.2.1　PCATS の前提

PCATS では，個人の活動を自由活動と固定活動に分けて考える．固定活動とは，就業や就学といった決められた活動であり，活動内容・場所・時間等の活動の要素が予め決められており，個人の自由意思では変更不可能であると仮定される．一方で，自由活動では，活動および移動の各要素を個人の自由意思で決定できる．

個々の活動および移動は以下の要素で表現される．

**活動**　開始時刻，終了時刻，活動内容，活動場所
**移動**　出発地，目的地，交通機関，出発時刻，到着時刻

ここで，自由活動の内容は，在宅，宅外でのスポーツ，宅外での趣味・娯楽，宅外での交際・訪問，外食，日常的な買い物，娯楽としての買い物，固定活動場所周辺での活動に分類される．活動場所，移動の出発地・目的地については，対象地域をゾーン単位に分割して表現される．藤井ら（1997）[16] は京阪神地域 174 市区町村をゾーンとした．交通機関は，公共交通機関・自動車・自転車・徒歩とされ，移動における所要時間・費用・乗換回数などのパラメータは，出発地・目的地・交通機関によって一義的に規定されているものとされている．なお，これらのパラメータを内生化することや，一つのトリップで複数の交通機関を利用する場合の考慮は，藤井らが課題として挙げている．

PCATS の入力・出力は，図 1 の通りである．即ち，個人・世帯属性，交通ネットワーク属性，地域属性，各個人の固定活動スケジュールを入力することで，全自由活動の要素と移動の要素を出力する．ここで，交通ネットワーク属性・地域属性は，活動場所・移動の出発地・目的地に合わせたゾーン単位で入力することになる．

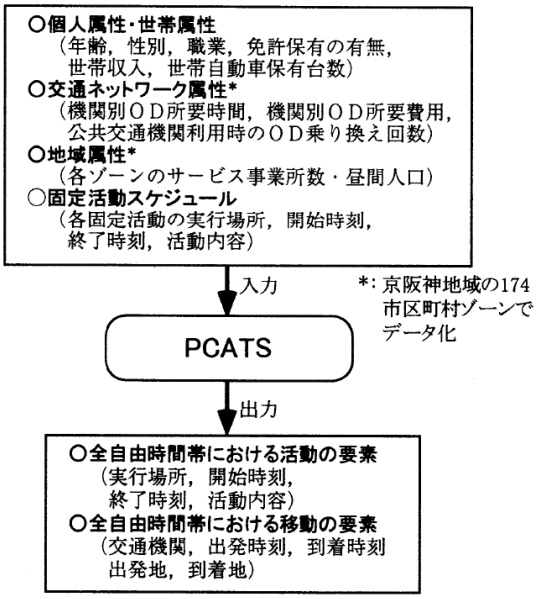


図 1　PCATS の入出力 [16]．PCATS では，仮想個人の 1 日を固定活動時間帯と自由活動時間帯に分け，個人・世帯・交通ネットワーク・地域の属性と，固定活動のスケジュールを入力とし，自由活動と移動の要素を生成する．

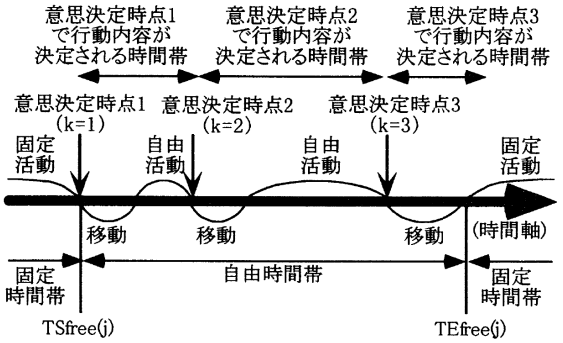


図 2　PCATS における自由時間帯と意思決定時点 [16]．固定活動時間帯が終了し自由活動時間帯へ移行する時点（この図では $k = 1$）と，1 つの自由活動が終了した時点（$k = 2, 3$）が「意思決定時点」となる．

### 2.2.2　PCATS における個人の意思決定過程

PCATS では，図 2 の通り，固定活動時間帯が終了し自由活動時間帯へ移行する時点と，自由活動時間帯において 1 つの自由活動が終了した時点を，「意思決定時点」と定義する．PCATS 内の個人は，各意思決定時点ごとに，次の活動と移動の要素を逐次的に決定していく．図 3 に，ある個人の $j$ 番目の自由活動時間帯における，$k$ 番目の意思決定時点での意思決定過程を示す．段階としては，「まず活動内容を決定し，それに基づいて活動場所とそこまでの交通機関を同時に決定し，最後に活動を実行する時間を決定する」と仮定される．

### 2.2.3　個々の意思決定のモデル化

ここで，意思決定過程における，活動時間の想定，活動内容の選択，交通機関・目的地の選択を再現するためのモデルについて述べる．

#### 活動時間の想定

PCATS では，「個人はそれぞれの活動を行う前に，予めその実行に要する時間を想定し，それに基づいて活動内容や場所を決定している」と仮定している．この，予め活動ごとに想定する所要時間を「想定活動時間」，その分布を「想定時間分布」と呼ぶ．想定活動時間は，7 つの自由活動（在宅，宅外でのスポーツ，宅外での趣味・娯楽，宅外での交際・訪問，外食，日常的な買い物，娯楽としての買い物）それぞれについて，以下のような式で示される Duration Model を用いてモデル化される．

$$y = Y_0 \exp(BX) \tag{1}$$

但し，

$y$ : 想定活動時間
$Y_0$ : 標準個体の想定活動時間
$B$ : パラメータベクトル
$X$ : 説明変数ベクトル

である．なお，$Y_0$ の分布としては Weibull 分布が仮定されている．説明変数は，個人・世帯属性（年齢・性別・職業・世帯収入・免許保有有無・世帯保有自動車数），活動パターン特性（現在時刻と次の固定活動開始時刻の中間時刻・その活動以前に当日に同活動を実行した累計時間・次の固定活動開始時刻と現在時刻の差から最低移動時間を差し引いた時間・現在地と次の固定活動場所の組合せを示すダミー変数）である．

### 活動内容の選択

活動内容選択は，図 4 に示す構造の Nested Logit Model でモデル化される．説明変数は，個人・世帯属性，現在地と次の固定活動場所の組合せを示すダミー変数，各自由活動の活動時間がプリズム制約により規定される最大活動時間よりも短い確率（$= ProbL$）である．ここで，$ProbL$ は以下の通り求められる．まず，各時

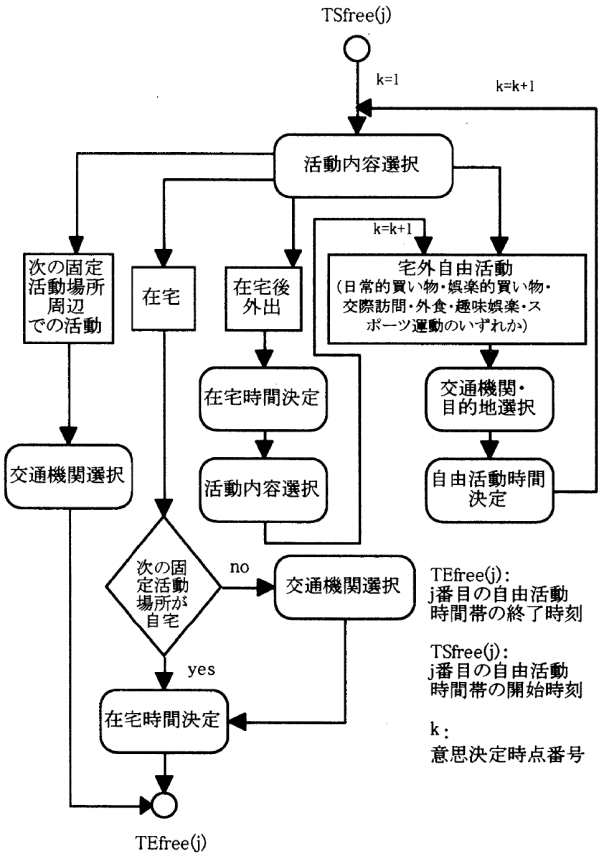


図 3　PCATS 上の個人の $j$ 番目の自由活動時間帯における，$k$ 番目の意思決定時点での意思決定過程 [16]．まず活動内容を決定し，それに基づいて活動場所とそこまでの交通機関を同時に決定し，最後に活動を実行する時間を決定する．

点における，選択肢活動ごとの最大活動時間 $D_{\text{free}}$ を，

$$D_{\text{free}} = TS - TN - D_{\text{od}} \tag{2}$$

とする．但し，

$TS$ : 次の固定活動の開始時刻
$TN$ : 現在時刻
$D_{\text{od}}$ : 最も早く到達できる交通機関での，現在地から次の固定活動場所までの移動時間

である．そして，想定時間分布に基づいて，それぞれの活動時間が $D_{\text{free}}$ を超過しない確率を求め，それを $ProbL$ とする．なお，$D_{\text{free}}$ が負の値をとる活動は，選択肢集合から削除される．これが，プリズム制約の考慮を表している．

#### 交通機関・目的地の選択

交通機関・目的地選択は，図 5 に示す構造の Nested Logit Model でモデル化される．説明変数は，個人・世帯属性，目的地の地域属性，交通ネットワーク属性，現在地から目的地を経て次の固定活動場所までの移動最小時間，活動時間が目的地で実行可能な最大活動時間よりも短い確率（$= ProbG$）である．ここで，$ProbG$ は，$ProbL$ と同様に，想定時間分布と式 (2) に示した $D_{\text{free}}$ から求められる．なお，交通機関選択肢集合は，以下の通り制限される．

- 次の固定活動場所に到達不可能な交通機関を削除する
- 営業時間帯外では，公共交通機関を削除する
- 現在地に自動車・自転車が存在しない場合は，自動車・自転車を削除する
- 現在地に自動車・自転車を駐車・駐輪できない場合（再び戻ってくる予定がある自宅・職場等の場所ではない場合），自動車・自転車以外を削除する

また，目的地選択肢集合からは，利用可能な交通機関のいずれを用いても到達不可能な地域が削除される．

### 2.2.4　活動実行時間の決定

まず，以上に述べた通り，現在地と既に選択した交通機関・次の活動場所から，活動開始時刻を決定する．次に，次の固定活動の場所と開始時刻から，実行可能な活動時間の最大値（最大活動時間）を求める．そして，想定時間分布モデルによって推定された想定時間分布を最大活動時間において打切り，想定時間分布を補正する．この補正後の想定時間分布に基づき，最終的な活動時間を決定する．

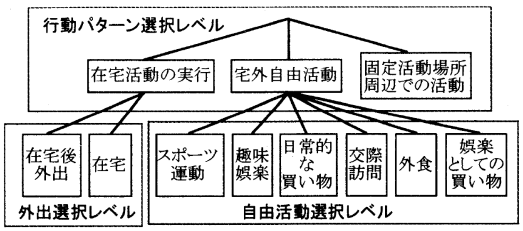


図 4　PCATS での活動内容選択モデルの選択構造 [16].

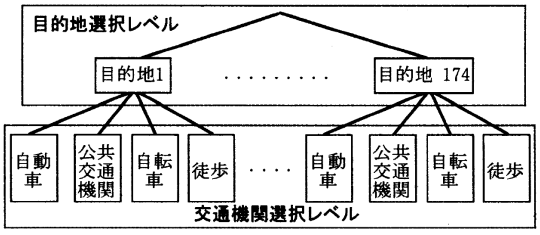


図 5　PCATS での交通機関・目的地選択モデルの選択構造 [16]．なお，174 は藤井ら（1997）によるゾーン数である．

## 2.3　交通関連データ

1.1 でも述べた通り，現代では様々な交通に関するデータが観測されている．本節では，それらの交通に関する諸データごとに，その特徴についてまとめる．

### 2.3.1　パーソントリップ調査

パーソントリップ調査（以下，PT 調査とする）は，公的機関によって行われる，個人の平日の 1 日の交通行動に関する調査である．「どのような人が」「どのような目的で」「どこからどこへ」「どのような交通手段で」移動したかなどを，図 6 のような調査票を用いて調べる．トリップ情報に加え，年代・性別・職業・自宅所在地といった個人・世帯情報も併せて収集しており，属性ごとの分析が可能となっている．PT 調査の結果は抽出データであるため，適切に拡大する必要がある．そこで，個人を年代・性別・居住地といった属性によって分類し，その属性における抽出率の逆数を「拡大係数」として設定し，個人 1 人に拡大係数分の人数を代表させる．

世界で初めて行われた PT 調査は 1953 年にアメリカのデトロイトで行われた Detroit Metropolitan Area Traffic Study と言われており [21]，国土交通省の都市交通調査・都市計画調査の Web ページによれば，日本では 1967 年に広島都市圏で行われたものが最初であり，以来日本各地の都市で行われている．三大圏での実施状況を記すと，東京都市圏では 1968 年以来，京阪神都市圏では 1970 年以来，中京都市圏では 1971 年以来，いずれも 10 年に 1 度 10 月前後に行われており，第 5 回調査が最新である．

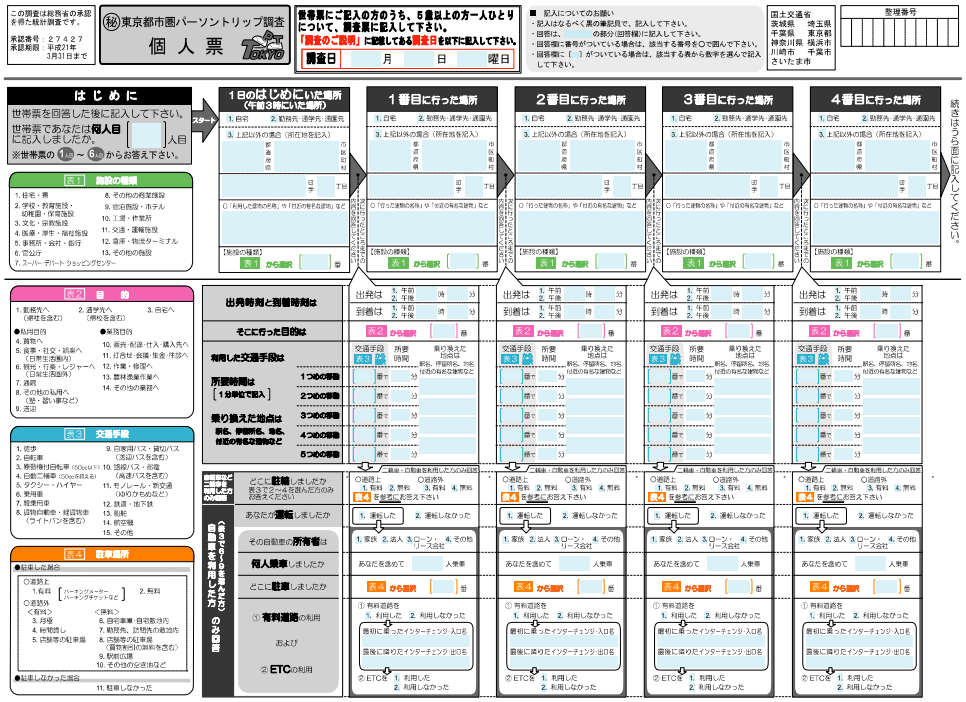


図 6　第 5 回東京都市圏 PT 調査の個人票 [18].

東京都市圏の最新調査である第 5 回東京都市圏パーソントリップ調査では，東京都・神奈川県・埼玉県・千葉県の全域（島嶼部を除く）と茨城県南部の合計 238 市区町村を対象とし，対象域内に居住する約 1600 万世帯のうち，約 140 万世帯を無作為抽出し，約 34 万世帯分の調査票を回収した．

PT 調査では網羅的な交通データを得ることができることが長所であるが，対象数が膨大で調査に時間・費用・手間がかかる，個人での回答であるため時刻等のデータの精度が比較的低い，という欠点も抱えている．特に集計にかかる時間は長く，過去の結果の公表時期を見ると，概要が纏まるまでだけでも 1-3 年を要している．そのために調査間隔も 10 年と長いため，実態との乖離に注意を払って取り扱う必要がある．

### 2.3.2　モバイル空間統計

モバイル空間統計は，図 7 のイメージのように，NTTdocomo が日本全国の 1 時間ごとのエリア人口を推計し提供しているサービスである．各基地局のエリアに所在する携帯電話の台数を周期的に集計し，docomo の普及率を加味することで，docomo 利用者以外を含めた人口を推計している．都市計画・防災計画等の公共分野での実証実験を経て，2013 年 10 月に事業化され，学術・産業分野での活用が期待されている．データの個人属性は年代・性別・居住地であり，エリア単位は東京 23 区内では 500[m] メッシュである．

モバイル空間統計を利用した研究には，その取得頻度の高さを活用した事例が多く見られる．一例として柏市における都市計画への利用を目指した清家（2013） [17] があり，時間帯別人口動態と土地利用計画の整合性や，拠点振興のためのイベント時の来街状況などを把握する手法としての有用性が期待されている．また，金森ら（2015） [22] は，函館市において，モバイル空間統計による滞留人口と，バスの乗降客数データを用いて OD 交通量の逆推定を行っている．

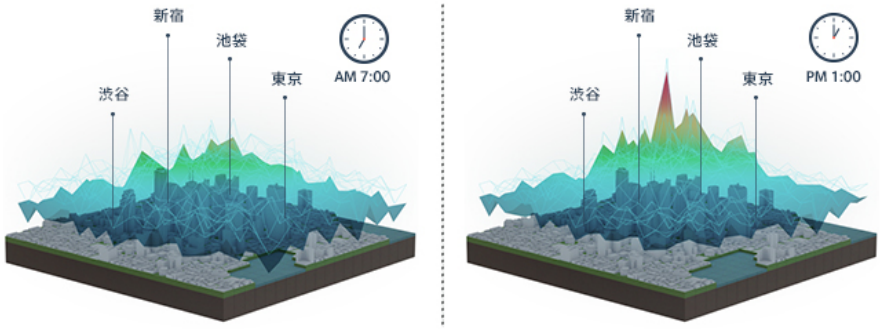


図 7　モバイル空間統計のイメージ．NTTdocomo の基地局通信の情報を利用し，1 時間ごとのエリア人口が推計される．

### 2.3.3　混雑統計

混雑統計は，ゼンリンデータが携帯電話の GPS 情報から収集した人の流れデータを提供するサービスである．GPS データを基にしているため位置情報が非集計であり，最小 250[m] のメッシュ単位や，都道府県・市区町村単位をはじめ，任意の形状でエリアを指定可能である．観測周期は最短で 1 時間単位まで指定可能である．欠点として，年代・性別等の属性情報が提供されない点が挙げられる．属性情報は，位置情報から推定された居住地・勤務地のみである．

混雑統計を用いた研究の例としては，静岡中部都市圏パーソントリップ調査データを基に道路交通量を推定した若生（2015） [24] がある．

### 2.3.4　道路交通センサス

道路交通センサスは，国土交通省が実施する全国道路・街路交通情勢調査の通称であり，調査内容は交通量・旅行速度などの実測を行う一般交通量調査と，アンケートにより OD を把握する自動車起終点調査に大別される．

1928 年に当時の内務省の外郭団体である道路改良会が実施した「全国交通調査」に端を発し，1980 年以降は概ね 5 年に 1 度，秋季に実施されている．最新の調査は，2015 年に実施された．

一般交通量調査では，高速道路・一般国道・都道府県道・一部の指定市の一般市道の幅員構成や整備状況を調査する道路状況調査，自動車・二輪車・歩行者の交通量を観測する交通量調査，自動車で実走して速度を測定する旅行速度調査の 3 つが実施される．自動車起終点調査では，路上やフェリーで行き来する自動車運転者からアンケートをとる路側 OD 調査と，自動車保有者から無作為抽出してアンケートをとるオーナーインタビュー OD 調査が実施される．

この調査の特徴として，平日のみならず休日も一部調査対象となることが挙げられる．交通量調査と旅行速度調査については，休日交通量が卓越していると道路管理者が指定した区間が，オーナーインタビュー OD 調査では対象の全員が，それぞれ休日 1 日の調査対象となる．

### 2.3.5　ETC2.0

ETC2.0 は，従来の ETC の料金自動収受機能に加えて，交通情報サービス等を組み合わせたドライバー支援システムである．その機能の 1 つに走行履歴収集があり，200[m] 走行時または 45 度以上の変針時に車載器に走行履歴を蓄積し，ITS スポットと呼ばれる路側機通過時に路車間通信でデータを収集している．収集情報は，時刻・位置情報・道路種別・進行方向・速度・加速度・車載器 ID である．

公的機関が時間的・空間的に非集計で収集している交通データとしては唯一のものであるが，普及率や対応範囲で課題が残っている．国土交通省によれば，ETC2.0 の普及率は 2017 年 10 月現在で 15.0% であり，2016 年同月の 6.8% から上がってきているとは言え，ETC 全体の 90.9% にはまだ遠く及ばない．また，ITS スポットが設置されているのは，図 8 に示す東京の例のように，現状では高速道路・直轄国道が中心であり，これらの路線を走行しない車両のデータは収集されない．

### 2.3.6　交通事業者保有データ

以上に挙げたデータは，公的機関や，位置情報を収集する企業によって提供されているものである．他に，交通事業者自身が持っているデータが存在する．ここでは便宜上，これらのデータを，時間的・空間的に非集計であるデータが存在するか，公開されているか，という点によって以下の通り分類する．

**観測困難**　事業者自身も一定程度に集計された値や代表値しか持っていないデータ
**非公開**　事業者は非集計値を持っているが，一般には集計値や代表値しか公開されないデータ
**公開**　非集計値が一般にも公開されているデータ

非集計データが観測困難な例としては，列車ごとの正確な乗車人員や，個人ごとの乗車経路（改札内乗換で，経路数が多数の場合）等が挙げられる．但し，近年では JR 東日本アプリの「山手線トレインネット」において列車ごとの混雑状況情報が提供されているなど，事業者や路線によっては観測困難から非公開，更には公開へと移ってきているデータも存在する．

非集計データが非公開な例としては，駅ごとの乗降人員や個人ごとの発着駅等が挙げられる．多くの場合，駅乗降人員は 1 年間を通じた 1 日平均値等が公開されているに過ぎず，事業者が保有しているであろう時間的に非集計なデータとは情報量の乖離が大きい．発着駅別トリップ数については，集計値・代表値でさえ公開されていない．しかし，これらのデータが観測可能になったのも，自動改札機や交通系 IC カードの普及によってである．それ以前は事業者自身も集計値しか観測できなかった点に着目すると，今後は個人を特定できないような秘匿処理を施した上で公開データとなる可能性が期待される．

公開されている非集計データとしては，運行位置情報が挙げられる．近年では鉄道事業者・バス事業者が列車の在線位置・バスの現在地をリアルタイムで提供している例も多く，これらを一定の時間間隔で取得すれば非集計データを得ることができる．

## 2.4　データ同化によるシミュレーションと交通データの統合

本節では，2.1 で述べた交通シミュレーションモデルと，2.3 で述べた交通関連データを統合した既往研究のうち，本研究でも用いるパーティクルフィルタによるデータ同化手法を用いたものを紹介する．

中村（2013） [19] は，PT 調査データを 1 分間隔に時空間内挿して生成した「人の流れデータ」を基に，出発ゾーンごとにランダムに出発時刻やその後の経路等を与えるトリップ生成モデルを開発した．更に，このモデルに対し，道路交通センサスの観測地点での 1 時間あたり通過交通量と駅での 1 時間当たり乗降客数を観測データとしてパーティクルフィルタで統合する手法を提案した．この手法を東京都市圏の「人の流れデータ」を全体の真値とする仮想都市圏に対して適用した．

Sudo *et al.*(2016) [10] は，大規模災害時の，通常とは大きく異なる人の動きをリアルタイムで把握すること

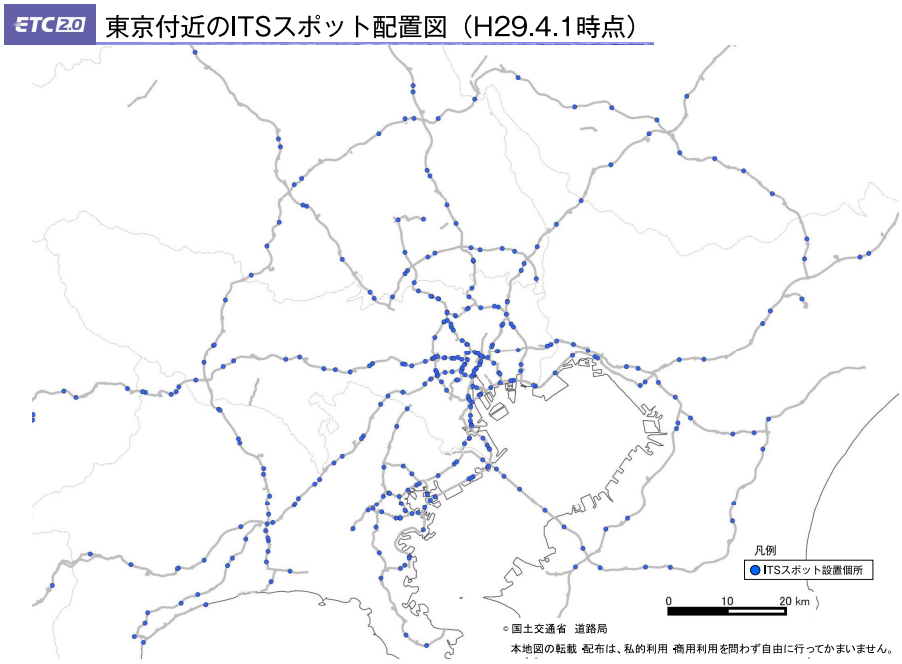


図 8　2017 年 4 月 1 日時点での東京圏における ITS スポットの配置図．車載器に蓄積された走行履歴情報は，これらの点を通過した時に収集される．（国土交通省 Web ページより）

を目的として，個人の動きを推定するシミュレータを提案してシステムモデルとし，対象地域の人口分布データを，観測間隔の長さと状態ベクトルの高次元さを要因とするパーティクル尤度の低さを克服した改良型パーティクルフィルタで統合した．この手法を 2011 年の東日本大震災当日の実 GPS データを用いて，東京都心部に対して適用し，午後 2 時から午後 12 時までの 30 秒ごとの個人位置を推定した．

淺原（2016） [12] は，任意の時刻において PT 調査と同等の網羅性を持った交通行動データを生成することを目的に，四段階推定法に基づくトリップ生成モデルに対し，擬似的な無線通信基地局を想定した人数カウントデータを観測データとしてパーティクルフィルタで統合する手法を提案した．この手法を 1998 年の「人の流れデータ」から構築したモデルに 2008 年の「人の流れデータ」から生成した人数カウントデータを統合する形で適用し，2008 年の「人の流れデータ」の再現を試みた．

原田（2017） [23] は，アクティビティモデルへの観測データ同化を目指して，PCATS に対してモバイル空間統計の観測データをパーティクルフィルタで統合する手法を提案した．2008 年の PT 調査データから PCATS の固定活動スケジュールファイルを作成し，千代田・中央・港の都心三区に対して適用した．

以上の研究をはじめとして，データ同化手法を交通分野に適用した研究は増えているが，アクティビティモデルに対して適用したものは少なく，中でも複数種類の観測データを同化したものは無い．この点が，他と比べた本研究の特徴である．

# 3　データ同化の理論

本章では，シミュレーションモデルと観測データを統合する手法であるデータ同化について述べる．本章の内容は，樋口（2011） [20]，矢野（2014） [14] を参考とした．

## 3.1　データ同化の概要

データ同化は Kitagawa(1993, 1996) [5] [6]，Gordon *et al.*(1993)[2] で提案され，現象のモデルに対して断片的な観測値を融合する手法として，気象学・海洋学分野で発展してきた．シミュレーションの精度・性能を観測データによって改善する，観測データの不足をシミュレーションで補うといった特徴を持っている．

## 3.2　状態空間モデル

現象をシミュレーションするにあたり，ある「状態」は，（実際に観測できるものもそうでないものも含めて）様々な状態量，例えば温度 $T$ や，風速ベクトル $(U, V)$ で表されるとする．これらの状態量は通常，時空間上で離散化される．いま，空間方向での離散化によって対象空間が $M$ 個のブロックに区切られたとし，そのうち $m$ 番目を考える．$m$ 番目のブロックにおいて温度 $T_m$ や風速ベクトル $(U_m, V_m)$ をはじめとする状態量を全て並べて，縦ベクトル $\boldsymbol{\xi}_k \overset{\text{def.}}{=} (T_m\ U_m\ V_m\ \ldots)'$ を定義する．但し，$'$ は転置である．ここで，シミュレーションに初期値を与えて時間更新を $t$ ステップ進めたときに，$\boldsymbol{\xi}_m$ を $m = 1$ から $m = M$ まで離散化されたブロック数分だけ縦に並べたベクトル量

$$\boldsymbol{x}_t \overset{\text{def.}}{=} \begin{pmatrix} \boldsymbol{\xi}_1 \\ \vdots \\ \boldsymbol{\xi}_m \\ \vdots \\ \boldsymbol{\xi}_M \end{pmatrix} \tag{3}$$

を定義する．この $\boldsymbol{x}_t$ を，状態ベクトルと呼ぶ．

一般にシミュレーションとは唯一の解を持つ決定論的な考えを持つ．しかし，そのようなシミュレーションで思うように現象が再現・予測できないとき，シミュレーションの計算結果に「遊び」を許し，自由度を持ったものへと拡張することを考える．即ち，解は複数存在し，確率分布として表されるようなモデルを考える．この確率論的なシミュレーションモデルを，システムモデルと呼ぶ．システムモデルは，状態ベクトル $\boldsymbol{x}_t$ の時間発展として，シミュレーションモデル $\boldsymbol{f}$ とシステムノイズ $\boldsymbol{v}$ を用いて，

$$\boldsymbol{x}_t = \boldsymbol{f}_t(\boldsymbol{x}_{t-1}, \boldsymbol{v}_t) \tag{4}$$

と定式化される．

次に，観測データを取り込むことを考える．時刻 $t$ における様々な地点での観測量をまとめたベクトルを $\boldsymbol{y}_t$ と表し，これを観測ベクトルと呼ぶ．状態ベクトル $\boldsymbol{x}_t$ は $k$ 次元ベクトル，観測ベクトル $\boldsymbol{y}_t$ は $l$ 次元ベクトルとすると，大抵の状況では $k > l$ である．即ち，状態は多くの変量からなるが，実際に観測されるのはその一部である．状態の一部が観測される場合，状態の全変量から観測される変量を取り出せるような $l \times k$ 行列 $H_t$ を導入し，$l$ 次元ベクトル $H_t\boldsymbol{x}_t$ を観測対応量とする．この行列 $H_t$ を観測行列と呼ぶ．但し，観測行列では $\boldsymbol{x}_t$ の要素を非線形変換した値が観測される場合を考えることができない．このときは，観測行列の代わり

に観測演算子 $\boldsymbol{h}_t$ を導入する．$\boldsymbol{h}_t$ は，状態の全変量から観測される変量を取り出す関数であり，$l$ 次元ベクトルである．

さて，状態ベクトル $\boldsymbol{x}_t$ が，一意に確定した値 $\hat{\boldsymbol{x}}_t$ だった場合は，観測される変量 $\boldsymbol{h}_t(\hat{\boldsymbol{x}}_t)$ も確定した値であり，それを確定した観測値である $\boldsymbol{y}_t$ と比較しても，その差分を議論することしかできない．そこで，「遊び」を許した状態ベクトルの時間発展であるシステムモデルと観測データを比較することを考える．システムモデルから得られるのは状態の確率分布の列 $p(\boldsymbol{x}_t)(t=1,\ldots,T)$ である．よって，観測される変量 $\boldsymbol{h}_t(\boldsymbol{x}_t)$ も一意には確定せず，確率分布の列 $p(\boldsymbol{h}_t(\boldsymbol{x}_t))$ に従うことになる．こうして $\boldsymbol{h}_t(\boldsymbol{x}_t)$ と $\boldsymbol{y}_t$ の整合性を吟味することになる．この両者の関係を，観測ノイズ $\boldsymbol{w}$ を用いて，

$$\boldsymbol{y}_t = \boldsymbol{h}_t(\boldsymbol{x}_t, \boldsymbol{w}_t) \tag{5}$$

と定式化する．これが観測モデルである．

式 (4) と式 (5) を連立させたモデルを状態空間モデルと呼ぶ．中でも，システムノイズや観測ノイズが線形に加わるとは限らず，ノイズの分布も Gauss 分布に限らないものを，特に非線形・非 Gauss モデルと呼ぶ．

これを更に一般化して，条件付き分布を用いて以下のような一般状態空間モデルを考えることができる．

$$\boldsymbol{x}_t \sim p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) \qquad \text{システムモデル} \tag{6}$$
$$\boldsymbol{y}_t \sim p(\boldsymbol{y}_t|\boldsymbol{x}_t) \qquad \text{観測モデル} \tag{7}$$

ここで，$p(\boldsymbol{a}|\boldsymbol{b})$ は，$\boldsymbol{b}$ が所与のときの $\boldsymbol{a}$ の条件付き確率分布を表し，$\sim$ は，左辺 $\boldsymbol{a}$ が，右辺の条件付き確率分布 $p(\boldsymbol{a}|\boldsymbol{b})$ に従うことを表す．

## 3.3　逐次 Bayes 推定

$\boldsymbol{a}$ と $\boldsymbol{b}$ の同時分布 $p(\boldsymbol{a},\boldsymbol{b})$ は，乗法定理を用いて

$$p(\boldsymbol{a},\boldsymbol{b}) = p(\boldsymbol{a}|\boldsymbol{b})p(\boldsymbol{b}) = p(\boldsymbol{b}|\boldsymbol{a})p(\boldsymbol{a}) \tag{8}$$

と分解できる．これを変形して，Bayes の定理

$$p(\boldsymbol{b}|\boldsymbol{a}) = \frac{p(\boldsymbol{a}|\boldsymbol{b})p(\boldsymbol{b})}{p(\boldsymbol{a})} \tag{9}$$

を得る．なお，同様に，同時分布 $p(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c})$ は，乗法定理を用いて

$$p(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}) = p(\boldsymbol{a}|\boldsymbol{b},\boldsymbol{c})p(\boldsymbol{b}|\boldsymbol{c}) = p(\boldsymbol{b}|\boldsymbol{a},\boldsymbol{c})p(\boldsymbol{a}|\boldsymbol{c}) \tag{10}$$

と分解できる．これを変形して，$\boldsymbol{c}$ で条件付けされた Bayes の定理

$$p(\boldsymbol{b}|\boldsymbol{a},\boldsymbol{c}) = \frac{p(\boldsymbol{a}|\boldsymbol{b},\boldsymbol{c})p(\boldsymbol{b}|\boldsymbol{c})}{p(\boldsymbol{a}|\boldsymbol{c})} \tag{11}$$

を得る．

式 (9) で $\boldsymbol{a} = \boldsymbol{y}_{1:T}$ および $\boldsymbol{b} = \boldsymbol{x}_{1:T}$ とおくことにより，

$$p(\boldsymbol{x}_{1:T}|\boldsymbol{y}_{1:T}) = \frac{p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:T})p(\boldsymbol{x}_{1:T})}{p(\boldsymbol{y}_{1:T})} \tag{12}$$

を得る．ここで，離散化された時間系での任意の時刻 $t$ におけるベクトル量 $\boldsymbol{z}_t$ を時刻 1 から時刻 $t$ まで並べたものを $\boldsymbol{z}_{1:t}$ と記した．即ち，

$$\boldsymbol{z}_{1:t} \overset{\text{def.}}{=} \{\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_t\} \tag{13}$$

である．式 (12) の左辺を事後分布，右辺分子の $p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:T})$ をデータ分布と呼ぶ．事後分布とは，データ集合 $\boldsymbol{y}_{1:T}$ を観測した後に，$\boldsymbol{x}_{1:T}$ に関する不確実性を条件付き分布 $p(\boldsymbol{x}_{1:T}|\boldsymbol{y}_{1:T})$ の形で評価することからついた名称である．また，データ分布 $p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:T})$ では $\boldsymbol{x}_{1:T}$ が所与であるため，$\boldsymbol{x}_{1:T}$ を尤度関数 $p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:T})$ のパラメータとして捉えることができる．また，$p(\boldsymbol{x}_{1:T})$ は，観測データ集合 $\boldsymbol{y}_{1:T}$ とは無関係であるので，事前分布と呼ぶ．

さて，データ同化において考えるべき問題は，時刻 1 から時刻 $t$ までの観測値 $\boldsymbol{y}_{1:t}$ を所与としたとき，式 (4) と式 (5) の状態空間モデル（あるいは式 (6) と式 (7) の一般状態空間モデル）を用いて，「未知の状態 $\boldsymbol{x}_t$ を推定する」ことである．これを状態推定という．状態推定において推定すべき分布，予測分布，フィルタ分布，平滑化分布を，それぞれ以下のように定義する．

$$\boldsymbol{x}_{t|t-1} \overset{\text{def.}}{=} p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1}) \qquad \text{予測分布} \tag{14}$$

$$\boldsymbol{x}_{t|t} \overset{\text{def.}}{=} p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}) \qquad \text{フィルタ分布} \tag{15}$$

$$\boldsymbol{x}_{t|T} \overset{\text{def.}}{=} p(\boldsymbol{x}_t|\boldsymbol{y}_{1:T}) \qquad \text{平滑化分布} \tag{16}$$

但し，$1 \leq t \leq T$ である．予測分布 $\boldsymbol{x}_{t|t-1}$ は，時刻 1 から時刻 $t-1$ までの観測値を所与として求めた，次期である時刻 $t$ での状態の分布である．フィルタ分布 $\boldsymbol{x}_{t|t}$ は，時刻 1 から時刻 $t$ までの観測値を所与として求めた，現時刻 $t$ での状態の分布である．平滑化分布 $\boldsymbol{x}_{t|T}$ は，時刻 1 から最終時刻 $T$ までの観測値を所与として求めた，時刻 $t$ での状態の分布である．

データ同化では，ある初期分布を与え，時刻 $t-1$ でのフィルタ分布 $\boldsymbol{x}_{t-1|t-1}$ からシステムモデル $p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$ を用いて時刻 $t$ での予測分布 $\boldsymbol{x}_{t|t-1}$ を得る（一期先予測），時刻 $t$ での予測分布 $\boldsymbol{x}_{t|t-1}$ と観測データ集合 $\boldsymbol{y}_{1:t}$ から観測モデル $p(\boldsymbol{y}_t|\boldsymbol{x}_t)$ を用いて時刻 $t$ でのフィルタ分布 $\boldsymbol{x}_{t|t}$ を得る（フィルタリング），という操作を逐次的に繰り返す．これを逐次 Bayes フィルタという．これにより，最後の時刻 $T$ における状態ベクトルのフィルタ分布 $\boldsymbol{x}_{T|T}$ が得られる．以下に，このアルゴリズムの漸化式を示す．

### 3.3.1　一期先予測

一期先予測のアルゴリズムは，以下の式で表せる．

$$\begin{aligned}
\boldsymbol{x}_{t|t-1} = p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1}) &= \int p(\boldsymbol{x}_t, \boldsymbol{x}_{t-1}|\boldsymbol{y}_{1:t-1}) d\boldsymbol{x}_{t-1} && \boldsymbol{x}_{t-1} \text{ に関する周辺化} \\
&= \int p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}, \boldsymbol{y}_{1:t-1}) p(\boldsymbol{x}_{t-1}|\boldsymbol{y}_{1:t-1}) d\boldsymbol{x}_{t-1} && \text{乗法定理} \\
&= \int p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) p(\boldsymbol{x}_{t-1}|\boldsymbol{y}_{1:t-1}) d\boldsymbol{x}_{t-1} && \text{時刻 } t \text{ への遷移における Marcov 性} \\
&= \int p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) \boldsymbol{x}_{t-1|t-1} d\boldsymbol{x}_{t-1} && \text{式 (15) の定義}
\end{aligned} \tag{17}$$

以上より，時刻 $t-1$ でのフィルタ分布 $\boldsymbol{x}_{t-1|t-1}$ からシステムモデル $p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$ を用いて時刻 $t$ での予測分布 $\boldsymbol{x}_{t|t-1}$ を得る，一期先予測の操作を行うことができる．

### 3.3.2　フィルタリング

フィルタリングのアルゴリズムは，以下の式で表せる．

$$
\begin{aligned}
\boldsymbol{x}_{t|t} = p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}) &= p(\boldsymbol{x}_t|\boldsymbol{y}_t, \boldsymbol{y}_{1:t-1}) && \boldsymbol{y}_{1:t} \text{ の分割} \\
&= \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t, \boldsymbol{y}_{1:t-1})p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})}{p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1})} && \text{式 (11) の } \boldsymbol{y}_{1:t-1} \text{ で周辺化した Bayes の定理} \\
&= \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})}{p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1})} && \text{時刻 } t \text{ での観測における Marcov 性} \\
&= \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})}{\int p(\boldsymbol{y}_t, \boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})d\boldsymbol{x}_t} && \boldsymbol{x}_t \text{ に関する周辺化} \\
&= \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})}{\int p(\boldsymbol{y}_t|\boldsymbol{x}_t, \boldsymbol{y}_{1:t-1})p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})d\boldsymbol{x}_t} && \text{乗法定理} \\
&= \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})}{\int p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})d\boldsymbol{x}_t} && \text{時刻 } t \text{ での観測における Marcov 性} \\
&= \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t)\boldsymbol{x}_{t|t-1}}{\int p(\boldsymbol{y}_t|\boldsymbol{x}_t)\boldsymbol{x}_{t|t-1}d\boldsymbol{x}_t} && \text{式 (14) の定義} \qquad (18)
\end{aligned}
$$

以上より，時刻 $t$ での予測分布 $\boldsymbol{x}_{t|t-1}$ と観測データ集合 $\boldsymbol{y}_{1:t}$ から観測モデル $p(\boldsymbol{y}_t|\boldsymbol{x}_t)$ を用いて時刻 $t$ でのフィルタ分布 $\boldsymbol{x}_{t|t}$ を得る，フィルタリングの操作を行うことができる．

### 3.3.3　固定区間平滑化

一期先予測とフィルタリングを最終時刻 $T$ まで繰り返すことによって $T$ でのフィルタ分布 $\boldsymbol{x}_{T|T}$ が得られたとき，そこから $\boldsymbol{x}_{T-1|T}$ を求め，そこから $\boldsymbol{x}_{T-2|T}$ を求め，と順次時間を遡る（平滑化）ことで，時刻 $t$ における状態ベクトルの平滑化分布 $\boldsymbol{x}_{t|T}$ が得られる．$T$ が変わらない，即ち，利用する観測データ数が変わらないことを明確にするために，固定区間平滑化とも呼ばれる．このアルゴリズムの漸化式は，以下の式で表せる．

$$
\begin{aligned}
\boldsymbol{x}_{t|T} = p(\boldsymbol{x}_t|\boldsymbol{y}_{1:T}) &= \int p(\boldsymbol{x}_t, \boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:T})d\boldsymbol{x}_{t+1} && \boldsymbol{x}_{t+1} \text{ に関する周辺化} \\
&= \int p(\boldsymbol{x}_t|\boldsymbol{x}_{t+1}, \boldsymbol{y}_{1:T})p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:T})d\boldsymbol{x}_{t+1} && \text{乗法定理} \\
&= \int p(\boldsymbol{x}_t|\boldsymbol{x}_{t+1}, \boldsymbol{y}_{1:t})p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:T})d\boldsymbol{x}_{t+1} && \text{以下の式 (20) より} \\
&= \int \frac{p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{y}_{1:t})}{p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t})}p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:T})d\boldsymbol{x}_{t+1} && \text{式 (11) の周辺化した Bayes の定理} \\
&= \int \frac{p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)}{p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t})}p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:T})d\boldsymbol{x}_{t+1} && \text{時刻 } t+1 \text{ への遷移における Marcov 性} \\
&= p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}) \int \frac{p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:T})}{p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t})}d\boldsymbol{x}_{t+1} && \boldsymbol{x}_{t+1} \text{ に無関係の部分を積分の外へ} \\
&= \boldsymbol{x}_{t|t} \int \frac{p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)\boldsymbol{x}_{t+1|T}}{\boldsymbol{x}_{t+1|t}} && \text{式 (14),(15),(16) の定義} \qquad (19)
\end{aligned}
$$

ここで，2 行目から 3 行目の $p(\boldsymbol{x}_t|\boldsymbol{x}_{t+1}, \boldsymbol{y}_{1:T}) = p(\boldsymbol{x}_t|\boldsymbol{x}_{t+1}, \boldsymbol{y}_{1:t})$ は，次の通り導ける．

$$\begin{aligned}
p(\boldsymbol{x}_t|\boldsymbol{x}_{t+1}, \boldsymbol{y}_{1:T}) &= p(\boldsymbol{x}_t|\boldsymbol{x}_{t+1}, \boldsymbol{y}_{1:t}, \boldsymbol{y}_{t+1:T}) && \boldsymbol{y}_{1:T} \text{ の分割} \\
&= \frac{p(\boldsymbol{y}_{t+1:T}|\boldsymbol{x}_t, \boldsymbol{x}_{t+1}, \boldsymbol{y}_{1:t})p(\boldsymbol{x}_t|\boldsymbol{x}_{t+1}, \boldsymbol{y}_{1:t})}{p(\boldsymbol{y}_{t+1:T}|\boldsymbol{x}_{t+1}, \boldsymbol{y}_{1:t})} && \text{式 (11) の周辺化した Bayes の定理} \\
&= \frac{p(\boldsymbol{y}_{t+1:T}|\boldsymbol{x}_{t+1}, \boldsymbol{y}_{1:t})p(\boldsymbol{x}_t|\boldsymbol{x}_{t+1}, \boldsymbol{y}_{1:t})}{p(\boldsymbol{y}_{t+1:T}|\boldsymbol{x}_{t+1}, \boldsymbol{y}_{1:t})} && \text{時刻 } t+1 \text{ での観測における Marcov 性} \\
&= p(\boldsymbol{x}_t|\boldsymbol{x}_{t+1}, \boldsymbol{y}_{1:t}) && \text{約分} \qquad (20)
\end{aligned}$$

式 (19) より，時刻 $t+1$ での平滑化分布 $\boldsymbol{x}_{t+1|T}$ と，既に計算した時刻 $t+1$ での予測分布 $\boldsymbol{x}_{t+1|t}$ と，時刻 $t$ でのフィルタ分布 $\boldsymbol{x}_{t|t}$ と，システムモデル $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)$ を用いて，時刻 $t$ での平滑化分布 $\boldsymbol{x}_{t|T}$ を得る，固定区間平滑化の操作を行うことができる．

### 3.3.4　固定ラグ平滑化

固定区間平滑化をそのまま数値的に実現するのは，状態ベクトルがきわめて低次元のときに限られる．状態ベクトルが高次元のときの平滑化は，固定区間平滑化分布 $\boldsymbol{x}_{t|T} = p(\boldsymbol{x}_t|\boldsymbol{y}_{1:T})$ の代わりに，固定ラグ平滑化分布 $\boldsymbol{x}_{t|t+L} = p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t+L})$ を用いて実行される．ここで，$L$ を固定ラグと呼ぶ．即ち，時刻 $t$ での平滑化において，最終時刻 $T$ までの全観測データではなく，$L$ だけ先の時刻 $t+L$ までの観測データを用いるものである．

固定ラグ平滑化分布の求め方は次の通りである．ある時刻 $t$ において，$L$ 個前までの過去の状態ベクトルを全て繋げた拡大状態ベクトル $\tilde{\boldsymbol{x}}_t' \overset{\text{def.}}{=} (\boldsymbol{x}_t'\ \boldsymbol{x}_{t-1}'\ \ldots\ \boldsymbol{x}_{t-L+1}'\ \boldsymbol{x}_{t-L}')$ を定義する．ここで，$\boldsymbol{x}_t$ に対する式 (4), (5) には変更は無いので，$\tilde{\boldsymbol{x}}_t$ に対する状態空間モデルを次のように定めることができる．

$$\tilde{\boldsymbol{x}}_t = \begin{pmatrix} \boldsymbol{x}_t \\ \boldsymbol{x}_{t-1} \\ \vdots \\ \boldsymbol{x}_{t-L+1} \\ \boldsymbol{x}_{t-L} \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_t(\boldsymbol{x}_{t-1}, \boldsymbol{v}_t) \\ \boldsymbol{x}_{t-1} \\ \vdots \\ \boldsymbol{x}_{t-L+1} \\ \boldsymbol{x}_{t-L} \end{pmatrix} \overset{\text{def.}}{=} \tilde{\boldsymbol{f}}_t(\tilde{\boldsymbol{x}}_{t-1}, \boldsymbol{v}_t) \qquad \text{システムモデル} \qquad (21)$$

$$\boldsymbol{y}_t = \boldsymbol{h}_t(\boldsymbol{x}_t, \boldsymbol{w}_t) \overset{\text{def.}}{=} \tilde{\boldsymbol{h}}_t(\tilde{\boldsymbol{x}}_t, \boldsymbol{w}_t) \qquad \text{観測モデル} \qquad (22)$$

但し，$\tilde{\boldsymbol{h}}_t$ は，観測ベクトル $\boldsymbol{y}_t$ の次元を $l$ として，拡大状態ベクトル $\tilde{\boldsymbol{x}}_t$ から時刻 $t$ において観測される $l$ 個の変量を取り出す関数で，$l$ 次元ベクトルである．式 (21), (22) は，$\tilde{\boldsymbol{x}}_t$ に対する通常の状態空間モデルと何ら変わらないので，逐次 Bayes フィルタで一期先予測とフィルタリングを繰り返せば，任意の時刻 $t$ におけるフィルタ分布 $\tilde{\boldsymbol{x}}_{t|t} = p(\tilde{\boldsymbol{x}}_t|\boldsymbol{y}_{1:t})$ が得られる．時刻 $t$ までの観測データ $\boldsymbol{y}_{1:t}$ を得た上での時刻 $t-L$ での固定ラグ平滑化分布 $\boldsymbol{x}_{t-L:t} = p(\boldsymbol{x}_{t-L}|\boldsymbol{y}_{1:t})$ は，以下の通り $\tilde{\boldsymbol{x}}_{t|t}$ を $t-L+1$ から $t$ までの状態ベクトルについて周辺化することで求められる．

$$p(\boldsymbol{x}_{t-L}|\boldsymbol{y}_{1:t}) = \int p(\tilde{\boldsymbol{x}}_t|\boldsymbol{y}_{1:t}) d\boldsymbol{x}_{t-L+1}\, d\boldsymbol{x}_{t-L+2}\, \ldots\, d\boldsymbol{x}_t \qquad (23)$$

## 3.4　パーティクルフィルタ

状態空間モデルを実際に用いるには，一期先予測の式 (17)，フィルタリングの式 (18)，平滑化の式 (19) を具体的に計算しなければならない．その最も簡単な方法として Kalman(1960) [4] による Kalman フィルタ

があるが，これはシステムモデル・観測モデルがともに線形で，かつ，システムノイズと観測ノイズがともに Gauss 分布である線形・Gauss 型の状態空間モデルにしか用いることができない．非線形モデルに対しても用いることができる方法としては，各状態変数の周りで線形化して計算する拡張 Kalman フィルタ，確率変数の確率分布を $N$ 個のサンプル集合を用いて Monte Carlo 近似するアンサンブル Kalman フィルタなどがある．しかし，これらも基本的にはシステムの状態と観測の間には線形な関係が成り立つことを仮定しており，それが成り立たない場合には上手く行かないことがある．そこで Kitagawa(1993, 1996) [5] [6]，Gordon *et al.*(1993)[2] によって開発されたのが，より一般的で，高次元の非線形・非 Gauss 型一般状態空間モデルにも応用可能な，パーティクルフィルタである．なお，このアルゴリズムには，提案当初は Monte Carlo フィルタ，ブートストラップフィルタ (bootstrap filter)，SIR(sampling importance resampling) などの呼び名があった．しかし，現在ではパーティクル（粒子）フィルタと呼ばれることが一般的である．

パーティクルフィルタの発想は，時刻 $t$ における推定すべき分布（予測分布・フィルタ分布・平滑化分布）を，$N$ 個の「粒子」の集合 $\{\boldsymbol{x}_t^{(i)}\}_{i=1}^N$ で Monte Carlo 近似する，というものである．この粒子の集合のことをアンサンブルと呼び，予測分布 $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})$ を Monte Carlo 近似するアンサンブル $\{\boldsymbol{x}_{t|t-1}^{(i)}\}_{i=1}^N$ を予測アンサンブル，フィルタ分布 $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})$ を Monte Carlo 近似するアンサンブル $\{\boldsymbol{x}_{t|t}^{(i)}\}_{i=1}^N$ をフィルタアンサンブルと呼ぶ．本来は分布そのものの解析解を得るのが最も良いが，非線形・非 Gauss 状態空間モデルでは殆どの場合それが不可能である．そのため，Monte Carlo 法で状態ベクトルの粒子を発生させ，それらの粒子を状態空間上に並べることで，求めるべき分布を構成する．

### 3.4.1　パーティクルフィルタのアルゴリズム

アンサンブル $\{\boldsymbol{x}^{(i)}\}_{i=1}^N$ による分布 $p(\boldsymbol{x})$ の Monte Carlo 近似は，

$$p(\boldsymbol{x}) \doteqdot \frac{1}{N}\sum_{i=1}^{N}\delta(\boldsymbol{x}-\boldsymbol{x}^{(i)}) \tag{24}$$

と表される．但し，$N$ は粒子数，$\doteqdot$ は Monte Carlo 近似，$\delta$ は Dirac のデルタ関数，$\boldsymbol{x}^{(i)}$ は $\boldsymbol{x}$ を近似するための $i$ 番目の粒子である．パーティクルフィルタでは，以下のように逐次的に各ステップの予測分布・フィルタ分布の Monte Carlo 近似を求めていく．

1. 初期分布 $\boldsymbol{x}_{0|0}$ を Monte Carlo 近似するアンサンブル $\{\boldsymbol{x}_{0|0}^{(i)}\}_{i=1}^N$ を生成する．
2. $t=1,\ldots,T$ について，次のステップを実行する．
   (a) 各粒子番号 $i$ $(i=1,\ldots,N)$ について，システムノイズを表現する乱数 $\boldsymbol{v}_t^{(i)} \sim p(\boldsymbol{v}_t)$ を生成する．
   (b) 各 $i$ $(i=1,\ldots,N)$ について，式 (4) のシステムモデルから $\boldsymbol{x}_{t|t-1}^{(i)} = \boldsymbol{f}_t(\boldsymbol{x}_{t-1|t-1}^{(i)}, \boldsymbol{v}_t^{(i)})$ を計算し，予測分布を Monte Carlo 近似するアンサンブル $\{\boldsymbol{x}_{t|t-1}^{(i)}\}_{i=1}^N$ を得る．
   (c) 各 $i$ $(i=1,\ldots,N)$ について，式 (5) の観測モデルに基づいて，それぞれの粒子の尤度 $\lambda_t^{(i)} = p(\boldsymbol{y}_t|\boldsymbol{x}_{t|t-1}^{(i)})$ を計算する．
   (d) 各 $i$ $(i=1,\ldots,N)$ について，正規化尤度 $\beta_t^{(i)} = \lambda_t^{(i)}/(\sum_{j=1}^N \lambda_t^{(j)})$ を求める．
   (e) 予測アンサンブル $\{\boldsymbol{x}_{t|t-1}^{(i)}\}_{i=1}^N$ から，各粒子 $\boldsymbol{x}_{t|t-1}^{(i)}$ が $\beta_t^{(i)}$ の確率で抽出されるように $N$ 回の復元抽出を行い，得られた $N$ 個のサンプルで，フィルタ分布を Monte Carlo 近似するアンサンブル $\{\boldsymbol{x}_{t|t}^{(i)}\}_{i=1}^N$ を構成する．

このうち，2. の (a)，(b) がシステムモデルによる一期先予測の操作であり，(c) から (e) が観測モデルによるフィルタリングの操作である．ここで，2. の (c) で出てくる $\lambda_t^{(i)} = p(\boldsymbol{y}_t|\boldsymbol{x}_{t|t-1}^{(i)})$ は，時刻 $t$ における予測分布

を Monte Carlo 近似するアンサンブルの $i$ 番目の粒子 $\boldsymbol{x}_{t|t-1}^{(i)}$ の尤度と呼ばれ，粒子 $\boldsymbol{x}_{t|t-1}^{(i)}$ がどの程度観測値 $\boldsymbol{y}_t$ に当てはまっているかを表す．なお，$\boldsymbol{x}_{t|t-1}^{(i)}$ の要素である状態量と，$\boldsymbol{y}_t$ の要素である観測値は，全て確定した値であるから，尤度 $\lambda_t^{(i)}$ も確定した値となる．

### 3.4.2　パーティクルフィルタの導出

まず，一期先予測による予測分布の導出について考える．フィルタ分布 $\boldsymbol{x}_{t-1|t-1} = p(\boldsymbol{x}_{t-1}|\boldsymbol{y}_{1:t-1})$ が $N$ 個の粒子 $\{\boldsymbol{x}_{t-1|t-1}^{(i)}\}_{i=1}^N$ によるアンサンブルで

$$p(\boldsymbol{x}_{t-1}|\boldsymbol{y}_{1:t-1}) \doteq \frac{1}{N}\sum_{i=1}^{N}\delta(\boldsymbol{x}_{t-1} - \boldsymbol{x}_{t-1|t-1}^{(i)}) \tag{25}$$

と Monte Carlo 近似されているとする．このとき，式 (17) より，

$$\begin{aligned}
p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1}) &= \int p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})p(\boldsymbol{x}_{t-1}|\boldsymbol{y}_{1:t-1})d\boldsymbol{x}_{t-1} \\
&= \int \left\{ \int p(\boldsymbol{x}_t, \boldsymbol{v}_t|\boldsymbol{x}_{t-1})d\boldsymbol{v}_t \right\} p(\boldsymbol{x}_{t-1}|\boldsymbol{y}_{1:t-1})d\boldsymbol{x}_{t-1} \\
&= \int \left\{ \int p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}, \boldsymbol{v}_t)p(\boldsymbol{v}_t|\boldsymbol{x}_{t-1})d\boldsymbol{v}_t \right\} p(\boldsymbol{x}_{t-1}|\boldsymbol{y}_{1:t-1})d\boldsymbol{x}_{t-1} \\
&= \iint p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}, \boldsymbol{v}_t)p(\boldsymbol{v}_t|\boldsymbol{x}_{t-1})p(\boldsymbol{x}_{t-1}|\boldsymbol{y}_{1:t-1})d\boldsymbol{x}_{t-1}d\boldsymbol{v}_t
\end{aligned} \tag{26}$$

となる．$\boldsymbol{v}_t$ と $\boldsymbol{y}_{1:t}$，および $\boldsymbol{v}_t$ と $\boldsymbol{x}_{t-1}$ は独立なので，上式の積分の内側は

$$\begin{aligned}
p(\boldsymbol{v}_t|\boldsymbol{x}_{t-1})p(\boldsymbol{x}_{t-1}|\boldsymbol{y}_{1:t-1}) &= p(\boldsymbol{v}_t)p(\boldsymbol{x}_{t-1}|\boldsymbol{y}_{1:t-1}) \\
&= p(\boldsymbol{x}_{t-1}, \boldsymbol{v}_t|\boldsymbol{y}_{1:t-1}) \\
&\doteq \frac{1}{N}\sum_{i=1}^{N}\delta\left(\begin{pmatrix}\boldsymbol{x}_{t-1}\\ \boldsymbol{v}_t\end{pmatrix} - \begin{pmatrix}\boldsymbol{x}_{t-1|t-1}^{(i)}\\ \boldsymbol{v}_t^{(i)}\end{pmatrix}\right)
\end{aligned} \tag{27}$$

と表現される．但し，$\boldsymbol{v}_t$ と $\boldsymbol{x}_{t-1}$ の独立性から，

$$\delta\left(\begin{pmatrix}\boldsymbol{x}_{t-1}\\ \boldsymbol{v}_t\end{pmatrix} - \begin{pmatrix}\boldsymbol{x}_{t-1|t-1}\\ \boldsymbol{v}_t\end{pmatrix}^{(i)}\right) = \delta\left(\begin{pmatrix}\boldsymbol{x}_{t-1}\\ \boldsymbol{v}_t\end{pmatrix} - \begin{pmatrix}\boldsymbol{x}_{t-1|t-1}^{(i)}\\ \boldsymbol{v}_t^{(i)}\end{pmatrix}\right) \tag{28}$$

であることを用いた．以上とシステムモデルの式 (4) を組み合わせて，

$$\begin{aligned}
p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1}) &\doteq \frac{1}{N}\sum_{i=1}^{N}\left\{\iint p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}, \boldsymbol{v}_t)\delta\left(\begin{pmatrix}\boldsymbol{x}_{t-1}\\ \boldsymbol{v}_t\end{pmatrix} - \begin{pmatrix}\boldsymbol{x}_{t-1|t-1}^{(i)}\\ \boldsymbol{v}_t^{(i)}\end{pmatrix}\right)d\boldsymbol{v}_t d\boldsymbol{x}_{t-1}\right\} \\
&= \frac{1}{N}\sum_{i=1}^{N} p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1} = \boldsymbol{x}_{t-1}^{(i)}, \boldsymbol{v}_t = \boldsymbol{v}_t^{(i)}) \\
&= \frac{1}{N}\sum_{i=1}^{N}\delta(\boldsymbol{x}_t - \boldsymbol{f}_t(\boldsymbol{x}_{t-1}^{(i)}, \boldsymbol{v}_t^{(i)})) \\
&= \frac{1}{N}\sum_{i=1}^{N}\delta(\boldsymbol{x}_t - \boldsymbol{x}_{t|t-1}^{(i)})
\end{aligned} \tag{29}$$

となる．以上より，予測分布 $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})$ が，フィルタアンサンブル $\{\boldsymbol{x}^{(i)}_{t-1|t-1}\}^N_{i=1}$ の各粒子 $\boldsymbol{x}^{(i)}_{t-1|t-1}$ をシステムモデル $\boldsymbol{x}^{(i)}_{t|t-1} = \boldsymbol{f}_t(\boldsymbol{x}^{(i)}_{t-1}, \boldsymbol{v}^{(i)}_t)$ によって計算して得られた粒子 $\boldsymbol{x}^{(i)}_{t|t-1}$ によって構成される予測アンサンブル $\{\boldsymbol{x}^{(i)}_{t|t-1}\}^N_{i=1}$ で Monte Carlo 近似できることが示された．

次に，フィルタリングによるフィルタ分布の導出について考える．予測分布 $\boldsymbol{x}_{t|t-1} = p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})$ がアンサンブル $\{\boldsymbol{x}^{(i)}_{t|t-1}\}^N_{i=1}$ で

$$p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1}) \doteq \frac{1}{N}\sum_{i=1}^{N}\delta(\boldsymbol{x}_t - \boldsymbol{x}^{(i)}_{t|t-1}) \tag{30}$$

と Monte Carlo 近似できることは上で示した．これを式 (18) に代入すると，

$$\begin{aligned}
p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}) &\doteq \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t)\sum_{i=1}^{N}\delta(\boldsymbol{x}_t - \boldsymbol{x}^{(i)}_{t|t-1})}{\int p(\boldsymbol{y}_t|\boldsymbol{x}_t)\sum_{j=1}^{N}\delta(\boldsymbol{x}_t - \boldsymbol{x}^{(j)}_{t|t-1})d\boldsymbol{x}_t} \\
&= \frac{1}{\sum_{j=1}^{N} p(\boldsymbol{y}_t|\boldsymbol{x}^{(j)}_{t|t-1})}\sum_{i=1}^{N} p(\boldsymbol{y}_t|\boldsymbol{x}^{(i)}_{t|t-1})\delta(\boldsymbol{x}_t - \boldsymbol{x}^{(i)}_{t|t-1}) \\
&= \frac{1}{\sum_{j=1}^{N} \lambda^{(j)}_t}\sum_{i=1}^{N} \lambda^{(i)}_t\delta(\boldsymbol{x}_t - \boldsymbol{x}^{(i)}_{t|t-1}) \\
&= \sum_{i=1}^{N} \beta^{(i)}_t\delta(\boldsymbol{x}_t - \boldsymbol{x}^{(i)}_{t|t-1})
\end{aligned} \tag{31}$$

となる．ここで，$\lambda^{(i)}_t, \beta^{(i)}_t$ はそれぞれ，3.4.1 のアルゴリズム説明における 2. の (c) と (d) で述べた，データ $\boldsymbol{y}_t$ が与えられたときの粒子 $\boldsymbol{x}^{(i)}_{t|t-1}$ の尤度，正規化尤度である．上式は，フィルタ分布 $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})$ が，予測アンサンブル $\{\boldsymbol{x}^{(i)}_{t|t-1}\}^N_{i=1}$ の各粒子に $\beta^{(i)}_t$ で重みを付けたもので Monte Carlo 近似できることを示している．

いま，

$$m^{(i)}_t \approx N\beta^{(i)}_t \quad \left(\sum_{i=1}^{N} m^{(i)}_t = N;\ m^{(i)}_t \geq 0\right) \tag{32}$$

を満たすような整数列 $\{m^{(i)}_t\}^N_{i=1}$ を考え，予測アンサンブル $\{\boldsymbol{x}^{(i)}_{t|t-1}\}^N_{i=1}$ の各粒子 $\boldsymbol{x}^{(i)}_{t|t-1}$ の複製が $m^{(i)}_t$ 個ずつ含まれるような新しいアンサンブル $\{\boldsymbol{x}^{(i)}_{t|t}\}^N_{i=1}$ を生成する．具体的には，予測アンサンブル $\{\boldsymbol{x}^{(i)}_{t|t-1}\}^N_{i=1}$ の各粒子が確率 $\beta^{(i)}_t$ で抽出されるように $N$ 個の粒子を復元抽出（この操作をリサンプリングと呼ぶ）すれば，その $N$ 個の粒子で新アンサンブル $\{\boldsymbol{x}^{(i)}_{t|t}\}^N_{i=1}$ が生成できる．このとき，

$$\begin{aligned}
p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}) &\approx \frac{1}{N}\sum_{i=1}^{N} m^{(i)}_t\delta(\boldsymbol{x}_t - \boldsymbol{x}^{(i)}_{t|t-1}) \\
&= \frac{1}{N}\sum_{i=1}^{N}\delta(\boldsymbol{x}_t - \boldsymbol{x}^{(i)}_{t|t})
\end{aligned} \tag{33}$$

となり，新アンサンブル $\{\boldsymbol{x}^{(i)}_{t|t}\}^N_{i=1}$ がフィルタ分布 $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})$ を Monte Carlo 近似するアンサンブルとなっていることが示された．

### 3.4.3　リサンプリングの方法

3.4.1 のアルゴリズム説明における 2. の (e) の操作をリサンプリングという．これを具体的に行うには，幾つかの方法が考えられる．最も単純な方法は，予測アンサンブル $\{\boldsymbol{x}_{t|t-1}^{(i)}\}_{i=1}^{N}$ から各粒子が $\beta_t^{(i)}$ の確率で抽出されるものとした抽出を $N$ 回繰り返すことである．しかし，この方法では，各粒子が抽出される実際の回数が，その期待値 $N\beta_t^{(i)}$ から外れてしまうことも多い．粒子 $\boldsymbol{x}_{t|t-1}^{(i)}$ の抽出が確率 $\beta_t^{(i)}$ の Bernoulli 試行と見做せることから，この方法で得たフィルタアンサンブル $\{\boldsymbol{x}_{t|t}^{(i)}\}_{i=1}^{N}$ に占める予測アンサンブルの $i$ 番目の粒子 $\boldsymbol{x}_{t|t-1}^{(i)}$ の複製の割合は，$\sqrt{\beta_t^{(i)}(1-\beta_t^{(i)})/N}$ 程度のばらつきを持つことになる．このことから，特に $N$ があまり大きくない場合において，得られたフィルタアンサンブルによって表現される分布が，本来のフィルタ分布の形状と大きく異なってしまう可能性がある．Monte Carlo 近似を行う目的は，確率分布の形状をよく近似することにあるので，これはあまり好ましくない．

そこで，粒子 $\boldsymbol{x}_{t|t-1}^{(i)}$ の抽出回数の比が正規化尤度 $\beta_t^{(i)}$ の比になるべく近くなるようにするために，単なるランダム抽出ではない以下のような方法をとることがある．

1. 抽出個数の期待値 $N\beta_t^{(i)}$ の整数部分を $\bar{m}_t^{(i)}$ として，まずは各粒子 $\boldsymbol{x}_{t|t-1}^{(i)}$ をそれぞれ $\bar{m}_t^{(i)}$ 個ずつ抽出する．これで，$\sum_{i=1}^{N}\bar{m}_t^{(i)}$ 個の抽出が完了する．
2. 残りの粒子を抽出するために，各粒子 $\boldsymbol{x}_{t|t-1}^{(i)}$ が確率 $(N\beta_t^{(i)}-\bar{m}_t^{(i)})/(N-\sum_{i=1}^{N}\bar{m}_t^{(i)})$ で抽出されるようなランダム抽出を $N-\sum_{i=1}^{N}\bar{m}_t^{(i)}$ 回繰り返す．

### 3.4.4　点推定値の与え方

さて，パーティクルフィルタでは一般に，逐次推定に利用されるのは 1 つの確定値ではなく，確率分布（の近似である粒子の集合，即ちアンサンブル）である．そのため，1 つの確定値である点推定値を求めたいときは，確率分布（アンサンブル）に対する何らかの操作を取り決める必要がある．時刻 $t$ におけるフィルタ分布は式 (18) より，

$$\boldsymbol{x}_{t|t} = p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}) = \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})}{p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1})} \tag{34}$$

と表され，左辺のフィルタ分布 $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})$ が事後分布，分母 $p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1})$ は $\boldsymbol{x}_t$ に無関係の確定値，分子の $p(\boldsymbol{y}_t|\boldsymbol{x}_t)$ はデータ分布（観測モデル），予測分布 $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})$ が事前分布である．事後分布を Monte Carlo 近似するフィルタアンサンブルは $\{\boldsymbol{x}_{t|t}^{(i)}\}_{i=1}^{N}$ であり，直前のリサンプリング，即ち予測アンサンブル $\{\boldsymbol{x}_{t|t-1}^{(i)}\}_{i=1}^{N}$ からの粒子 $\boldsymbol{x}_{t|t-1}^{(i)}$ の抽出確率 $\beta_t^{(i)}$ での復元抽出を考えると，$\beta_t^{(i)}$ が正規化尤度を表し，予測アンサンブル $\{\boldsymbol{x}_{t|t-1}^{(i)}\}_{i=1}^{N}$ が事前分布の Monte Carlo 近似である．

式 (34) の確率分布から 1 つの確定値としての代表値を定める方法は幾つか考えられる．$\boldsymbol{x}_t$ が連続量であれば重み付き平均 $\sum_{i=1}^{N}\beta_t^{(i)}\boldsymbol{x}_t^{(i)}$ とするのが一般的（但し，非 Gauss 分布では多峰性を持つ可能性があり，その場合は確率の低い谷を推定する危険がある）だが，本研究の場合 $\boldsymbol{x}_t$ は離散量であり，Gauss 分布も仮定していないので，重み付き平均は使えない．その場合，事後分布 $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})$ を最大とする解（MAP 解）を点推定値とするのが一般的である．

事後分布の Monte Carlo 近似はフィルタアンサンブル $\{\boldsymbol{x}_{t|t}^{(i)}\}_{i=1}^{N}$ であるから，このフィルタアンサンブルから（$N$ 回に限らず）何度も抽出を繰り返したときに，最も多くの回数抽出されるであろう粒子 $\hat{\boldsymbol{x}}_{t|t}^{(i)}$ が，求める点推定値ということになる．言い換えれば，$\hat{\boldsymbol{x}}_{t|t}^{(i)}$ は，フィルタアンサンブルを構成する $N$ 個の粒子が

最も密になっている点である．ところで，3.4.2 より，フィルタアンサンブル $\{\boldsymbol{x}_{t|t}^{(i)}\}_{i=1}^N$ は，予測アンサンブル $\{\boldsymbol{x}_{t|t-1}^{(i)}\}_{i=1}^N$ から，粒子 $\boldsymbol{x}_{t|t-1}^{(i)}$ が確率 $\beta_t^{(i)}$ で選択される復元抽出を $N$ 回繰り返して構成されたものである．故に，フィルタアンサンブルを構成する $N$ 個の粒子が最も密になっている点 $\hat{\boldsymbol{x}}_{t|t}^{(i)}$ は，予測アンサンブル $\{\boldsymbol{x}_{t|t-1}^{(i)}\}_{i=1}^N$ の粒子のうち，復元抽出された個数が最も多かったものということになる．

なお，復元抽出は，その最大個数が $N$ 個に制限された上でのランダム抽出であるため，必ずしも確率 $\beta_t^{(i)}$ を完全に反映した結果とはならないことに注意が必要である．だからと言って，抽出確率 $\beta_t^{(i)}$ の大きさをそのまま点推定値 $\hat{\boldsymbol{x}}_{t|t}^{(i)}$ の決定に用いてはいけない．このリサンプリングの不自由さと不確実性こそが，シミュレーションの影響を残すことに繋がる．確率 $\beta_t^{(i)}$ のみを完全に反映した選び方では，シミュレーションの結果，即ち予測分布（予測アンサンブルにおける粒子の散らばり方とも言える）を殆ど無視して，観測値に最も近いものを選ぶことになってしまうのである．尤度が低い粒子でも，つまり観測値から離れていても，一定の確率では抽出される可能性がある．その結果，シミュレーション結果である予測分布，つまり事前分布が，一定程度高ければ，つまりその付近にある程度の数の粒子が固まっていれば，それに応じた確率でその付近の粒子が抽出され，事後分布を形成するのである．

## 3.5　パーティクルスムーザ

パーティクルフィルタは 3.3.4 で述べた固定ラグ平滑化に拡張することができる．パーティクルフィルタの枠組みに基づく平滑化アルゴリズムをパーティクルスムーザと呼ぶ．

3.3.4 と同様に，拡大状態ベクトルを $\tilde{\boldsymbol{x}}_t' \overset{\text{def.}}{=} (\boldsymbol{x}_t'\ \boldsymbol{x}_{t-1}'\ \cdots\ \boldsymbol{x}_{t-L+1}'\ \boldsymbol{x}_{t-L}')$ と定義すると，時刻 $t-1$ におけるフィルタ分布を Monte Carlo 近似するアンサンブルは $\{\tilde{\boldsymbol{x}}_{t-1|t-1}^{(i)}\}_{i=1}^N = \{(\boldsymbol{x}_{t-1|t-1}^{(i)'}\ \cdots\ \boldsymbol{x}_{t-L-1|t-1}^{(i)'})\}_{i=1}^N$ となる．このとき，時刻 $t$ における予測分布を Monte Carlo 近似するアンサンブル $\{\tilde{\boldsymbol{x}}_{t|t-1}^{(i)}\}_{i=1}^N = \{(\boldsymbol{x}_{t|t-1}^{(i)'}\ \cdots\ \boldsymbol{x}_{t-L|t-1}^{(i)'})\}_{i=1}^N$ の各粒子 $\tilde{\boldsymbol{x}}_{t|t-1}^{(i)}$ は，式 (21) の拡大システムモデルを適用して，

$$\tilde{\boldsymbol{x}}_{t|t-1}^{(i)} = \begin{pmatrix} \boldsymbol{x}_{t|t-1}^{(i)} \\ \boldsymbol{x}_{t-1|t-1}^{(i)} \\ \vdots \\ \boldsymbol{x}_{t-L|t-1}^{(i)} \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_t(\boldsymbol{x}_{t-1|t-1}^{(i)}, \boldsymbol{v}_t^{(i)}) \\ \boldsymbol{x}_{t-1|t-1}^{(i)} \\ \vdots \\ \boldsymbol{x}_{t-L|t-1}^{(i)} \end{pmatrix} \tag{35}$$

のように得られる．

次に，得られた予測アンサンブル $\{\tilde{\boldsymbol{x}}_{t|t-1}^{(i)}\}_{i=1}^N$ から時刻 $t$ の観測 $\boldsymbol{y}_t$ に基づいてリサンプリングを行う．ここで，Marcov 性から $p(\boldsymbol{y}_t|\tilde{\boldsymbol{x}}_t) = p(\boldsymbol{y}_t|\boldsymbol{x}_t)$ が成り立つことから，予測アンサンブルの粒子 $\tilde{\boldsymbol{x}}_{t|t-1}^{(i)}$ の尤度は

$$p(\boldsymbol{y}_t|\tilde{\boldsymbol{x}}_{t|t-1}^{(i)}) = p(\boldsymbol{y}_t|\boldsymbol{x}_{t|t-1}^{(i)}) = \lambda_t^{(i)} \tag{36}$$

となり，拡大前の粒子 $\boldsymbol{x}_{t|t-1}^{(i)}$ の尤度と同じになる．故に，3.4 で述べた正規化尤度 $\beta_t^{(i)}$ をそのまま用いればよい．こうして得られたフィルタアンサンブル $\{\tilde{\boldsymbol{x}}_{t|t}^{(i)}\}_{i=1}^N = \{(\boldsymbol{x}_{t|t}^{(i)'}\ \cdots\ \boldsymbol{x}_{t-L|t}^{(i)'})\}_{i=1}^N$ から，$\{\boldsymbol{x}_{t-L|t}^{(i)'}\}_{i=1}^N$ の部分を取り出すと，平滑化分布 $p(\boldsymbol{x}_{t-L}|\boldsymbol{y}_{1:t})$ を Monte Carlo 近似するアンサンブルが得られる．

なお，実際に時刻 $t$ までの観測データを用いて時刻 $t-L$ における平滑化分布を求めるには，時刻 $t-L+1$ から時刻 $t$ まで，拡大状態ベクトルによる一期先予測とフィルタリングを繰り返すことになる．このとき，拡大状態ベクトル内の時刻 $t-L$ の部分のみに着目すると，アンサンブル $\{\boldsymbol{x}_{t-L|t}^{(i)}\}_{i=1}^N$ は，$\{\boldsymbol{x}_{t-L|t-L}^{(i)}\}_{i=1}^N$ から $L$ 回のリサンプリングを繰り返して得られたものになっている．このため，ラグ長 $L$ を長く取りすぎると，リ

サンプリングを繰り返すうちにアンサンブルの構成粒子の殆どが，ある特定の粒子の複製で占められてしまうという，退化の問題が顕在化する．したがって，$L$ をあまり長くとりすぎないよう注意すべきである．

# 4　PCATS と交通データからなる状態空間モデルの平滑化

本章では，第 2 章で述べた交通シミュレーションモデルと交通関連データを，第 3 章で述べたデータ同化を用いて統合する手法について述べる．本研究で提案するのは，アクティビティモデル PCATS と交通データで構成される状態空間モデルを，パーティクルスムーザを用いて平滑化する手法である．

## 4.1　手法の概要

本研究では，システムモデルとして 2.2 で述べた PCATS を用いる．状態空間モデルにおける状態ベクトル $\boldsymbol{x}_t$ は，時刻 $t$ における全対象個人の現在地ゾーン，活動内容，その後の固定活動スケジュール等，PCATS が保持している全データとして定義した．観測ベクトル $\boldsymbol{y}_t$ は，様々な種類のものを利用できるようにし，その種類に応じて観測モデルを設定した．データ同化の実装手法としてはモデルの非線形性・非正規性を考慮してパーティクルフィルタを用いることとし，第 1 章で述べたような背景から，平滑化手法を採用することで，時空間プリズム制約への対応を試みる．

## 4.2　システムモデル

システムモデルには，アクティビティモデルの PCATS を用いる．その概要は 2.2 に述べた通りである．本研究の実装において，PCATS は 1 日を午前 3 時から翌日午前 3 時までの 24 時間とし，その中で 1 分単位で設定した任意の時刻において，中断して結果を出力することができる．そのため，シミュレーションの開始時刻・シミュレーションの終了（中断）時刻を設定することで，状態空間モデルにおける時刻 $t$ を表現できる．必要な入力データは，以下の 2 つの場合によって異なる．即ち，

**初期入力**　午前 3 時から PCATS による推定を開始する場合
**中断明け入力**　（前回中断した）任意の時刻から PCATS による推定を開始（再開）する場合

である．初期入力の場合に必要な入力データは表 1 の通りであり，この場合，粒子数を $N$ として，PCATS を $N$ 回実行すると，その出力は，設定した中断時刻を $t = 1$ とした予測分布 $\boldsymbol{x}_{1|0}$ を Monte Carlo 近似するアンサンブルを表すことになる．中断明け入力の場合に必要な入力データは，表 2 に示す，4.3 で述べる出力データのうち，活動詳細の部分である．本研究の提案手法では，時刻 $t-1$ から時刻 $t$ への予測分布 $\boldsymbol{x}_{t|t-1}$ の Monte Carlo 近似を，PCATS の出力データファイルとして得る．そこに時刻 $t$ での観測データを 4.5 で述べる観測モデルを用いて取り込んだフィルタ分布 $\boldsymbol{x}_{t|t}$ の Monte Carlo 近似も，データ形式としては PCATS 出力と同じである．そのため，これを中断明け入力ファイルとして用いることができ，次の予測分布 $\boldsymbol{x}_{t+1|t}$ の Monte Carlo 近似を得ることができる．

3.4.1 に示したパーティクルフィルタのアルゴリズムのうち，1. の操作が PCATS の初期入力によって，2. の (a)，(b) の操作が PCATS の中断明け入力によって得られる．

## 4.3　状態ベクトル

状態ベクトル $\boldsymbol{x}_t$ は，時刻 $t$ における全対象個人の現在地ゾーン，活動内容，その後の固定活動スケジュール等，PCATS が保持している全データとして定義した．PCATS の出力は個人別活動別の活動詳細ファイル

表 1　PCATS における入力データの一覧．これは，1 期目の予測分布 $\boldsymbol{x}_{1|0}$ を近似するアンサンブルを作成する際に必要となる．

| 種類 | データ内容 | 備考 |
|---|---|---|
| 個人属性 | 個人 ID | |
| | カテゴリ | 就業者，就学者，その他 |
| | 居住地 | ゾーン番号 |
| | 就業・就学地 | ゾーン番号 |
| | 性別 | |
| | 年代 | 5 歳刻み |
| | 職業 | PT 調査と同一区分 |
| | 免許保有有無 | |
| | 拡大係数 | |
| 世帯属性 | 世帯人数 | |
| | 保有自動車数 | |
| | 5 歳以下人数 | |
| | 就業者人数 | |
| | 高校生人数 | |
| | 中学生人数 | |
| | 世帯主年齢 | |
| | 世帯主性別 | |
| 固定活動スケジュール | 活動内容 | 就業，就学，在宅 |
| | 開始時刻 | |
| | 終了時刻 | |
| | 活動場所 | ゾーン番号 |
| 地域属性 | 面積 | |
| | 人口 | |
| | サービス事業所数 | |
| | 人口密度 | |
| | サービス事業所密度 | |
| 交通ネットワーク属性（OD 別） | 公共交通所要時間 | |
| | 公共交通運賃 | |
| | 公共交通平均乗換回数 | |
| | 自動車所要時間 | |
| | 徒歩所要時間 | |
| | 道のり | |

と，個人別の時空間プリズム制約ファイルから構成される．その全容は表 2 の通りであり，状態ベクトルはこのうちの活動詳細ファイルの部分である．時空間プリズム制約ファイルは便宜上出力されるが，情報は活動詳細ファイルで足りており，PCATS での計算上の次期入力ファイルとしては不要である．

なお，実装上，活動詳細ファイルには，PCATS の開始（再開）時刻から終了（中断）時刻までにかかわる活動のみならず，午前 3 時から終了（中断）時刻までにかかわる全ての活動が記録される．この点で，PCATS 出力ファイルは，拡大状態ベクトル

$$\tilde{\boldsymbol{x}}_t = \begin{pmatrix} \boldsymbol{x}_t \\ \boldsymbol{x}_{t-1} \\ \vdots \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_1 \end{pmatrix} \tag{37}$$

を表していると言える．

表 2　PCATS における出力データの一覧．これらのデータが状態ベクトルの各要素に相当し，その粒子の集合が予測アンサンブルを構成する．また，観測データを同化したフィルタアンサンブルも，予測アンサンブルからの粒子の復元抽出で生成されるため，粒子内の個々の要素を構成するデータは，この表の通りである．

| 種類 | データ内容 | 備考 |
|---|---|---|
| 活動詳細（個人別活動別） | 個人 ID | |
| | 活動内容 | 就業，就学，自由活動 |
| | 活動開始時刻 | |
| | 活動終了時刻 | |
| | 活動場所 | ゾーン番号 |
| | 活動施設種類 | |
| | 固定活動ダミー | |
| | 活動後の移動有無 | あり，なし，処理中断 |
| | 移動開始時刻 | 処理中断の場合あり |
| | 移動終了時刻 | 処理中断の場合あり |
| | 移動手段 | 処理中断の場合あり |
| | 活動開始時点での自動車の場所 | ゾーン番号 |
| | 活動開始時点での自動車の場所 | 施設種類 |
| | 活動が中断中か否か | |
| | 中断中の活動種類 | 固定活動，自由活動，在宅，移動中 |
| 時空間プリズム制約（個人別） | 個人 ID | |
| | 中断中の活動開始時刻 | |
| | 中断中の活動終了時刻 | |
| | 中断中の活動場所 | ゾーン番号 |
| | この活動終了後に移動可能な場所 | ゾーン番号列挙 |

## 4.4　観測ベクトル

一般状態空間モデルでは，一つの時系列の中で，観測ベクトルとして様々な種類のデータを設定することができ，それぞれに応じて観測モデルを設定することが可能である．観測ベクトルには高いデータ取得頻度が必要となることから，モバイル空間統計と IC カードデータの 2 種類を観測ベクトルとして設定した．

### 4.4.1　モバイル空間統計

モバイル空間統計は，2.3.2 で述べたように，1 時間ごとの 500[m] メッシュ単位の人口データである．本研究では PCATS に合わせて対象地域をゾーン単位で表現するため，メッシュ人口をゾーン人口に面積按分する必要がある．得られる観測ベクトルは，対象ゾーン数を $J$ として，$J$ 次元のベクトル

$$\boldsymbol{y}_t^{\mathrm{mobile}} = (y_t^{(1)} \ \cdots \ y_t^{(j)} \ \cdots \ y_t^{(J)})' \tag{38}$$

となる．但し，$y_t^{(j)}$ は，時刻 $t$ における $j$ 番目のゾーンの人口を表す．

### 4.4.2　IC カードデータ

IC カードデータとは，2.3.6 で述べた交通事業者保有データを想定したものである．実際に入手することは現時点では不可能であるが，本研究では第 5 章で述べるように，PT 調査データを全数の真値とする仮想都市圏を仮定することで，疑似観測データとして利用した．その内容は，PT 調査において公共交通機関を代表交通機関とするトリップの，個人 ID・拡大係数・出発地・出発時刻・目的地・到着時刻を並べたものである．このままでは時間的に非集計なデータであるために，観測ベクトルとして利用するには何らかの方法で集計する必要がある．その方法には様々なものが考えられる．今回は，対象ゾーン数を $J$ として，

$$\boldsymbol{y}_t^{\mathrm{IC_{zone}}} = (y_{t-1:t}^{(1_{\mathrm{O}})} \, y_{t-1:t}^{(1_{\mathrm{D}})} \ \cdots \ y_{t-1:t}^{(j_{\mathrm{O}})} \, y_{t-1:t}^{(j_{\mathrm{D}})} \ \cdots \ y_{t-1:t}^{(J_{\mathrm{O}})} \, y_{t-1:t}^{(J_{\mathrm{D}})})' \tag{39}$$

$$\boldsymbol{y}_t^{\mathrm{IC_{OD}}} = (y_{t-1:t}^{(1,1)} \ \cdots \ y_{t-1:t}^{(1,J)} \, y_{t-1:t}^{(2,1)} \ \cdots \ y_{t-1:t}^{(j,k)} \ \cdots \ y_{t-1:t}^{(J,1)} \ \cdots \ y_{t-1:t}^{(J,J)})' \tag{40}$$

を観測ベクトルとして設定した．$\boldsymbol{y}_t^{\mathrm{IC_{zone}}}$ は，到着時刻基準で各ゾーン別発着別のトリップ人数を集計した $2J$ 次元のベクトルであり，$y_{t-1:t}^{(j_{\mathrm{O}})}$ は，$j$ 番目のゾーンを出発地とし，時刻 $t-1$ から $t$ までに目的地に到着した人の数である．同様に，$y_{t-1:t}^{(j_{\mathrm{D}})}$ は，$j$ 番目のゾーンを目的地とし，時刻 $t-1$ から $t$ までに目的地に到着した人の数である．$\boldsymbol{y}_t^{\mathrm{IC_{OD}}}$ は，到着時刻基準で各 OD（発着ゾーンの組合せ）別のトリップ人数を集計した $J^2$ 次元のベクトルであり，$y_{t-1:t}^{(j,k)}$ は，$j$ 番目のゾーンを出発地，$k$ 番目のゾーンを目的地とし，時刻 $t-1$ から $t$ までに目的地に到着した人の数である．

## 4.5　観測モデル

観測モデルは，その時刻 $t$ に得られた観測ベクトルの種類に応じて，ベクトル間の重み付き Euclid 平方距離の逆数である類似度を尤度と定義することで定式化した．その詳細は以下の通りである．3.4.1 に示したパーティクルフィルタのアルゴリズムのうち，2. の (c) の操作が，本節で述べるものである．

まず，時刻 $t$ に得られた観測データが 1 種類の場合について述べる．モバイル空間統計の観測ベクトル $\boldsymbol{y}_t^{\mathrm{mobile}}$ に対しては，次の通りである．PCATS の出力である時刻 $t$ での予測アンサンブル $\{\boldsymbol{x}_{t|t-1}^{(i)}\}_{i=1}^{N}$ に含まれる粒子 $\boldsymbol{x}_{t|t-1}^{(i)}$ について，観測演算子 $\boldsymbol{h}_t^{\mathrm{mobile}}(\boldsymbol{x}_{t|t-1}^{(i)})$ を，時刻 $t$ での $\boldsymbol{x}_{t|t-1}^{(i)}$ のゾーンごとの人口を集計して並べる $J$ 次元ベクトル関数として定義する．但し，$J$ はゾーン数である．$\boldsymbol{h}_t^{\mathrm{mobile}}(\boldsymbol{x}_{t|t-1}^{(i)})$ の要素を並べたも

のを，$\boldsymbol{h}_t^{\mathrm{mobile}}(\boldsymbol{x}_{t|t-1}^{(i)}) = (h_t^{(1)}(\boldsymbol{x}_{t|t-1}^{(i)}) \ \ldots \ h_t^{(j)}(\boldsymbol{x}_{t|t-1}^{(i)}) \ \ldots \ h_t^{(J)}(\boldsymbol{x}_{t|t-1}^{(i)}))'$ とする．このとき，粒子 $\boldsymbol{x}_{t|t-1}^{(i)}$ の尤度 $\lambda_t^{(i)}$ を，次の通り定義する．

$$\lambda_t^{(i)} = p(\boldsymbol{y}_t^{\mathrm{mobile}}|\boldsymbol{x}_{t|t-1}^{(i)}) \overset{\mathrm{def.}}{=} \left( \sum_{j=1}^{J} \frac{[y_t^{(j)} - h_t^{(j)}(\boldsymbol{x}_{t|t-1}^{(i)})]^2}{(y_t^{(j)})^2} \right)^{-1} \tag{41}$$

このような $1/(y_t^{(j)})^2$ による重み付き Euclid 平方距離を採用した理由は，ある複数ゾーンにおいて人口の誤差が同数であった場合でも，人口の少ないゾーンの誤差の重みを大きく設定できるからである．

IC カードデータの観測ベクトル $\boldsymbol{y}_t^{\mathrm{IC_{zone}}}, \boldsymbol{y}_t^{\mathrm{IC_{OD}}}$ に対しても同様である．観測演算子 $\boldsymbol{h}_t^{\mathrm{IC_{zone}}}(\boldsymbol{x}_{t|t-1}^{(i)})$ を，$\boldsymbol{x}_{t|t-1}^{(i)}$ に含まれる，時刻 $t-1$ から $t$ までに目的地に到着した，各ゾーン別発着別の，公共交通を利用したトリップ人数を集計して並べる $2J$ 次元ベクトル関数として，また，観測演算子 $\boldsymbol{h}_t^{\mathrm{IC_{OD}}}(\boldsymbol{x}_{t|t-1}^{(i)})$ を，$\boldsymbol{x}_{t|t-1}^{(i)}$ に含まれる，時刻 $t-1$ から $t$ までに目的地に到着した，各 OD 別の，公共交通を利用したトリップ人数を集計して並べる $J^2$ 次元ベクトル関数として，それぞれ定義する．但し，$J$ はゾーン数である．また，$\boldsymbol{h}_t^{\mathrm{IC_{zone}}}(\boldsymbol{x}_{t|t-1}^{(i)})$ の要素として $h_t^{(j_{\mathrm{O}})}(\boldsymbol{x}_{t|t-1}^{(i)}), h_t^{(j_{\mathrm{D}})}(\boldsymbol{x}_{t|t-1}^{(i)})$ を 4.4.2 の $y_{t-1:t}^{(j_{\mathrm{O}})}, y_{t-1:t}^{(j_{\mathrm{D}})}$ と同様に，$\boldsymbol{h}_t^{\mathrm{IC_{OD}}}(\boldsymbol{x}_{t|t-1}^{(i)})$ の要素として $h_t^{(j,k)}(\boldsymbol{x}_{t|t-1}^{(i)})$ を 4.4.2 の $y_{t-1:t}^{(j,k)}$ と同様に，それぞれ定義する．このとき，粒子 $\boldsymbol{x}_{t|t-1}^{(i)}$ の尤度 $\lambda_t^{(i)}$ を，次の通り定義する．

$$\lambda_t^{(i)} = \begin{cases} p(\boldsymbol{y}_t^{\mathrm{IC_{zone}}}|\boldsymbol{x}_{t|t-1}^{(i)}) & \overset{\mathrm{def.}}{=} \left( \sum_{j=1}^{J} \left( \frac{[y_{t-1:t}^{(j_{\mathrm{O}})} - h_t^{(j_{\mathrm{O}})}(\boldsymbol{x}_{t|t-1}^{(i)})]^2}{(y_{t-1:t}^{(j_{\mathrm{O}})})^2} + \frac{[y_{t-1:t}^{(j_{\mathrm{D}})} - h_t^{(j_{\mathrm{D}})}(\boldsymbol{x}_{t|t-1}^{(i)})]^2}{(y_{t-1:t}^{(j_{\mathrm{D}})})^2} \right) \right)^{-1} \\ & \quad \text{（ゾーン別発着別トリップ人数の場合）} \\ p(\boldsymbol{y}_t^{\mathrm{IC_{OD}}}|\boldsymbol{x}_{t|t-1}^{(i)}) & \overset{\mathrm{def.}}{=} \left( \sum_{j=1}^{J} \sum_{k=1}^{J} \frac{[y_{t-1:t}^{(j,k)} - h_t^{(j,k)}(\boldsymbol{x}_{t|t-1}^{(i)})]^2}{(y_{t-1:t}^{(j,k)})^2} \right)^{-1} \\ & \quad \text{（OD 別トリップ人数の場合）} \end{cases} \tag{42}$$

次に，時刻 $t$ に得られた観測データが 2 種類の場合について述べる．この場合，まずは観測データが 1 種類の場合と同様に，モバイル空間統計の観測に関する粒子 $\boldsymbol{x}_{t|t-1}^{(i)}$ の尤度 $\lambda_t^{\mathrm{mobile}(i)}$ を式 (41) の $\lambda_t^{(i)}$ で，IC カードデータの観測に関する粒子 $\boldsymbol{x}_{t|t-1}^{(i)}$ の尤度 $\lambda_t^{\mathrm{IC}(i)}$ を式 (42) の $\lambda_t^{(i)}$ で，それぞれ定義する．その上で，最終的な粒子 $\boldsymbol{x}_{t|t-1}^{(i)}$ の尤度 $\lambda_t^{(i)}$ を，

$$\lambda_t^{(i)} = (1-\alpha)\lambda_t^{\mathrm{mobile}(i)} + \alpha\lambda_t^{\mathrm{IC}(i)} \tag{43}$$

で与える．但し，$\alpha$ は 2 種類の観測データの重み付けをするために外生的に与えるパラメータである．

## 4.6　手法の流れ

以上に述べた状態ベクトル・観測ベクトル・システムモデル・観測モデルからなる一般状態空間モデルを，パーティクルスムーザを用いて平滑化することで，状態推定を行う．本節では，その具体的な流れを示す．

まず，PCATS 出力ファイルが表す式 (37) の拡大状態ベクトル $\tilde{\boldsymbol{x}}$ に対するパーティクルフィルタの手続きとして，以下のような逐次的方法で，各ステップの予測分布・フィルタ分布の Monte Carlo 近似を求めていく．

1. 開始時刻を午前 3 時，中断時刻を $t=1$ とした PCATS を $N$ 回実行し，予測アンサンブル $\{\tilde{\boldsymbol{x}}_{1|0}^{(i)}\}_{i=1}^{N}$ を生成する．

2. $t = 1, \ldots, T$ について，次のステップを実行する．但し，$t = 1$ については，(b) から実行する．
   (a) 各粒子番号 $i$ $(i = 1, \ldots, N)$ について，フィルタアンサンブルを構成する粒子 $\tilde{\boldsymbol{x}}_{t-1|t-1}^{(i)}$ に対して PCATS を実行し，予測アンサンブル $\{\tilde{\boldsymbol{x}}_{t|t-1}^{(i)}\}_{i=1}^{N}$ を得る．
   (b) 各 $i$ $(i = 1, \ldots, N)$ について，時刻 $t$ での観測データの種類に応じた 4.5 での定義に基づいて，それぞれの粒子の尤度 $\lambda_t^{(i)}$ を計算する．
   (c) 各 $i$ $(i = 1, \ldots, N)$ について，正規化尤度 $\beta_t^{(i)} = \lambda_t^{(i)} / (\sum_{j=1}^{N} \lambda_t^{(j)})$ を求める．
   (d) 予測アンサンブル $\{\tilde{\boldsymbol{x}}_{t|t-1}^{(i)}\}_{i=1}^{N}$ から，3.4.3 の後半で述べた方法で，リサンプリングを行う．抽出個数の期待値 $N\beta_t^{(i)}$ の整数部分を $\bar{m}_t^{(i)}$ として，各粒子 $\tilde{\boldsymbol{x}}_{t|t-1}^{(i)}$ をそれぞれ $\bar{m}_t^{(i)}$ 個ずつ抽出する．これで，$\sum_{i=1}^{N} \bar{m}_t^{(i)}$ 個の抽出が完了する．
   (e) 残りの粒子を抽出するために，各粒子 $\boldsymbol{x}_{t|t-1}^{(i)}$ が確率 $(N\beta_t^{(i)} - \bar{m}_t^{(i)})/(N - \sum_{i=1}^{N} \bar{m}_t^{(i)})$ で抽出されるようなランダム抽出を $N - \sum_{i=1}^{N} \bar{m}_t^{(i)}$ 回繰り返す．
   (f) 得られた $N$ 個の粒子で，フィルタアンサンブル $\{\tilde{\boldsymbol{x}}_{t|t}^{(i)}\}_{i=1}^{N}$ を構成する．

次に，本手法での最終的な点推定値を求める手法を説明する．採用するのは，3.4.4 の通り，最大個数複製された粒子を点推定値とする方法である．具体的には以下の通りである．上記の逐次的手順で最終時刻 $T$ でのフィルタアンサンブル $\{\tilde{\boldsymbol{x}}_{T|T}^{(i)}\}_{i=1}^{N}$ が得られたら，その構成粒子のうち，直前のリサンプリング，つまり $\{\tilde{\boldsymbol{x}}_{T|T-1}^{(i)}\}_{i=1}^{N}$ からの復元抽出において，最も多くの個数抽出された粒子の複製（この粒子の新アンサンブル $\{\tilde{\boldsymbol{x}}_{T|T}^{(i)}\}_{i=1}^{N}$ における番号 $i$ を $i^{\mathrm{mode}}$ とする）について，その要素である（拡大前の）状態ベクトルのうち，任意の時刻 $t$ に関するものを取り出す．即ち，

$$\tilde{\boldsymbol{x}}_{T|T}^{(i^{\mathrm{mode}})} = \begin{pmatrix} \boldsymbol{x}_{T|T}^{(i^{\mathrm{mode}})} \\ \boldsymbol{x}_{T-1|T}^{(i^{\mathrm{mode}})} \\ \vdots \\ \boldsymbol{x}_{t|T}^{(i^{\mathrm{mode}})} \\ \vdots \\ \boldsymbol{x}_{2|T}^{(i^{\mathrm{mode}})} \\ \boldsymbol{x}_{1|T}^{(i^{\mathrm{mode}})} \end{pmatrix} \tag{44}$$

から，任意の時刻 $t$ に関する部分 $\boldsymbol{x}_{t|T}^{(i^{\mathrm{mode}})}$ を取り出す．これを，本研究の提案手法での，時刻 $t$ における最終的な点推定値 $\hat{\boldsymbol{x}}_{t|T}$ とする．なお，最大複製個数をとる粒子が複数あった場合には，それらの中で粒子の尤度 $\lambda_t^{(i)}$ が最大の粒子を採用する．

本研究の提案手法が，従来手法と決定的に違うのは，拡大状態ベクトル $\tilde{\boldsymbol{x}}$ を用いた平滑化手法を採用することで，時空間プリズム制約からの逸脱を心配すること無く一期先予測とフィルタリングを繰り返すことができる点である．従来手法では，各 $t$ ごとに，フィルタアンサンブル $\{\boldsymbol{x}_{t|t}^{(i)}\}_{i=1}^{N}$ の構成粒子のうち，直前の予測アンサンブルから $\{\boldsymbol{x}_{t|t-1}^{(i)}\}_{i=1}^{N}$ のリサンプリング時に最も複製個数が多かった粒子 $\boldsymbol{x}_{t|t}^{(i_t^{\mathrm{mode}})}$ を，時刻 $t$ における点推定値としていた．しかし，それでは $\boldsymbol{x}_{t|t}^{(i_t^{\mathrm{mode}})}$ と $\boldsymbol{x}_{t+1|t+1}^{(i_{t+1}^{\mathrm{mode}})}$ に整合性が取れる保証はどこにも無い．即ち，$\boldsymbol{x}_{t|t}^{(i_t^{\mathrm{mode}})}$ では各個人の位置と時空間プリズム制約が与えられるが，$\boldsymbol{x}_{t+1|t+1}^{(i_{t+1}^{\mathrm{mode}})}$ での個人位置が，その時空間プリズム制約を満たしているとは限らないのである．従来手法ではこの問題に対し，各時刻 $t$，各粒子 $i$ について，各個人が時空間プリズム制約を満たしているか否かを都度調べ，満たしていない個人については観測データの同化を行わず，シミュレーションによる結果をそのまま利用して結果としていた．しかし，この作業は手間も

計算負荷も大きい上に，観測データによる同化を行っていないという点で，単なるシミュレーションと変わっていない．

一方で本研究の提案手法では，最終的なフィルタアンサンブル $\{\tilde{\boldsymbol{x}}_{T|T}^{(i)}\}_{i=1}^{N}$ が得られるまでは点推定値を示さない．そして，$\{\tilde{\boldsymbol{x}}_{T|T}^{(i)}\}_{i=1}^{N}$ の全ての粒子は，PCATS による一期先予測とその複製抽出によってのみ生成されてきたので，任意の時刻で時空間プリズム制約を満たすことが，PCATS 自身によって保証されている．最終的な点推定値 $\hat{\boldsymbol{x}}_{t|T}$ は $\{\tilde{\boldsymbol{x}}_{T|T}^{(i)}\}_{i=1}^{N}$ から取り出された粒子 $\boldsymbol{x}_{t|T}^{(i^{\mathrm{mode}})}$ であるから，当然に時空間プリズム制約を満たすことになる．

# 5　適用

本章では，第 4 章で述べた提案手法を実際に適用した対象のデータの詳細と，適用の結果について述べる．結果の評価は，各時刻におけるゾーン人口と，ゾーン別発着別トリップ人数で行った．

なお，本研究では，

- IC カードデータを疑似的に生成し，観測データとして用いることの有効性を検証するため
- 適用結果の検証方法として，全数の真値との比較を採用するため

といった目的から，PT 調査の結果を全数の真値とする，仮想都市圏を適用対象とした．以下，必要に応じて，実在の都市圏に対する PT 調査を真 PT 調査，そこから真 PT 調査の抽出率に倣ってランダム抽出した PT 調査を疑似 PT 調査と呼んで区別する．

## 5.1　対象のデータ・設定条件

### 5.1.1　対象地域

対象としたのは，東京 23 区 115 ゾーンであり，ゾーンレベルは PT 踏査における計画基本ゾーンを採用した．計画基本ゾーンとは，小ゾーン（夜間人口約 15,000 人を目安とし，地区計画の単位となるゾーンレベル）を数個集めて構成し，広域における計画単位として，また地域としてのまとまりのある交通計画の単位となるゾーンレベルである [18]．本研究では，この 23 区 115 ゾーンの内内交通を対象とした．そのため，対象地域内に居住する個人を対象とし，その中でも，対象地域外に固定活動を持つ個人は対象外とした．

### 5.1.2　パーソントリップ調査データ

2008 年（平成 20 年）10-11 月に実施された，第 5 回東京都市圏パーソントリップ調査データを使用した．この PT 調査では，東京都・神奈川県・埼玉県・千葉県の全域（島嶼部を除く）と茨城県南部の合計 238 市区町村を対象とし，対象域内に居住する約 1600 万世帯のうち，約 140 万世帯を無作為抽出し，34 万 0619 世帯，73 万 3873 人，180 万 1258 トリップ分の調査票を回収した．世帯数基準での抽出率は約 2.1% である．この真 PT 調査データから，疑似 PT 調査データとして 7500 世帯分（抽出率約 2.2%）のデータをランダム抽出した．疑似 PT 調査の疑似調査票の規模は，7500 世帯，1 万 6176 人，3 万 9526 トリップ分である．疑似 PT 調査の拡大係数は，居住市区町村別・性別・年齢階層別に属性を仮定し，その属性を持つ真 PT 調査の人数を疑似 PT 調査の人数で割って小数点以下を四捨五入した整数値で与えた．

### 5.1.3　PCATS 固定活動スケジュール

PCATS 固定活動スケジュールファイルは，次のように作成した．まず，PT 調査データに含まれる全トリップから，固定活動開始トリップ候補として，

- 到着地区分が「勤務先」または「通学先」である
- 移動の目的が「勤務先へ」または「通学先へ」である
- 直前のトリップが業務等へ向かうものではない

の条件を満たすものを抽出した．3 つ目の条件を加えたのは，取引先からの帰社等のトリップを取り除くためである．次に，固定活動終了トリップ候補として，

- 出発地区分が「勤務先・通学先」である
- 移動の目的が「帰宅」または「私用」である
- 移動先が業務等ではない

の条件を満たすものを抽出した．これらを対照し，

- 固定活動開始トリップ候補と固定活動終了トリップ候補が同じ個人によって行われている
- 固定活動開始トリップ候補の到着地と固定活動終了トリップ候補の出発地が同一

であるものを抽出して，これを固定活動とした．

更に，PCATS における 1 日の最初と最後の 10 分，即ち午前 3 時から午前 3 時 10 分までと，翌午前 2 時 50 分から翌午前 3 時までは，全ての個人が自宅での固定活動を行うものとした．これは，1 日の最初と最後には必ず自宅にいるという仮定を置いたことになる．1 日の最初と最後の自宅滞在以外に固定活動を持つ東京都市圏全域の個人の数は，真 PT 調査で 32 万 3872 人，疑似 PT 調査で 6960 人であった．

本研究での適用対象とした入力データは，疑似 PT 調査から作成した固定活動スケジュールファイルのうち，23 区内に居住し，固定活動を 23 区外に持つ個人を除いたもので，そこに含まれる個人数は，自宅滞在のみの個人を含めて 2893 人，彼らの拡大係数の合計は 12 万 4885 人であった．推定においては，この個人を各々拡大係数に応じて拡大することで，真 PT 調査を全数の真値とする都市圏の交通の再現を図る．なお，真 PT 調査から作成した固定活動スケジュールファイルのうち，23 区内に居住し，固定活動を 23 区外に持つ個人を除いた数は，自宅滞在のみの個人を含めて 12 万 9279 人であった．この時点での再現率は 96.6% である．

### 5.1.4　その他 PCATS 入力データ

各ゾーンの地域属性データ（面積，人口，サービス事業所数，人口密度，サービス事業所密度）は，次のように作成した．面積は，ArcGIS から求めた．人口は，東京都による「住民基本台帳による東京都の世帯と人口　平成 20 年 1 月」の「第 5 表　区市町村，町丁別の世帯数及び男女別人口」から，PT 調査における計画基本ゾーンと町丁目の対応表を使って求めた．サービス事業所数は，経済産業省による「平成 26 年商業統計メッシュデータ」による小売業事業所数の 500[m] メッシュデータを面積按分して求めた．但し，面積按分においては，水域を多く含むメッシュ等において，過小評価となっていることに注意する必要がある．

各 OD の交通ネットワーク属性データ（公共交通所要時間，公共交通運賃，公共交通平均乗換回数，自動車所要時間，徒歩所要時間，道のり）は，次のように作成した．公共交通運賃は，同一ゾーン内については初乗り運賃を，それ以外の OD については，表 3 に示すように定めた代表駅同士の運賃を，2017 年 12 月現在の情報に基づいて採用し，設定した．公共交通所要時間，公共交通平均乗換回数，自動車所要時間，徒歩所要時間は，真 PT 調査から当該 OD で代表交通機 g 関別にトリップを全て抽出し，それらの平均を採用した．なお，真 PT 調査でデータが 4 件以下しか存在しない（0 件も含む）OD の組合せについては，真 PT 調査のデータ数が不十分と判断し，次のように扱った．自動車所要時間と徒歩所要時間については，表 3 に示すように定めた代表駅の地点をゾーン代表点として，2018 年 1 月現在の情報で Google Maps API を用いて計算した．なお，自動車については，有料道路を経由しない設定とした．公共交通については，選択される可能性が極めて低いと判断し，所要時間を 9999 分，乗換回数を 99 回と設定した．

表 3　東京 23 区の計画基本ゾーン一覧と，設定したその代表駅．交通ネットワーク属性は，この代表駅同士の OD を中心に構成した．

| 区 | 通し番号 | ゾーンコード | ゾーン名 | 代表駅 |
|---|---|---|---|---|
| 千代田区 | 1 | 10 | 千代田 1 | 東京（大手町） |
| | 2 | 11 | 千代田 2 | 市ヶ谷 |
| | 3 | 12 | 千代田 3 | 御茶ノ水 |
| | 4 | 13 | 千代田 4 | 有楽町（日比谷） |
| 中央区 | 5 | 20 | 中央 1 | 馬喰町（馬喰横山） |
| | 6 | 21 | 中央 2 | 日本橋 |
| | 7 | 22 | 中央 3 | 銀座 |
| | 8 | 23 | 中央 4 | 八丁堀 |
| | 9 | 24 | 中央 5 | 月島 |
| 港区 | 10 | 30 | 港 1 | 青山一丁目 |
| | 11 | 31 | 港 2 | 浜松町（大門） |
| | 12 | 32 | 港 3 | 品川 |
| | 13 | 33 | 港 4 | 田町（三田） |
| | 14 | 34 | 港 5 | 白金高輪 |
| 目黒区 | 15 | 110 | 目黒 1 | 目黒[*1] |
| | 16 | 111 | 目黒 2 | 自由が丘 |
| | 17 | 112 | 目黒 3 | 中目黒 |
| 品川区 | 18 | 120 | 品川 1 | 五反田 |
| | 19 | 121 | 品川 2 | 天王洲アイル |
| | 20 | 122 | 品川 3 | 大井町 |
| | 21 | 123 | 品川 4 | 中延 |
| | 22 | 124 | 品川 5 | 大井競馬場前 |
| 大田区 | 23 | 130 | 大田 1 | 馬込 |
| | 24 | 131 | 大田 2 | 大森 |
| | 25 | 132 | 大田 3 | 昭和島 |
| | 26 | 133 | 大田 4 | 羽田空港 |
| | 27 | 134 | 大田 5 | 雑色 |
| | 28 | 135 | 大田 6 | 下丸子 |
| | 29 | 136 | 大田 7 | 田園調布 |
| | 30 | 137 | 大田 8 | 大岡山 |
| | 31 | 138 | 大田 9 | 蒲田（京急蒲田） |
| 豊島区 | 32 | 210 | 豊島 1 | 駒込 |

表 3 は次ページに続く

[*1] 当該ゾーン（概ね，東横線祐天寺―目黒線武蔵小山―目黒線―山手線―日比谷線―東横線に囲まれたエリア）内に駅が無いため，便宜上，目黒駅を代表駅とする．なお，目黒駅の実所在地は 120（品川 1）ゾーン内である．

表 3　東京 23 区の計画基本ゾーン一覧と，設定したその代表駅（続き）

| 区 | 通し番号 | ゾーンコード | ゾーン名 | 代表駅 |
|---|---|---|---|---|
| | 33 | 211 | 豊島 2 | 池袋 |
| | 34 | 212 | 豊島 3 | 千川 |
| 文京区 | 35 | 220 | 文京 1 | 東大前 |
| | 36 | 221 | 文京 2 | 本郷三丁目 |
| | 37 | 222 | 文京 3 | 後楽園（春日） |
| | 38 | 223 | 文京 4 | 新大塚 |
| 新宿区 | 39 | 230 | 新宿 1 | 神楽坂 |
| | 40 | 231 | 新宿 2 | 四ツ谷 |
| | 41 | 232 | 新宿 3 | 新宿（西武新宿） |
| | 42 | 233 | 新宿 4 | 都庁前 |
| | 43 | 234 | 新宿 5 | 高田馬場 |
| | 44 | 235 | 新宿 6 | 早稲田 |
| 渋谷区 | 45 | 240 | 渋谷 1 | 代々木 |
| | 46 | 241 | 渋谷 2 | 渋谷 |
| | 47 | 242 | 渋谷 3 | 恵比寿 |
| | 48 | 243 | 渋谷 4 | 代々木上原 |
| 荒川区 | 49 | 310 | 荒川 1 | 南千住 |
| | 50 | 311 | 荒川 2 | 日暮里 |
| | 51 | 312 | 荒川 3 | 熊野前 |
| 台東区 | 52 | 320 | 台東 1 | 入谷 |
| | 53 | 321 | 台東 2 | 浅草 |
| | 54 | 322 | 台東 3 | 浅草橋 |
| | 55 | 323 | 台東 4 | 上野（京成上野） |
| 墨田区 | 56 | 330 | 墨田 1 | 鐘ヶ淵 |
| | 57 | 331 | 墨田 2 | 八広 |
| | 58 | 332 | 墨田 3 | 押上 |
| | 59 | 333 | 墨田 4 | 本所吾妻橋 |
| | 60 | 334 | 墨田 5 | 錦糸町 |
| | 61 | 335 | 墨田 6 | 両国 |
| | 62 | 336 | 墨田 7 | 曳舟（京成曳舟） |
| 江東区 | 63 | 340 | 江東 1 | 亀戸 |
| | 64 | 341 | 江東 2 | 大島 |
| | 65 | 342 | 江東 3 | 南砂町 |
| | 66 | 343 | 江東 4 | 新木場 |
| | 67 | 344 | 江東 5 | 東陽町 |
| | 68 | 345 | 江東 6 | 越中島 |

表 3 は次ページに続く

表 3　東京 23 区の計画基本ゾーン一覧と，設定したその代表駅（続き）

| 区 | 通し番号 | ゾーンコード | ゾーン名 | 代表駅 |
|---|---|---|---|---|
| | 69 | 346 | 江東 7 | 有明（国際展示場） |
| | 70 | 347 | 江東 8 | 門前仲町 |
| | 71 | 348 | 江東 9 | 清澄白河 |
| | 72 | 349 | 江東 10 | 住吉 |
| 中野区 | 73 | 410 | 中野 1 | 鷺ノ宮 |
| | 74 | 411 | 中野 2 | 中野 |
| | 75 | 412 | 中野 3 | 中野坂上 |
| 杉並区 | 76 | 420 | 杉並 1 | 高円寺 |
| | 77 | 421 | 杉並 2 | 永福町 |
| | 78 | 422 | 杉並 3 | 高井戸 |
| | 79 | 423 | 杉並 4 | 荻窪 |
| | 80 | 424 | 杉並 5 | 井荻 |
| 世田谷区 | 81 | 430 | 世田谷 1 | 下北沢 |
| | 82 | 431 | 世田谷 2 | 三軒茶屋 |
| | 83 | 432 | 世田谷 3 | 二子玉川 |
| | 84 | 433 | 世田谷 4 | 等々力 |
| | 85 | 434 | 世田谷 5 | 成城学園前 |
| | 86 | 435 | 世田谷 6 | 千歳烏山 |
| | 87 | 436 | 世田谷 7 | 桜上水 |
| | 88 | 437 | 世田谷 8 | 経堂 |
| 練馬区 | 89 | 510 | 練馬 1 | 平和台 |
| | 90 | 511 | 練馬 2 | 練馬 |
| | 91 | 512 | 練馬 3 | 富士見台 |
| | 92 | 513 | 練馬 4 | 石神井公園 |
| | 93 | 514 | 練馬 5 | 大泉学園 |
| | 94 | 515 | 練馬 6 | 光が丘 |
| 板橋区 | 95 | 520 | 板橋 1 | ときわ台 |
| | 96 | 521 | 板橋 2 | 板橋 |
| | 97 | 522 | 板橋 3 | 成増（地下鉄成増） |
| | 98 | 523 | 板橋 4 | 高島平 |
| 北区 | 99 | 530 | 北 1 | 赤羽 |
| | 100 | 531 | 北 2 | 王子 |
| | 101 | 532 | 北 3 | 田端 |
| 足立区 | 102 | 610 | 足立 1 | 綾瀬 |
| | 103 | 611 | 足立 2 | 北千住 |
| | 104 | 612 | 足立 3 | 小菅 |

表 3 は次ページに続く

表 3　東京 23 区の計画基本ゾーン一覧と，設定したその代表駅（続き）

| 区 | 通し番号 | ゾーンコード | ゾーン名 | 代表駅 |
|---|---|---|---|---|
| | 105 | 613 | 足立 4 | 扇大橋 |
| | 106 | 614 | 足立 5 | 西新井 |
| 葛飾区 | 107 | 620 | 葛飾 1 | 金町（京成金町） |
| | 108 | 621 | 葛飾 2 | 京成高砂 |
| | 109 | 622 | 葛飾 3 | 新小岩 |
| | 110 | 623 | 葛飾 4 | 亀有 |
| 江戸川区 | 111 | 630 | 江戸川 1 | 平井 |
| | 112 | 631 | 江戸川 2 | 船堀 |
| | 113 | 632 | 江戸川 3 | 小岩 |
| | 114 | 633 | 江戸川 4 | 瑞江 |
| | 115 | 634 | 江戸川 5 | 葛西 |

表 3 はこれで終わり

2.2 で述べた想定時間分布モデル，活動内容選択モデル，交通機関・目的地選択モデルのうち，想定時間分布モデルと活動内容選択モデルのパラメータについては，オリジナルの PCATS でも使用された，大阪湾岸地域の居住者を対象としたパネル調査の，1995 年 6 月に実施した第 3 回調査の一部である平日を対象としたアクティビティダイアリー調査で得られたデータを用いて構築されたものを用いた．このパネル調査は，阪神高速道路湾岸線の供用，ならびに，阪神・淡路大震災が個人の生活行動，交通行動に及ぼした影響を把握することを目的として実施されたものであり，アクティビティダイアリー調査では 916 個人からデータを得ている [16]．本研究での対象は東京 23 区居住者であるが，この 2 つのモデルに関しては京阪神都市圏と東京都市圏で差異は無いと仮定し，パラメータをそのまま利用した．交通機関・目的地選択モデルのパラメータについては，交通政策審議会陸上交通分科会鉄道部会によって推定された東京都市圏 PT 調査データを用いた非集計ロジットモデルを利用した．このモデルにより推計された交通機関別分担率を用いて，オリジナルの PCATS のパラメータに定数項補正を行うことで，東京都市圏に対応した交通機関・目的地選択モデルとした．

#### 5.1.5　観測データ

観測データとしては，疑似モバイル空間統計データと，疑似 IC カードデータを用いた．疑似モバイル空間統計データは，真 PT 調査データから，ある時刻における 23 区内居住者のゾーン人口（拡大係数の合計ではなく調査票個人数）を抽出することで作成した．疑似 IC カードデータは，真 PT 調査データから，公共交通機関を代表交通機関とするトリップの，個人 ID・拡大係数・出発地・出発時刻・目的地・到着時刻を抽出することで作成した．

疑似 IC カードデータの観測時の観測ベクトルは，ゾーン別発着別の公共交通トリップ人数である $\boldsymbol{y}_t^{\mathrm{IC}_{\mathrm{zone}}}$ を用いた．OD 別の $\boldsymbol{y}_t^{\mathrm{IC}_{\mathrm{OD}}}$ では観測ベクトルの次元が $115^2 = 13225$ 通りとかなり多く，また疑似 PT 調査データから作った PCATS 固定活動スケジュールを基にした PCATS 上の個人数は 2893 人と少ないため，多くの OD についてトリップ人数が 0 となって粒子尤度の計算に不都合を生じるからである．

#### 5.1.6　設定条件

以上のデータを用いて，次に示す 13 通りの条件で，提案手法による状態推定を行った．

**同化時刻**　午前 9 時・正午・午後 5 時・午後 9 時の 4 回
**パーティクル数**　100 個，1000 個，5000 個の 3 通り（但し，5000 個は観測データ 2 種類使用のときのみ）
**使用観測データと重み**　モバイル空間統計のみ，IC カードデータのみ，両者（$\alpha = 0.8, 0.5, 0.2$）の 5 通り

## 5.2　結果

結果の検証は，ゾーン人口と，ゾーン別発着別公共交通トリップ人数の 2 種類を指標値として行った．それぞれの条件での，同化時刻 $t$ における指標値の推定値 $h_t^{(k)}(\boldsymbol{x}_{t|T}^{(i^{\mathrm{mode}})})$ が，全数の真値と見做す真 PT 調査の値 $y_t^{(k)}$ をどの程度再現できているかを，次に定義する再現率 $r_t$ を指標として調べた．但し，$k$ は指標値ベクトルの要素の番号，$K$ はその最大値であり，$J$ をゾーン数とすると，ゾーン人口については $K = J$，ゾーン別発着別公共交通トリップ人数については $K = 2J$ となる．今回の適用では $J = 115$ である．

$$r_t \overset{\mathrm{def.}}{=} 1 - \sum_{k=1}^{K} \left| \frac{h_t^{(k)}(\boldsymbol{x}_{t|T}^{(i^{\mathrm{mode}})})}{\sum_{k=1}^{K} h_t^{(k)}(\boldsymbol{x}_{t|T}^{(i^{\mathrm{mode}})})} - \frac{y_t^{(k)}}{\sum_{k=1}^{K} y_t^{(k)}} \right| \tag{45}$$

計算環境は表 4 の通りであり，時刻 $t$ の 1 ステップ実行にかかった時間は，$N = 100$ では PCATS12-15 秒，フィルタリング 16-21 秒，$N = 1000$ では PCATS 約 2 分，フィルタリング約 10 分，$N = 5000$ では PCATS 約 10 分，フィルタリング約 3 時間 30 分であった．

表 4　計算を実行した環境．

| | |
|---|---|
| OS | Microsoft Windows® 8.1 Enterprise |
| プロセッサ | Intel® Core™i7-2600 CPU @ 3.40GHz |
| メモリ | 16.0 GB |
| プログラミング言語 | C++ |
| 統合開発環境 | Microsoft Visual Studio Express 2015 |

**従来手法との比較**

まず，観測データが 1 種類の場合における，提案手法（パーティクルスムーザ）と従来手法（パーティクルフィルタ）の比較結果を表 5 に示す．モバイル空間統計データを使用した場合と IC カードデータを使用した場合を比較すると，提案手法では IC カードデータを使用した場合の方が再現率が高い傾向にある．また，モバイル空間統計データを使用するとゾーン人口の再現率が，IC カードデータを使用するとゾーン発着トリップ人数の再現率がそれぞれ高くなるように思えるが，必ずしもそうなっている訳ではない．粒子数の増減による再現率の高低にはあまり関係性が見られない他，同一条件内での再現率の高低も粒子数の変化にあまり影響を受けていない．提案手法と従来手法を比較すると，提案手法では，従来手法に比べて，最大で 7% 程度再現率の低下が見られる．しかしながら，全体的に大きな再現率低下は見られず，逆に再現率が向上しているものもある．

表 5 観測データが 1 種類の場合の，従来手法と提案手法のそれぞれにおける，式 (45) の再現率 $r_t$ で示した推定結果の一覧．「人口」はゾーン人口の再現率を，「発着人数」はゾーン別発着別公共交通トリップ人数の再現率を示す．単位は [%] である．

| | 手法 | 従来手法（パーティクルフィルタ） | | | | 提案手法（パーティクルスムーザ） | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 使用観測データ | モバイル空間統計 | | IC カードデータ | | モバイル空間統計 | | IC カードデータ | |
| $N$ | 検証時刻 $t$ | 人口 | 発着人数 | 人口 | 発着人数 | 人口 | 発着人数 | 人口 | 発着人数 |
| 100 | 午前 9 時 | 81.6 | 58.1 | 81.1 | 59.5 | 81.4 | 58.4 | 81.7 | 53.9 |
| | 正午 | 80.0 | 44.5 | 80.0 | 49.2 | 80.1 | 41.9 | 79.4 | 41.8 |
| | 午後 5 時 | 82.2 | 41.1 | 82.7 | 42.5 | 80.4 | 44.9 | 81.9 | 47.1 |
| | 午後 9 時 | 82.4 | 50.5 | 82.1 | 54.3 | 82.4 | 50.5 | 82.1 | 54.3 |
| | 平均再現率 | 81.6 | 48.6 | 81.5 | 51.4 | 81.1 | 48.9 | 81.3 | 49.3 |
| | | 65.1 | | 66.4 | | 65.0 | | 65.3 | |
| 1000 | 午前 9 時 | 82.3 | 57.1 | 81.9 | 53.2 | 81.8 | 53.5 | 81.1 | 52.9 |
| | 正午 | 79.8 | 44.4 | 80.1 | 48.4 | 80.3 | 47.4 | 78.9 | 47.0 |
| | 午後 5 時 | 81.2 | 43.6 | 81.5 | 41.5 | 81.4 | 44.0 | 81.2 | 46.7 |
| | 午後 9 時 | 81.1 | 55.1 | 81.0 | 58.1 | 81.1 | 55.1 | 81.0 | 58.1 |
| | 平均再現率 | 81.1 | 50.1 | 81.1 | 50.3 | 81.2 | 50.0 | 80.6 | 51.2 |
| | | 65.6 | | 65.7 | | 65.6 | | 65.9 | |

表 6 従来手法における，各時刻での推定結果とプリズム制約の矛盾．時点 $t_2$ での推定結果における各個人の時刻 $t_1$ での位置が，時点 $t_1$ での推定結果の時空間プリズム制約を満たしているものと，満たしていないものに分類し，それぞれの人数を集計した．

| | 使用観測データ | | モバイル空間統計 | | | IC カードデータ | | |
|---|---|---|---|---|---|---|---|---|
| $N$ | $t_1$ | $t_2$ | 制約内 | 制約外 | 矛盾割合 | 制約内 | 制約外 | 矛盾割合 |
| 100 | 午前 9 時 | 正午 | 4656 人 | 269 人 | 5.46% | 3813 人 | 111 人 | 2.83% |
| | 午前 9 時 | 午後 5 時 | 4707 人 | 218 人 | 4.43% | 3813 人 | 111 人 | 2.83% |
| | 午前 9 時 | 午後 9 時 | 4714 人 | 211 人 | 4.28% | 3826 人 | 98 人 | 2.50% |
| | 正午 | 午後 5 時 | 950 人 | 71 人 | 6.95% | 1498 人 | 0 人 | 0.00% |
| | 正午 | 午後 9 時 | 987 人 | 34 人 | 3.33% | 1498 人 | 0 人 | 0.00% |
| | 午後 5 時 | 午後 9 時 | 4470 人 | 825 人 | 15.58% | 3861 人 | 1136 人 | 22.73% |
| 1000 | 午前 9 時 | 正午 | 4589 人 | 146 人 | 3.08% | 4624 人 | 133 人 | 2.80% |
| | 午前 9 時 | 午後 5 時 | 4484 人 | 251 人 | 5.30% | 4606 人 | 151 人 | 3.17% |
| | 午前 9 時 | 午後 9 時 | 4564 人 | 171 人 | 3.61% | 4570 人 | 187 人 | 3.93% |
| | 正午 | 午後 5 時 | 1238 人 | 37 人 | 2.90% | 1102 人 | 37 人 | 3.25% |
| | 正午 | 午後 9 時 | 1238 人 | 37 人 | 2.90% | 1139 人 | 0 人 | 0.00% |
| | 午後 5 時 | 午後 9 時 | 3911 人 | 901 人 | 18.72% | 4125 人 | 659 人 | 13.78% |

表 7 観測データが 2 種類の場合の，式 (43) の $\alpha$ の値別の，式 (45) の再現率 $r_t$ で示した推定結果の一覧．「人口」はゾーン人口の再現率を，「発着人数」はゾーン別発着別公共交通トリップ人数の再現率を示す．単位は [%] である．

| | | $\lambda_t^{(i)} = (1-\alpha)\lambda_t^{\mathrm{mobile}(i)} + \alpha\lambda_t^{\mathrm{IC}(i)}$ | | | | | |
|---|---|---|---|---|---|---|---|
| | | $\alpha = 0.8$ | | $\alpha = 0.5$ | | $\alpha = 0.2$ | |
| $N$ | 検証時刻 $t$ | 人口 | 発着人数 | 人口 | 発着人数 | 人口 | 発着人数 |
| 100 | 午前 9 時 | 81.6 | 56.3 | 81.9 | 58.8 | 83.0 | 55.2 |
| | 正午 | 80.1 | 47.0 | 80.2 | 50.2 | 80.5 | 46.9 |
| | 午後 5 時 | 81.6 | 40.8 | 81.8 | 44.9 | 82.1 | 46.0 |
| | 午後 9 時 | 82.2 | 56.7 | 80.9 | 54.7 | 81.4 | 53.6 |
| | 平均再現率 | 81.4 | 50.2 | 81.2 | 52.2 | 81.8 | 50.4 |
| | | 65.8 | | 66.7 | | 66.1 | |
| 1000 | 午前 9 時 | 82.8 | 52.1 | 81.1 | 57.5 | 81.3 | 54.0 |
| | 正午 | 80.0 | 44.9 | 80.1 | 46.2 | 80.2 | 40.7 |
| | 午後 5 時 | 81.8 | 39.4 | 81.7 | 42.5 | 82.2 | 46.6 |
| | 午後 9 時 | 81.5 | 53.0 | 81.8 | 54.6 | 81.2 | 55.9 |
| | 平均再現率 | 81.5 | 47.4 | 81.2 | 50.2 | 81.2 | 49.3 |
| | | 64.4 | | 65.7 | | 65.3 | |
| 5000 | 午前 9 時 | 81.6 | 57.2 | 82.6 | 55.9 | 81.9 | 52.4 |
| | 正午 | 79.9 | 41.5 | 80.0 | 48.6 | 80.7 | 42.0 |
| | 午後 5 時 | 82.2 | 46.2 | 81.6 | 44.0 | 81.6 | 43.6 |
| | 午後 9 時 | 81.5 | 54.0 | 80.8 | 52.3 | 81.0 | 55.3 |
| | 平均再現率 | 81.3 | 49.7 | 81.3 | 50.2 | 81.3 | 48.3 |
| | | 65.5 | | 65.7 | | 64.8 | |

なお，表 5 では時刻ごとに推定結果が示されているが，それぞれ相互の時空間プリズム制約との整合性は保証されておらず，この推定結果同士の遷移が可能とは限らない．表 6 はこのような矛盾が生じる割合を示したもので，時点 $t_2$ での推定結果における各個人の時刻 $t_1$ での位置が，時点 $t_1$ での推定結果の時空間プリズム制約を満たしているか否かを集計したものである．従来手法ではこのような矛盾が起こり得る一方で，提案手法では，表 5 の推定結果同士の遷移が，時空間プリズム制約の中で可能である．

**複数種類の観測データがある場合**

次に，観測データが 2 種類の場合における，式 (43) の $\alpha$ の値別の比較結果を表 7 に示す．

表 5 と表 7 を比較すると，複数種類の観測データを使用した場合，1 種類のみを使用する場合に比べて全体的に再現率が向上する傾向にあることがわかる．表 7 に着目して，$\alpha$ の値について見ると，今回の結果では，$N$ の値によらず $\alpha = 0.5$ が最も平均再現率が良いことがわかる．また，必ずしも $\alpha$ の値を上げれば（= より IC カードデータの観測を重視すれば）ゾーン発着トリップ数の再現率が高くなるという訳ではない．粒子数の増減による再現率の高低にはあまり関係性が見られないこと，同一条件内での再現率の高低も粒子数の変化にあまり影響を受けていないことは，表 5 とほぼ同様である．また，ゾーン人口の再現率は，全ての場合で正午

が最も低い．また，ゾーン別発着別トリップ人数の再現率は，$N = 100$ の提案手法，$N = 1000, 5000, \alpha = 0.2$ の提案手法，$N = 5000, \alpha = 0.8$ の提案手法では正午が，その他の場合では午後 5 時が最も低い．

以上の結果のうち，粒子数 $N = 5000$，観測データの重み $\alpha = 0.5$ の条件における，午前 9 時での同化について，推定値と真値の関係をプロットしたものを図 9，図 10 に示す．

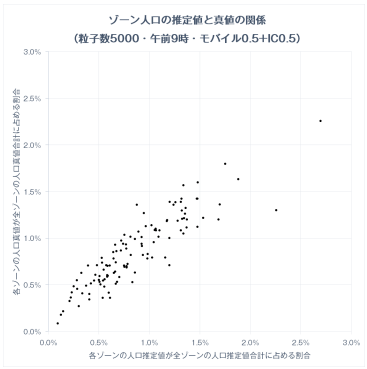


図 9　$N = 5000, \alpha = 0.5$ の設定条件での，午前 9 時におけるゾーン人口の推定値と真値の関係．プロット数 $K$ はゾーン数である 115 であり，各プロットは，そのゾーンにおける，ゾーン人口推定値が全人口推定値に占める割合と，ゾーン人口真値が全人口真値に占める割合の関係を表している．

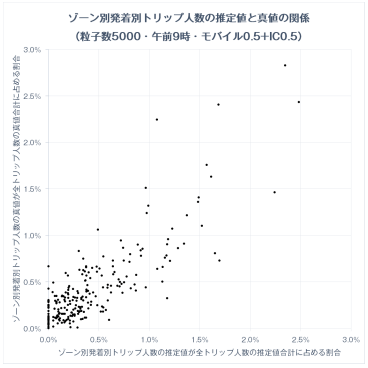


図 10　$N = 5000, \alpha = 0.5$ の設定条件での，午前 9 時におけるゾーン別発着別トリップ人数の推定値と真値の関係．プロット数 $K$ はゾーン数の 2 倍である 230 であり，各プロットは，そのゾーン・発着別における，トリップ人数推定値が全トリップ人数推定値に占める割合と，トリップ人数真値が全トリップ人数真値に占める割合の関係を表している．

## 5.3　考察

まず，モバイル空間統計データのみを使用した場合と IC カードデータのみを使用した場合で，IC カード利用の方が再現率が高いことについて考える．比較指標のうち，ゾーン人口の次元は，モバイル空間統計の観測ベクトル $\boldsymbol{y}_t^{\mathrm{mobile}}$ の次元と同じ，ゾーン数に等しい 115 である．それに対して，ゾーン別発着別トリップ人数の次元は，IC カードデータの観測ベクトル $\boldsymbol{y}_t^{\mathrm{IC_{zone}}}$ の次元と同じ，$115 \times 2 = 230$ である．一方で，観測ベクトルの中身であるゾーン人口やゾーン別発着別トリップ人数の対象地域における合計は，人口が 12 万-14 万人であるのに対し，トリップ人数は 3 万-5 万程度である．故に，ゾーン別発着別トリップ人数では，1 つの要素に含まれる値が小さくなってしまう，ときにはトリップ人数が 0 になってしまう場合が発生し，疑似 PT 調査の拡大係数を単位として真 PT 調査の数を再現することが難しくなる，というスパース性の問題がある．その結果，IC カードデータを用いない（モバイル空間統計のみを観測データとする）場合には，特にゾーン別発着別トリップ人数をうまく推定することが難しくなり，再現率を下げてしまうものと思われる．

次に，提案手法では従来手法に比べて再現率が低下していることについて考察する．従来手法は，各時刻 $t$ での同化においては，前後の時刻との時空間プリズム制約を考えることなく，観測データの統合後の事後分布で最も確率の高いものを推定結果としている．これは提案手法に比べると，言わば合理性を犠牲にして無理に観測値に近いものを選んでいるようなものである．このため，提案手法よりも再現率が高くなるのはある意味当然の結果と言える．一方で提案手法では，時空間プリズム制約に反することが無いながらも，従来手法からの再現率低下も多くの場合で 1% 程度，最大でも 5% 程度に留めている．このため，提案手法は従来手法よりも合理性が高いと言える．

各条件の平均再現率を見ると，複数種類の観測データを用いることは，かなり有効な手法であることがわかった．特に，$N = 100, \alpha = 0.5$ の平均再現率 66.7% は提案手法の 13 条件の中だけでなく，プリズム制約を考えない従来手法を含めても最良値である．しかし一方で，$\alpha$ の選び方については，$\alpha = 0.5$ が最も再現率が高いという結果こそ得られたが，$\alpha = 0.2$ と $\alpha = 0.8$ のときの再現率の高低が粒子数 $N$ によって異なるなど，現時点では十分な示唆が得られない．このため，データごとに試行錯誤してチューニングする必要がある．

時刻による再現率の違いについても述べる．まず，ゾーン人口の再現率は，全ての場合で正午が最も低いことについてである．これは，昼休みに移動する人が多いからだと考えられる．PCATS では基本的に通勤・通学を固定活動として想定しており，固定活動の予定がある個人に対しては，不確実性を伴わずに必ず「固定活動中」の結果を返す．一方で実際の人の動きでは，昼休みには，固定活動時間帯中であっても固定活動地から離れることが比較的多いと思われる．また，ゾーン別発着別トリップ人数の再現率は殆どの場合で午後 5 時が最も低い．この理由は様々に考えられるが，一つには，サンプル数が特に少ないことだと考えられる．この時刻におけるゾーン発着トリップ数の再現率は，正午から午後 5 時の間に目的地に到着したトリップの数が基準になっている．この時間帯は他の時間帯に比べて，固定活動中の個人が多く，発生するトリップの数が特に少ないと考えられる．このため，上にも述べたようなスパース性の問題，即ち，1 つの要素に含まれる値が小さくなってしまい，疑似 PT 調査の拡大係数を単位として真 PT 調査の数を再現することが難しくなる現象が生じているのではないだろうか．

以上で度々述べたスパース性の問題は，図 10 からも読み取れる．横軸の値が 0 であるプロットが散見される他，横軸の値が小さいプロットでは，値が飛び飛びになっている，即ち，そのゾーン・発着別に該当する推定結果が 1 人ないし数人しかおらず，その拡大係数倍を以て推定値が与えられていることが示されている．

# 6 結論

本章では，本研究から得られた成果と今後の課題を述べる．

## 6.1 成果

本研究では，アクティビティモデルと異種交通データからなる状態空間モデルにおいて，パーティクルスムーザを用いた平滑化により，交通行動を推定する手法を提案した．また，提案した手法を，東京都市圏 PT 調査を全数の真値と見做す仮想都市圏に対して適用した．適用結果について，各時刻におけるゾーン人口と，ゾーン別発着別トリップ人数を指標として，再現率を算出し，手法の有効性の検証を行った．具体的な成果は次の通りである．

### アクティビティモデルと異種交通データからなる状態空間モデルにおける平滑化手法の構築

状態空間モデルの枠組みで，アクティビティモデルと，数種類の異なる交通データを統合し，状態ベクトルの系列を平滑化した．本研究では，アクティビティモデルの PCATS をシステムモデル，全対象個人の，活動開始・終了時刻，活動場所，活動後の移動開始・終了時刻などで構成される生活行動スケジュールを状態ベクトルとして表現し，観測データを，パーティクルスムーザを用いて統合した．統合に際しては，アクティビティモデルが持つ特徴である，時空間プリズム制約に反しない手法を理論的に構築した．その結果，従来手法では平均で 5% 程度，最大で 10% 以上の個人が時空間プリズム制約外に配置されるような推定が行われるのに比べて，より合理的な推定が可能となった．また，複数種類の観測データを使用することも容易になった．

### アクティビティモデル PCATS の東京 23 区への適用

PCATS は元来，主に京阪神地域を対象として開発された．また，入力ファイルとして，固定活動スケジュールの他に，地域属性データや交通ネットワーク属性データを必要とする．そのため，新しい地域への適用にあたっては，各種データの整備が必要となる．本研究では，東京 23 区を対象として，計画基本ゾーンレベルで，ゾーンの面積・人口・サービス事業所数・人口密度・サービス事業所密度からなる地域属性データと，ゾーン相互間の公共交通所要時間・公共交通運賃・公共交通平均乗換回数・自動車所要時間・徒歩所要時間・道のりからなる交通ネットワーク属性データを整備し，PCATS の東京 23 区への適用を可能とした．

### アクティビティモデルを用いたデータ同化における複数種類の観測データ使用の有効性の確認

アクティビティモデルに対する交通データの同化に際し，複数種類の観測データを使用することの有効性を確認した．本研究では，東京 23 区を対象とし，実際に行われた PT 調査を全数の真値と見做す疑似的な都市圏に対し，まず，真 PT 調査から，疑似 PT 調査データ・モバイル空間統計を想定した 1 時間ごとのゾーン人口データ・疑似 IC カードデータを作成した．次に，疑似 PT 調査データから作成した固定活動スケジュールを PCATS の入力とし，観測データとしてゾーン人口データと，疑似 IC カードデータによるゾーン別発着別トリップ人数の 2 種類を用いて，提案手法を適用した．この適用において，複数種類の観測データを統合すると，1 種類のみを同化したときに比べて，ゾーン人口の再現率が向上することを確認した．

## 6.2　今後の課題

本研究では前述のような成果を挙げた一方で，今後の応用に向けては以下のような課題がある．

### システムモデルの改善

システムモデルとして用いた PCATS には，内部モデルとして「想定時間分布モデル」「活動内容選択モデル」「交通機関・目的地選択モデル」が含まれている．これらのモデルのパラメータは，藤井ら（1997） [16] によって開発された当時の，1995 年の調査データによって推定されたものが使われている．それから 20 年以上が経過しており，人々のライフスタイルは大きく変化していると思われるため，これらのモデルについても再考が必要である．

また，疑似 PT 調査における拡大係数の設定にも検討の余地が残る．今回は，居住市区町村別・性別・年齢階層別に分類して拡大係数を計算した．カテゴリを増やし過ぎると，「真 PT 調査には個人が含まれているが，疑似 PT 調査には誰もいない」というカテゴリが多発してしまうためである．しかし，東京都市圏における生活行動を考慮したとき，自動車保有の有無は交通行動に大きく影響すると考えられるため，これを拡大係数のカテゴリに入れることも検討すべきだろう．

### データのスパース性への対応

今回は，疑似 PT における個人数の少なさもあり，何らかのカテゴリ分け操作を行った際に，各カテゴリごとの人数が希薄になってしまうことがしばしば見られた．この結果，例えば，拡大係数が推定結果に大きく影響を及ぼす，観測ベクトルの要素の殆どが 0 になる，などの無理が生じた箇所がある．このスパース性が原因で，予測分布と IC カードによる観測データの類似度の評価に OD 別トリップ人数を利用することや，IC カードデータの同化間隔を狭めることなどを断念した．このため，今回の対応方法である「観測モデルにおける集計基準を，OD 別トリップ人数からゾーン別発着別トリップ人数に変更し，カテゴリの数を減らして中身の人数を多くする」等の対応や，十分な対象個人数が得られるようなデータを利用することが必要となる．

### 観測データの拡張

本研究では，モバイル空間統計データはゾーン人口と直接比較して，IC カードデータはゾーン別発着別トリップ人数と直接比較してフィルタリングを行った．しかし，実際のモバイル空間統計データには居住地データが含まれており，IC カードデータにも，記名式であれば性別・年代が，定期券であれば更に居住地・固定活動地のデータも含まれている．本研究では個人数の少なさから上述のスパース性の問題が大きかったため，これらの属性データ別のフィルタリングを行うことはできなかった．しかし，実都市圏への適用が可能になった際には個人数も増え，スパース性についてはある程度解消される見込みである．その際には，これらの個人属性データも利用して，属性別にフィルタリングを行うことで，精度の改善が見込まれる．

### 適用対象地域の検討

例えば，今回の対象地域である東京 23 区程度の区域では，内外・外内交通や通過交通も多数存在するであろう．それに対し，本研究で得られるのは内内交通における推定値のみである．このため，本研究の手法を実際の都市部に利用しようとすると，実際の交通量や滞留人口の推定にはまだ遠いことが予想される．このため，適用対象となる地域を拡大することで，相対的に内内交通の割合を上げることが考えられる．但し，地域

を広げるとその分ゾーン数も増えるため，例えば，属性別データのスパース性がより高くなる，観測データが対象地域の一部の分しか得られないような場合がある，といった問題が生じ得ることが予想される．このため，部分観測データが得られた場合の同化手法の構築や，ネットワーク縮約を通じて対象 OD 組合せをより低次元化した上での同化を試みることなどが課題となる．

また，本研究ではゾーン単位として計画基本ゾーンを用いたが，これを小ゾーンに分割する，あるいは中ゾーンに統合するといった状況でも推定を行い，集計単位における結果の違いも検討すべきであろう．

# 参考文献


[1] Thomas Adler and Moshe Ben-Akiva. A theoretical and empirical model of trip chaining behavior. *Transportation Research Part B: Methodological*, Vol. 13, No. 3, pp. 243–257, 1979.

[2] Neil J. Gordon, David J. Salmond, and Adrian F. M. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In *IEE Proceedings-F (Radar and Signal Processing)*, Vol. 140, pp. 107–113. IET, 1993.

[3] Torsten Hägerstrand. What about people in regional science? *Papers in regional science*, Vol. 24, No. 1, pp. 7–24, 1970.

[4] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, Vol. D-82, pp. 35–45, 1960.

[5] Genshiro Kitagawa. A Monte Carlo filtering and smoothing method for non-Gaussian nonlinear state space models. Research memorandum, The Institute of Statistical Mathematics, 1993.

[6] Genshiro Kitagawa. Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. *Journal of computational and graphical statistics*, Vol. 5, No. 1, pp. 1–25, 1996.

[7] R. Kitamura, R. M. Pendyala, and E. I. Pas. Application of AMOS, an activity-based TCM evaluation tool to the Washington, D.C. In *Proceedings of Seminar E held at the PTRC European Transport Forum*, Vol. 392, pp. 177–190, 1995.

[8] Gerald M. McCarthy. Multiple-regression analysis of household trip generation-a critique. *Highway Research Record*, No. 297, pp. 31–43, 1969.

[9] J. P. Robinson, R. Kitamura, and T. F. Golob. Daily travel in the Netherlands and California: A time diary perspective. *Hague Consulting Group*, 1992.

[10] Akihito Sudo, Takehiro Kashiyama, Takahiro Yabe, Hiroshi Kanasugi, Xuan Song, Tomoyuki Higuchi, Shin'ya Nakano, Masaya Saito, and Yoshihide Sekimoto. Particle filter for real-time human mobility prediction following unprecedented disaster. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, p. 5. ACM, 2016.

[11] Janusz Supernak. Temporal utility profiles of activities and travel: uncertainty and decision making. *Transportation Research Part B: Methodological*, Vol. 26, No. 1, pp. 60–76, 1992.

[12] 淺原彰規, 林秀樹. 粒子フィルタを用いた都市域人流推定方式. 情報処理学会論文誌, Vol. 57, No. 1, pp. 54–65, 2016.

[13] 西井和夫, 近藤勝直. 鉄道利用通勤者の時空間プリズムに着目した交通パターン分析. 土木計画学研究・論文集, Vol. 7, pp. 139–146, 1989.

[14] 矢野浩一. 粒子フィルタの基礎と応用: フィルタ・平滑化・パラメータ推定. 日本統計学会誌, Vol. 44, No. 1, pp. 189–216, 2014.

[15] 藤井聡. 改めて 「交通行動調査」 を考え直す–良質な交通計画のために, 豊かな交通行動調査を （特集 交通行動調査）. 交通工学, Vol. 46, No. 2, pp. 4–7, 2011.

[16] 藤井聡, 大塚祐一郎, 北村隆一, 門間俊幸. 時間的空間的制約を考慮した生活行動軌跡を再現するための行動シミュレーションの構築. 土木計画学研究・論文集, Vol. 14, pp. 643–652, 1997.

[17] 清家剛, 三牧浩也, 原裕介, 森田祥子. 基礎自治体におけるモバイル空間統計の活用可能性に関する研究.

日本建築学会技術報告集, Vol. 19, No. 42, pp. 737–742, 2013.

[18] 東京都市圏交通計画協議会. PT データ利用の手引き. 東京都市圏パーソントリップ調査, 2012.

[19] 中村敏和, 関本義秀, 薄井智貴, 柴崎亮介. パーティクルフィルターを用いた都市圏レベルの人の流れの推定手法の構築. 土木学会論文集 D3 (土木計画学), Vol. 69, No. 3, pp. 227–236, 2013.

[20] 樋口知之, 上野玄太, 中野慎也, 中村和幸, 吉田亮. データ同化入門 ―次世代のシミュレーション技術―. 朝倉書店, 2011.

[21] 北村隆一. 交通需要予測の課題: 次世代手法の構築にむけて. 土木学会論文集, Vol. IV-30, No. 530, pp. 17–30, 1996.

[22] 金森亮, 水野敬太, 野田五木樹, 中島秀之. 滞留人口データを利用した交通手段別 OD 交通量推計手法の提案. 研究報告知能システム (ICS), Vol. 2015, No. 8, pp. 1–6, 2015.

[23] 原田遼. 詳細な交通行動推定のためのアクティビティシミュレーションと観測データの統合に関する研究. 修士論文, 東京大学大学院工学系研究科社会基盤学専攻, 2017.

[24] 若生凌. 時空間メッシュ集計データを用いたデータ同化手法による人流推定. 修士論文, 東京大学大学院工学系研究科社会基盤学専攻, 2015.

# ソースコード

ここでは，本研究の提案手法を実装する際に用いたプログラムのソースコードを示す．

## filtering.cpp

PCATS の中断時データと，その時点での観測データを入力とし，パーティクルスムーザによるフィルタリング後の PCATS 中断時データを出力する計算は，以下のように C++ で実装した．

```
/*
以下のファイルを入力とし，
・PCATS の実行結果（途中時刻での中断結果）のファイル
（個人ID の末尾にパーティクル番号が付加され，パーティクル数分複製されたもの）
・ゾーン別人口の観測データのファイル
・IC カードトリップ情報の観測データのファイル

以下を実行してファイルに出力するプログラム
・PCATS の中断時に関わる活動のみを抜き出して固定活動とそれ以外に分類
・固定活動についてはPCATS 中断時の 各ゾーンの人口(拡大係数の総和)を算出
・固定活動以外についてはPCATS 中断時の 各パーティクルの各ゾーンの人口を算出
・パーティクルの重みを計算
・各パーティクルの復元抽出個数を決定，リサンプリング
・リサンプリングされたパーティクルに基づき，PCATS 中断時の データを修正し，ファイルを出力
*/


#define _CRT_SECURE_NO_WARNINGS // セキュリティ関連のエラーでプログラムが止まらないように

#include "␣stdafx.h␣"
#include <vector>
#include <math.h>
#include <iostream>
#include <stdio.h>
#include <cstdlib>
#include <stdlib.h>
#include <fstream>
#include <sstream>
#include <iostream>
#include <algorithm>
#include <string>
#include <time.h>
#include "␣Mt.h␣" // http://www.sat.t.u−tokyo.ac.jp/˜omi/random_variables_generation.html にあるメルセンヌ・ツイスタを用いた乱数生成のため
using namespace std;

// 関数のプロトタイプ宣言
int maximum(int *all, int number); // 各対象ゾーンに応じた値の配列中で，最大値を持つゾーンのゾーンコードを返す関数
int MAP(double *par, int number); // double 配列中で最大値を持つものが何番目の要素であるかを返す関数
double distance(int *x, int *y); // 2つの整数配列のユークリッド距離の平方を返す関数
void file_set_up(); // 入出力ファイルの設定をする関数
void extract_suspended_act(); // PCATS 中断時の データを全パーティクル分抜き出す関数
void classify_fixed_or_not(); // 抜き出したデータを，固定活動中とそれ以外に分類する関数
void extract_PCATSout_at_this_particle_non_fixed(int p); // PCATS 中断時の 固定活動以外のデー
```

```
   タから，引数とするパーティクルNo.におけるものだけを抜き出す関数
44 void count_zone_population_at_this_particle_non_fixed(int p); // 引数とするパーティクルNo.にお
   ける，固定活動以外のゾーン別人口を数える関数
45 void count_zone_population_fixed(); // 固定活動のゾーン別人口を数える関数
46 void count_trip(); // IC カードデータとの比較のために，各パーティクルにおける
   OD 別トリップ数を数える関数
47 void calculate_weight_of_mobile(vector<double> &weight_of_mobile, vector<double> &
   normalized_weight_of_mobile); // 引数とするベクトル(要素数はパーティクル数)の各要素に対し，モバ
   イル空間統計による各パーティクルの正規化重みの値を代入する関数
48 void calculate_weight_of_ic(vector<double> &weight_of_ic, vector<double> &
   normalized_weight_of_ic); // 引数とするベクトル(要素数はパーティクル数)の各要素に対し，
   IC カードデータによる各パーティクルの正規化重みの値を代入する関数
49 void calculate_weight_of_particle(); // パーティクルの重みを計算する関数
50 void resample_particle(); // 各パーティクルの復元抽出個数を決定し，リサンプリングする関数
51 void output_resampled_file_for_PCATS(); // リサンプリングされたパーティクルに基づき，
   PCATS 中断時の データを修正し，PCATS の体裁のファイルを出力する関数
52
53
54 ////////////////////////
55     // ここから基本変数設定
56 ////////////////////////
57
58 // 各種定数の設定
59 const int START = 1020; // 入力となるPCATS の開始時刻
60 const int END = 1260; // 入力となるPCATS の終了（＝中断）時刻
61 const int IC_START = 1020; // IC カードデータのトリップ数を数える時間帯の開始時刻
62 const int IC_END = 1260; // IC カードデータのトリップ数を数える時間帯の終了時刻
63 const int MOBILE_OBS_TIME = 1260; // モバイル空間統計の観測時刻
64 const int OBS_DATA_FLAG = 3; // 1：モバイル空間統計のみ，2：IC カードデータのみ，3：モバイル空間統
   計 +IC カードデータ同時
65 const int IC_FLAG = 2; // 1：OD ごとのトリップ数の類似度で尤度算出，2：ゾーンごと発着別のトリップ数
   の類似度で尤度算出
66 const double WEIGHT_OF_OBS_DATA_IC = 0.2; // 2つの観測データの重みづけ
67 const int ASSIMILATE_SWITCH = 1; // 0：ゾーン人口やトリップ数の計算のみ行う，1：観測データの同化ま
   で行う
68
69 const int N_OF_PARTICLE = 5000; // パーティクル数
70 const int N_OF_ZONE = 115; // ゾーン数
71 const int N_OF_DIGIT_OF_INDV_ID = 11; // PCATS でのパーティクル複製時，出力される個人
   ID は「この桁数+パーティクル番号」になるため，文字列比較時に必要
72 const int N_OF_DIGIT_OF_PTCL_NO = 4; // パーティクル番号の桁数，ここを変更したときは 928, 1977,
   2090, 2115行目(出力桁数)も書き換える必要がある
73 const int N_OF_PCATS_PARAMETER = 23; // PCATS の（個人 ID と中断時の活動を除いた）パラメータの数
74 const int MAX_N_OF_ACT_OF_THE_DAY = 100; // PCATS で，1個人 1パーティクルにおける，活動の数の最大
   値
75
76 // 不変だが，値を後から入れるので宣言だけして初期化しない変数
77 int N_OF_ROW_OF_PCATS_OUTPUT; // 入力ファイルとなるPCATS 中断時の ファイルの行数
78 int N_OF_PERSON_nFX; // 固定活動以外の人数
79 int N_OF_PERSON_FX; // 固定活動の人数
80 int MODE_PTCL_NO; // 尤度最大のパーティクルの(1から数えた)旧パーティクルでの番号
81
82 // ログファイルでの時刻表示用設定
83 time_t timer;
84 struct tm *t_st;
85 char *wday[] = { "␣日␣" , "␣月␣" , "␣火␣" , "␣水␣" , "␣木␣" , "␣金␣" , "␣土␣" };
86
```

```
87 // ここから入出力ファイル用変数の設定(
     file_set_up 内で宣言するとローカル変数になり他の関数内で使えなくなるので，ここで設定)
88
89 // 作成(追記)：LOG_FILE：このプログラムの実行結果を保存するためのファイル
90 FILE *LOG_FILE;
91
92 // 読込：ORIGINAL_PCATS_OUTPUT：PCATS によって出力された，個人ごと時間帯ごとの活動内容が書
     かれたファイル
93 FILE *ORIGINAL_PCATS_OUTPUT;
94 char PCATSout[200];
95
96 // 作成・読込：PCATS_SUSPENDED：ORIGINAL_PCATS_OUTPUT から，
     PCATS 中断時刻に関わる行のみを抜き出したファイル
97 FILE *PCATS_SUSPENDED;
98 char suspended[200];
99
100 // 作成・読込：POUT_nFX：PCATS_SUSPENDED から，固定活動以外を抜き出したファイル
101 FILE *POUT_nFX;
102 char nonfixed[200]; // 1行の末尾にゾーン番号がパーティクル数分だけ連なるので，ゾーンコードの桁数とパ
     ーティクル数によっては配列の大きさが不足するので注意
103
104 // 作成・読込：POUT_FX：PCATS_SUSPENDED から，固定活動を抜き出したファイル
105 FILE *POUT_FX;
106 char fix[200]; // 1行の末尾にゾーン番号がパーティクル数分だけ連なるので，ゾーンコードの桁数とパーティ
     クル数によっては配列の大きさが不足するので注意
107
108 // 作成：POP_OF_ALLZONE_ALLPTCL_nFX：PCATS 中断時の 各ゾーンの固定活動以外の人口を，パーテ
     ィクル数分並べたファイル
109 FILE *POP_OF_ALLZONE_ALLPTCL_nFX;
110 char zoneall[200]; // 1行にゾーン人口(1万人 5桁オーダー)がゾーン数分だけ連なるので，ゾーン数によって
     は配列の大きさが不足するので注意
111
112 // 作成・読込：POUT_nFX_THIS_PTCL：PCATS 中断時刻に関わる固定活動以外のデータを，そのパーティ
     クルにおける分だけ全個人分並べたファイル
113 FILE *POUT_nFX_THIS_PTCL;
114 char nonfixed_this_ptcl[200];
115
116 // 作成・読込：POP_OF_ALLZONE_THIS_PTCL_nFX：そのパーティクルにおけるPCATS 中断時の 各ゾー
     ンの固定活動以外の人口
117 FILE *POP_OF_ALLZONE_THIS_PTCL_nFX;
118 char zoneall_this_ptcl[200]; // 1行にゾーン人口(1万人 5桁オーダー)がゾーン数分だけ連なるので，ゾー
     ン数によっては配列の大きさが不足するので注意
119
120 // 作成・読込：POP_OF_ALLZONE_FX：PCATS 中断時の 各ゾーンの固定活動の人口
121 FILE *POP_OF_ALLZONE_FX;
122 char zoneall_fix[200]; // 1行にゾーン人口(1万人 5桁オーダー)がゾーン数分だけ連なるので，ゾーン数によ
     っては配列の大きさが不足するので注意
123
124 // 読込：POP_OF_ALLZONE_OBS：観測データで，各ゾーンの人口が 1行にゾーン数分だけ並んだファイル
125 FILE *POP_OF_ALLZONE_OBS;
126 char zoneall_obs[200]; // 1行にゾーン人口(1万人 5桁オーダー)がゾーン数分だけ連なるので，ゾーン数によ
     っては配列の大きさが不足するので注意
127
128 // 読込：ZONECODE：ゾーン番号のOD 組合せが ORDER BY o_zone, d_zone で並んだファイル
129 FILE *ZONECODE;
130 char zonecode[200];
131
```

```
132 // 読込：IC_OBS：観測データで，個人ID，拡大係数，発時刻，着時刻，発ゾーン，着ゾーン，交通手段が，トリップ数分の行数だけ並んだファイル
133 FILE *IC_OBS;
134 char ic_obs[200];
135
136 // 作成・読込：N_OF_TRIP_BY_OD_THIS_PTCL：各OD 組合せごとに，トリップ数のPCATS での推定値と観測値を OD 組合せ数分の行数だけ並べたファイル
137 FILE *N_OF_TRIP_BY_OD_THIS_PTCL;
138 char n_of_trip_by_od[200];
139
140 // 作成・読込：N_OF_TRIP_BY_ZONE_THIS_PTCL：各ゾーン発着別に，トリップ数のPCATS での推定値と観測値をゾーン数の 2 倍の行数だけ並べたファイル
141 FILE *N_OF_TRIP_BY_ZONE_THIS_PTCL;
142 char n_of_trip_by_zone[200];
143
144 // 作成：WEIGHT_OF_MOBILE：モバイル空間統計に関する各パーティクルの重みと正規化重みを，パーティクル数分の行数だけ並べたファイル
145 FILE *WEIGHT_OF_MOBILE;
146 char weight_mobile[200]; // 配列の大きさは，重みを小数点以下何桁まで表示するかに依存
147
148 // 作成：WEIGHT_OF_IC：IC カードデータに関する各パーティクルの重みと正規化重みを，パーティクル数分の行数だけ並べたファイル
149 FILE *WEIGHT_OF_IC;
150 char weight_ic[200]; // 配列の大きさは，重みを小数点以下何桁まで表示するかに依存
151
152 // 作成・読込：WEIGHT_OF_PTCLS：各パーティクルの重みと正規化重みを，パーティクル数分の行数だけ並べたファイル
153 FILE *WEIGHT_OF_PTCLS;
154 char weight_ptcl[200]; // 配列の大きさは，重みを小数点以下何桁まで表示するかに依存
155
156 // 作成・読込：CORRESPONDENCE_TABLE_OF_PTCLS：リサンプリング前後における，新旧パーティクルの対応表
157 FILE *CORRESPONDENCE_TABLE_OF_PTCLS;
158 char correspondence[200];
159
160 // 作成・読込：NEW_THIS_PTCL：POUT_nFX_THIS_PTCL を復元抽出してできた，新しいパーティクル
161 FILE *NEW_THIS_PTCL;
162 char new_ptcl[200];
163
164 // 作成：REVISED_PCATS_OUTPUT：リサンプリングされたパーティクルに基づいてPCATS 中断時の データを修正した，このプログラムの最終出力ファイル
165 FILE *REVISED_PCATS_OUTPUT;
166 char PCATSout_revised[200];
167
168
169
170
171 ////////////////////////
172     // ここまで基本設定
173     // ここからメイン関数_tmain の内容
174 ////////////////////////
175
176
177 int _tmain(int argc, _TCHAR* argv[])
178 {
179     time(&timer); // 現在時刻の取得
180     t_st = localtime(&timer); // 現在時刻を現地時刻の構造体に変換
```

```
181
182     // 作成(追記)：LOG_FILE：このプログラムの実行結果を保存するためのファイル
183     LOG_FILE = fopen( "␣C:\\cpp\\filtering\\filtering\\log.txt␣" , "␣a␣" );
184     if (LOG_FILE == NULL) {
185         cout << "␣ファイルLOG_FILE（このプログラムの実行結果を保存するためのファイル）の作成失敗␣"
              << endl;
186         system( "␣pause␣" );
187     }
188     else {
189         cout << "␣ファイルLOG_FILE（このプログラムの実行結果を保存するためのファイル）の作成成功␣"
              << endl;
190     }
191
192     fprintf(LOG_FILE, "␣\n\n\n*****************************\n␣" );
193     fprintf(LOG_FILE, "␣PCATS 出力へのフィルタリング開始\n␣" );
194     fprintf(LOG_FILE, "␣    開始日：␣%04d 年%02d 月%02d 日（%s)\n␣" , t_st->tm_year+1900, t_st
          ->tm_mon+1, t_st->tm_mday, wday[t_st->tm_wday]);
195     fprintf(LOG_FILE, "␣  開始時刻：␣%02d 時%02d 分%02d 秒\n␣" , t_st->tm_hour, t_st->tm_min,
          t_st->tm_sec);
196     fprintf(LOG_FILE, "␣設定条件は以下の通り\n␣" );
197     fprintf(LOG_FILE, "␣入力PCATS 開始時刻：␣%d, ␣入力PCATS 中断時刻：␣%d, ␣\n パーティクル数：␣%d, ␣
          ゾーン数：␣%d\n␣" , START, END, N_OF_PARTICLE, N_OF_ZONE);
198     if (OBS_DATA_FLAG == 1) {
199         fprintf(LOG_FILE, "␣同化観測データ：␣モバイル空間統計のみ\n モバイル空間統計観測時刻：␣%d\n\
              n␣" , MOBILE_OBS_TIME);
200     }
201     else if (OBS_DATA_FLAG == 2) {
202         fprintf(LOG_FILE, "␣同化観測データ：␣IC カードデータのみ\
              nIC カードデータ集計時間帯（着時刻基準）開始時刻：␣%d, ␣終了時刻：␣%d\n␣" , IC_START,
              IC_END);
203         if (IC_FLAG == 1) {
204             fprintf(LOG_FILE, "␣IC カードデータ同化方法：␣OD 別トリップ数の類似度で尤度算出\n\n␣"
                  );
205         }
206         else if (IC_FLAG == 2) {
207             fprintf(LOG_FILE, "␣IC カードデータ同化方法：␣ゾーン別発着別トリップ数の類似度で尤度算
                  出\n\n␣" );
208         }
209         else {
210             cout << "␣IC_FLAG の値を確認してください␣" << endl;
211             system( "␣pause␣" );
212         }
213     }
214     else if (OBS_DATA_FLAG == 3) {
215         fprintf(LOG_FILE, "␣同化観測データ：␣モバイル空間統計␣+␣IC カードデータ\
              nIC カードデータ集計時間帯（着時刻基準）開始時刻：␣%d, ␣終了時刻：␣%d\n␣" , IC_START,
              IC_END);
216         if (IC_FLAG == 1) {
217             fprintf(LOG_FILE, "␣IC カードデータ同化方法：␣OD 別トリップ数の類似度で尤度算出\n␣" );
218         }
219         else if (IC_FLAG == 2) {
220             fprintf(LOG_FILE, "␣IC カードデータ同化方法：␣ゾーン別発着別トリップ数の類似度で尤度算
                  出\n␣" );
221         }
222         else {
223             cout << "␣IC_FLAG の値を確認してください␣" << endl;
224             system( "␣pause␣" );
```

```
225         }
226         fprintf(LOG_FILE, "␣観測データの重みづけ：␣：␣モバイル空間統計␣%.2lf：␣IC カードデータ␣%.2
              lf\n\n␣" , 1.0 - WEIGHT_OF_OBS_DATA_IC, WEIGHT_OF_OBS_DATA_IC);
227     }
228     else {
229         cout << "␣OBS_DATA_FLAG の値を確認してください␣" << endl;
230         system( "␣pause␣" );
231     }
232
233     cout << "␣入出力ファイルの設定開始␣" << endl;
234
235     file_set_up(); // 関数file_set_up の実行
236
237     cout << "␣入出力ファイルの設定完了，␣PCATS 中断時データ抜き出し開始␣" << endl;
238
239     extract_suspended_act(); // 関数extract_suspended_act の実行
240
241     fprintf(LOG_FILE, "␣入力データサイズは以下の通り\n␣" );
242     fprintf(LOG_FILE, "␣入力したPCATS のデータ行数：␣%d 行\n␣" , N_OF_ROW_OF_PCATS_OUTPUT); //
          ログにデータサイズを記載
243
244     cout << "␣PCATS 中断時データ抜き出し完了，␣固定活動か否かの分類開始␣" << endl;
245
246     classify_fixed_or_not(); // 関数classify_fixed_or_not の実行
247
248     cout << "␣固定活動か否かの分類完了，␣固定活動以外について，␣PCATS 中断時の␣パーティクルごとの各ゾ
          ーン人口集計開始␣" << endl;
249
250     if (END != 1620) {
251         for (int p = 0; p < N_OF_PARTICLE; p++) { // パーティクルの数だけ以下を繰り返し
252
253             // ファイルの作成・読込：POUT_nFX_THIS_PTCL：
                  PCATS 中断時刻に関わる固定活動以外のデータを，そのパーティクルにおける分だけ全個人分並
                  べたファイル
254             // パーティクルNo.に関するループ内なので，
                  file_set_up 内では上手く書けなかったのでこちらに記述
255             // 上手く書く方法があれば，file_set_up 内に書いた方が，入出力ファイルのパスやファイル名を変
                  更する際に見易いのでそうしたいが……
256             sprintf(
257                 nonfixed_this_ptcl,
258                  "␣C:\\cpp\\filtering\\filtering\\output\\PCATSoutputparticle\\%d-%d\\
                      particle_non_fixed_%d-%d(particle_No.%d).data␣" , START, END, START, END,
                       p + 1
259                 );
260             POUT_nFX_THIS_PTCL = fopen(nonfixed_this_ptcl, "␣w+␣" );
261             if (POUT_nFX_THIS_PTCL == NULL) {
262                 cout << "␣パーティクル番号␣" << p + 1 <<
263                      "␣におけるファイル
                          POUT_nFX_THIS_PTCL（PCATS 中断時刻に関わる固定活動以外のデータを，␣そのパーテ
                          ィクルにおける分だけ全個人分並べたファイル）の作成失敗␣" << endl;
264                 system( "␣pause␣" );
265             }
266
267             // ファイルの作成：POP_OF_ALLZONE_THIS_PTCL_nFX：そのパーティクルにおける
                  PCATS 中断時の 各ゾーンの固定活動以外の人口
268             // パーティクルNo.に関するループ内なので，
                  file_set_up 内では上手く書けなかったのでこちらに記述
```

```
269             // 上手く書く方法があれば，file_set_up 内に書いた方が，入出力ファイルのパスやファイル名を変更する際に見易いのでそうしたいが……
270             sprintf(
271                 zoneall_this_ptcl,
272                  "␣C:\\cpp\\filtering\\filtering\\output\\population_of_zone\\%d-%d\\population_of_allzone_non_fixed_%d-%d(particle_No.%d).csv␣" , START, END, START, END, p + 1
273                 );
274             POP_OF_ALLZONE_THIS_PTCL_nFX = fopen(zoneall_this_ptcl, "␣w␣" );
275             if (POP_OF_ALLZONE_THIS_PTCL_nFX == NULL) {
276                 cout << "␣パーティクル番号␣" << p + 1 <<
277                      "␣におけるファイル
                          POP_OF_ALLZONE_THIS_PTCL_nFX（そのパーティクルにおける PCATS 中断時の␣各ゾーンの固定活動以外の人口）の作成失敗␣" << endl;
278                 system( "␣pause␣" );
279             }
280
281             extract_PCATSout_at_this_particle_non_fixed(p); // そのパーティクルNo.における関数 extract_PCATSout_at_this_particle_non_fixed の実行
282
283             count_zone_population_at_this_particle_non_fixed(p); // そのパーティクルNo.における関数count_zone_population_at_this_particle_non_fixed の実行
284
285             if ((int((p + 1) * 10000.0 / N_OF_PARTICLE)) % 100 == 0) { // 剰余演算を使って 1%ごとに以下を表示
286                 cout << ((p + 1) * 1.0 / N_OF_PARTICLE) * 100 << "␣%完了␣" << endl;
287             }
288         }
289     }
290
291     fclose(POUT_nFX);
292     fseek(POP_OF_ALLZONE_ALLPTCL_nFX, 0, SEEK_SET); // POP_OF_ALLZONE_ALLPTCL_nFX は読書両方のモード w+なので，書き込みが終わった今，次の読込に備えてポインタを先頭にしておく
293
294     cout << "␣固定活動以外について，␣PCATS 中断時の␣パーティクルごとの各ゾーン人口集計完了，␣固定活動について各ゾーン人口集計開始␣" << endl;
295
296     count_zone_population_fixed(); // 関数count_zone_population_fixed の実行
297
298     fprintf(LOG_FILE, "␣固定活動以外を行っている人数：␣%d 人\n␣" , N_OF_PERSON_nFX); // ログにデータサイズを記載
299     fprintf(LOG_FILE, "␣固定活動を行っている人数：␣%d 人\n\n␣" , N_OF_PERSON_FX); // ログにデータサイズを記載
300
301     cout << "␣固定活動について各ゾーン人口集計完了，␣パーティクルごとのトリップ数計算開始␣" << endl;
302
303     count_trip(); // 関数count_trip の実行
304
305     if (ASSIMILATE_SWITCH == 0) {
306         cout << "␣パーティクルごとのトリップ数計算完了␣" << endl;
307         cout << "␣同化は行わない␣" << endl;
308         fprintf(LOG_FILE, "␣ゾーン人口とトリップ数集計のみ行い，␣同化は行わなかった．\n␣" );
309     }
310     else if (ASSIMILATE_SWITCH == 1) {
311         cout << "␣パーティクルごとのトリップ数計算完了，␣パーティクルの重み計算開始␣" << endl;
312
```

```
313         calculate_weight_of_particle(); // 関数calculate_weight_of_particle の実行
314
315         cout << "␣パーティクルの重み計算完了,␣各パーティクルの復元抽出個数決定・リサンプリング開始␣"
              << endl;
316
317         resample_particle(); // 関数resample_particle の実行
318
319         fprintf(LOG_FILE, "␣尤度最大のパーティクル番号は,␣1から数えた旧パーティクルNo.で,␣【%d】\n\n
              ␣" , MODE_PTCL_NO);
320
321         cout << "␣各パーティクルの復元抽出個数決定・リサンプリング完了,␣中断明け
              PCATS ファイル作成開始␣" << endl;
322
323         output_resampled_file_for_PCATS(); // 関数output_resampled_file_for_PCATS の実行
324
325         cout << "␣中断明けPCATS ファイル作成・出力完了␣" << endl;
326     }
327     time(&timer); // 現在時刻の取得
328     t_st = localtime(&timer); // 現在時刻を現地時刻の構造体に変換
329
330     fprintf(LOG_FILE, "␣プログラム終了\n␣" );
331     fprintf(LOG_FILE, "␣    終了日 :␣%04d 年%02d 月%02d 日 (%s)\n␣" , t_st->tm_year + 1900,
          t_st->tm_mon + 1, t_st->tm_mday, wday[t_st->tm_wday]);
332     fprintf(LOG_FILE, "␣  終了時刻 :␣%02d 時%02d 分%02d 秒\n␣" , t_st->tm_hour, t_st->tm_min,
          t_st->tm_sec);
333     fprintf(LOG_FILE, "␣*****************************\n␣" );
334
335     fclose(LOG_FILE);
336
337     cout << "␣正常終了␣" << endl;
338     return 0;
339 }
340
341 ////////////////////////
342     // ここまでメイン関数_tmain の内容
343     // ここから関数file_set_up の内容（入出力ファイルの設定をする関数）
344 ////////////////////////
345
346 void file_set_up() {
347
348     // 読込：ORIGINAL_PCATS_OUTPUT：PCATS によって出力された，個人ごと時間帯ごとの活動内容
          が書かれたファイル
349     sprintf(
350         PCATSout,
351          "␣C:\\cpp\\filtering\\filtering\\input\\PCATSoutput\\23cities_%d-%d(%d).data␣" ,
               START, END, N_OF_PARTICLE
352         );
353     ORIGINAL_PCATS_OUTPUT = fopen(PCATSout, "␣r␣" );
354     if (ORIGINAL_PCATS_OUTPUT == NULL) {
355         cout << "␣ファイル
              ORIGINAL_PCATS_OUTPUT（PCATS から出力の個人別・時間帯別活動内容）の入力失敗␣" << endl;
356         system( "␣pause␣" );
357     }
358     else {
359         cout << "␣ファイル
              ORIGINAL_PCATS_OUTPUT（PCATS から出力の個人別・時間帯別活動内容）の入力成功␣" << endl;
360     }
```

```
361
362     // 作成・読込：PCATS_SUSPENDED：ORIGINAL_PCATS_OUTPUT から，
          PCATS 中断時刻に関わる行のみを抜き出したファイル
363     sprintf(
364         suspended,
365          "␣C:\\cpp\\filtering\\filtering\\output\\PCATSsuspended\\suspended_23cities_%d-%d
              (%d).data␣" , START, END, N_OF_PARTICLE
366         );
367     PCATS_SUSPENDED = fopen(suspended, "␣w+␣" );
368     if (PCATS_SUSPENDED == NULL) {
369         cout << "␣ファイルPCATS_SUSPENDED（PCATS 中断時刻に関わる行のみを抜き出し）の作成失敗␣" <<
              endl;
370         system( "␣pause␣" );
371     }
372     else {
373         cout << "␣ファイルPCATS_SUSPENDED（PCATS 中断時刻に関わる行のみを抜き出し）の作成成功␣" <<
              endl;
374     }
375
376     // 作成・読込：POUT_nFX：PCATS_SUSPENDED から，固定活動以外を抜き出したファイル
377     sprintf(
378         nonfixed,
379          "␣C:\\cpp\\filtering\\filtering\\output\\PCATSoutputparticle\\particle_non_fixed_%
              d-%d(%d).data␣" , START, END, N_OF_PARTICLE
380         );
381     POUT_nFX = fopen(nonfixed, "␣w+␣" );
382     if (POUT_nFX == NULL) {
383         cout << "␣ファイルPOUT_nFX（PCATS 中断時刻での固定活動以外のパーティクル）の作成失敗␣" <<
              endl;
384         system( "␣pause␣" );
385     }
386     else {
387         cout << "␣ファイルPOUT_nFX（PCATS 中断時刻での固定活動以外のパーティクル）の作成成功␣" <<
              endl;
388     }
389
390     // 作成・読込：POUT_FX：PCATS_SUSPENDED から，固定活動を抜き出したファイル
391     sprintf(
392         fix,
393          "␣C:\\cpp\\filtering\\filtering\\output\\PCATSoutputparticle\\particle_fixed_%d-%d
              (%d).data␣" , START, END, N_OF_PARTICLE
394         );
395     POUT_FX = fopen(fix, "␣w+␣" );
396     if (POUT_FX == NULL) {
397         cout << "␣ファイルPOUT_FX（PCATS 中断時刻での固定活動のパーティクル）の作成失敗␣" << endl;
398         system( "␣pause␣" );
399     }
400     else {
401         cout << "␣ファイルPOUT_FX（PCATS 中断時刻での固定活動のパーティクル）の作成成功␣" << endl;
402     }
403
404     // 作成・読込：POP_OF_ALLZONE_ALLPTCL_nFX：PCATS 中断時の 各ゾーンの固定活動以外の人口
          を，パーティクル数分並べたファイル
405     sprintf(
406         zoneall,
407          "␣C:\\cpp\\filtering\\filtering\\output\\population_of_zone\\
              population_of_allzone_allparticle_non_fixed_%d-%d(%d).data␣" , START, END,
```

```
            N_OF_PARTICLE
408         );
409     POP_OF_ALLZONE_ALLPTCL_nFX = fopen(zoneall, "␣w+␣" );
410     if (POP_OF_ALLZONE_ALLPTCL_nFX == NULL) {
411         cout << "␣ファイルPOP_OF_ALLZONE_ALLPTCL_nFX（PCATS 中断時の␣各ゾーンの固定活動以外の人口
              を,␣パーティクル数分並べたファイル）の作成失敗␣" << endl;
412         system( "␣pause␣" );
413     }
414     else {
415         cout << "␣ファイルPOP_OF_ALLZONE_ALLPTCL_nFX（PCATS 中断時の␣各ゾーンの固定活動以外の人口
              を,␣パーティクル数分並べたファイル）の作成成功␣" << endl;
416     }
417
418     // 作成・読込：POP_OF_ALLZONE_FX：PCATS 中断時の 各ゾーンの固定活動の人口
419     sprintf(
420         zoneall_fix,
421          "␣C:\\cpp\\filtering\\filtering\\output\\population_of_zone\\
              population_of_allzone_fixed_%d-%d.csv␣" , START, END
422         );
423     POP_OF_ALLZONE_FX = fopen(zoneall_fix, "␣w+␣" );
424     if (POP_OF_ALLZONE_FX == NULL) {
425         cout << "␣ファイルPOP_OF_ALLZONE_FX（PCATS 中断時の␣各ゾーンの固定活動の人口）の作成失敗␣"
              << endl;
426         system( "␣pause␣" );
427     }
428     else {
429         cout << "␣ファイルPOP_OF_ALLZONE_FX（PCATS 中断時の␣各ゾーンの固定活動の人口）の作成成功␣"
              << endl;
430     }
431
432     if (MOBILE_OBS_TIME != 1620) {
433     // 読込：POP_OF_ALLZONE_OBS：観測データで，各ゾーンの人口が 1行にゾーン数分だけ並んだファイ
        ル
434         sprintf(
435             zoneall_obs,
436              "␣C:\\cpp\\filtering\\filtering\\input\\obsdata\\pseudo_mobile_spatial_stat(%d
                  ).csv␣" , MOBILE_OBS_TIME
437             );
438         POP_OF_ALLZONE_OBS = fopen(zoneall_obs, "␣r␣" );
439         if (POP_OF_ALLZONE_OBS == NULL) {
440             cout << "␣ファイルPOP_OF_ALLZONE_OBS（観測データで,␣各ゾーンの人口が 1行にゾーン数分だ
                  け並んだファイル）の作成失敗␣" << endl;
441             system( "␣pause␣" );
442         }
443         else {
444             cout << "␣ファイルPOP_OF_ALLZONE_OBS（観測データで,␣各ゾーンの人口が 1行にゾーン数分だ
                  け並んだファイル）の作成成功␣" << endl;
445         }
446     }
447
448     // 読込：ZONECODE：ゾーン番号のOD 組合せが ORDER BY o_zone, d_zone で並んだファイル
449     sprintf(
450         zonecode,
451          "␣C:\\cpp\\filtering\\filtering\\input\\zone_code.csv␣"
452         );
453     ZONECODE = fopen(zonecode, "␣r␣" );
454     if (ZONECODE == NULL) {
```

```
455         cout << "␣ファイルZONECODE（ゾーン番号の OD 組合せが ORDER␣BY␣o_zone,␣
              d_zone で並んだファイル）の作成失敗␣" << endl;
456         system( "␣pause␣" );
457     }
458     else {
459         cout << "␣ファイルZONECODE（ゾーン番号の OD 組合せが ORDER␣BY␣o_zone,␣
              d_zone で並んだファイル）の作成成功␣" << endl;
460     }
461
462
463     // 読込：IC_OBS：観測データで，個人ID，拡大係数，発時刻，着時刻，発ゾーン，着ゾーン，交通手段が，
          トリップ数分の行数だけ並んだファイル
464     sprintf(
465         ic_obs,
466          "␣C:\\cpp\\filtering\\filtering\\input\\obsdata\\pseudo_IC.csv␣"
467         );
468     IC_OBS = fopen(ic_obs, "␣r␣" );
469     if (IC_OBS == NULL) {
470         cout << "␣ファイルIC_OBS（観測データで,␣各ゾーンの人口が 1行にゾーン数分だけ並んだファイル）
              の作成失敗␣" << endl;
471         system( "␣pause␣" );
472     }
473     else {
474         cout << "␣ファイルIC_OBS（観測データで,␣各ゾーンの人口が 1行にゾーン数分だけ並んだファイル）
              の作成成功␣" << endl;
475     }
476
477     // 作成：WEIGHT_OF_MOBILE：モバイル空間統計に関する各パーティクルの重みと正規化重みを，パー
          ティクル数分の行数だけ並べたファイル
478     sprintf(
479         weight_mobile,
480          "␣C:\\cpp\\filtering\\filtering\\output\\weight_of_particle\\weight_of_mobile_%d-%
              d(%d).csv␣" , START, END, N_OF_PARTICLE
481         );
482     WEIGHT_OF_MOBILE = fopen(weight_mobile, "␣w␣" );
483     if (WEIGHT_OF_MOBILE == NULL) {
484         cout << "␣ファイル
              WEIGHT_OF_MOBILE（モバイル空間統計に関する各パーティクルの重みと正規化重みを,␣パーティクル
              数分の行数だけ並べたファイル)の作成失敗␣" << endl;
485         system( "␣pause␣" );
486     }
487     else {
488         cout << "␣ファイル
              WEIGHT_OF_MOBILE（モバイル空間統計に関する各パーティクルの重みと正規化重みを,␣パーティクル
              数分の行数だけ並べたファイル)の作成成功␣" << endl;
489     }
490
491     // 作成：WEIGHT_OF_IC：IC カードデータに関する各パーティクルの重みと正規化重みを，パーティク
          ル数分の行数だけ並べたファイル
492     sprintf(
493         weight_ic,
494          "␣C:\\cpp\\filtering\\filtering\\output\\weight_of_particle\\weight_of_ic_%d-%d(%d
              ).csv␣" , START, END, N_OF_PARTICLE
495         );
496     WEIGHT_OF_IC = fopen(weight_ic, "␣w␣" );
497     if (WEIGHT_OF_IC == NULL) {
498         cout << "␣ファイルWEIGHT_OF_IC（IC カードデータに関する各パーティクルの重みと正規化重みを,␣
```

```
        パーティクル数分の行数だけ並べたファイル)の作成失敗␣" << endl;
499         system( "␣pause␣" );
500     }
501     else {
502         cout << "␣ファイルWEIGHT_OF_IC（IC カードデータに関する各パーティクルの重みと正規化重みを,␣
              パーティクル数分の行数だけ並べたファイル)の作成成功␣" << endl;
503     }
504
505     // 作成・読込：WEIGHT_OF_PTCLS：各パーティクルの重みと正規化重みを，パーティクル数分の行数だ
          け並べたファイル
506     sprintf(
507         weight_ptcl,
508          "␣C:\\cpp\\filtering\\filtering\\output\\weight_of_particle\\weight_of_particle_%d
              -%d(%d).csv␣" , START, END, N_OF_PARTICLE
509         );
510     WEIGHT_OF_PTCLS = fopen(weight_ptcl, "␣w+␣" );
511     if (WEIGHT_OF_PTCLS == NULL) {
512         cout << "␣ファイルWEIGHT_OF_PTCLS（各パーティクルの重みと正規化重みを,␣パーティクル数分の行
              数だけ並べたファイル)の作成失敗␣" << endl;
513         system( "␣pause␣" );
514     }
515     else {
516         cout << "␣ファイルWEIGHT_OF_PTCLS（各パーティクルの重みと正規化重みを,␣パーティクル数分の行
              数だけ並べたファイル)の作成成功␣" << endl;
517     }
518
519     // 作成・読込：CORRESPONDENCE_TABLE_OF_PTCLS：リサンプリング前後における，新旧パーテ
          ィクルの対応表
520     sprintf(
521         correspondence,
522          "␣C:\\cpp\\filtering\\filtering\\output\\resampled_particle\\correspondence_table_
              %d-%d(%d).csv␣" , START, END, N_OF_PARTICLE
523         );
524     CORRESPONDENCE_TABLE_OF_PTCLS = fopen(correspondence, "␣w+␣" );
525     if (CORRESPONDENCE_TABLE_OF_PTCLS == NULL) {
526         cout << "␣ファイルCORRESPONDENCE_TABLE_OF_PTCLS（リサンプリング前後における,␣新旧パーティ
              クルの対応表)の作成失敗␣" << endl;
527         system( "␣pause␣" );
528     }
529     else {
530         cout << "␣ファイルCORRESPONDENCE_TABLE_OF_PTCLS（リサンプリング前後における,␣新旧パーティ
              クルの対応表)の作成成功␣" << endl;
531     }
532
533     // 作成：REVISED_PCATS_OUTPUT：リサンプリングされたパーティクルに基づいてPCATS 中断時の
          データを修正した，このプログラムの最終出力ファイル
534     sprintf(
535         PCATSout_revised,
536          "␣C:\\cpp\\filtering\\filtering\\output\\revised_PCATS_output\\revised_PCATS_%d-%d
              (%d).data␣" , START, END, N_OF_PARTICLE
537         );
538     REVISED_PCATS_OUTPUT = fopen(PCATSout_revised, "␣w␣" );
539     if (REVISED_PCATS_OUTPUT == NULL) {
540         cout << "␣ファイル
              REVISED_PCATS_OUTPUT（リサンプリングされたパーティクルに基づいて PCATS 中断時の␣データを
              修正した,␣このプログラムの最終出力ファイル)の作成失敗␣" << endl;
541         system( "␣pause␣" );
```

```
542     }
543     else {
544         cout << "␣ファイル
              REVISED_PCATS_OUTPUT（リサンプリングされたパーティクルに基づいて PCATS 中断時の␣データを
              修正した,␣このプログラムの最終出力ファイル）の作成成功␣" << endl;
545     }
546
547     return;
548 }
549
550 ////////////////////////
551     // ここまで関数file_set_up の内容（入出力ファイルの設定をする関数）
552     // ここから関数maximum の内容（各対象ゾーンに応じた値の配列中で，最大値を持つゾーンのゾーンコー
          ドを返す関数）
553 ////////////////////////
554
555 // 関数maximum の定義
556 // 各対象ゾーンに応じた値を要素に持つ配列all と，対象ゾーン数number を引数とし，
557 // all の中で最大の要素を持つゾーンのゾーンコードを整数の返り値とする関数
558
559 int maximum(int *all, int number) { // 整数の配列all と，整数number を引数，整数を返り値とする関数，
      maximum
560     int max = all[0]; // 整数max の初期値は all の 0 番目の要素
561     for (int i = 1; i < number; i++) { // インクリメント変数i が number の値になるまで，
          max の値を更新
562         if (max < all[i]) {
563             max = all[i];
564         }
565     }
566     if (max == all[0]) { return 10; } // max が all の何番目の要素かによって，返り値を設定
567     else if (max == all[1]) { return 11; }
568     else if (max == all[2]) { return 12; }
569     else if (max == all[3]) { return 13; }
570     else if (max == all[4]) { return 20; }
571     else if (max == all[5]) { return 21; }
572     else if (max == all[6]) { return 22; }
573     else if (max == all[7]) { return 23; }
574     else if (max == all[8]) { return 24; }
575     else if (max == all[9]) { return 30; }
576     else if (max == all[10]) { return 31; }
577     else if (max == all[11]) { return 32; }
578     else if (max == all[12]) { return 33; }
579     else if (max == all[13]) { return 34; }
580     else if (max == all[14]) { return 110; }
581     else if (max == all[15]) { return 111; }
582     else if (max == all[16]) { return 112; }
583     else if (max == all[17]) { return 120; }
584     else if (max == all[18]) { return 121; }
585     else if (max == all[19]) { return 122; }
586     else if (max == all[20]) { return 123; }
587     else if (max == all[21]) { return 124; }
588     else if (max == all[22]) { return 130; }
589     else if (max == all[23]) { return 131; }
590     else if (max == all[24]) { return 132; }
591     else if (max == all[25]) { return 133; }
592     else if (max == all[26]) { return 134; }
593     else if (max == all[27]) { return 135; }
```

```
594     else if (max == all[28]) { return 136; }
595     else if (max == all[29]) { return 137; }
596     else if (max == all[30]) { return 138; }
597     else if (max == all[31]) { return 210; }
598     else if (max == all[32]) { return 211; }
599     else if (max == all[33]) { return 212; }
600     else if (max == all[34]) { return 220; }
601     else if (max == all[35]) { return 221; }
602     else if (max == all[36]) { return 222; }
603     else if (max == all[37]) { return 223; }
604     else if (max == all[38]) { return 230; }
605     else if (max == all[39]) { return 231; }
606     else if (max == all[40]) { return 232; }
607     else if (max == all[41]) { return 233; }
608     else if (max == all[42]) { return 234; }
609     else if (max == all[43]) { return 235; }
610     else if (max == all[44]) { return 240; }
611     else if (max == all[45]) { return 241; }
612     else if (max == all[46]) { return 242; }
613     else if (max == all[47]) { return 243; }
614     else if (max == all[48]) { return 310; }
615     else if (max == all[49]) { return 311; }
616     else if (max == all[50]) { return 312; }
617     else if (max == all[51]) { return 320; }
618     else if (max == all[52]) { return 321; }
619     else if (max == all[53]) { return 322; }
620     else if (max == all[54]) { return 323; }
621     else if (max == all[55]) { return 330; }
622     else if (max == all[56]) { return 331; }
623     else if (max == all[57]) { return 332; }
624     else if (max == all[58]) { return 333; }
625     else if (max == all[59]) { return 334; }
626     else if (max == all[60]) { return 335; }
627     else if (max == all[61]) { return 336; }
628     else if (max == all[62]) { return 340; }
629     else if (max == all[63]) { return 341; }
630     else if (max == all[64]) { return 342; }
631     else if (max == all[65]) { return 343; }
632     else if (max == all[66]) { return 344; }
633     else if (max == all[67]) { return 345; }
634     else if (max == all[68]) { return 346; }
635     else if (max == all[69]) { return 347; }
636     else if (max == all[70]) { return 348; }
637     else if (max == all[71]) { return 349; }
638     else if (max == all[72]) { return 410; }
639     else if (max == all[73]) { return 411; }
640     else if (max == all[74]) { return 412; }
641     else if (max == all[75]) { return 420; }
642     else if (max == all[76]) { return 421; }
643     else if (max == all[77]) { return 422; }
644     else if (max == all[78]) { return 423; }
645     else if (max == all[79]) { return 424; }
646     else if (max == all[80]) { return 430; }
647     else if (max == all[81]) { return 431; }
648     else if (max == all[82]) { return 432; }
649     else if (max == all[83]) { return 433; }
650     else if (max == all[84]) { return 434; }
```

```
651     else if (max == all[85]) { return 435; }
652     else if (max == all[86]) { return 436; }
653     else if (max == all[87]) { return 437; }
654     else if (max == all[88]) { return 510; }
655     else if (max == all[89]) { return 511; }
656     else if (max == all[90]) { return 512; }
657     else if (max == all[91]) { return 513; }
658     else if (max == all[92]) { return 514; }
659     else if (max == all[93]) { return 515; }
660     else if (max == all[94]) { return 520; }
661     else if (max == all[95]) { return 521; }
662     else if (max == all[96]) { return 522; }
663     else if (max == all[97]) { return 523; }
664     else if (max == all[98]) { return 530; }
665     else if (max == all[99]) { return 531; }
666     else if (max == all[100]) { return 532; }
667     else if (max == all[101]) { return 610; }
668     else if (max == all[102]) { return 611; }
669     else if (max == all[103]) { return 612; }
670     else if (max == all[104]) { return 613; }
671     else if (max == all[105]) { return 614; }
672     else if (max == all[106]) { return 620; }
673     else if (max == all[107]) { return 621; }
674     else if (max == all[108]) { return 622; }
675     else if (max == all[109]) { return 623; }
676     else if (max == all[110]) { return 630; }
677     else if (max == all[111]) { return 631; }
678     else if (max == all[112]) { return 632; }
679     else if (max == all[113]) { return 633; }
680     else if (max == all[114]) { return 634; }
681
682     else {
683         cout << "␣関数maximum に代入された配列の大きさと,␣想定されるゾーン数が一致しません␣" <<
              endl;
684         system( "␣pause␣" );
685         return 9999999; // 仮にreturn するだけの値であり，これが返る前にpause で止まる
686     }
687 }
688
689
690 ////////////////////////
691     // ここまで関数maximum の内容（各対象ゾーンに応じた値の配列中で，最大値を持つゾーンのゾーンコー
        ドを返す関数）
692     // ここから関数MAP の内容（整数配列中で最大値を持つものが何番目の要素であるかを返す関数）
693 ////////////////////////
694
695 // 関数MAP の定義
696 // int の配列 par と，その要素数number を引数とし，
697 // par の中で最大の要素を持つものが（1 番目から数えて）何番目であるかを整数の返り値とする関数
698 // 最大の要素が複数ある場合は，要素番号が最も小さいものを返す
699
700 int MAP(int *par, int number) { // int の配列 par と，整数number を引数，整数を返り値とする関数，
      MAP
701     int MAP = par[0]; // int 型変数 MAP の初期値は par の 0 番目の要素
702     int MAPnum = 1; // 整数MAPnum の初期値は 1
703     for (int i = 1; i < number; i++) { // インクリメント変数i が number の値になるまで，
          MAP と MAPnum の値を更新
```

```
704         if (MAP < par[i]) {
705             MAP = par[i]; // MAP の値は，par の要素のうち最大のものになる
706             MAPnum = i + 1; // MAPnum の値は，
                  par の要素のうち最大のものが（0 番目から数えて）i 番目のとき，i+1になる
707         }
708     }
709
710     return MAPnum;
711 }
712
713
714 ////////////////////////
715     // ここまで関数MAP の内容（整数配列中で最大値を持つものが何番目の要素であるかを返す関数）
716     // ここから関数distance の内容（2 つの整数配列のユークリッド距離の平方を返す関数）
717 ////////////////////////
718
719 // 関数distance の定義
720 // 整数配列x と，整数配列y を引数とし，
721 // x と y のユークリッド距離の平方の 10 万分の 1 を 8 バイト小数の返り値とする関数
722
723 double distance(int *x, int *y) {
724     if (sizeof x / sizeof x[0] == sizeof y / sizeof y[0]) { //
          x と y の要素数が同じである場合のみ関数を実行
725         double sum = 0; // 要素の差の 2乗を足していく変数
726         double sum2 = 0; // sum の 10 万分の 1
727         for (int i = 0; i < (sizeof x / sizeof x[0]); i++) { // インクリメント変数がx,
              y の要素数に達するまで，sum に差の 2 乗を足す
728             int pp; // 要素の差の 2乗を表す変数
729             pp = (x[i] - y[i]) * (x[i] - y[i]);
730             sum += pp;
731         }
732         sum2 = (sum / 100000);
733         return sum2;
734     }
735     else { // x と y の要素数が違うときはエラーを返して処理を中断
736         cout << "␣要素数が違うためベクトルの距離計算不可␣" << endl;
737         system( "␣pause␣" );
738         return 9999999; // 仮にreturn するだけの値であり，これが返る前にpause で止まる
739     }
740 }
741
742
743 ////////////////////////
744     // ここまで関数distance の内容（2 つの整数配列のユークリッド距離の平方を返す関数）
745     // ここから関数extract_suspended_act の内容（PCATS 中断時の データを抜き出す関数）
746 ////////////////////////
747
748
749 void extract_suspended_act() {
750
751     char indv_id[N_OF_DIGIT_OF_INDV_ID + 10]; // 個人ID：配列の大きさは個人ID の桁数に依存
752     int col[N_OF_PCATS_PARAMETER]; // 整数が入っているカラムをまとめて配列にしたもの
753                 // 0：カテゴリ，1：自宅ゾーン，2：就業就学地ゾーン，3：性別，4：年齢，5：職業，6：免
                      許有無，7：世帯内自動車台数，
754                 // 8：拡大係数，9：世帯人数，10：活動内容，11：活動開始時刻，12：活動終了時刻，13：
                      活動場所ゾーン，14：活動施設，
755                 // 15：固定活動ダミー，16：活動後移動有無，17：移動開始時刻，18：移動終了時刻，19：
```

```
                移動手段，20：車ゾーン，21：車場所，22：中断ダミー
756     char suspend_act[8]; // 中断時の活動：” MOVING ” の 6文字が最長
757     int i = 0; // インクリメント変数
758
759     while ( // ファイルの読込
760         fscanf(ORIGINAL_PCATS_OUTPUT, "␣%s␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%s␣" ,
761             indv_id, &col[0], &col[1], &col[2], &col[3], &col[4], &col[5], &col[6], &col[7], &col[8], &col[9], &col[10],
762             &col[11], &col[12], &col[13], &col[14], &col[15], &col[16], &col[17], &col[18], &col[19], &col[20], &col[21], &col[22], suspend_act) != EOF
763         )
764     {
765         if (END != 1620) { // 1620は調査翌日AM3:00，即ち PCATS の最終時刻を表す．つまり最終時刻以外で中断されていれば以下を実行
766             if (col[22] == 1) { // 中断時点の活動ならば以下の通り書き込みを行う
767                 fprintf(PCATS_SUSPENDED, "␣%s␣%d␣%d␣%d␣%d␣%d␣%d␣%s\n␣" ,
768                     indv_id, col[0], col[1], col[4], col[8], col[10], col[13], suspend_act);
769                 // 個人ID，カテゴリ，自宅ゾーン，年齢，拡大係数，活動内容，活動場所ゾーン，中断時の活動
770             }
771
772         }
773         else { // 最終時刻で中断されていた場合には以下を実行
774             if (col[12] == 1620) { // 活動終了時刻が 1620の活動について以下の通り書き込みを行う
775                 fprintf(PCATS_SUSPENDED, "␣%s␣%d␣%d␣%d␣%d␣%d␣%d␣%s\n␣" ,
776                     indv_id, col[0], col[1], col[4], col[8], col[10], col[13], suspend_act);
777                 // 個人ID，カテゴリ，自宅ゾーン，年齢，拡大係数，活動内容，活動場所ゾーン，中断時の活動
778             }
779
780         }
781
782         if (i == 500000) { cout << "␣50万行目処理完了␣" << endl; }
783         if (i == 1000000) { cout << "␣100万行目処理完了␣" << endl; }
784         if (i == 1500000) { cout << "␣150万行目処理完了␣" << endl; }
785         if (i == 2000000) { cout << "␣200万行目処理完了␣" << endl; }
786         if (i == 2500000) { cout << "␣250万行目処理完了␣" << endl; }
787         if (i == 3000000) { cout << "␣300万行目処理完了␣" << endl; }
788         if (i == 3500000) { cout << "␣350万行目処理完了␣" << endl; }
789         if (i == 4000000) { cout << "␣400万行目処理完了␣" << endl; }
790         if (i == 4500000) { cout << "␣450万行目処理完了␣" << endl; }
791         if (i == 5000000) { cout << "␣500万行目処理完了␣" << endl; }
792         if (i == 5500000) { cout << "␣550万行目処理完了␣" << endl; }
793         if (i == 6000000) { cout << "␣600万行目処理完了␣" << endl; }
794         if (i == 6500000) { cout << "␣650万行目処理完了␣" << endl; }
795         if (i == 7000000) { cout << "␣700万行目処理完了␣" << endl; }
796         if (i == 7500000) { cout << "␣750万行目処理完了␣" << endl; }
797         if (i == 8000000) { cout << "␣800万行目処理完了␣" << endl; }
798
799         i++;
800     }
801
802     N_OF_ROW_OF_PCATS_OUTPUT = i; // ログに記録するため，入力となるPCATS ファイルの行数を記録
803
804     fseek(ORIGINAL_PCATS_OUTPUT, 0, SEEK_SET); // ORIGINAL_PCATS_OUTPUT は PCATS 中断明けデータを作る時にも使うので，次の読込に備えてポインタを先頭にしておく
805     fseek(PCATS_SUSPENDED, 0, SEEK_SET); // PCATS_SUSPENDED は読書両方のモード w+なので，
```

```
        書き込みが終わった今，次の読込に備えてポインタを先頭にしておく
806     return;
807 }
808 
809 
810 ////////////////////////
811     // ここまで関数extract_suspended_act の内容（PCATS 中断時の データを抜き出す関数）
812     // ここから関数classify_fixed_or_not の内容（抜き出したデータを，固定活動中とそれ以外に分類する関数）
813 ////////////////////////
814 
815 
816 void classify_fixed_or_not() {
817 
818     // PCATS_SUSPENDED（PCATS 中断時刻に関わる活動）から読み込んだ値を入れるための変数を宣言
819     char indv_id[N_OF_DIGIT_OF_INDV_ID + 10]; // 個人ID
820     char temp_id[N_OF_DIGIT_OF_INDV_ID + 10] = "␣DifferentFromAnyIds␣" ; // 比較用なので，初期
          値はどのID とも被らない文字列にしておく
821     int category; // カテゴリ
822     int address_zone; // 自宅ゾーン
823     int age; // 年齢
824     int scaling; // 拡大係数
825     int activity; // 活動内容
826     int act_zone; // 活動場所ゾーン
827     char suspend_act[8]; // 中断時の活動
828     int i = 0;
829     int j = 0; // インクリメント変数
830 
831     while ( // ファイルの読込
832         fscanf(PCATS_SUSPENDED, "␣%s␣%d␣%d␣%d␣%d␣%d␣%d␣%s␣" ,
833             indv_id, &category, &address_zone, &age, &scaling, &activity, &act_zone,
                  suspend_act) != EOF
834         )
835     {
836         if (strcmp(suspend_act, "␣FIX␣" ) == 0 || END == 1620) { // 文字列比較：中断時の活動
              suspend_act が ” FIX ” であるか，中断時刻が 1620ならば以下を実行
837             if (i == 0) { // インクリメントの最初では以下の通り書き込みを行う
838                 fprintf(POUT_FX, "␣%s␣%d␣%d␣%d␣%d␣%d␣%d␣" , indv_id, category, address_zone,
                        age, scaling, activity, act_zone);
839             }
840             else if (i > 0 && strncmp(indv_id, temp_id, N_OF_DIGIT_OF_INDV_ID) == 0) { // 参照
                  している個人ID（のうち，最後のパーティクル番号を除いた部分)が，直前に参照した個人
                  ID と等しければ，ゾーンのみ追加
841                 fprintf(POUT_FX, "␣␣%d␣" , act_zone);
842             }
843             else { // 直前に参照した個人ID と異なれば，改行してから以下の通り書き込みを行う
844                 fprintf(POUT_FX, "␣\n%s␣%d␣%d␣%d␣%d␣%d␣%d␣" , indv_id, category,
                      address_zone, age, scaling, activity, act_zone);
845             }
846 
847             i++;
848 
849         }
850         else { // 中断時の活動suspend_act が ” FIX ” でないならば以下を実行
851             if (j == 0) { // インクリメントの最初では以下の通り書き込みを行う
852                 fprintf(POUT_nFX, "␣%s␣%d␣%d␣%d␣%d␣%d␣%d␣" , indv_id, category, address_zone
                        , age, scaling, activity, act_zone);
853             }
```

```
854             else if (j > 0 && strncmp(indv_id, temp_id, N_OF_DIGIT_OF_INDV_ID) == 0) { // 参照
                  している個人ID（のうち，最後のパーティクル番号を除いた部分)が，直前に参照した個人
                  ID と等しければ，ゾーンのみ追加
855                 fprintf(POUT_nFX, "␣␣%d␣" , act_zone);
856             }
857             else { // 直前に参照した個人ID と異なれば，改行してから以下の通り書き込みを行う
858                 fprintf(POUT_nFX, "␣\n%s␣%d␣%d␣%d␣%d␣%d␣%d␣" , indv_id, category,
                      address_zone, age, scaling, activity, act_zone);
859             }
860
861             j++;
862
863         }
864
865         if (i + j == 1000000) { cout << "␣100万行目処理完了␣" << endl; }
866         if (i + j == 2000000) { cout << "␣200万行目処理完了␣" << endl; }
867         if (i + j == 3000000) { cout << "␣300万行目処理完了␣" << endl; }
868         if (i + j == 4000000) { cout << "␣400万行目処理完了␣" << endl; }
869         if (i + j == 5000000) { cout << "␣500万行目処理完了␣" << endl; }
870         if (i + j == 6000000) { cout << "␣600万行目処理完了␣" << endl; }
871         if (i + j == 7000000) { cout << "␣700万行目処理完了␣" << endl; }
872         if (i + j == 8000000) { cout << "␣800万行目処理完了␣" << endl; }
873
874         strcpy(temp_id, indv_id); // temp_id に indv_id の値をコピー
875
876     }
877
878     fprintf(POUT_FX, "␣\n␣" ); // 最後に改行
879     fprintf(POUT_nFX, "␣\n␣" );
880
881     fclose(PCATS_SUSPENDED);
882     fseek(POUT_FX, 0, SEEK_SET); // POUT_FX は読書両方のモード w+なので，書き込みが終わった今，
        次の読込に備えてポインタを先頭にしておく
883     fseek(POUT_nFX, 0, SEEK_SET); // POUT_nFX は読書両方のモード w+なので，書き込みが終わった
        今，次の読込に備えてポインタを先頭にしておく
884
885     return;
886 }
887
888
889 ////////////////////////
890     // ここまで関数classify_fixed_or_not の内容（抜き出したデータを，固定活動中とそれ以外に分類する関数）
891     // ここから関数extract_PCATSout_at_this_particle_non_fixed の内容（PCATS 中断時の 固定活動以外の
        データから，引数とするパーティクルNo.におけるものだけを抜き出す関数）
892 ////////////////////////
893
894
895 void extract_PCATSout_at_this_particle_non_fixed(int p) {
896
897
898     // POUT_nFX から読み込んだ値を入れる変数の宣言
899     char indv_id[N_OF_DIGIT_OF_INDV_ID + 10]; // 個人ID
900     int category; // カテゴリ
901     int address_zone; // 自宅ゾーン
902     int age; // 年齢
903     int scaling; // 拡大係数
904     int activity; // 活動内容
```

```
905     int act_zone[N_OF_PARTICLE]; // 活動場所ゾーン
906 
907     // 読み込んだ個人ID は，末尾にパーティクル番号が「001」として入っているので，正しいパーティクル番号をつける
908     char indv_id_without_ptcl_num[N_OF_DIGIT_OF_INDV_ID + 10]; // そのために，純粋な個人ID 部分のみを格納する配列を用意
909 
910     while ( // ファイルの読込
911         fscanf(POUT_nFX, "␣%s␣%d␣%d␣%d␣%d␣%d␣␣" , indv_id, &category, &address_zone, &age, &scaling, &activity) != EOF
912         // 外側ループで個人ID，カテゴリ，自宅ゾーン，年齢，拡大係数，活動内容を読込
913         )
914     {
915         int i = 0; // パーティクル数分のインクリメント変数
916         while ( // 内側ループでパーティクル数分だけ活動場所ゾーンを読込
917             i < N_OF_PARTICLE && fscanf(POUT_nFX, "␣%d␣␣" , &act_zone[i]) != EOF // ここの条件の順序を入れ替えると，fscanf が先に実行されてしまう分，ポインタの位置がずれるので注意
918             )
919         {i += 1;}
920         // 本当はこの関数定義内で想定しているパーティクル数と，実際のファイルのパーティクル数が一致しなかった時のエラーも書きたい
921         // あとは，現在のパーティクル番号p の情報のみ書き込む
922 
923         strncpy(indv_id_without_ptcl_num, indv_id, N_OF_DIGIT_OF_INDV_ID); // パーティクル番号を除いた，純粋な個人ID の桁数分だけコピーして格納
924         indv_id_without_ptcl_num[N_OF_DIGIT_OF_INDV_ID] = '\0'; // 末尾にヌル文字を記録
925 
926         fprintf(POUT_nFX_THIS_PTCL, "␣%s%04d␣%d␣%d␣%d␣%d␣%d␣%d\n␣" ,
            indv_id_without_ptcl_num, p + 1, category, address_zone, age, scaling, activity,
            act_zone[p]);
927         // 個人ID は，末尾に正しいパーティクル番号を，頭に 0を付けて 3桁に揃えて付加し，出力
928     }
929 
930     fseek(POUT_nFX, 0, SEEK_SET); // POUT_nFX は次の p ループでも使うので，次の読込に備えてポインタを先頭にしておく
931     fseek(POUT_nFX_THIS_PTCL, 0, SEEK_SET); // POUT_nFX_THIS_PTCL は読書両方のモード w+なので，書き込みが終わった今，次の読込に備えてポインタを先頭にしておく
932 
933     return;
934 }
935 
936 
937 ////////////////////////
938     // ここまで関数extract_PCATSout_at_this_particle_non_fixed の内容（PCATS 中断時の 固定活動以外のデータから，引数とするパーティクルNo.におけるものだけを抜き出す関数）
939     // ここから関数count_zone_population_at_this_particle_non_fixed の内容（引数とするパーティクル No.における，固定活動以外のゾーン別人口を数える関数）
940 ////////////////////////
941 
942 
943 void count_zone_population_at_this_particle_non_fixed(int p) {
944 
945     // POUT_nFX_THIS_PTCL から読み込んだ値を入れる変数の宣言
946     char indv_id[N_OF_DIGIT_OF_INDV_ID + 10]; // 個人ID
947     int category; // カテゴリ
948     int address_zone; // 自宅ゾーン
949     int age; // 年齢
```

```
950     int scaling; // 拡大係数
951     int activity; // 活動内容
952     int act_zone; // 活動場所ゾーン
953     // POP_OF_ALLZONE_THIS_PTCL_nFX へ書き込む値を入れる変数の宣言
954     int zone_pop[N_OF_ZONE] = { 0 }; // 各ゾーンの人口値，0で初期化
955 
956     int i = 0; // 固定活動以外の人数を数えるためのインクリメント変数
957 
958     while ( // ファイルの読込
959         fscanf(POUT_nFX_THIS_PTCL, "␣%s␣%d␣%d␣%d␣%d␣%d␣%d␣" ,
960             indv_id, &category, &address_zone, &age, &scaling, &activity, &act_zone) != EOF
961         )
962     { // このパーティクル番号において，ゾーンごとに人口を集計
963         if (act_zone == 10) { zone_pop[0] += scaling; }
964         else if (act_zone == 11) { zone_pop[1] += scaling; }
965         else if (act_zone == 12) { zone_pop[2] += scaling; }
966         else if (act_zone == 13) { zone_pop[3] += scaling; }
967         else if (act_zone == 20) { zone_pop[4] += scaling; }
968         else if (act_zone == 21) { zone_pop[5] += scaling; }
969         else if (act_zone == 22) { zone_pop[6] += scaling; }
970         else if (act_zone == 23) { zone_pop[7] += scaling; }
971         else if (act_zone == 24) { zone_pop[8] += scaling; }
972         else if (act_zone == 30) { zone_pop[9] += scaling; }
973         else if (act_zone == 31) { zone_pop[10] += scaling; }
974         else if (act_zone == 32) { zone_pop[11] += scaling; }
975         else if (act_zone == 33) { zone_pop[12] += scaling; }
976         else if (act_zone == 34) { zone_pop[13] += scaling; }
977         else if (act_zone == 110) { zone_pop[14] += scaling; }
978         else if (act_zone == 111) { zone_pop[15] += scaling; }
979         else if (act_zone == 112) { zone_pop[16] += scaling; }
980         else if (act_zone == 120) { zone_pop[17] += scaling; }
981         else if (act_zone == 121) { zone_pop[18] += scaling; }
982         else if (act_zone == 122) { zone_pop[19] += scaling; }
983         else if (act_zone == 123) { zone_pop[20] += scaling; }
984         else if (act_zone == 124) { zone_pop[21] += scaling; }
985         else if (act_zone == 130) { zone_pop[22] += scaling; }
986         else if (act_zone == 131) { zone_pop[23] += scaling; }
987         else if (act_zone == 132) { zone_pop[24] += scaling; }
988         else if (act_zone == 133) { zone_pop[25] += scaling; }
989         else if (act_zone == 134) { zone_pop[26] += scaling; }
990         else if (act_zone == 135) { zone_pop[27] += scaling; }
991         else if (act_zone == 136) { zone_pop[28] += scaling; }
992         else if (act_zone == 137) { zone_pop[29] += scaling; }
993         else if (act_zone == 138) { zone_pop[30] += scaling; }
994         else if (act_zone == 210) { zone_pop[31] += scaling; }
995         else if (act_zone == 211) { zone_pop[32] += scaling; }
996         else if (act_zone == 212) { zone_pop[33] += scaling; }
997         else if (act_zone == 220) { zone_pop[34] += scaling; }
998         else if (act_zone == 221) { zone_pop[35] += scaling; }
999         else if (act_zone == 222) { zone_pop[36] += scaling; }
1000        else if (act_zone == 223) { zone_pop[37] += scaling; }
1001        else if (act_zone == 230) { zone_pop[38] += scaling; }
1002        else if (act_zone == 231) { zone_pop[39] += scaling; }
1003        else if (act_zone == 232) { zone_pop[40] += scaling; }
1004        else if (act_zone == 233) { zone_pop[41] += scaling; }
1005        else if (act_zone == 234) { zone_pop[42] += scaling; }
1006        else if (act_zone == 235) { zone_pop[43] += scaling; }
```

```
        else if (act_zone == 240) { zone_pop[44] += scaling; }
        else if (act_zone == 241) { zone_pop[45] += scaling; }
        else if (act_zone == 242) { zone_pop[46] += scaling; }
        else if (act_zone == 243) { zone_pop[47] += scaling; }
        else if (act_zone == 310) { zone_pop[48] += scaling; }
        else if (act_zone == 311) { zone_pop[49] += scaling; }
        else if (act_zone == 312) { zone_pop[50] += scaling; }
        else if (act_zone == 320) { zone_pop[51] += scaling; }
        else if (act_zone == 321) { zone_pop[52] += scaling; }
        else if (act_zone == 322) { zone_pop[53] += scaling; }
        else if (act_zone == 323) { zone_pop[54] += scaling; }
        else if (act_zone == 330) { zone_pop[55] += scaling; }
        else if (act_zone == 331) { zone_pop[56] += scaling; }
        else if (act_zone == 332) { zone_pop[57] += scaling; }
        else if (act_zone == 333) { zone_pop[58] += scaling; }
        else if (act_zone == 334) { zone_pop[59] += scaling; }
        else if (act_zone == 335) { zone_pop[60] += scaling; }
        else if (act_zone == 336) { zone_pop[61] += scaling; }
        else if (act_zone == 340) { zone_pop[62] += scaling; }
        else if (act_zone == 341) { zone_pop[63] += scaling; }
        else if (act_zone == 342) { zone_pop[64] += scaling; }
        else if (act_zone == 343) { zone_pop[65] += scaling; }
        else if (act_zone == 344) { zone_pop[66] += scaling; }
        else if (act_zone == 345) { zone_pop[67] += scaling; }
        else if (act_zone == 346) { zone_pop[68] += scaling; }
        else if (act_zone == 347) { zone_pop[69] += scaling; }
        else if (act_zone == 348) { zone_pop[70] += scaling; }
        else if (act_zone == 349) { zone_pop[71] += scaling; }
        else if (act_zone == 410) { zone_pop[72] += scaling; }
        else if (act_zone == 411) { zone_pop[73] += scaling; }
        else if (act_zone == 412) { zone_pop[74] += scaling; }
        else if (act_zone == 420) { zone_pop[75] += scaling; }
        else if (act_zone == 421) { zone_pop[76] += scaling; }
        else if (act_zone == 422) { zone_pop[77] += scaling; }
        else if (act_zone == 423) { zone_pop[78] += scaling; }
        else if (act_zone == 424) { zone_pop[79] += scaling; }
        else if (act_zone == 430) { zone_pop[80] += scaling; }
        else if (act_zone == 431) { zone_pop[81] += scaling; }
        else if (act_zone == 432) { zone_pop[82] += scaling; }
        else if (act_zone == 433) { zone_pop[83] += scaling; }
        else if (act_zone == 434) { zone_pop[84] += scaling; }
        else if (act_zone == 435) { zone_pop[85] += scaling; }
        else if (act_zone == 436) { zone_pop[86] += scaling; }
        else if (act_zone == 437) { zone_pop[87] += scaling; }
        else if (act_zone == 510) { zone_pop[88] += scaling; }
        else if (act_zone == 511) { zone_pop[89] += scaling; }
        else if (act_zone == 512) { zone_pop[90] += scaling; }
        else if (act_zone == 513) { zone_pop[91] += scaling; }
        else if (act_zone == 514) { zone_pop[92] += scaling; }
        else if (act_zone == 515) { zone_pop[93] += scaling; }
        else if (act_zone == 520) { zone_pop[94] += scaling; }
        else if (act_zone == 521) { zone_pop[95] += scaling; }
        else if (act_zone == 522) { zone_pop[96] += scaling; }
        else if (act_zone == 523) { zone_pop[97] += scaling; }
        else if (act_zone == 530) { zone_pop[98] += scaling; }
        else if (act_zone == 531) { zone_pop[99] += scaling; }
        else if (act_zone == 532) { zone_pop[100] += scaling; }
```

```
1064        else if (act_zone == 610) { zone_pop[101] += scaling; }
1065        else if (act_zone == 611) { zone_pop[102] += scaling; }
1066        else if (act_zone == 612) { zone_pop[103] += scaling; }
1067        else if (act_zone == 613) { zone_pop[104] += scaling; }
1068        else if (act_zone == 614) { zone_pop[105] += scaling; }
1069        else if (act_zone == 620) { zone_pop[106] += scaling; }
1070        else if (act_zone == 621) { zone_pop[107] += scaling; }
1071        else if (act_zone == 622) { zone_pop[108] += scaling; }
1072        else if (act_zone == 623) { zone_pop[109] += scaling; }
1073        else if (act_zone == 630) { zone_pop[110] += scaling; }
1074        else if (act_zone == 631) { zone_pop[111] += scaling; }
1075        else if (act_zone == 632) { zone_pop[112] += scaling; }
1076        else if (act_zone == 633) { zone_pop[113] += scaling; }
1077        else if (act_zone == 634) { zone_pop[114] += scaling; }
1078        else {
1079            cout << "␣関数count_zone_population_at_this_particle_non_fixed の内容と,␣想定されるゾーン数が一致しません␣" << endl;
1080            system( "␣pause␣" );
1081        }
1082
1083        i++;
1084    }
1085
1086    N_OF_PERSON_nFX = i;
1087
1088    for (int i = 0; i < N_OF_ZONE; i++) {
1089        fprintf(POP_OF_ALLZONE_THIS_PTCL_nFX, // 以下の通り，このパーティクルにおける各ゾーンの人口を書き込む
1090            "␣%d,␣" , zone_pop[i]);
1091    }
1092
1093    for (int i = 0; i < N_OF_ZONE; i++) {
1094        fprintf(POP_OF_ALLZONE_ALLPTCL_nFX, // 以下の通り，このパーティクルにおける各ゾーンの人口を書き込む
1095            "␣%d␣␣" , zone_pop[i]);
1096    }
1097    fprintf(POP_OF_ALLZONE_ALLPTCL_nFX, "␣\n␣" );
1098
1099
1100
1101    fclose(POUT_nFX_THIS_PTCL);
1102    fclose(POP_OF_ALLZONE_THIS_PTCL_nFX);
1103
1104    return;
1105 }
1106
1107
1108 ////////////////////////
1109    // ここまで関数count_zone_population_at_this_particle_non_fixed の内容（引数とするパーティクル No.における，固定活動以外のゾーン別人口を数える関数）
1110    // ここから関数count_zone_population_fixed の内容（固定活動のゾーン別人口を数える関数）
1111 ////////////////////////
1112
1113
1114 void count_zone_population_fixed(){
1115
1116    // POUT_FX から読み込んだ値を入れる変数の宣言
```

```
1117    char indv_id[N_OF_DIGIT_OF_INDV_ID + 10]; // 個人ID
1118    int category; // カテゴリ
1119    int address_zone; // 自宅ゾーン
1120    int age; // 年齢
1121    int scaling; // 拡大係数
1122    int activity; // 活動内容
1123    int act_zone[N_OF_PARTICLE]; // 活動場所ゾーン
1124    // POP_OF_ALLZONE_FX へ書き込む値を入れる変数の宣言
1125    int zone_pop[N_OF_ZONE] = { 0 }; // 各ゾーンの人口値，0で初期化
1126
1127    int j = 0; // 固定活動の人数を数えるためのインクリメント変数
1128
1129    while ( // ファイルの読込
1130        fscanf(POUT_FX, "␣%s␣%d␣%d␣%d␣%d␣%d␣␣" , indv_id, &category, &address_zone, &age, &
              scaling, &activity) != EOF
1131        // 外側ループで個人ID，カテゴリ，自宅ゾーン，年齢，拡大係数，活動内容を読込
1132        )
1133    {
1134        int i = 0; // パーティクル数分のインクリメント変数
1135        while ( // 内側ループでパーティクル数分だけ活動場所ゾーンを読込
1136            i < N_OF_PARTICLE && fscanf(POUT_FX, "␣%d␣␣" , &act_zone[i]) != EOF // ここの条件
                  の順序(&&の前後)を入れ替えると，fscanf が先に実行されてしまう分，ポインタの位置がずれる
                  ので注意
1137            )
1138            { i += 1;}
1139        // 固定活動ではact_zone の要素は全て同じはず（PCATS を何度回そうが同じ結果だから）なので，0番
              目の要素だけを使ってゾーンごとに人口を集計
1140        // だから本当は上のwhile ループ自体，N_OF_PARTICLE 回も回す必要ないんだけど……
1141        if (act_zone[0] == 10) { zone_pop[0] += scaling; }
1142        else if (act_zone[0] == 11) { zone_pop[1] += scaling; }
1143        else if (act_zone[0] == 12) { zone_pop[2] += scaling; }
1144        else if (act_zone[0] == 13) { zone_pop[3] += scaling; }
1145        else if (act_zone[0] == 20) { zone_pop[4] += scaling; }
1146        else if (act_zone[0] == 21) { zone_pop[5] += scaling; }
1147        else if (act_zone[0] == 22) { zone_pop[6] += scaling; }
1148        else if (act_zone[0] == 23) { zone_pop[7] += scaling; }
1149        else if (act_zone[0] == 24) { zone_pop[8] += scaling; }
1150        else if (act_zone[0] == 30) { zone_pop[9] += scaling; }
1151        else if (act_zone[0] == 31) { zone_pop[10] += scaling; }
1152        else if (act_zone[0] == 32) { zone_pop[11] += scaling; }
1153        else if (act_zone[0] == 33) { zone_pop[12] += scaling; }
1154        else if (act_zone[0] == 34) { zone_pop[13] += scaling; }
1155        else if (act_zone[0] == 110) { zone_pop[14] += scaling; }
1156        else if (act_zone[0] == 111) { zone_pop[15] += scaling; }
1157        else if (act_zone[0] == 112) { zone_pop[16] += scaling; }
1158        else if (act_zone[0] == 120) { zone_pop[17] += scaling; }
1159        else if (act_zone[0] == 121) { zone_pop[18] += scaling; }
1160        else if (act_zone[0] == 122) { zone_pop[19] += scaling; }
1161        else if (act_zone[0] == 123) { zone_pop[20] += scaling; }
1162        else if (act_zone[0] == 124) { zone_pop[21] += scaling; }
1163        else if (act_zone[0] == 130) { zone_pop[22] += scaling; }
1164        else if (act_zone[0] == 131) { zone_pop[23] += scaling; }
1165        else if (act_zone[0] == 132) { zone_pop[24] += scaling; }
1166        else if (act_zone[0] == 133) { zone_pop[25] += scaling; }
1167        else if (act_zone[0] == 134) { zone_pop[26] += scaling; }
1168        else if (act_zone[0] == 135) { zone_pop[27] += scaling; }
1169        else if (act_zone[0] == 136) { zone_pop[28] += scaling; }
```

```
1170        else if (act_zone[0] == 137) { zone_pop[29] += scaling; }
1171        else if (act_zone[0] == 138) { zone_pop[30] += scaling; }
1172        else if (act_zone[0] == 210) { zone_pop[31] += scaling; }
1173        else if (act_zone[0] == 211) { zone_pop[32] += scaling; }
1174        else if (act_zone[0] == 212) { zone_pop[33] += scaling; }
1175        else if (act_zone[0] == 220) { zone_pop[34] += scaling; }
1176        else if (act_zone[0] == 221) { zone_pop[35] += scaling; }
1177        else if (act_zone[0] == 222) { zone_pop[36] += scaling; }
1178        else if (act_zone[0] == 223) { zone_pop[37] += scaling; }
1179        else if (act_zone[0] == 230) { zone_pop[38] += scaling; }
1180        else if (act_zone[0] == 231) { zone_pop[39] += scaling; }
1181        else if (act_zone[0] == 232) { zone_pop[40] += scaling; }
1182        else if (act_zone[0] == 233) { zone_pop[41] += scaling; }
1183        else if (act_zone[0] == 234) { zone_pop[42] += scaling; }
1184        else if (act_zone[0] == 235) { zone_pop[43] += scaling; }
1185        else if (act_zone[0] == 240) { zone_pop[44] += scaling; }
1186        else if (act_zone[0] == 241) { zone_pop[45] += scaling; }
1187        else if (act_zone[0] == 242) { zone_pop[46] += scaling; }
1188        else if (act_zone[0] == 243) { zone_pop[47] += scaling; }
1189        else if (act_zone[0] == 310) { zone_pop[48] += scaling; }
1190        else if (act_zone[0] == 311) { zone_pop[49] += scaling; }
1191        else if (act_zone[0] == 312) { zone_pop[50] += scaling; }
1192        else if (act_zone[0] == 320) { zone_pop[51] += scaling; }
1193        else if (act_zone[0] == 321) { zone_pop[52] += scaling; }
1194        else if (act_zone[0] == 322) { zone_pop[53] += scaling; }
1195        else if (act_zone[0] == 323) { zone_pop[54] += scaling; }
1196        else if (act_zone[0] == 330) { zone_pop[55] += scaling; }
1197        else if (act_zone[0] == 331) { zone_pop[56] += scaling; }
1198        else if (act_zone[0] == 332) { zone_pop[57] += scaling; }
1199        else if (act_zone[0] == 333) { zone_pop[58] += scaling; }
1200        else if (act_zone[0] == 334) { zone_pop[59] += scaling; }
1201        else if (act_zone[0] == 335) { zone_pop[60] += scaling; }
1202        else if (act_zone[0] == 336) { zone_pop[61] += scaling; }
1203        else if (act_zone[0] == 340) { zone_pop[62] += scaling; }
1204        else if (act_zone[0] == 341) { zone_pop[63] += scaling; }
1205        else if (act_zone[0] == 342) { zone_pop[64] += scaling; }
1206        else if (act_zone[0] == 343) { zone_pop[65] += scaling; }
1207        else if (act_zone[0] == 344) { zone_pop[66] += scaling; }
1208        else if (act_zone[0] == 345) { zone_pop[67] += scaling; }
1209        else if (act_zone[0] == 346) { zone_pop[68] += scaling; }
1210        else if (act_zone[0] == 347) { zone_pop[69] += scaling; }
1211        else if (act_zone[0] == 348) { zone_pop[70] += scaling; }
1212        else if (act_zone[0] == 349) { zone_pop[71] += scaling; }
1213        else if (act_zone[0] == 410) { zone_pop[72] += scaling; }
1214        else if (act_zone[0] == 411) { zone_pop[73] += scaling; }
1215        else if (act_zone[0] == 412) { zone_pop[74] += scaling; }
1216        else if (act_zone[0] == 420) { zone_pop[75] += scaling; }
1217        else if (act_zone[0] == 421) { zone_pop[76] += scaling; }
1218        else if (act_zone[0] == 422) { zone_pop[77] += scaling; }
1219        else if (act_zone[0] == 423) { zone_pop[78] += scaling; }
1220        else if (act_zone[0] == 424) { zone_pop[79] += scaling; }
1221        else if (act_zone[0] == 430) { zone_pop[80] += scaling; }
1222        else if (act_zone[0] == 431) { zone_pop[81] += scaling; }
1223        else if (act_zone[0] == 432) { zone_pop[82] += scaling; }
1224        else if (act_zone[0] == 433) { zone_pop[83] += scaling; }
1225        else if (act_zone[0] == 434) { zone_pop[84] += scaling; }
1226        else if (act_zone[0] == 435) { zone_pop[85] += scaling; }
```

```
1227        else if (act_zone[0] == 436) { zone_pop[86] += scaling; }
1228        else if (act_zone[0] == 437) { zone_pop[87] += scaling; }
1229        else if (act_zone[0] == 510) { zone_pop[88] += scaling; }
1230        else if (act_zone[0] == 511) { zone_pop[89] += scaling; }
1231        else if (act_zone[0] == 512) { zone_pop[90] += scaling; }
1232        else if (act_zone[0] == 513) { zone_pop[91] += scaling; }
1233        else if (act_zone[0] == 514) { zone_pop[92] += scaling; }
1234        else if (act_zone[0] == 515) { zone_pop[93] += scaling; }
1235        else if (act_zone[0] == 520) { zone_pop[94] += scaling; }
1236        else if (act_zone[0] == 521) { zone_pop[95] += scaling; }
1237        else if (act_zone[0] == 522) { zone_pop[96] += scaling; }
1238        else if (act_zone[0] == 523) { zone_pop[97] += scaling; }
1239        else if (act_zone[0] == 530) { zone_pop[98] += scaling; }
1240        else if (act_zone[0] == 531) { zone_pop[99] += scaling; }
1241        else if (act_zone[0] == 532) { zone_pop[100] += scaling; }
1242        else if (act_zone[0] == 610) { zone_pop[101] += scaling; }
1243        else if (act_zone[0] == 611) { zone_pop[102] += scaling; }
1244        else if (act_zone[0] == 612) { zone_pop[103] += scaling; }
1245        else if (act_zone[0] == 613) { zone_pop[104] += scaling; }
1246        else if (act_zone[0] == 614) { zone_pop[105] += scaling; }
1247        else if (act_zone[0] == 620) { zone_pop[106] += scaling; }
1248        else if (act_zone[0] == 621) { zone_pop[107] += scaling; }
1249        else if (act_zone[0] == 622) { zone_pop[108] += scaling; }
1250        else if (act_zone[0] == 623) { zone_pop[109] += scaling; }
1251        else if (act_zone[0] == 630) { zone_pop[110] += scaling; }
1252        else if (act_zone[0] == 631) { zone_pop[111] += scaling; }
1253        else if (act_zone[0] == 632) { zone_pop[112] += scaling; }
1254        else if (act_zone[0] == 633) { zone_pop[113] += scaling; }
1255        else if (act_zone[0] == 634) { zone_pop[114] += scaling; }
1256        else {
1257            cout << "␣関数count_zone_population_fixed の内容と,␣想定されるゾーン数が一致しません␣
                  " << endl;
1258            system( "␣pause␣" );
1259        }
1260
1261        j++;
1262    }
1263
1264    N_OF_PERSON_FX = j;
1265
1266    for (int i = 0; i < N_OF_ZONE; i++) { // 各ゾーンの人口を書き込む
1267        fprintf(POP_OF_ALLZONE_FX, "␣%d,␣" , zone_pop[i]);
1268    }
1269
1270    fclose(POUT_FX);
1271    fseek(POP_OF_ALLZONE_FX, 0, SEEK_SET); // POP_OF_ALLZONE_FX は読書両方のモード w+なの
          で，書き込みが終わった今，次の読込に備えてポインタを先頭にしておく
1272
1273    return;
1274 }
1275
1276
1277 ////////////////////////
1278     // ここまで関数count_zone_population_fixed の内容（固定活動のゾーン別人口を数える関数）
1279     // ここから関数count_trip の内容（IC カードデータとの比較のために，各パーティクルにおける
          OD 別トリップ数を数える関数）
1280 ////////////////////////
```

```
1281
1282
1283
1284 void count_trip() {
1285     // 全パーティクル分の，OD 別トリップ数を格納する 3 次元ベクトルの宣言
1286     vector<vector<vector<int>>> n_of_trip_by_od_all_ptcl(N_OF_PARTICLE, vector<vector<int>>(
           N_OF_ZONE * N_OF_ZONE, vector<int>(3, 0)));
1287     // n_of_trip_by_od_all_ptcl[i][j][k]とすると，
1288     // i：(0から数えた)パーティクル番号
1289     // j：OD 組合せの通し番号
1290     // k：カラムの列番号(0：O ゾーン，1：D ゾーン，2：推定結果の拡大係数の合計)
1291     // 各パーティクルについて，中身の 2次元ベクトルは，以下の通り
1292     // 要素数はゾーン数の 2乗，全ての要素の初期値は【「要素数が 3，全ての要素が 0で初期化」というベクトル】
1293     // 要するに，((O ゾーン，D ゾーン，推定結果の拡大係数の合計), (…), (…), …ゾーン数の 2乗分…, (…))と
           いう行列
1294
1295     // 観測データの，OD 別トリップ数を格納する 2 次元ベクトルの宣言
1296     vector<vector<int>> n_of_trip_by_od_obs(N_OF_ZONE * N_OF_ZONE, vector<int>(3, 0));
1297     // n_of_trip_by_od_obs[i][j]とすると，
1298     // i：OD 組合せの通し番号
1299     // j：カラムの列番号(0：O ゾーン，1：D ゾーン，2：観測データのトリップ数)
1300     // 要素数はゾーン数の 2乗，全ての要素の初期値は【「要素数が 3，全ての要素が 0で初期化」というベクトル】
1301     // 要するに，((O ゾーン，D ゾーン，観測データのトリップ数), (…), (…), …ゾーン数の 2乗分…, (…))とい
           う行列
1302
1303     // ZONECODE の読込，ゾーン番号の代入
1304     cout << "␣ゾーンコードファイルの読込開始␣" << endl;
1305     for (int i = 0; i < N_OF_PARTICLE; i++) {
1306         int j = 0;
1307         while (
1308             j < N_OF_ZONE * N_OF_ZONE && fscanf(ZONECODE, "␣%d,%d\n␣" , &
                  n_of_trip_by_od_all_ptcl[i][j][0], &n_of_trip_by_od_all_ptcl[i][j][1]) != EOF
1309             )
1310         {
1311             j++;
1312         }
1313         fseek(ZONECODE, 0, SEEK_SET);
1314     }
1315
1316     int ii = 0;
1317     while (
1318         ii < N_OF_ZONE * N_OF_ZONE && fscanf(ZONECODE, "␣%d,%d\n␣" , &n_of_trip_by_od_obs[
              ii][0], &n_of_trip_by_od_obs[ii][1]) != EOF
1319         )
1320     {
1321         ii++;
1322     }
1323     fclose(ZONECODE);
1324
1325     // IC_OBS の読込
1326     cout << "␣IC カード観測データファイルの読込開始␣" << endl;
1327     char header[200]; // 読み込むだけで使わない変数
1328     fgets(header, 200, IC_OBS); // ヘッダがあるファイルなので，ヘッダを読み込むだけ
1329
1330     // IC_OBS から読み取った値を入れる変数の宣言
1331     char indv_id_obs[N_OF_DIGIT_OF_INDV_ID + 10]; // 個人ID：配列の大きさは個人ID の桁数に依存
1332     int scaling_obs;
```

```
1333    int o_time_obs;
1334    int d_time_obs;
1335    int o_zone_obs;
1336    int d_zone_obs;
1337    int method_obs;
1338
1339    int i = 0;
1340
1341        while (
1342            fscanf(IC_OBS, "␣%[^,],%d,%d,%d,%d,%d,%d\n␣" , indv_id_obs, &scaling_obs, &
                  o_time_obs, &d_time_obs, &o_zone_obs, &d_zone_obs, &method_obs) != EOF
1343            )
1344        {
1345            if (d_time_obs > IC_START && d_time_obs <= IC_END) { // トリップの到着時刻が検証対象
                  内ならば
1346
1347              for (int i = 0; i < N_OF_ZONE * N_OF_ZONE; i++) { //
                    n_of_trip_by_od_obs にある OD 組合せを順に見ていき，
                    OD が対応するところで観測データの数を加算
1348                  if (o_zone_obs == n_of_trip_by_od_obs[i][0] && d_zone_obs ==
                        n_of_trip_by_od_obs[i][1]) {
1349                    n_of_trip_by_od_obs[i][2] ++; // もし観測データを拡大するときは，ここで拡
                          大係数を加える
1350                    break; // OD 組合せはユニークなので，一度見つけたらその先は見る必要が無い
1351                }
1352
1353              }
1354              // break;の抜け先はここ
1355          }
1356
1357          i++;
1358
1359          if (i == 3000) { cout << "␣3000行目処理完了␣" << endl; }
1360          if (i == 6000) { cout << "␣6000行目処理完了␣" << endl; }
1361          if (i == 9000) { cout << "␣9000行目処理完了␣" << endl; }
1362          if (i == 12000) { cout << "␣1万 2000行目処理完了␣" << endl; }
1363
1364      }
1365  fclose(IC_OBS);
1366
1367  cout << "␣トリップ数カウントのためのPCATS 出力結果ファイルの読込開始␣" << endl;
1368
1369
1370  // ORIGINAL_PCATS_OUTPUT から読み込んだ値を入れる変数の宣言
1371  char indv_id[N_OF_DIGIT_OF_INDV_ID + 10]; // 個人ID：配列の大きさは個人ID の桁数に依存
1372  int col[N_OF_PCATS_PARAMETER]; // 整数が入っているカラムをまとめて配列にしたもの
1373                                 // 0：カテゴリ，1：自宅ゾーン，2：就業就学地ゾーン，3：性別，4：年
                                      齢，5：職業，6：免許有無，7：世帯内自動車台数，
1374                                 // 8：拡大係数，9：世帯人数，10：活動内容，11：活動開始時刻，12：
                                      活動終了時刻，13：活動場所ゾーン，14：活動施設，
1375                                 // 15：固定活動ダミー，16：活動後移動有無，17：移動開始時刻，18：
                                      移動終了時刻，19：移動手段，20：車ゾーン，21：車場所，22：中断
                                      ダミー
1376  char suspend_act[8]; // 中断時の活動：” MOVING ” の 6文字が最長
1377                       // 読み込んだ値から抽出・作成する変数の宣言
1378  char indv_id_without_ptcl_num[N_OF_DIGIT_OF_INDV_ID + 10]; // 個人
        ID には末尾にパーティクル番号が入っているため，純粋な個人ID 部分のみを格納する配列を用意
```

```
1379    int ptcl_no; // (1から数えた)パーティクル番号：個人ID の末尾に記載
1380    int o_time;
1381    int d_time;
1382    int o_zone;
1383    int method;
1384    int trip_flag = 0; // 0：トリップあり，1：トリップなし
1385    int j = 0; // インクリメント変数
1386
1387
1388    while ( // PCATS 出力結果ファイルの読込
1389        fscanf(ORIGINAL_PCATS_OUTPUT, "␣%s␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%s␣" ,
1390            indv_id, &col[0], &col[1], &col[2], &col[3], &col[4], &col[5], &col[6], &col[7], &col[8], &col[9], &col[10],
1391            &col[11], &col[12], &col[13], &col[14], &col[15], &col[16], &col[17], &col[18], &col[19], &col[20], &col[21], &col[22], suspend_act) != EOF
1392        )
1393    {
1394        // パーティクル番号の値取得
1395        char temp[N_OF_DIGIT_OF_PTCL_NO + 3]; // パーティクル番号は，まずは文字列として読み取る必要があるので，一時的にここに格納するために宣言
1396        strncpy(temp, indv_id + N_OF_DIGIT_OF_INDV_ID, N_OF_DIGIT_OF_PTCL_NO); // indv_id の頭から個人 ID の桁数分だけ後ろ（＝パーティクル番号の頭）から，パーティクル番号をコピーして格納
1397        temp[N_OF_DIGIT_OF_PTCL_NO] = '\0'; // 末尾にヌル文字を記録
1398        ptcl_no = atoi(temp); // temp に格納されていた文字列としての（1 から数えた）パーティクル番号を，ptcl_no に int として格納
1399
1400
1401        // 純粋個人ID の値取得
1402        strncpy(indv_id_without_ptcl_num, indv_id, N_OF_DIGIT_OF_INDV_ID); // パーティクル番号を除いた，純粋な個人ID の桁数分だけコピーして格納
1403        indv_id_without_ptcl_num[N_OF_DIGIT_OF_INDV_ID] = '\0'; // 末尾にヌル文字を記録
1404
1405        if (trip_flag == 1) { // 1つ前の行が「活動後トリップあり」ならば，
1406            if (d_time > IC_START && d_time <= IC_END && method == 1) { // トリップの到着時刻が検証対象内で，かつ公共交通による移動ならば，
1407                for (int i = 0; i < N_OF_ZONE * N_OF_ZONE; i++) { // n_of_trip_by_od_all_ptcl にある OD 組合せを順に見ていき，OD が対応するところでトリップ情報を n_of_trip_by_od_all_ptcl に書き込む
1408                    if (o_zone == n_of_trip_by_od_all_ptcl[ptcl_no - 1][i][0] && col[13] == n_of_trip_by_od_all_ptcl[ptcl_no - 1][i][1]) { // 到着ゾーンは，この行の活動ゾーンに等しい
1409                        n_of_trip_by_od_all_ptcl[ptcl_no - 1][i][2] += col[8];
1410                        break; // OD 組合せはユニークなので，一度見つけたらその先は見る必要が無い
1411                    }
1412
1413                }
1414                // break;の抜け先はここ
1415            }
1416            trip_flag = 0; // フラグを 0に戻す
1417        }
1418
1419        if (col[16] == 1) { // この行が「活動後トリップあり」ならば，トリップの発地と時刻と手段を記憶し，フラグを 1にする
1420            o_time = col[17];
```

```
1421            d_time = col[18];
1422            o_zone = col[13];
1423            method = col[19];
1424            trip_flag = 1;
1425        }
1426
1427        j++;
1428
1429        if (j == 200000) { cout << "␣20万行目処理完了␣" << endl; }
1430        if (j == 400000) { cout << "␣40万行目処理完了␣" << endl; }
1431        if (j == 600000) { cout << "␣60万行目処理完了␣" << endl; }
1432        if (j == 800000) { cout << "␣80万行目処理完了␣" << endl; }
1433    }
1434    fseek(ORIGINAL_PCATS_OUTPUT, 0, SEEK_SET); //
          ORIGINAL_PCATS_OUTPUT は PCATS 中断明けデータを作る時にも使うので，次の読込に備えてポ
          インタを先頭にしておく
1435    // PCATS 出力結果ファイルの読込終了
1436
1437    cout << "␣OD 別トリップ数集計完了，␣ゾーン別発着別トリップ数集計開始␣" << endl;
1438
1439
1440    // 全パーティクル分の，ゾーン別発着別トリップ数を格納する 3次元ベクトルの宣言
1441    vector<vector<vector<int>>> n_of_trip_by_zone_all_ptcl(N_OF_PARTICLE, vector<vector<int
          >>(N_OF_ZONE * 2, vector<int>(2, 0)));
1442    // n_of_trip_by_od_all_ptcl[i][j][k]とすると，
1443    // i：(0から数えた)パーティクル番号
1444    // j：ゾーンの通し番号(発着別なので注意)
1445    // k：カラムの列番号(0：ゾーン番号，1：推定結果の拡大係数の合計)
1446    // 各パーティクルについて，中身の 2次元ベクトルは，以下の通り
1447    // 要素数はゾーン数の 2倍，全ての要素の初期値は【「要素数が 2，全ての要素が 0で初期化」というベクトル】
1448    // 要するに，((ゾーン番号，推定結果の拡大係数の合計), (…), (…), …ゾーン数の 2倍分…, (…))という行列
1449
1450    for (int k = 0; k < N_OF_PARTICLE; k++) {
1451        for (int i = 0; i < N_OF_ZONE; i++) { // n_of_trip_by_od_all_ptcl にある OD 組合せのうち，
              D ゾーン（即ち，ゾーン番号が順に入っている)を見ていく
1452            n_of_trip_by_zone_all_ptcl[k][2 * i][0] = n_of_trip_by_od_all_ptcl[k][i][1]; //
                  今見ているゾーン番号(i 行目）を，n_of_trip_by_zone に 2 行書き込む
1453            n_of_trip_by_zone_all_ptcl[k][2 * i + 1][0] = n_of_trip_by_od_all_ptcl[k][i
                  ][1];
1454            for (int j = 0; j < N_OF_ZONE * N_OF_ZONE; j++) { //
                  n_of_trip_by_od_all_ptcl ベクトルを上から順に見ていく
1455                if (n_of_trip_by_od_all_ptcl[k][j][0] == n_of_trip_by_od_all_ptcl[k][i][1]) {
                        // j 行目の O ゾーンが今見ているゾーン番号（i 行目）ならば
1456                    n_of_trip_by_zone_all_ptcl[k][2 * i][1] += n_of_trip_by_od_all_ptcl[k][j
                          ][2]; // 今見ているゾーン番号(i 行目）を O ゾーンとするトリップ数の推定値を集計
1457                }
1458                if (n_of_trip_by_od_all_ptcl[k][j][1] == n_of_trip_by_od_all_ptcl[k][i][1]) {
                        // j 行目の D ゾーンが今見ているゾーン番号（i 行目）ならば
1459                    n_of_trip_by_zone_all_ptcl[k][2 * i + 1][1] += n_of_trip_by_od_all_ptcl[
                          k][j][2]; // 今見ているゾーン番号(
                          i 行目）を D ゾーンとするトリップ数の推定値を集計
1460                }
1461            }
1462        }
1463    }
1464
1465
```

```
1466    // 観測データの，ゾーン別発着別トリップ数を格納する 2次元ベクトルの宣言
1467    vector<vector<int>> n_of_trip_by_zone_obs(N_OF_ZONE * N_OF_ZONE, vector<int>(2, 0));
1468    // n_of_trip_by_od_obs[i][j]とすると，
1469    // i：OD 組合せの通し番号
1470    // j：カラムの列番号(0：ゾーン番号，1：観測データのトリップ数)
1471    // 要素数はゾーン数の 2倍，全ての要素の初期値は【「要素数が 2，全ての要素が 0で初期化」というベクトル】
1472    // 要するに，((ゾーン番号，観測データのトリップ数), (…), (…), …ゾーン数の 2倍分…, (…))という行列
1473
1474    for (int i = 0; i < N_OF_ZONE; i++) { // n_of_trip_by_od_obs にある OD 組合せのうち，
          D ゾーン（即ち，ゾーン番号が順に入っている）を見ていく
1475        n_of_trip_by_zone_obs[2 * i][0] = n_of_trip_by_od_obs[i][1]; // 今見ているゾーン番号(
              i 行目）を，n_of_trip_by_zone に 2 行書き込む
1476        n_of_trip_by_zone_obs[2 * i + 1][0] = n_of_trip_by_od_obs[i][1];
1477        for (int j = 0; j < N_OF_ZONE * N_OF_ZONE; j++) { //
              n_of_trip_by_od_obs ベクトルを上から順に見ていく
1478            if (n_of_trip_by_od_obs[j][0] == n_of_trip_by_od_obs[i][1]) { //
                  j 行目の O ゾーンが今見ているゾーン番号（i 行目）ならば
1479                n_of_trip_by_zone_obs[2 * i][1] += n_of_trip_by_od_obs[j][2]; // 今見ているゾ
                      ーン番号(i 行目）を O ゾーンとするトリップ数の推定値を集計
1480            }
1481            if (n_of_trip_by_od_obs[j][1] == n_of_trip_by_od_obs[i][1]) { //
                  j 行目の D ゾーンが今見ているゾーン番号（i 行目）ならば
1482                n_of_trip_by_zone_obs[2 * i + 1][1] += n_of_trip_by_od_obs[j][2]; // 今見てい
                      るゾーン番号(i 行目）を D ゾーンとするトリップ数の推定値を集計
1483            }
1484        }
1485    }
1486
1487    cout << "␣ゾーン別発着別トリップ数集計完了，␣ファイル書き出し開始␣" << endl;
1488
1489    for (int i = 0; i < N_OF_PARTICLE; i++) {
1490        // ファイルの作成：N_OF_TRIP_BY_OD_THIS_PTCL：各OD 組合せごとに，トリップ数の
              PCATS での推定値と観測値を OD 組合せ数分の行数だけ並べたファイル
1491        // パーティクルNo.に関するループ内なので，file_set_up 内では上手く書けなかったのでこちらに記述
1492        // 上手く書く方法があれば，file_set_up 内に書いた方が，入出力ファイルのパスやファイル名を変更する
              際に見易いのでそうしたいが……
1493        sprintf(
1494            n_of_trip_by_od,
1495             "␣C:\\cpp\\filtering\\filtering\\output\\n_of_trip\\%d-%d\\n_of_trip_by_od_%d
                  -%d(No.%d).csv␣" , IC_START, IC_END, IC_START, IC_END, i + 1
1496            );
1497        N_OF_TRIP_BY_OD_THIS_PTCL = fopen(n_of_trip_by_od, "␣w␣" );
1498        if (N_OF_TRIP_BY_OD_THIS_PTCL == NULL) {
1499            cout << "␣ファイルN_OF_TRIP_BY_OD_THIS_PTCL（各 OD 組合せごとに，␣トリップ数の
                      PCATS での推定値と観測値を OD 組合せ数分の行数だけ並べたファイル）の作成失敗␣" << endl;
1500            system( "␣pause␣" );
1501        }
1502
1503        // ファイルの作成：N_OF_TRIP_BY_ZONE_THIS_PTCL：各ゾーン発着別に，トリップ数の
              PCATS での推定値と観測値をゾーン数の 2 倍の行数だけ並べたファイル
1504        // パーティクルNo.に関するループ内なので，file_set_up 内では上手く書けなかったのでこちらに記述
1505        // 上手く書く方法があれば，file_set_up 内に書いた方が，入出力ファイルのパスやファイル名を変更する
              際に見易いのでそうしたいが……
1506        sprintf(
1507            n_of_trip_by_zone,
1508             "␣C:\\cpp\\filtering\\filtering\\output\\n_of_trip\\%d-%d\\n_of_trip_by_zone_%
                  d-%d(No.%d).csv␣" , IC_START, IC_END, IC_START, IC_END, i + 1
```

```
1509            );
1510        N_OF_TRIP_BY_ZONE_THIS_PTCL = fopen(n_of_trip_by_zone, "␣w␣" );
1511        if (N_OF_TRIP_BY_ZONE_THIS_PTCL == NULL) {
1512            cout << "␣ファイルN_OF_TRIP_BY_ZONE_THIS_PTCL（各ゾーン発着別に,␣トリップ数の
                  PCATS での推定値と観測値をゾーン数の 2 倍の行数だけ並べたファイル）の作成失敗␣" << endl
                  ;
1513            system( "␣pause␣" );
1514        }
1515
1516        // ヘッダの書き込み
1517        fprintf(N_OF_TRIP_BY_OD_THIS_PTCL, "␣o_zone,d_zone,PCATS,obs_IC\n␣" );
1518        fprintf(N_OF_TRIP_BY_ZONE_THIS_PTCL, "␣zone,O_or_D,PCATS,obs_IC\n␣" );
1519
1520        // 結果の書き込み
1521        for (int j = 0; j < N_OF_ZONE * N_OF_ZONE; j++) {
1522            fprintf(N_OF_TRIP_BY_OD_THIS_PTCL, "␣%d,%d,%d,%d\n␣" ,
1523                n_of_trip_by_od_all_ptcl[i][j][0], n_of_trip_by_od_all_ptcl[i][j][1],
                      n_of_trip_by_od_all_ptcl[i][j][2], n_of_trip_by_od_obs[j][2]);
1524        }
1525
1526        for (int j = 0; j < N_OF_ZONE; j++) {
1527            fprintf(N_OF_TRIP_BY_ZONE_THIS_PTCL, "␣%d,O,%d,%d\n%d,D,%d,%d\n␣" ,
1528                n_of_trip_by_zone_all_ptcl[i][2 * j][0], n_of_trip_by_zone_all_ptcl[i][2 * j
                      ][1], n_of_trip_by_zone_obs[2 * j][1],
1529                n_of_trip_by_zone_all_ptcl[i][2 * j + 1][0], n_of_trip_by_zone_all_ptcl[i][2
                       * j + 1][1], n_of_trip_by_zone_obs[2 * j + 1][1]);
1530        }
1531
1532        fclose(N_OF_TRIP_BY_OD_THIS_PTCL);
1533        fclose(N_OF_TRIP_BY_ZONE_THIS_PTCL);
1534    }
1535
1536    return;
1537 }
1538
1539
1540  ////////////////////////
1541  // ここまで関数count_trip_at_this_particle の内容（IC カードデータとの比較のために，各パーティクルにお
      けるOD 別トリップ数を数える関数）
1542  // ここから関数
      calculate_weight_of_mobile の内容（引数とするベクトル（要素数はパーティクル数）の各要素に対し，モバ
      イル空間統計による各パーティクルの重みの値を代入する関数）
1543  ////////////////////////
1544
1545 void calculate_weight_of_mobile(vector<double> &weight_of_mobile, vector<double> &
      normalized_weight_of_mobile) {
1546    cout << "␣モバイル空間統計によるパーティクルの重み計算開始␣" << endl;
1547
1548    // POP_OF_ALLZONE_ALLPTCL_nFX から読み込んだ値を入れる 2 次元ベクトルの宣言
1549    vector<vector<int>> pop_of_all_zone_all_particle_non_fixed(N_OF_PARTICLE, vector<int>(
        N_OF_ZONE, 0));
1550    // 2次元ベクトルの宣言：要素数はパーティクル数分，全ての要素の初期値は【「要素数がゾーン数分，全ての
        要素が 0で初期化」というベクトル】
1551    // 要するに，((…ゾーン数分の要素…), (…), (…), …パーティクル数分…, (…))という行列
1552
1553    // ファイルの読込
1554    for (int i = 0; i < N_OF_PARTICLE; i++) { // パーティクル番号ループ
```

```
1555        int j = 0; // ゾーン番号初期化
1556        while (j < N_OF_ZONE && fscanf(POP_OF_ALLZONE_ALLPTCL_nFX, "␣%d␣␣" , &
              pop_of_all_zone_all_particle_non_fixed[i][j]) != EOF) { // ゾーン番号ループ，読み取った値の格納
1557 // ここの条件の順序(&&の前後)を入れ替えると，fscanf が先に実行されてしまう分，ポインタの位置がずれるので注意
1558            j += 1;
1559        }
1560    }
1561    fclose(POP_OF_ALLZONE_ALLPTCL_nFX);
1562
1563    // POP_OF_ALLZONE_OBS から読み込んだ値を入れる配列の宣言
1564    int zone_pop_obs[N_OF_ZONE] = { 0 }; // 要素は 0で初期化
1565
1566    for (int i = 0; i < N_OF_ZONE; i++) { // ファイルの読込
1567        fscanf(POP_OF_ALLZONE_OBS, "␣%d,␣␣" , &zone_pop_obs[i]);
1568    }
1569    fclose(POP_OF_ALLZONE_OBS);
1570
1571    // POP_OF_ALLZONE_FX から読み込んだ値を入れる配列の宣言
1572    int zone_pop_fixed[N_OF_ZONE] = { 0 }; // 要素は 0で初期化
1573
1574    for (int i = 0; i < N_OF_ZONE; i++) { // 要素は 0で初期化
1575        fscanf(POP_OF_ALLZONE_FX, "␣%d,␣␣" , &zone_pop_fixed[i]);
1576    }
1577    fclose(POP_OF_ALLZONE_FX);
1578
1579    // 観測値から固定活動を引いたものをフィルタリング対象とする
1580    int zone_pop_obs_minus_fixed[N_OF_ZONE] = { 0 }; // フィルタリング対象とする配列の宣言，初期化
1581
1582    for (int i = 0; i < N_OF_ZONE; i++) {
1583        zone_pop_obs_minus_fixed[i] = zone_pop_obs[i] - zone_pop_fixed[i];
1584    }
1585
1586    // パーティクルの重みを計算するための変数の宣言
1587    double pp1 = 0.0;
1588    double pp2 = 0.0;
1589
1590    // パーティクルの重みの計算(ベクトルどうしの重み付きユークリッド距離の平方の逆数で類似度として定義)
1591    // 重み付きユークリッド距離の「重み」(ベクトルの各要素の重み)と，パーティクルの「重み」は，意味が違うので注意すること
1592    for (int i = 0; i < N_OF_PARTICLE; i++) { // パーティクル番号ループ
1593        double sum = 0.0; // 変数リセット
1594        for (int j = 0; j < N_OF_ZONE; j++) { // ゾーン番号ループ
1595            pp1 = pow((pop_of_all_zone_all_particle_non_fixed[i][j] -
                  zone_pop_obs_minus_fixed[j]), 2.0); // フィルタリング対象ベクトルと観測ベクトルの要素の差の 2乗
1596            pp2 = pp1 / pow(zone_pop_obs_minus_fixed[j], 2.0); // 各要素の重みを，要素の差の 2乗に掛けたもの
1597            sum += pp2; // pp2 の総和：両ベクトルの重み付きユークリッド距離の平方
1598        }
1599        if (sum != 0) {
1600            weight_of_mobile[i] = 1.0 / sum; // パーティクルの重みは，ベクトルどうしの重み付きユークリッド距離の平方の逆数で定義
1601        }
```

```
1602        else {
1603            cout << "␣パーティクル番号(1から数えたもの)␣" << i + 1 << "␣のモバイル空間統計同化において,␣フィルタリング対象ベクトルと␣"
1604                "␣観測ベクトルが完全に一致したため,␣パーティクルの重みを定義不可能␣" << endl;
1605            system( "␣pause␣" );
1606        }
1607    }
1608
1609    // パーティクルの重みの正規化
1610    double sum_w = 0.0; // 計算用の変数の宣言
1611
1612    for (int i = 0; i < N_OF_PARTICLE; i++) {
1613        sum_w += weight_of_mobile[i]; // sum_w：パーティクルの重みの総和
1614    }
1615    for (int j = 0; j < N_OF_PARTICLE; j++) {
1616        normalized_weight_of_mobile[j] = weight_of_mobile[j] / sum_w; // 各パーティクルの重みを総和で割ったものが正規化重み
1617    }
1618
1619    // ファイルへの書き込み
1620    for (int i = 0; i < N_OF_PARTICLE; i++) {
1621        fprintf(WEIGHT_OF_MOBILE, "␣%.12lf,%.12lf\n␣" , weight_of_mobile[i],
1622          normalized_weight_of_mobile[i]); // 重みは小数点以下 12桁表示
1623    }
1624    fclose(WEIGHT_OF_MOBILE);
1625
1626    cout << "␣モバイル空間統計によるパーティクルの重み計算完了␣" << endl;
1627
1628    return;
1629 }
1630
1631  ////////////////////////
1632  // ここまで関数
       calculate_weight_of_mobile の内容（引数とするベクトル（要素数はパーティクル数）の各要素に対し，モバイル空間統計による各パーティクルの重みの値を代入する関数）
1633  // ここから関数
       calculate_weight_of_ic の内容（引数とするベクトル（要素数はパーティクル数）の各要素に対し，
       IC カードデータによる各パーティクルの重みの値を代入する関数）
1634  ////////////////////////
1635
1636 void calculate_weight_of_ic(vector<double> &weight_of_ic, vector<double> &
       normalized_weight_of_ic) {
1637    cout << "␣IC カードデータによるパーティクルの重み計算開始␣" << endl;
1638
1639    if (IC_FLAG == 1) { // 尤度算出方法が，OD 別の場合
1640        // パーティクルの重みを計算するための変数の宣言
1641        double pp1 = 0.0;
1642        double pp2 = 0.0;
1643
1644        for (int i = 0; i < N_OF_PARTICLE; i++) {
1645            // ファイルの読込：N_OF_TRIP_BY_OD_THIS_PTCL：各OD 別に，トリップ数の
                 PCATS での推定値と観測値をゾーン数の 2 乗の行数だけ並べたファイル
1646            // パーティクルNo.に関するループ内なので，
                 file_set_up 内では上手く書けなかったのでこちらに記述
1647            // 上手く書く方法があれば，file_set_up 内に書いた方が，入出力ファイルのパスやファイル名を変更する際に見易いのでそうしたいが……
1648            sprintf(
```

```
1648                n_of_trip_by_od,
1649                 "␣C:\\cpp\\filtering\\filtering\\output\\n_of_trip\\%d-%d\\n_of_trip_by_od_
                      %d-%d(No.%d).csv␣" , IC_START, IC_END, IC_START, IC_END, i + 1
1650                );
1651            N_OF_TRIP_BY_OD_THIS_PTCL = fopen(n_of_trip_by_od, "␣r␣" ); // ここを ” w ” や ” w
                  + ” にすると，プログラム前半で作った内容が消えて上書きされるので注意
1652            if (N_OF_TRIP_BY_OD_THIS_PTCL == NULL) {
1653                cout << "␣ファイルN_OF_TRIP_BY_OD_THIS_PTCL（各 OD 別に,␣トリップ数の
                      PCATS での推定値と観測値をゾーン数の 2 乗の行数だけ並べたファイル）の読込失敗␣" <<
                      endl;
1654                system( "␣pause␣" );
1655            }
1656
1657            // ファイルの読込
1658            char header[200]; // 読み込むだけで使わない変数
1659            fgets(header, 200, N_OF_TRIP_BY_OD_THIS_PTCL); // ヘッダがあるファイルなので，ヘッダ
                  を読み込むだけ
1660
1661            int o_zone; // 読み込むだけで使わない
1662            int d_zone; // 読み込むだけで使わない
1663            int n_of_trip_PCATS;
1664            int n_of_trip_obs;
1665
1666            double sum = 0.0; // 変数リセット
1667
1668            while (fscanf(N_OF_TRIP_BY_OD_THIS_PTCL, "␣%d,%d,%d,%d\n␣" , &o_zone, &d_zone, &
                  n_of_trip_PCATS, &n_of_trip_obs) != EOF) { // ゾーン別発着別ループ
1669                // パーティクルの重みの計算(ベクトルどうしの重み付きユークリッド距離の平方の逆数で類似
                      度として定義)
1670                // 重み付きユークリッド距離の「重み」(ベクトルの各要素の重み)と，パーティクルの「重み」は，
                      意味が違うので注意すること
1671                pp1 = pow((n_of_trip_PCATS - n_of_trip_obs), 2.0); // フィルタリング対象ベクト
                      ルと観測ベクトルの要素の差の 2乗
1672                if (n_of_trip_obs != 0) {
1673                    pp2 = pp1 / pow(n_of_trip_obs, 2.0); // 各要素の重みを，要素の差の 2乗に掛けた
                          もの
1674                }
1675                else {
1676                    pp2 = 0; // 観測データが 0件だった要素の重み(ここは本当は要検討)
1677                }
1678                sum += pp2; // pp2 の総和：両ベクトルの重み付きユークリッド距離の平方
1679            }
1680            if (sum != 0) {
1681                weight_of_ic[i] = 1.0 / sum; // パーティクルの重みは，ベクトルどうしの重み付きユー
                      クリッド距離の平方の逆数で定義
1682            }
1683            else {
1684                cout << "␣パーティクル番号(1から数えたもの)␣" << i + 1 << "␣の
                      IC カードデータ同化において,␣フィルタリング対象ベクトルと␣"
1685                     "␣観測ベクトルが完全に一致したため,␣パーティクルの重みを定義不可能␣" << endl;
1686                system( "␣pause␣" );
1687            }
1688            fclose(N_OF_TRIP_BY_OD_THIS_PTCL);
1689        }
1690
1691
1692    }
```

```
1693    else if (IC_FLAG == 2) { // 尤度算出方法が，ゾーン別発着別の場合
1694        // パーティクルの重みを計算するための変数の宣言
1695        double pp1 = 0.0;
1696        double pp2 = 0.0;
1697
1698        for (int i = 0; i < N_OF_PARTICLE; i++) {
1699            // ファイルの読込：N_OF_TRIP_BY_ZONE_THIS_PTCL：各ゾーン発着別に，トリップ数の
                  PCATS での推定値と観測値をゾーン数の 2 倍の行数だけ並べたファイル
1700            // パーティクルNo.に関するループ内なので，
                  file_set_up 内では上手く書けなかったのでこちらに記述
1701            // 上手く書く方法があれば，file_set_up 内に書いた方が，入出力ファイルのパスやファイル名を変
                  更する際に見易いのでそうしたいが……
1702            sprintf(
1703                n_of_trip_by_zone,
1704                 "␣C:\\cpp\\filtering\\filtering\\output\\n_of_trip\\%d-%d\\
                      n_of_trip_by_zone_%d-%d(No.%d).csv␣" , IC_START, IC_END, IC_START, IC_END
                      , i + 1
1705            );
1706            N_OF_TRIP_BY_ZONE_THIS_PTCL = fopen(n_of_trip_by_zone, "␣r␣" ); // ここを ” w ” や
                  ” w+ ” にすると，プログラム前半で作った内容が消えて上書きされるので注意
1707            if (N_OF_TRIP_BY_ZONE_THIS_PTCL == NULL) {
1708                cout << "␣ファイルN_OF_TRIP_BY_ZONE_THIS_PTCL（各ゾーン発着別に,␣トリップ数の
                      PCATS での推定値と観測値をゾーン数の 2 倍の行数だけ並べたファイル）の読込失敗␣" <<
                      endl;
1709                system( "␣pause␣" );
1710            }
1711
1712            // ファイルの読込
1713            char header[200]; // 読み込むだけで使わない変数
1714            fgets(header, 200, N_OF_TRIP_BY_ZONE_THIS_PTCL); // ヘッダがあるファイルなので，ヘッ
                  ダを読み込むだけ
1715
1716            int zone; // 読み込むだけで使わない
1717            char o_or_d[3]; // 読み込むだけで使わない
1718            int n_of_trip_PCATS;
1719            int n_of_trip_obs;
1720
1721            double sum = 0.0; // 変数リセット
1722
1723            while (fscanf(N_OF_TRIP_BY_ZONE_THIS_PTCL, "␣%d,%[^,],%d,%d\n␣" , &zone, o_or_d,
                      &n_of_trip_PCATS, &n_of_trip_obs) != EOF) { // ゾーン別発着別ループ
1724                // パーティクルの重みの計算(ベクトルどうしの重み付きユークリッド距離の平方の逆数で類似
                      度として定義)
1725                // 重み付きユークリッド距離の「重み」(ベクトルの各要素の重み)と，パーティクルの「重み」は，
                      意味が違うので注意すること
1726                pp1 = pow((n_of_trip_PCATS - n_of_trip_obs), 2.0); // フィルタリング対象ベクト
                      ルと観測ベクトルの要素の差の 2乗
1727                if (n_of_trip_obs != 0) {
1728                    pp2 = pp1 / pow(n_of_trip_obs, 2.0); // 各要素の重みを，要素の差の 2乗に掛けた
                          もの
1729                }
1730                else {
1731                    pp2 = 0; // 観測データが 0件だった要素の重み(ここは本当は要検討)
1732                }
1733                sum += pp2; // pp2 の総和：両ベクトルの重み付きユークリッド距離の平方
1734            }
1735            if (sum != 0) {
```

```
1736                weight_of_ic[i] = 1.0 / sum; // パーティクルの重みは，ベクトルどうしの重み付きユークリッド距離の平方の逆数で定義
1737            }
1738            else {
1739                cout << "␣パーティクル番号(1から数えたもの)␣" << i + 1 << "␣の IC カードデータ同化において,␣フィルタリング対象ベクトルと␣"
1740                    "␣観測ベクトルが完全に一致したため,␣パーティクルの重みを定義不可能␣" << endl;
1741                system( "␣pause␣" );
1742            }
1743            fclose(N_OF_TRIP_BY_ZONE_THIS_PTCL);
1744        }
1745    }
1746
1747
1748    // パーティクルの重みの正規化
1749    double sum_w = 0.0; // 計算用の変数の宣言
1750
1751    for (int i = 0; i < N_OF_PARTICLE; i++) {
1752        sum_w += weight_of_ic[i]; // sum_w：パーティクルの重みの総和
1753    }
1754    for (int j = 0; j < N_OF_PARTICLE; j++) {
1755        normalized_weight_of_ic[j] = weight_of_ic[j] / sum_w; // 各パーティクルの重みを総和で割ったものが正規化重み
1756    }
1757
1758    // ファイルへの書き込み
1759    for (int i = 0; i < N_OF_PARTICLE; i++) {
1760        fprintf(WEIGHT_OF_IC, "␣%.12lf,%.12lf\n␣" , weight_of_ic[i], normalized_weight_of_ic[i]); // 重みは小数点以下 12桁表示
1761    }
1762    fclose(WEIGHT_OF_IC);
1763
1764    cout << "␣IC カードデータによるパーティクルの重み計算完了␣" << endl;
1765
1766    return;
1767 }
1768
1769  ////////////////////////
1770  // ここまで関数
        calculate_weight_of_ic の内容（引数とするベクトル（要素数はパーティクル数）の各要素に対し，
        IC カードデータによる各パーティクルの重みの値を代入する関数）
1771  // ここから関数calculate_weight_of_particle の内容（パーティクルの重みを計算する関数）
1772  ////////////////////////
1773
1774 void calculate_weight_of_particle(){
1775
1776    // 観測データに対する，パーティクルの重みを入れるベクトルの宣言
1777    //「要素数がパーティクル数分，全ての要素が 0で初期化」というベクトル
1778    vector<double> weight_of_mobile(N_OF_PARTICLE, 0);
1779    vector<double> normalized_weight_of_mobile(N_OF_PARTICLE, 0);
1780    vector<double> weight_of_ic(N_OF_PARTICLE, 0);
1781    vector<double> normalized_weight_of_ic(N_OF_PARTICLE, 0);
1782
1783    if (OBS_DATA_FLAG == 1 || OBS_DATA_FLAG == 3) {
1784        // 関数calculate_weight_of_mobile の実行により，引数とするベクトルにモバイル空間統計による各パーティクルの重みを代入
1785        calculate_weight_of_mobile(weight_of_mobile, normalized_weight_of_mobile);
```

```
1786    }
1787
1788    if (OBS_DATA_FLAG == 2 || OBS_DATA_FLAG == 3) {
1789        // 関数calculate_weight_of_ic の実行により，引数とするベクトルに
              IC カードデータによる各パーティクルの重みを代入
1790        calculate_weight_of_ic(weight_of_ic, normalized_weight_of_ic);
1791    }
1792
1793    // 最終的なパーティクルの重みを入れるベクトルの宣言
1794    vector<double> weight(N_OF_PARTICLE, 0);
1795    vector<double> weight_normalized(N_OF_PARTICLE, 0);
1796
1797    if (OBS_DATA_FLAG == 1) {
1798        for (int i = 0; i < N_OF_PARTICLE; i++) {
1799            weight[i] = weight_of_mobile[i];
1800            weight_normalized[i] = normalized_weight_of_mobile[i];
1801        }
1802    }
1803    else if (OBS_DATA_FLAG == 2) {
1804        for (int i = 0; i < N_OF_PARTICLE; i++) {
1805            weight[i] = weight_of_ic[i];
1806            weight_normalized[i] = normalized_weight_of_ic[i];
1807        }
1808    }
1809    else if (OBS_DATA_FLAG == 3) {
1810        for (int i = 0; i < N_OF_PARTICLE; i++) {
1811            weight[i] = (1.0 - WEIGHT_OF_OBS_DATA_IC) * weight_of_mobile[i] +
                  WEIGHT_OF_OBS_DATA_IC * weight_of_ic[i];
1812            weight_normalized[i] = (1.0 - WEIGHT_OF_OBS_DATA_IC) *
                  normalized_weight_of_mobile[i] + WEIGHT_OF_OBS_DATA_IC *
                  normalized_weight_of_ic[i];
1813        }
1814    }
1815
1816    // ファイルへの書き込み
1817    for (int i = 0; i < N_OF_PARTICLE; i++) {
1818        fprintf(WEIGHT_OF_PTCLS, "␣%.12lf,%.12lf\n␣" , weight[i], weight_normalized[i]); //
              重みは小数点以下 12桁表示
1819    }
1820
1821    fseek(WEIGHT_OF_PTCLS, 0, SEEK_SET); // WEIGHT_OF_PTCLS は読書両方のモード w+なので，
          書き込みが終わった今，次の読込に備えてポインタを先頭にしておく
1822
1823
1824    return;
1825 }
1826
1827
1828 ////////////////////////
1829    // ここまで関数calculate_weight_of_particle の内容（パーティクルの重みを計算する関数）
1830    // ここから関数resample_particle の内容（各パーティクルの復元抽出個数を決定し，リサンプリングする関
          数）
1831 ////////////////////////
1832
1833 void resample_particle() {
1834
1835    // データ同化入門(樋口，2011)の 6章「粒子フィルタ」6.5.2「リサンプリングの方法」の，整数部分と小数部分
```

```
    を分ける方法を採用
1836
1837    // WEIGHT_OF_PTCLS から読み込んだ値を入れる変数の宣言
1838    double weight_non_normalized[N_OF_PARTICLE]; // 読込時に便宜上なだけ，使わない
1839    double weight[N_OF_PARTICLE]; // 正規化重みを読み込んで入れておく配列
1840
1841    // 複製抽出するパーティクルの数を管理するための変数の宣言
1842    int num_of_resampled_particle[N_OF_PARTICLE]; // この配列の旧パーティクル番号番目(0から数えて
          )の要素の値の数だけ，旧パーティクルを複製し，新パーティクルとする
1843    double num_of_resampled_particle_fraction[N_OF_PARTICLE]; // 再抽出する個数の期待値のうち，
          小数部分を入れておく
1844    int sum1 = 0; // 複製後のパーティクルの個数を数える変数
1845
1846    int i = 0; // パーティクル番号のインクリメント変数
1847    while (
1848        i < N_OF_PARTICLE && fscanf(WEIGHT_OF_PTCLS, "␣%lf,%lf␣" , &weight_non_normalized[i
              ], &weight[i]) != EOF
1849        // ここの条件の順序(&&の前後)を入れ替えると，fscanf が先に実行されてしまう分，ポインタの位置が
              ずれるので注意
1850        )
1851    {
1852        double integer = 0; // 再抽出する個数の期待値の整数部分，整数だが次の
              modf の引数としては double である必要がある
1853        double fraction = 0.0; // 再抽出する個数の期待値の小数部分
1854
1855        fraction = modf(weight[i] * N_OF_PARTICLE, &integer); // パーティクルの重みと総個数の積
              が期待値であるから，それを整数部分と小数部分に分ける
1856
1857        num_of_resampled_particle[i] = int(integer); // まず，整数部分は抽出個数として確定
1858        sum1 += int(integer); // ここまでで確定した総抽出個数を記録しておく
1859        num_of_resampled_particle_fraction[i] = fraction; // 小数部分も後で使うので記録
1860
1861        i += 1;
1862    }
1863
1864    fclose(WEIGHT_OF_PTCLS);
1865
1866    // 最頻のパーティクルの(1から数えた)番号を定数に記憶しておく
1867    MODE_PTCL_NO = MAP(num_of_resampled_particle, N_OF_PARTICLE); // 変数MODE_PTCL_NO，
          関数MAP は冒頭で定義済み
1868    cout << "␣最頻のパーティクルは,␣(1から数えて)No.␣" << MODE_PTCL_NO << endl;
1869
1870    int sum2 = sum1; // sum1 は整数部分で確定した総抽出個数（この後は不変），
          sum2 はこの後抽出する分も含めたそれまでの総抽出個数
1871
1872    init_genrand((unsigned)time(NULL)); // 現在時刻を用いて乱数を初期化
1873    while (1) { // LOOPEND まで次の for 文を無限ループ
1874        for (int j = 0; j < N_OF_PARTICLE; j++) {
1875            if (num_of_resampled_particle_fraction[j] / (N_OF_PARTICLE - sum1) >
                  genrand_real1()) {
1876                // genrand_real1()は 0以上 1以下の範囲を一様に取る乱数なので，この文は確率【
                      num_of_resampled_particle_fraction[j] / (N_OF_PARTICLE − sum1)】で真
1877                num_of_resampled_particle[j] += 1; // 真のとき，この番号のパーティクルの抽出個数
                      を 1つ増やす
1878                sum2 += 1; // それまでの総抽出個数も 1増やす
1879            }
1880
```

```
1881            if (sum2 == N_OF_PARTICLE) { // このif 文は else if ではない，上のif 文と独立な文なので注意
1882                goto LOOP_END; // 総抽出個数が総パーティクル数に達したら，無限ループを抜ける
1883            }
1884        }
1885    }
1886    LOOP_END: // 無限ループの抜け先
1887
1888    cout << "␣パーティクルの複製抽出個数決定␣" << endl;
1889
1890    // 新旧パーティクル番号対応表への書き込み
1891    fprintf(CORRESPONDENCE_TABLE_OF_PTCLS, "␣%s,%s,%s,%s\n␣" , "␣Old␣Particle␣No.␣(count␣
          from␣1)␣" , "␣amount␣" , "␣New␣No.␣(count␣from␣1):␣from␣" , "␣New␣No.␣(count␣from␣
          1):␣to␣" ); // ヘッダの書き込み
1892    int new_ptcl_no = 1; // 新番号を(1から)数えるための変数の宣言
1893    int old_ptcl_no[N_OF_PARTICLE]; // この配列の新パーティクル番号番目(0から数えて)の要素の値は，
          複製元の旧パーティクル番号(0から数えて)とする
1894
1895    for (int j = 0; j < N_OF_PARTICLE; j++) { // このループで，番号対応表ファイルへの書き込みと，
          old_ptcl_no 配列の整備の，2つの作業を行う
1896        // j：旧パーティクルNo.(0から数えて)のインクリメント変数
1897        if (num_of_resampled_particle[j] == 0) { // その旧No.(0から数えて)の再抽出個数が 0のとき
1898            fprintf(CORRESPONDENCE_TABLE_OF_PTCLS, "␣%d,%d,%s,%s\n␣" , // 番号対応表ファイルへ
                  の書き込み(1から数えて)
1899                j + 1, num_of_resampled_particle[j], "␣---␣" , "␣---␣" );
1900
1901            // old_ptcl_no 配列に対しては何もしない
1902        }
1903        else { // その旧No.(0から数えて)の再抽出個数が 0でないとき
1904            fprintf(CORRESPONDENCE_TABLE_OF_PTCLS, "␣%d,%d,%d,%d\n␣" , // 番号対応表ファイルへ
                  の書き込み(1から数えて)
1905                j + 1, num_of_resampled_particle[j], new_ptcl_no, new_ptcl_no +
                      num_of_resampled_particle[j] - 1);
1906
1907            for (int k = new_ptcl_no - 1; k - new_ptcl_no + 1 < num_of_resampled_particle[j];
                  k++) { // old_ptcl_no 配列の整備
1908                old_ptcl_no[k] = j;
1909            }
1910
1911            new_ptcl_no += num_of_resampled_particle[j]; // 新番号(1から数えて)を抽出個数だけ進
                  める
1912        }
1913    }
1914
1915    fseek(CORRESPONDENCE_TABLE_OF_PTCLS, 0, SEEK_SET); //
          CORRESPONDENCE_TABLE_OF_PTCLS は読書両方のモード w+なので，書き込みが終わった今，
          次の読込に備えてポインタを先頭にしておく
1916
1917    // ここから新パーティクルの内容ファイル作成・書き込み
1918    if (END != 1620) {
1919        for (int p = 0; p < N_OF_PARTICLE; p++) {
1920
1921            // (0から数えて)p 番目の新パーティクル（1 から数えて No. p + 1)に，
1922            // (0から数えて)old_ptcl_no[p]番目の旧パーティクル(1から数えてNo. old_ptcl_no[p] + 1)の内容
                  を複製する
1923
1924
1925            // ファイルの読込：POUT_nFX_THIS_PTCL：
```

```
                PCATS 中断時刻に関わる固定活動以外のデータを，そのパーティクルにおける分だけ全個人分並べたファイル
1926            // パーティクルNo.に関するループ内なので，
                file_set_up 内では上手く書けなかったのでこちらに記述
1927            // 上手く書く方法があれば，file_set_up 内に書いた方が，入出力ファイルのパスやファイル名を変更する際に見易いのでそうしたいが……
1928            // プログラムの前半，関数extract_PCATSout_at_this_particle_non_fixed で書き込みし，関数count_zone_population_at_this_particle_non_fixed で読み込んで一度 fclose したファイル
1929            sprintf(
1930                nonfixed_this_ptcl,
1931                 "␣C:\\cpp\\filtering\\filtering\\output\\PCATSoutputparticle\\%d-%d\\particle_non_fixed_%d-%d(particle_No.%d).data␣" , START, END, START, END, old_ptcl_no[p] + 1
1932                );
1933            POUT_nFX_THIS_PTCL = fopen(nonfixed_this_ptcl, "␣r␣" ); // ここを ” w ” や ” w+ ” にすると，プログラム前半で作った内容が消えて上書きされるので注意
1934            if (POUT_nFX_THIS_PTCL == NULL) {
1935                cout << "␣(1から数えた)旧パーティクル番号␣" << old_ptcl_no[p] + 1 <<
1936                     "␣におけるファイル
                          POUT_nFX_THIS_PTCL（PCATS 中断時刻に関わる固定活動以外のデータを,␣そのパーティクルにおける分だけ全個人分並べたファイル）の読込失敗␣" << endl;
1937                system( "␣pause␣" );
1938            }
1939
1940            // ファイルの作成・読込：NEW_THIS_PTCL：
                POUT_nFX_THIS_PTCL を復元抽出してできた，新しいパーティクル
1941            // パーティクルNo.に関するループ内なので，
                file_set_up 内では上手く書けなかったのでこちらに記述
1942            // 上手く書く方法があれば，file_set_up 内に書いた方が，入出力ファイルのパスやファイル名を変更する際に見易いのでそうしたいが……
1943            sprintf(
1944                new_ptcl,
1945                 "␣C:\\cpp\\filtering\\filtering\\output\\resampled_particle\\%d-%d\\new_particle_non_fixed_%d-%d(particle_No.%d).data␣" , START, END, START, END, p + 1
1946                );
1947            NEW_THIS_PTCL = fopen(new_ptcl, "␣w+␣" );
1948            if (NEW_THIS_PTCL == NULL) {
1949                cout << "␣(1から数えた)新パーティクル番号␣" << p + 1 <<
1950                     "␣におけるファイルNEW_THIS_PTCL（POUT_nFX_THIS_PTCL を復元抽出してできた,␣新しいパーティクル）の作成失敗␣" << endl;
1951                system( "␣pause␣" );
1952            }
1953
1954            // POUT_nFX_THIS_PTCL から読み込んだ値を入れる変数の宣言
1955            char indv_id[N_OF_DIGIT_OF_INDV_ID + 10]; // 個人ID
1956            int category; // カテゴリ
1957            int address_zone; // 自宅ゾーン
1958            int age; // 年齢
1959            int scaling; // 拡大係数
1960            int activity; // 活動内容
1961            int act_zone; // 活動場所ゾーン
1962
1963            // 読み込んだ個人ID は，末尾 3桁のパーティクル番号が旧パーティクルのものになっているので，新パーティクル番号をつける
1964            char indv_id_without_ptcl_num[N_OF_DIGIT_OF_INDV_ID + 10]; // そのために，純粋な個人ID 部分のみを格納する配列を用意
```

```
1965
1966            while ( // 旧パーティクルファイルの読込
1967                fscanf(POUT_nFX_THIS_PTCL, "␣%s␣%d␣%d␣%d␣%d␣%d␣%d␣" ,
1968                    indv_id, &category, &address_zone, &age, &scaling, &activity, &act_zone)
                        != EOF
1969                )
1970            { // 新パーティクルファイルへの書き込み
1971
1972                strncpy(indv_id_without_ptcl_num, indv_id, N_OF_DIGIT_OF_INDV_ID); // パーティクル番号を除いた，純粋な個人ID の桁数分だけコピーして格納
1973                indv_id_without_ptcl_num[N_OF_DIGIT_OF_INDV_ID] = '\0'; // 末尾にヌル文字を記録
1974
1975                fprintf(NEW_THIS_PTCL, "␣%s%04d␣%d␣%d␣%d␣%d␣%d␣%d\n␣" ,
1976                    indv_id_without_ptcl_num, p + 1, category, address_zone, age, scaling,
                        activity, act_zone);
1977                // 個人ID は，末尾に正しいパーティクル番号を，頭に 0を付けて 3桁に揃えて付加し，出力
1978            }
1979
1980            fclose(POUT_nFX_THIS_PTCL);
1981            fclose(NEW_THIS_PTCL);
1982
1983            if ((int((p + 1) * 10000.0 / N_OF_PARTICLE)) % 100 == 0) { // 剰余演算を使って 1%ごとに以下を表示
1984                cout << ((p + 1) * 1.0 / N_OF_PARTICLE) * 100 << "␣%完了␣" << endl;
1985            }
1986
1987        }
1988    }
1989
1990    return;
1991 }
1992
1993
1994 ////////////////////////
1995     // ここまで関数resample_particle の内容（各パーティクルの復元抽出個数を決定し，リサンプリングする関数）
1996     // ここから関数output_resampled_file_for_PCATS の内容（リサンプリングされたパーティクルに基づき，PCATS 中断時の データを修正し，PCATS の体裁のファイルを出力する関数）
1997 ////////////////////////
1998
1999 void output_resampled_file_for_PCATS() {
2000     /*
2001     PCATS 中断時ファイルを頭から 1 行ずつ読み取った上で，次の操作を行う
2002     ・個人ID から末尾の旧パーティクル番号も読み込む
2003     ・旧パーティクル番号を複製する個数を参照する
2004         −複製個数が 0個なら，次の行の読込へ
2005         −複製個数が 1個なら，読み取った内容のパーティクル番号を新番号に書き換えて出力ファイルへ書き込む
2006             −−中断ダミーが 1なら，新パーティクル番号を 1進める
2007             −−次の行の読込へ
2008         −複製個数が 2個以上，かつ，中断ダミーが 0
2009             −−読み取った内容を保存用配列に一旦格納して次の行の読込へ
2010         −複製個数が 2個以上，かつ，中断ダミーが 1
2011             −−読み取った内容を保存用配列に格納し，以下の操作を複製個数回繰り返す
2012                 −−−保存用配列の中断ダミーが 0の間，0要素目から順に新番号に書き換えて出力ファイルへ書き込む
```

```
2013                ---保存用配列の中断ダミーが 1なら，その要素を新番号に書き換えて出力ファイルへ書き込
                        み，新パーティクル番号を 1進める
2014            --保存用配列を初期化し，次の行の読込へ
2015    */
2016
2017
2018    // まず，旧パーティクル番号に対して，新パーティクルとして何個複製するかの対応表を読み込む
2019
2020    // CORRESPONDENCE_TABLE_OF_PTCLS から読み込んだ値を入れる変数の宣言
2021    int num_of_resampled_particle[N_OF_PARTICLE]; // この配列の旧パーティクル番号番目(0から数えて
          )の要素の値の数だけ，旧パーティクルを複製し，新パーティクルとする
2022    int i = 0; // 旧パーティクル番号に関するインクリメント変数
2023    char header[200]; // 読み込むだけで使わない変数
2024    int old_ptcl_no_temp; // 読み込むだけで使わない変数
2025    char first_new_ptcl_no[5]; // 読み込むだけで使わない変数( " --- " もあるのでchar 型)
2026    char last_new_ptcl_no[5]; // 読み込むだけで使わない変数( " --- " もあるのでchar 型)
2027
2028    fgets(header, 200, CORRESPONDENCE_TABLE_OF_PTCLS); // ヘッダがあるファイルなので，ヘッダを読
          み込むだけ
2029
2030    while (
2031        fscanf(CORRESPONDENCE_TABLE_OF_PTCLS, "␣%d,␣%d,␣%s,␣%s␣" , &old_ptcl_no_temp, &
              num_of_resampled_particle[i], first_new_ptcl_no, last_new_ptcl_no) != EOF
2032        )
2033    {
2034        i++;
2035    }
2036
2037    // 次に，PCATS 中断時ファイルを読み込む変数を用意する
2038
2039    char indv_id[N_OF_DIGIT_OF_INDV_ID + 10]; // 個人ID：配列の大きさは個人ID の桁数に依存
2040    // 読み込んだ個人ID は，末尾にパーティクル番号が「001」として入っているので，新しいパーティクル番号
          をつける
2041    char indv_id_without_ptcl_num[N_OF_DIGIT_OF_INDV_ID + 10]; // そのために，純粋な個人
          ID 部分のみを格納する配列を用意
2042    int old_ptcl_no; // (1から数えた)旧パーティクル番号
2043    int new_ptcl_no = 1; // (1から数えた)新パーティクル番号
2044    int col[N_OF_PCATS_PARAMETER] = { 0 }; // 整数が入っているカラムをまとめて配列にしたもの，0で
          初期化
2045                // 0：カテゴリ，1：自宅ゾーン，2：就業就学地ゾーン，3：性別，4：年齢，5：職業，6：免
                      許有無，7：世帯内自動車台数，
2046                // 8：拡大係数，9：世帯人数，10：活動内容，11：活動開始時刻，12：活動終了時刻，13：
                      活動場所ゾーン，14：活動施設，
2047                // 15：固定活動ダミー，16：活動後移動有無，17：移動開始時刻，18：移動終了時刻，19：
                      移動手段，20：車ゾーン，21：車場所，22：中断ダミー
2048    char suspend_act[8]; // 中断時の活動：" MOVING " の 6文字が最長
2049    int j = 0; // 行数のインクリメント変数
2050
2051    // 次に，複製個数が 2個以上のときのため，保存用配列を用意する
2052    int keep = 0; // 保存用配列の要素番号，即ち同一個人同一パーティクルの活動の数を表す
2053    vector<vector<int>> keep_col(MAX_N_OF_ACT_OF_THE_DAY, vector<int>(N_OF_PCATS_PARAMETER,
          0)); // col の中身を保存するベクトル
2054    // 2次元ベクトルの宣言：要素数は最大の活動数分，全ての要素の初期値は【「要素数が
          col の中身たる N_OF_PCATS_PARAMETER，全ての要素が 0で初期化」というベクトル】
2055    // 要するに，((最大N_OF_PCATS_PARAMETER 個の，col にあたる要素…), (…), (…), …最大活動数分
          …, (…))という行列
2056
```

```
2057    char keep_suspend_act[MAX_N_OF_ACT_OF_THE_DAY][8]; // suspend_act の中身を保存する配列（C
          ++では文字列が文字の配列なので，文字列の配列は二次元配列になる）
2058
2059
2060    // ここがこの関数の本体(PCATS 中断時ファイルの書き換え)
2061
2062    while ( // ファイルの読込
2063        fscanf(ORIGINAL_PCATS_OUTPUT, "␣%s␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣
              %d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%s␣" ,
2064          indv_id, &col[0], &col[1], &col[2], &col[3], &col[4], &col[5], &col[6], &col
                [7], &col[8], &col[9], &col[10],
2065          &col[11], &col[12], &col[13], &col[14], &col[15], &col[16], &col[17], &col[18],
                 &col[19], &col[20], &col[21], &col[22], suspend_act) != EOF
2066        )
2067    {
2068        // 純粋個人ID の値入力
2069        strncpy(indv_id_without_ptcl_num, indv_id, N_OF_DIGIT_OF_INDV_ID); // パーティクル番
              号を除いた，純粋な個人ID の桁数分だけコピーして格納
2070        indv_id_without_ptcl_num[N_OF_DIGIT_OF_INDV_ID] = '\0'; // 末尾にヌル文字を記録
2071
2072        // パーティクル番号の値入力
2073        char temp[N_OF_DIGIT_OF_PTCL_NO + 3]; // パーティクル番号は，まずは文字列として読み取る必
              要があるので，一時的にここに格納するために宣言
2074        strncpy(temp, indv_id + N_OF_DIGIT_OF_INDV_ID, N_OF_DIGIT_OF_PTCL_NO); //
              indv_id の頭から個人 ID の桁数分だけ後ろ（＝パーティクル番号の頭）から，パーティクル番号をコ
              ピーして格納
2075        temp[N_OF_DIGIT_OF_PTCL_NO] = '\0'; // 末尾にヌル文字を記録
2076        old_ptcl_no = atoi(temp); //
              temp に格納されていた文字列としての（1 から数えた）旧パーティクル番号を，
              old_ptcl_no に int として格納
2077
2078        if (old_ptcl_no == 1) { // 旧パーティクル番号が 1の行に来たら(純粋個人ID が変わったら)，新パ
              ーティクル番号を 1にする
2079            new_ptcl_no = 1;
2080        }
2081
2082        if (num_of_resampled_particle[old_ptcl_no - 1] == 0) { // その旧パーティクル番号の複製個
              数が 0個なら
2083        // 何もしないで次の行の読込へ
2084        }
2085
2086        else if (num_of_resampled_particle[old_ptcl_no - 1] == 1) { // その旧パーティクル番号の
              複製個数が 1個なら
2087            // 以下の通り，新パーティクル番号で出力ファイルへ書き込む
2088            fprintf(REVISED_PCATS_OUTPUT, "␣%s%04d␣%2d␣%6d␣%6d␣%3d␣%3d␣%3d␣%2d␣%3d␣%5d␣%3d␣
                  %3d␣%5d␣%5d␣%6d␣%3d␣%2d␣%2d␣%5d␣%5d␣%3d␣%5d␣%3d␣%3d␣%s\n␣" , // 空白数等の書式
                  をPCATS に合わせる
2089              indv_id_without_ptcl_num, new_ptcl_no, col[0], col[1], col[2], col[3], col
                    [4], col[5], col[6], col[7], col[8], col[9],
2090              col[10], col[11], col[12], col[13], col[14], col[15], col[16], col[17], col
                    [18], col[19], col[20], col[21], col[22], suspend_act
2091              );
2092            if (col[12] == 1620) { // 活動終了時刻が 1620(同じ個人・同じパーティクルはこの行で最後)な
                  ら
2093                new_ptcl_no++; // 新パーティクル番号を 1進める
2094            }
2095        // 次の行の読込へ
```

```
2096        }
2097
2098        else if (num_of_resampled_particle[old_ptcl_no - 1] > 1) { // その旧パーティクル番号の複
              製個数が 2個以上なら
2099            // 以下の通り，読み取った内容を保存用配列に格納する
2100            for (int k = 0; k < N_OF_PCATS_PARAMETER; k++) { //
                  col の中身は keep_col ベクトルの keep 番目に格納
2101                keep_col[keep][k] = col[k];
2102            }
2103            strcpy(keep_suspend_act[keep], suspend_act); //
                  suspend_act の中身は keep_suspend_act の keep 番目に格納（文字列なので）
2104            keep++; // keep の値を 1 進める：keep は最終的にこの個人・パーティクルの活動数を表す
2105
2106            if (col[12] != 1620) { // 活動終了時刻が 1620でない（＝次の行も同じ個人・同じパーティクル）
                  なら
2107            // 直ちに次の行の読込へ
2108            }
2109            else if (col[12] == 1620) { // 活動終了時刻が 1620(同じ個人・同じパーティクルはこの行で最
                  後)なら
2110                // 以下の通り，保存用配列の内容を出力ファイルへ【パーティクルの複製個数だけ繰り返し】書
                      き込む
2111                for (int p = 0; p < num_of_resampled_particle[old_ptcl_no - 1]; p++) { // パー
                      ティクルの複製個数だけ繰り返し
2112                    for (int kk = 0; kk < keep; kk++) { // kk：保存用配列の要素番号を使って，
                          keep：この個人・パーティクルの活動数の分だけ書き込み
2113                        fprintf(REVISED_PCATS_OUTPUT, "␣%s%04d␣%2d␣%6d␣%6d␣%3d␣%3d␣%3d␣%2d␣%3
                              d␣%5d␣%3d␣%3d␣%5d␣%5d␣%6d␣%3d␣%2d␣%2d␣%5d␣%5d␣%3d␣%5d␣%3d␣%3d␣%s\n
                              ␣" , // 空白数等の書式をPCATS に合わせる
2114                            indv_id_without_ptcl_num, new_ptcl_no,
2115                            keep_col[kk][0], keep_col[kk][1], keep_col[kk][2], keep_col[kk
                                  ][3], keep_col[kk][4], keep_col[kk][5], keep_col[kk][6],
                                  keep_col[kk][7],
2116                            keep_col[kk][8], keep_col[kk][9], keep_col[kk][10], keep_col[kk
                                  ][11], keep_col[kk][12], keep_col[kk][13], keep_col[kk][14],
                                  keep_col[kk][15],
2117                            keep_col[kk][16], keep_col[kk][17], keep_col[kk][18], keep_col[kk
                                  ][19], keep_col[kk][20], keep_col[kk][21], keep_col[kk][22],
                                  keep_suspend_act[kk]
2118                            );
2119                    }
2120                    new_ptcl_no++; // 新パーティクル番号を 1進める
2121                }
2122                keep = 0; // 活動終了時刻が 1620のとき，パーティクル複製個数ぶんの繰り返しが終わった
                      ら，活動数keep の初期化
2123            }
2124        }
2125
2126        if (j == 500000) { cout << "␣50万行目処理完了␣" << endl; }
2127        if (j == 1000000) { cout << "␣100万行目処理完了␣" << endl; }
2128        if (j == 1500000) { cout << "␣150万行目処理完了␣" << endl; }
2129        if (j == 2000000) { cout << "␣200万行目処理完了␣" << endl; }
2130        if (j == 2500000) { cout << "␣250万行目処理完了␣" << endl; }
2131        if (j == 3000000) { cout << "␣300万行目処理完了␣" << endl; }
2132        if (j == 3500000) { cout << "␣350万行目処理完了␣" << endl; }
2133        if (j == 4000000) { cout << "␣400万行目処理完了␣" << endl; }
2134        if (j == 4500000) { cout << "␣450万行目処理完了␣" << endl; }
2135        if (j == 5000000) { cout << "␣500万行目処理完了␣" << endl; }
```

```
2136        if (j == 5500000) { cout << "␣550万行目処理完了␣" << endl; }
2137        if (j == 6000000) { cout << "␣600万行目処理完了␣" << endl; }
2138        if (j == 6500000) { cout << "␣650万行目処理完了␣" << endl; }
2139        if (j == 7000000) { cout << "␣700万行目処理完了␣" << endl; }
2140        if (j == 7500000) { cout << "␣750万行目処理完了␣" << endl; }
2141        if (j == 8000000) { cout << "␣800万行目処理完了␣" << endl; }
2142
2143        j++;
2144
2145    }
2146
2147    fclose(ORIGINAL_PCATS_OUTPUT);
2148    fclose(REVISED_PCATS_OUTPUT);
2149
2150    return;
2151 }
```

# 謝辞

この論文は，以下に記す多くの方々に支えられてようやく完成させることができたものです．ささやかながら，ここに謝意を述べます．

主査である布施孝志先生は，この修士論文の指導のみならず，学部 4 年以来，研究と学生生活一般とを問わず最もお世話になった方と言えます．本分たる研究以外を忙しくし，決して研究熱心で真面目な学生とは言えない私に対しても（呆れや諦めを感じさせてしまったときも多々あるかもしれませんが），丁寧かつ的確なコメントをくださり，自分の考えを客観的に文章化することの重要性と難しさに気付かせていただけました．

副査である関本義秀先生は，主に研究する意義がどこにあるのかという点で，多くの有益な助言をくださいました．布施先生のお言葉が主に研究の妥当性や結果の適切な伝え方についてのものだったのとは，また違った側面であり，新鮮かつありがたいご指摘でした．研究以外の忙しさにかまけて伺う頻度が低くなってしまったことを申し訳なく思っております．

清水英範先生がゼミでくださるコメントは，研究を日々進める上での心の支えでした．特に，「論文で大事なのは序論と結論で，中身は少々まずくたって発表ではどうせ皆聞いちゃいないし理解もできてない」という先生らしいご冗談は，卒論の頃から大好きなお言葉です．また，ゼミ発表の時点でいつも「15 号教室の最終発表でどう見せるか」を意識させてくださるのも，とても助かりました．

地域/情報研究室の先輩である神谷啓太さんは，ご自身の博士論文がお忙しいのにもかかわらず，最後まで私を見捨てることなく指導してくださいました．卒論のときと同じ，同室にすぐに研究の相談をできる方がいらっしゃるという環境は，非常にありがたかったです．そして，その環境に最後まで甘えてばかりですみませんでした．

既に研究室を去ってしまった方にも，大変お世話になりました．中でも，東京工業大学環境・社会理工学院土木・環境工学系の中西航特任助教（と書くと仰々しいですね．私にとっては「中西さん」です）は，卒論の指導において私に研究・論文の何たるかを教えてくださった方です．中西さん無しにはこの修論も書けなかったでしょう．異動される前にやっていた卒論の続きの研究，結局放置してしまってごめんなさい．私には 2 つ並行は無理でした．また，昨年の修了生である原田遼さんの研究無しにはこの修論は完成しなかったことも，言うまでもありません．

3 年間過ごした研究室の皆さまにも，大変お世話になりました．特に同期の大倉には，自分が研究室に来ない日が多いこともあって，相変わらずご心配・ご迷惑をおかけしました．ごめん．ありがとう．そして，Joan や Amr，秘書の石田さんの他，先輩のたかまみさん，永良さん，平松さん，池澤さん，泉さん，後輩の妹背，梶さん，柚花さん，仁美さん，横澤，荒木，黒ちゃん，鶴野，山野と，思えば多くの仲間に恵まれたものです．交通・地域ラボとして部屋を同じくする交通研の皆さま，特に鳩山先生，原先生，柳沼先生にも，たくさんのご支援を頂いています．

PCATS の実装にあたっては，東北工業大学工学部都市マネジメント学科の菊池輝教授が昨年の原田さんの修士論文のためにご提供くださったコードを，改良した上で利用させていただきました．この紙上をお借りしてお礼申し上げます．

また，内定先である中日本高速道路株式会社の社員の方にも，研究に関しての有益なアドバイスを幾つか頂きました．こちらもこの紙上をお借りしてお礼申し上げます．

学部に比べて一般には息が詰まりがちな大学院の生活で，私が息を抜き過ぎていたのが，アルバイト先であり遊び場所でもある東京ディズニーリゾートでした．所属部署であるアトラクション運営部ディズニーシー・トランジットスチーマーラインユニットの船長・ゴンドリエの皆さま，ありがとうございました．11 月からのヴェネツィアン・ゴンドラへのクロストレーニングは，前々から希望していたとは言え，大いに研究の~~妨げ~~息抜きになりました．無理言って行かせていただいて，ありがとうございました．

また，学部生活の大半を占めた駒場祭委員会・五月祭常任委員会での活動で苦楽を共にした仲間達を筆頭に，フィロムジカ交響楽団，鉄道研究会，インタークラスの友人ら，4 年間の学生生活を支えてくれた全ての人にも感謝しています．

家族と離れて暮らす私にとって，日々の私生活を支えてくれたのは林茉里奈さんでした．特に就活が忙しかった時期や，最後の追い込みの時期は，貴女がいたから頑張れました．祖父母も，日々私のことを応援してくれました．中でも 11 月に天寿を全うした節子ばあちゃん，本当にありがとう．そして何より両親へ．私のわがままで東大しか受けなかったことに始まった駿台での 1 年間の浪人に加え，大阪大学での 1 年間の下宿生活と私の勝手な再受験，それに伴う大阪へと東京への 2 度もの長距離引越という，時間的にも金銭的にも大きな回り道をした私を，小言を挟みつつも何だかんだで支え続けてくれました．決して裕福ではない家計でありながら，学生生活の最後まで特に不自由の無い生活を送らせてくださったことに，本当に感謝しています．育児計画では 22 年で自立する予定だったそうですが，4 年も超過してしまいました．そんな私も，春からようやく経済的に自立します．ここまで育ててくれた両親に最大の謝意を送って，筆を置きたいと思います．

2018 年 1 月 23 日

4 年ぶりの大雪の降りしきる本郷キャンパスにて

福留　義章