カルトグラムの作成手法に関する研究
－GISを用いた統計データの新たな表現に向けて－

Studies on Cartogram Construction Algorithms
for Better Visualization of Statistical Data on GIS

2005年3月

井 上　　亮

論文要旨

近年の情報通信技術の進展に伴い，国や地方自治体等が収集する多種多様な統計データを Web 上から無料で入手可能になってきている．例えば，平成 16 年に開設された「統計データ・ポータルサイト ～政府統計の総合窓口～」(http://portal.stat.go.jp/) では，政府機関等が公表している全ての統計データを検索でき，その多くを無償でダウンロードできる．また，現在では過去の統計データも電磁的媒体で安価に入手することもできる．このように，一般市民や分析者が多くの統計データを利活用し，地域の現状・変遷を分析・把握することができる環境が整ってきた．

これら大量の統計データは，行政区域等の集計単位に基づき地理情報システム (GIS) 上で管理されている．GIS では基本機能として統計データの視覚化を提供しており，プレゼンテーションやデータマイニングの道具として利用されてきた．また，WebGIS の普及により GIS の統計データ視覚化手法を用いて Web 上で統計データを閲覧できる環境が整ってきており，GIS の統計データの視覚化機能はより多くのユーザーが利用できる一般的な手法へとなりつつある．

以上のように情報通信技術の発展につれて，膨大な統計データの入手・利用が容易になると同時に，GIS 利用者，中でも統計データ視覚化手法の利用者の裾野が広がりつつある．この結果，今後 GIS を用いたより効果的な統計データ視覚化への要求はますます高まると考えられる．この要求に応え GIS の更なる普及・利用を促すために，本研究では統計データの視覚化手法の中からカルトグラムに着目し，GIS を用いた統計データの新たな視覚化手法を提案する．

カルトグラムとは，地図を変形して統計データの特徴を表現する視覚化手法である．カルトグラムには大きく分けて 2 種類あり，地点間の近接性を示す統計データを地図上の地点間距離の長短で表現するディスタンスカルトグラムと，行政区域内の人口等の統計データを地図上の地域の面積の大小で表現するエリアカルトグラムに分類される．また，エリアカルトグラムは，地域間の隣接関係を保持したまま地図を連続的に変形し統計データを表現する連続エリアカルトグラムと，隣接関係を捨象し各地域を非連続に記して統計データを表現する簡便法の非連続エリアカルトグラムに分けられる．

カルトグラムは古くから効果的な統計データ視覚化手法として注目されてきた．カルトグラムの読図者はカルトグラム上と地理的地図上の地図形状の比較を行い，その違いをもとに表現されている統計データの特徴を認識する．そのため，読図者が対象地域の地理的地図形状に対して先験的な知識を備えている場合カルトグラムによる視覚化は特に効果的で，印象的な視覚化が可能である．

以前は手作業による作図が行われていたが，1960・70 年代に計算機を用いた作成手法が提案されて以来，数多くの作成手法が提案されてきた．しかし，既存作成手法の計算過程の複雑さや数学的不明快さ，作図結果の統計データ視認性の低さ，また作成手法のソフトウェア未整備等が主因となり，カルトグラムを利用した視覚化はほとんど行われていないのが現状である．

本論文ではカルトグラムの統計データ視覚化手法としての潜在能力に注目し，一般市民や分析者がカルトグラムを利用した統計データ視覚化を容易に行うことができる環境を整備し，カルトグラムの実用化を実現することを最終的な目標とする．この目標の達成に向けて，簡潔な操作でデータ視認性の高いカルトグラムを作成できる手法の構築を行う．また，その成果をもとに，最も普及している GIS ソフトウェアの一つである ArcGIS の拡張機能として，カルトグラム作成ソフトウェアを整備する．

第 1 章では，以上に述べた研究の背景と目的に続き，本論文におけるカルトグラム作成手法の開発方針について纏める．

地図上の地点間距離や地域面積だけでは地点配置や地域形状を一意に定められないため，カルトグラム作成問題は正則化が必要な不良設定問題である．本論文では，この正則化のために地図形状変化の抑制を導入し，過剰な地図形状変形がなく統計データの特徴を認識しやすいカルトグラム作成を目指す．

また，カルトグラム作成ソフトウェアの開発のためには，作成手法は高い操作性を備えていなければならない．変数設定の簡潔さや高速な計算という操作性を実現するには，作成手法は数学的に明快でなければならない．

即ち,「過剰な地図形状変形を排除する正則化条件の設定」による「数学的明快さを持ったカルトグラム作成手法」の追求を，本論文におけるカルトグラム作成手法開発の基本的方針とする．

第 2 章では，ディスタンスカルトグラムと連続エリアカルトグラム，および非連続エリアカルトグラムのうち近年注目されているサークルエリアカルトグラムの 3 種類について既往研究の整理を行い，既存作成手法の問題点について指摘する．

第 3 章では，ディスタンスカルトグラム作成手法の構築を行う．まず，ディスタンスカルトグラム作成問題を非線形最小二乗問題で記述する．この非線形最小二乗問題を数値解法の一つである Levenberg-Marquardt 法を用いて正則化を行い，一つの作成手法を導く．一方，ディスタンスカルトグラム作成問題にカルトグラム上の地点間リンクの方位角変化を抑制する正則化を導入し，地理的地図形状に近く統計データの特徴を判別しやすい作図を行うことができる新たなディスタンスカルトグラム作成手法を提案する．また，Levenberg-Marquardt 法に基づく解法と新たな提案手法を都市間の交通所要時間データを用いて比較し，提案手法が簡潔な操作で高速にデータ視認性が高いディスタンスカルトグラムを作成できることを確認する．

第 4 章では，連続エリアカルトグラム作成手法の構築を行う．具体的には，まず簡潔な作成手法を構築するために任意形状の地域を三角網分割することを前提とし，連続エリアカルトグラム作成問題を三角形の面積を統計データに合わせて変形する問題として定式化する．さらに，地理的地図からの変形を抑制する正則化条件として，三角網上の各辺の方位角変化を抑制する項を導入する．これらをもとに，地理的地図からの変形が小さく比較対照が容易な連続エリアカルトグラムを作成できる数学的に明快な手法を提案する．人口データ等を用いて提案手法の評価を行い，簡潔操作でかつ高速に連続エリアカルトグラム作成を実行できることを確認する．

第 5 章では，非連続エリアカルトグラムの一種で，各地域の統計データを円の面積の大小で表現するサークルエリアカルトグラムについて新たな作成手法を提案する．サークルエリアカルトグラム作成問題とは，隣接する地域を表す円が接するように円を配置する問題である．即ち，隣接する円の半径の和が円の中心間距離となるように円の中心座標を決定するディスタンスカルトグラム作成問題であると言い換えることができる．そこで，第 3 章のディスタンスカルトグラム作成手法を応用し，地理的位置関係を保ったサークルエリアカルトグラムの作成手法を提案する．更に円の重なりを排除するための条件を追加したサークルエリアカルトグラム作成手法を構築し，人口データを用いてその評価を行う．

第 6 章では，カルトグラム上への空間データの内挿法を構築し，その適用例を示す．アフィン変換と普遍クリギングを利用した空間内挿法をもとに，位相が反転している基準点の除去とカルトグラム上への空間データの内挿を同時に行う手法を提案する．第 3・4 章で作成したカルトグラムに対

して提案手法を適用し，内挿により更に印象的な統計データ視覚化が可能であることを例示する．

第 7 章では，第 3・4 章で提案したカルトグラム作成手法を多くの GIS ユーザーの利用に供するため，GIS 上で動作するカルトグラム作成ツールの開発を行う．具体的には，最も普及している GIS ソフトウェアの一つである ArcGIS の拡張機能としてカルトグラム作成手法を実装し，GIS 上で管理されている様々な統計データを元に簡潔な操作でカルトグラムを作成し視覚化することを可能にする．ユーザーインターフェースを開発しデータ入力・作成カルトグラムの出力に関する操作性を確保するとともに，統計データの変遷をより効果的に視覚化できるカルトグラムのアニメーション表示・出力機能を整備し，カルトグラムの統計データ視覚化手法としての実用性・有効性を向上させる．

第 8 章では，各章の成果を総括し，本論文の結論とする．

以上のように，本論文では統計データの視覚化手法としてカルトグラムに着目し，地図上の距離で統計データを表現するディスタンスカルトグラムと，面積で統計データを表現するエリアカルトグラムの双方について，統計データの視認性が高い作図が可能な数学的に明快な作成手法を提案した．また，その成果をもとに，簡潔な操作でカルトグラム作成を行うソフトウェアを ArcGIS の拡張機能として開発した．

目 次

図 目 次

表 目 次

第1章　序論 －研究の背景と目的－

近年の情報通信技術の進展に伴い，国や地方自治体等により収集された多種多様な統計データがWeb上に掲載され，無料で提供されるようになってきている．例えば，総務省統計局が平成16年1月20日に開設した「統計データ・ポータルサイト ～政府統計の総合窓口～」(http://portal.stat.go.jp/)にアクセスすると，政府機関等によって公表されているあらゆる統計データを検索することができ，その多くを無償でダウンロードできる．また，一部省庁では公表と同時に統計データがWeb上に掲載されるようになっており，迅速な統計データの公開への流れも進んでいる．以前に収集された時系列の統計データも，例えば財団法人統計情報研究開発センターなどから電磁的媒体で提供されており，安価に入手することができる．その結果，一般市民や分析者が多くの統計データを利活用し，地域の現状・変遷を分析・把握することができる環境が整ってきた．

これら大量の統計データは，行政区域等の集計単位に基づいて地理情報システム (GIS: geographical information system) 上で管理されている．GISでは基本機能の一つとして統計データの視覚化機能が提供されており，プレゼンテーションやデータマイニングの道具として利用されてきた．従来，GISソフトウェアには，地図上の地域を統計データの値に基づいて色分けするコロプレスマップや，地図上に点を表示し事象の空間密度分布を表現するドットマップ，地図上に棒グラフや円グラフなどを記載する手法等が実装されている (例えば 大場, 2003; 高阪, 2002)．また，例えば総務省統計局の「統計GISプラザ」(http://gisplaza.stat.go.jp/GISPlaza/) のようにWebGISを利用した情報提供も進んでおり，Webページにアクセスすれば誰でも棒グラフ・円グラフ・コロプレスマップ等の統計データ視覚化手法を通してデータを閲覧できる環境が整ってきている．このようにGISの普及に伴い，地図を用いた統計データの視覚化手法は，GISソフトウェアを所有する一部の計算機ユーザーのみが利用できるものから，GISソフトウェアを持たない多くのWebユーザーが利用できる一般的な手法へとなりつつある．

以上のように情報通信技術の発展につれて，膨大な統計データの入手・利用が容易になってきており，統計データ群から有益な情報を抽出するデータマイニングに対する需要を生んでいる．それと同時に，WebGISに代表される技術によりGISの一般的利用が進み，GIS利用者，中でも統計データ視覚化手法の利用者の裾野を大きく広げつつある．この結果，今後GISを用いたより効果的な統計データ視覚化への要求はますます高まると考えられる．この要求に応えGISの更なる普及・利用を促すために，本研究では統計データの視覚化手法の中からカルトグラム (cartogram) に着目し，GISを用いた統計データの新たな視覚化手法を提案する．

カルトグラムとは，地図を変形して統計データの空間分布の特徴を視覚的に表現する手法で，計量地理学において提案されてきた (Monmonier, 1977; Dorling, 1991; Tobler, 2004)．このカルトグラムには大きく分けて2種類あり，例えば交通所要時間等の地点間の近接性を示す統計データを地図上の地点間距離の長短で表現するディスタンスカルトグラム (distance cartogram) と，行政区域内の人口等の統計データを地図上の地域の面積の大小で表現するエリアカルトグラム (area

cartogram) に分類される (Monmonier, 1977; Tobler, 2004)．また，エリアカルトグラムは，地域間の隣接関係を保持したまま地図を連続的に変形し統計データを表現する連続エリアカルトグラム (continuous area cartogram, contiguous area cartogram) と，隣接関係を捨象し各地域を非連続に記して統計データを表現する簡便法の非連続エリアカルトグラム (non-continuous area cartogram) に分けられる (Dent, 1999; Tobler, 2004)．

カルトグラムによる統計データの視覚化では，読図者はカルトグラム上と地理的地図上の地図形状の比較を行い，それらの違いを基に表現されている統計データの特徴を認識する．読図者が対象地域の地理的地図形状に対して先験的な知識を備えている場合，カルトグラムによる視覚化は特に効果的である．日常見慣れた地図形状と異なるカルトグラム上の地図形状は違和感を生み，統計データの特徴を直感的に把握することを促す．カルトグラムは古くから効果的な統計データ視覚化手法として注目されてきており，手作業による作図が行われていた．1960・70 年代に計算機を用いた作成手法が提案されると，その後多くの研究が行われてきたが，1980 年代半ばには計算機性能の限界から一度研究は下火となる．しかし，計算機性能が向上した 1990 年代後半から再び統計データの視覚化手法として注目を浴び，近年多くの作成手法が提案されてきている．しかし，既存作成手法の計算過程の複雑さや数学的不明快さ，作図結果の統計データ視認性の低さ，また既存作成手法のソフトウェア整備がほとんど行われていないこと等が主因となり，カルトグラムを利用した視覚化はほとんど行われていないのが現状である．

そこで，本論文ではカルトグラムの統計データ視覚化手法としての潜在能力に注目し，一般市民や分析者がカルトグラムを利用した統計データ視覚化を容易に行うことができる環境を整備し，カルトグラムの実用化を実現することを最終的な目標とする．

この目標の達成に向けて，ディスタンスカルトグラム・エリアカルトグラムに関して，簡潔な操作でデータ視認性の高いカルトグラムを作成できる手法の構築を行う．また，その成果をもとに，最も普及している GIS ソフトウェアの一つである ArcGIS の拡張機能として，カルトグラム作成ソフトウェアを整備する．

ここで，本論文におけるカルトグラム作成手法の開発方針を以下に記す．

まず，カルトグラム作成問題とは，解が一意に定まらない不良設定問題である．地図上の地点間距離や地域面積だけでは，カルトグラム上の地点配置や地域形状を一意に定めることはできない．例えば，四辺形のネットワークに与えられた地点間距離を再現する地点配置は無数に存在し，多角形の地域に与えられた面積を再現する地域形状もまた無数に存在する．そのため，カルトグラム作成手法の構築には，正則化条件の設定が不可欠である．そこで本論文では，カルトグラム作成問題の正則化のため，地図形状変化の抑制する条件を導入する．前述のようにカルトグラムとは，地理的な地図を始めとする読図者が認知している対象地域の地図形状との比較を通して，カルトグラム上に表現されている統計データの特徴を認識する視覚化手法である．カルトグラム上の地図形状が比較対象となる地図形状から大きく変形している場合，読図者はその変形が意味するところを解釈できなくなり，視覚化手法として機能しない．そのため，カルトグラム作成では，統計データの表現とは無関係な地図形状の変形を極力排除するべきであり，地図形状変化の抑制はカルトグラム作成問題の正則化には有効である．

また，カルトグラム作成ソフトウェアの開発には，操作性の高い作成手法を構築する必要がある．操作性とは具体的に，データ入力や変数設定等の初期値設定に試行錯誤を要さず容易であること，また短時間で計算が可能であることを意味する．容易な初期値設定を達成するためには，最低

限の変数を用い，またそれらの初期値設定が持つ数学的な意味が明らかであることが求められる．また，短時間で計算可能であるためには，試行錯誤的な計算過程ではなく，最低限，必ず解に収束する方向に計算過程が進む手法でなければならない．これら操作性に関わる条件を満たすためには，作成手法は数学的に明快でなければならない．

すなわち，この「過剰な地図形状変形を排除する正則化条件の設定」による「数学的明快さを持ったカルトグラム作成手法」の追求が，本論文におけるカルトグラム作成手法開発の基本的方針となる．この方針を踏まえ，本論文ではディスタンスカルトグラム・エリアカルトグラム双方の作成手法の提案を行う．

第2章　カルトグラムに関する既往研究

第 2 章では，ディスタンスカルトグラムとエリアカルトグラム作成手法に関する既往研究の整理を行う．

2.1　ディスタンスカルトグラムに関する既往研究

ディスタンスカルトグラムとは，例えば地点間の所要時間や旅行費用等の地点間の近接性を表す統計データを，図上の距離によって把握できるように地点を配置する主題図のことをいう．ディスタンスカルトグラムを地理的地図と比較することにより地域による近接性指標の違いを，ディスタンスカルトグラムの時系列的な比較によって近接性指標の時代変遷を視覚的に示すことができる．

ディスタンスカルトグラムの代表例は，地点間所要時間を縮尺として地点を配置する時間地図 (time-space map) である．従来，ディスタンスカルトグラムは主に時間地図作成手法として研究されてきた．

ディスタンスカルトグラム作成の基本的問題は，地点間に近接性が与えられている幾つかの地点 (対象地点) を平面上にいかに配置するかという問題である (例えば 清水, 1992; 伊藤, 2001a)．従来のアプローチは，大きく 2 つに分けられる．

1 つは，対象地点全ての 2 地点間に与えられた近接性を再現しようとする方法である．本論文では，この方法を完全ネット型ディスタンスカルトグラムと呼ぶ．この解法として一般に多次元尺度構成法 (MDS: multidimensional scaling) が適用される (例えば Ewing, 1974; Muller, 1978)．この解法は数学的に明快であり，また，適当な最適基準のもとにディスタンスカルトグラムの地点配置 (図形形状) は一意に確定される．あとは，回転・平行移動の拘束により，設定する座標系にディスタンスカルトグラムを固定すればよい．しかし，全ての地点間の近接性指標を再現しようとするため，対象地点数が多い場合には局所的には再現精度が著しく低下し，違和感のあるディスタンスカルトグラムが得られる場合がある．そのため，部分的なデータのみを用いて MDS を繰り返しディスタンスカルトグラムを作成する手法も提案されている (Spiekermann and Wegener, 1993, 1994)．

もう 1 つの方法は，鉄道網や道路網などを参考に対象地点を結ぶ連結ネットワークを考え，このリンクに相当する地点間の近接性データのみを再現しようとする方法である．本論文では，この方法を部分ネット型ディスタンスカルトグラムと呼ぶ．大都市間の近接性，地理的に近い都市間の近接性といったように，多くの人が注目する近接性を精度よく再現する操作が容易である．もちろん，リンクが定義されない地点間の近接性は一切無視されるため，全ての対象について効果的とは言い切れないが，一般的には，違和感の少ないディスタンスカルトグラムを得やすい．筆者らの理解では，部分ネット型ディスタンスカルトグラムは桝谷他 (1995, 1997)，古藤 (1995, 1997) がパイオニアであり，この分野でのわが国の貢献という意味においても注目される．しかし，四角形ネットワークの各リンクに与えられた近接性を再現しようとする場合を考えれば明らかなように，部分

ネット型ディスタンスカルトグラムは，近接性だけでは必ずしも図形形状を確定することができない問題である．桝谷他 (1995, 1997)，古藤 (1995, 1997) による解法は，いろいろな状況に対してそれぞれ対応方法を用意するという個別問題解決型の手順構成になっており，数学的に明快な解法とはなっていないのが現状である．

2.2　エリアカルトグラムに関する既往研究

エリアカルトグラムとは，各地域の統計データの大小をカルトグラム上の面積の大小を用いて表現する手法である．エリアカルトグラムは単に “cartogram” と呼ばれたり，あるいは “value-by-area map” と呼ばれている．このエリアカルトグラムは，通常の地図を変形させてデータを表示する連続エリアカルトグラムと，地域形状や隣接関係等を捨象し簡略的にデータを表現する非連続エリアカルトグラムの 2 種類に大別できる．

連続エリアカルトグラムには今まで数多くの作成手法 (Tobler, 1973, 1986; Dougenik *et al.*, 1985; Gusein-Zade and Tikunov, 1993; Kocmoud and House, 1998; Keim *et al.*, 2004; Gastner and Newman, 2004) が提案されてきた．しかし，複雑なパラメータ設定が必要なものや，計算に時間を要するもの，データを高精度で表現できないものなど様々な問題を抱えている．また，これらの作成手法が一般のユーザーが利用できるように GIS の拡張機能等のソフトウェアとして提供されている例 (Du and Liu, 1999) はほとんど見られないのが現状である．

一方，非連続エリアカルトグラムの作成手法も数多く提案されてきた (Olson, 1976; Dorling, 1991)．その中で現在注目されている手法は，Dorling (1991) によって提案されたサークルエリアカルトグラム (circle area cartogram) である．サークルエリアカルトグラムは，各地域の形状の情報を省略し，各地域を円で表現し円の面積でデータを表現する手法である．この手法は連続エリアカルトグラムに比べて簡便な計算で作成が可能であり，その作成プログラムが公開されている (Dorling, 1996) ため，データの視覚化手法として利用されている例もある (Yano *et al.*, 2001)．また，MAPresso(http://www.mapresso.com/) や GeoDa(Anselin, 2004; Anselin *et al.*, forthcoming) などのソフトウェア上に実装されるようになってきており，広く利用され始めている．しかし，既存手法はデータによっては変形が非常に大きく，円と地域の対応がつかなくなってしまうという問題を抱えている．

第 2.2 節では，これら 2 種類のカルトグラムに対する既存の作成手法の整理を行い，その問題点を指摘する．

2.2.1　連続エリアカルトグラム

以前，連続エリアカルトグラムは手作業によって作成されていたが，Tobler (1973) は，計算機を用いた連続エリアカルトグラム作成手法を提案している．カルトグラム上の座標を (x, y)，通常の地図上の座標を (u, v)，カルトグラムで表現する統計データの (u, v) 上での密度を $h(u, v)$ とおき，面積の情報だけでは地域形状を一意に定めることは不可能であるから変形の少ない一意な解を求めるため，角度変化を最小化する Dirichlet 積分最小化を設定し，エリアカルトグラム作成問題を式 (2.1) と定義している．

$$\min \int_R \left(\frac{\partial x^2}{\partial u} + \frac{\partial y^2}{\partial u} + \frac{\partial x^2}{\partial v} + \frac{\partial y^2}{\partial v} \right) dudv \tag{2.1a}$$

$$s.t. \quad \frac{\partial x}{\partial u}\frac{\partial y}{\partial v} - \frac{\partial x}{\partial v}\frac{\partial y}{\partial u} = h(u,v) \tag{2.1b}$$

実際にエリアカルトグラム作成する際には，地域を正方格子に分割した後，式 (2.1) の近似解法を各格子に適用し頂点の移動量を計算している．しかし，この近似解法では統計データの表現に不必要な地域形状の変形を抑えることができず，頂点の位置関係が反転し位相が破壊されることを防げない．そこで，頂点の座標変化量を 1 点ずつ個別に計算し，位相破壊が生じる場合には座標変化量を小さくするという場当たり的な手法で調整している．また，簡便解法 (Tobler, 1986) として xy 方向に独立に計算する手法も提案しているが，統計データを高精度に表現できない (図 2.1(a))．

Dougenik *et al.* (1985) では Tobler (1973) の解法を改良し高速化を果たしているが，アルゴリズムは不明快さを拭えずまた必ずしも視覚的に分かりやすいエリアカルトグラムを作成することができない (図 2.1(b))．また，Gusein-Zade and Tikunov (1993) は，各地域の重心・頂点間距離を統計データに基づき伸縮させる手法を用いている (図 2.1(c))．しかし，この手法に基づいて作成した連続エリアカルトグラム上の地域形状は重心を中心とした円に近づいてしまうため，形状変化が大きく視覚的に優れた手法とは言えない．

これまでの手法では，地域形状変化を抑えて視覚的に分かりやすいエリアカルトグラムを作成することを十分考慮していなかった．しかし，Kocmoud and House (1998) では，統計データに面積を合わせる計算と地域形状変化を抑える計算を交互に行い，視覚的に分かりやすいエリアカルトグラム作成を行う手法を提案している (図 2.1(d))．この手法では，地域形状を可能な限り保持しながら高い精度でデータを表現するエリアカルトグラムを作成できる．しかし，そのアルゴリズムは，地域形状維持のために「辺の方位角変化抑制」「辺長の比率変化抑制」を，位相破壊の防止のために「地域形状の内角変化抑制」「辺長変化抑制」「辺の交差防止」を導入するなど，都合 5 つの制約条件を設定してエリアカルトグラム作成を行っている．これらの制約条件は複雑で数学的に明快ではなく，またそれらの重み付けも論文中で明らかにされていない．また，本手法によるとアメリカ合衆国州人口エリアカルトグラムの作成に CPU300MHz の計算機で 6 時間もの計算を要し，視覚化手法として実用的とは言えない．

Keim *et al.* (2004) では，CartoDraw と呼ぶ作成アルゴリズムを提案している (図 2.1(e))．Kocmoud and House (1998) と同様に，統計データの面積による表示とともに地域形状変化の抑制を考慮したアルゴリズムを構築しており，視覚的に分かりやすいエリアカルトグラムを作成することが可能である．さらに，Kocmoud and House (1998) より約 2000 倍高速に計算できるとしており，既存の最良手法といえる．

CartoDraw のアルゴリズムでは，scanline と呼ぶ直線を設定し，地域の面積を統計データに合わせるように scanline から一定距離内の頂点を scanline の垂線方向に移動させる手法を提案している．但し，地域形状変化が基準を超えた場合は変形を止めるというルールを設定している．このアルゴリズムの難点は，初期設定である scanline に対する結果の依存性が強くこの設定によりエリアカルトグラム形状が大きく異なる点，また不適切に設定された場合には統計データを低い精度で表現するエリアカルトグラムしか作成できない可能性がある点である．

Gastner and Newman (2004) では，物理学の拡散方程式の解法を利用し，カルトグラム上全て

の点における密度が一定になるように点の移動量を求めて連続エリアカルトグラムを作成する手法を提案している (図 2.1(f))．この手法は，数式は複雑であるが，数学的に明快である．また，多地域のデータを用いても，論文中に詳述されていないが，短時間に計算が可能だとしている．しかし，そのカルトグラム上の地域形状は Gusein-Zade and Tikunov (1993) に近い形状をしている．この手法では，カルトグラム上の地域形状を規定することはできないので，必ずしも分かりやすい形状となるとは限らない．

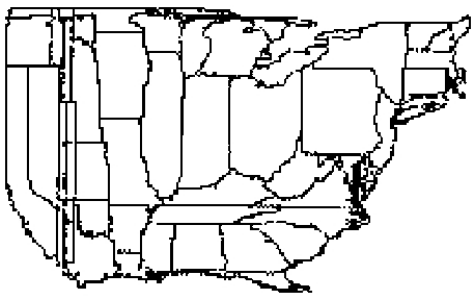
このように，既存のエリアカルトグラム作成手法は，視覚的に分かりやすいエリアカルトグラムを作成できるようになってきているが，アルゴリズムが複雑な手法や，初期値設定に多くの工夫を要する手法等，未だ実用的とは言えない．

2.2.2　サークルエリアカルトグラム

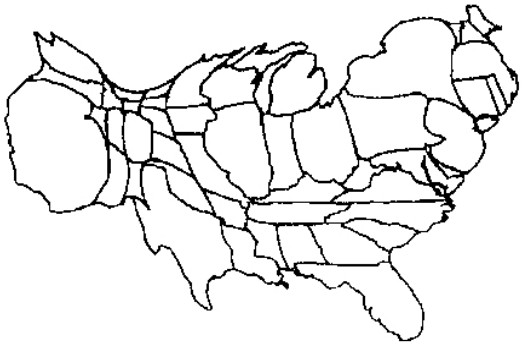
エリアカルトグラムの簡便法として，地域の隣接関係の表現を省略し，短時間での作成を可能とする非連続エリアカルトグラムが提案されている．Dorling (1991, 1995b) は，各地域を円で表現し，円の面積の大小で統計データの大小を表す，サークルエリアカルトグラムを提案した．この手法では，全ての地域が円で表現されるため，各地域の面積の比較を行うことも容易になるという利点を備えている．

もちろん，地図上に円を描き，その大きさで統計データを表現するという手法は ArcGIS 上でも実装されており，簡単に作成することが可能である．地域の地理的位置，例えば地域の重心などの代表点を利用して円の中心座標を設定し，その後，円の縮尺を調整する手法である．この手法による作図結果を図 2.2(a)(b) に示す．図 2.2(a) では，円の縮尺を大きく描いているため，当然ながら地理的な面積は小さいが人口が多い地域が密集する地方では円が複雑に重なってしまい，円の面積や色の情報を判別することができない．しかし，円の縮尺を小さくし円の重なりを緩和すると (図 2.2(b))，当然，地理的な面積は大きいが人口が少ない地域が多い地方では，円が非常に小さくなり，色を認識することが不可能になる．このように，地理的位置関係をもとに円を配置した場合は，データの視認性が著しく低下する場合がある．

そこで，Dorling (1991, 1995b) は新たな作成手法を提案した．すなわち，　『円の重なりを解消するよう円を移動させる』が，　『隣接する地域を表す円はできる限り接するように配置する』ことの 2 点を指針とする手法である．作図に当たっては，これら二つの条件を満たすように円を一つずつ順番に移動させて円を配置している．この手法では，円の重なりを解消すると同時に，余白を小さく円を大きく表示することが可能になる．Dorling (1991, 1995b) の手法による結果を図 2.2(c) に示す．確かに円の重なりはほぼ解消されており，円の色の判別も可能である．計算時間も CPU2.8GHz の計算機で 4 秒程度と短時間で計算することができる．しかし，明確な目的関数を持たずに，逐次，円の中心座標を更新しているため，全体の地理的位置関係を大きく損なってしまう．例えば，図 2.2(c) は特に周縁部で大きく変形しているため，円と市区町村の対応がつきにくい．八王子市は周縁に押し出されているなど，奥多摩・秩父の市町村を表す円を見つけることは難しい．また，中心部でも，例えば市川市・船橋市が南北に配置されるなど局所的な位置関係も大きく変化しており，読図を難しくしている．Dorling (1991, 1995b) の手法では，円の位置関係を大きく崩し，分かりにくい作図を行う可能性がある．

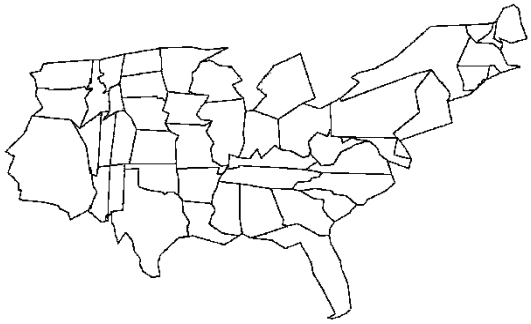


(a) Tobler (1973)
(1980 年人口)
(Kocmoud and House (1998) より引用)

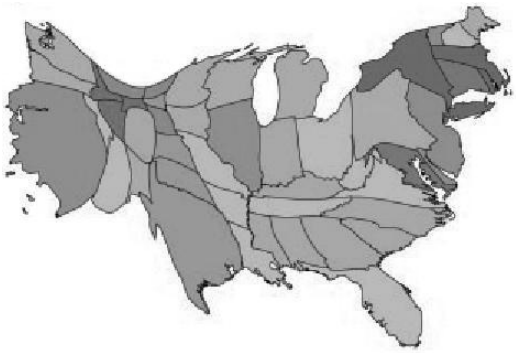


(b) Gusein-Zade and Tikunov (1993)
(1980 年人口)
(Kocmoud and House (1998) より引用)

(c) Kocmoud and House (1998)
(1980 年人口)

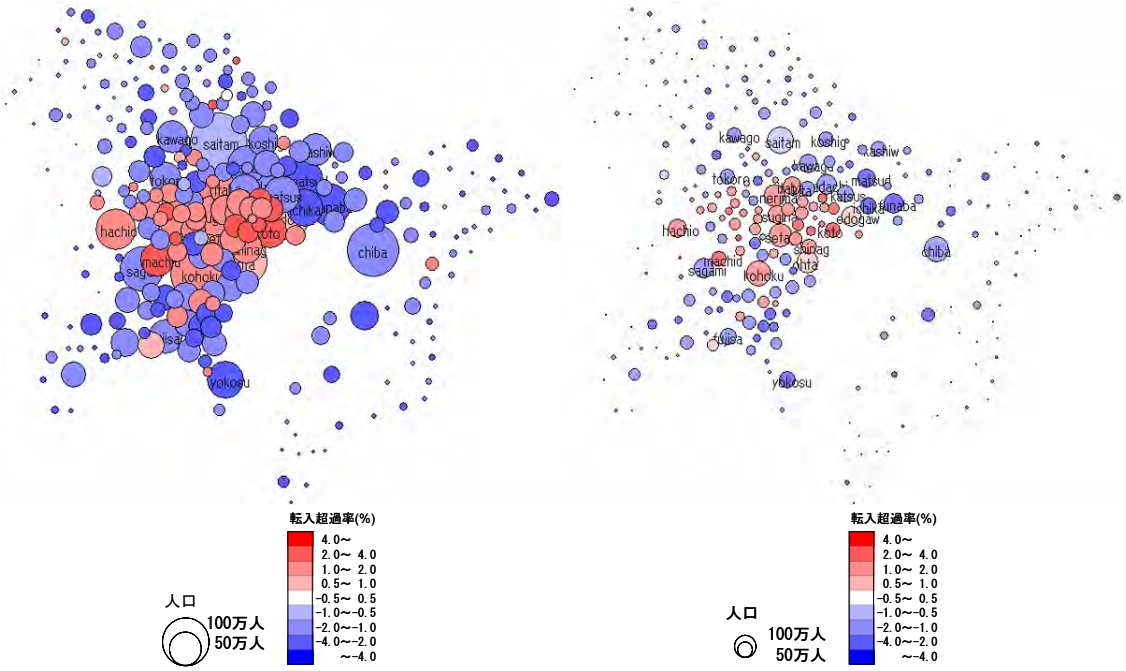


(d) Keim *et al.* (2004)
(1980 年人口)

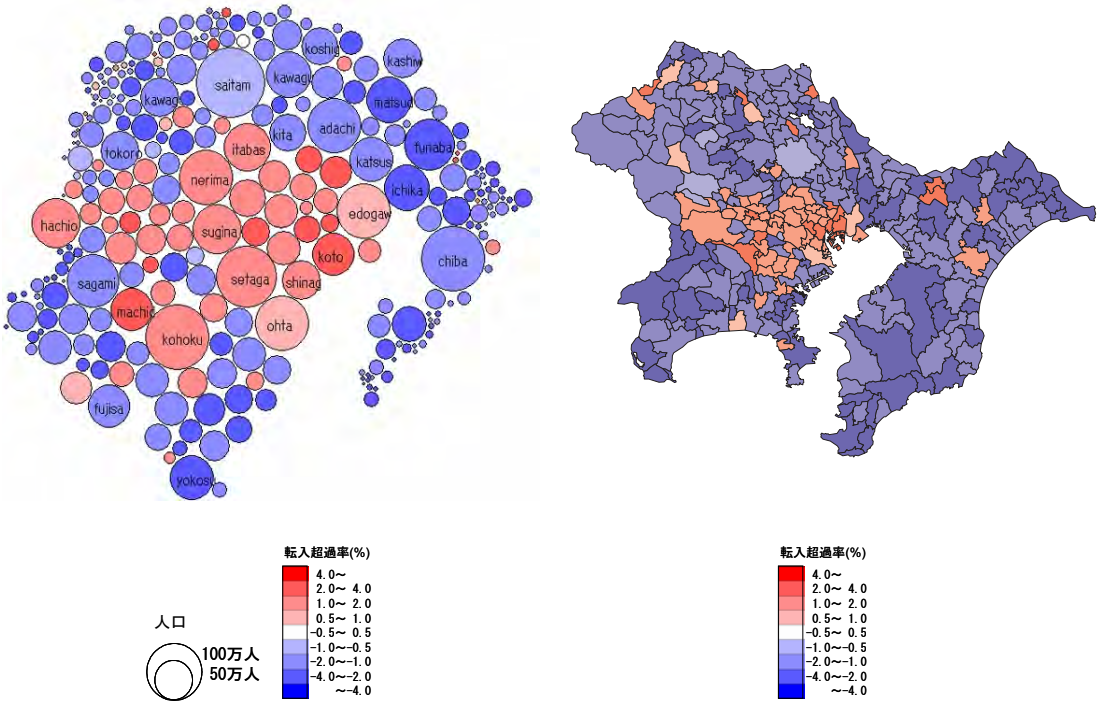


(e) Gastner and Newman (2004)
(2000 年人口)

図 2.1: 既存の連続エリアカルトグラム作成手法
(アメリカ合衆国人口)



(a) 地理的配置 重なり大　　(b) 地理的配置 重なり排除



(c) Dorling (1991) アルゴリズム　　(d) コロプレスマップ

図 2.2: 既存のサークルエリアカルトグラム作成手法
(2000 年南関東市区町村人口・転入超過率)

第3章　ディスタンスカルトグラム

第 2.1 節で述べたように，ディスタンスカルトグラムには，完全ネット型と部分ネット型の 2 種類が提案されている．しかし，完全ネット型ディスタンスカルトグラムは，全ての対象地点間に近接性を与えた部分ネット型ディスタンスカルトグラムといえるので，部分ネット型ディスタンスカルトグラムの数学的な解法を用意すれば，それはディスタンスカルトグラム作成のための汎用解法になる．汎用解法があれば，完全ネット型ディスタンスカルトグラムと部分ネット型ディスタンスカルトグラムを適宜使い分けることができるばかりでなく，両者の利点を活かした折衷法的な応用も容易になる．

第 3 章では，ディスタンスカルトグラム作成問題を非線形最小二乗問題で表現し，これをベースにディスタンスカルトグラムの汎用解法を提示する．

3.1　基本的な考え方

部分ネット型ディスタンスカルトグラムは，第 2.1 節で述べた通り，地点間の近接性指標のみから図形形状を確定できない，解の一意性が保証されない不適切な問題である．そこで，不適切性を解消するための何らかの正則化条件を必要とする．このような非線形最小二乗問題の数値解法はいくつか提案されているので，これらを適用することによりディスタンスカルトグラム作成のための一つの汎用解法を提示することができる．

第 3.2 節では，解の一意性が保証されない非線形最小二乗問題の最も代表的な解法の一つであり、多くの数理計画用のプログラム・パッケージにも採用されている Levenberg-Marquardt 法を用いて，ディスタンスカルトグラム作成のための一つの解法を提示する．しかし，このような一般的な数値解法が，ディスタンスカルトグラム作成という特定の目的に対して実効性の高い解法であるかどうかは議論を要する．

ディスタンスカルトグラムは実地図との比較によって意味を持つ．そのため，地域間の近接性指標の違いを実地図との比較によって分かりやすく表現する必要がある．そこで，近接性指標の再現とは無関係な実地図とディスタンスカルトグラムの相違を極力排除しなくてはならない．また，ディスタンスカルトグラムは，あくまで視覚化の方法であり，政策的な意思決定を直接支援するものではない．個人の思考や集団での議論の過程で適宜用いられるものであり，個人的な興味や遊び感覚での利用も多いだろう．ユーザーの関心・興味によって適宜利用するといった類のものである．そのため，ディスタンスカルトグラム作成の解法は，徹底してユーザー・フレンドリーなものである必要がある．初期値の与え方に試行錯誤的な苦労を強いる解法や，長い計算時間を必要とする解法は，実際のところ役に立たない解法と言わざるをえない．

以上を踏まえ，第 3.3 節では，Levenberg-Marquardt 解法をベースに，その数学的な明快さを保った上で，かつディスタンスカルトグラム作成に要請される特性を備えた実効性の高い汎用解法

を提案する．すなわち，実地図との比較が容易で，かつ計算時間が非常に短い解法を提示する．

第 3.4 節では，Levenberg-Marquardt 解法と筆者による提案解法を鉄道ディスタンスカルトグラムの実問題に適用し，両者の比較を通して，提案解法の有効性を検討する．また，完全ネット型，部分ネット型のディスタンスカルトグラムが持つ必然的な限界について例示する．

第 3.5 節では，筆者の提案解法により，完全ネット型・部分ネット型ディスタンスカルトグラムの各々の利点を活かした折衷法的な応用が可能であることを示す．また，この折衷法的応用が効果的であることを実証することにより，提案解法の意義を明確化する．

3.2　Levenberg-Marquardt 法による汎用解法

本節では，ディスタンスカルトグラム作成問題を非線形最小二乗問題で記述し，Levenberg-Marquardt 法 (藤目, 1999) を用いた一つの解法を構築する．

3.2.1　非線形最小二乗問題による表現

近接性が与えられた，地点 i, j 間のリンク ij の集合を L と表す．ディスタンスカルトグラム作成問題とは，リンク ij のディスタンスカルトグラム上地点間距離 d_{ij} が近接性指標 t_{ij} を再現するように，ディスタンスカルトグラム上の地点配置を定める問題である．

$$\min \sum_{ij \in L} (t_{ij} - d_{ij})^2 \tag{3.1}$$

この d_{ij} は地点のディスタンスカルトグラム上の x, y 座標を用いて，

$$d_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \tag{3.2}$$

と書けることから，ディスタンスカルトグラム作成問題は式 (3.3) の非線形最小二乗問題で表現できる．

$$\min \sum_{ij \in L} \left(t_{ij} - \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \right)^2 \tag{3.3}$$

本論文では，式 (3.3) を基本形として解法を構築する．

3.2.2　Levenberg-Marquardt 法による汎用解法

ここでは Levenberg-Marquardt 法をベースに 1 つの汎用解法を提示する．

Levenberg-Marquardt 法とは，目的関数の変数に近似値を与えて線形化し，その近似値からの変化量のノルム最小化を正則条件として，非線形最小二乗問題を数値的に解く手法である．

まず，変数 x_i, y_i に近似値 x'_i, y'_i を与える．

$$x_i = x'_i + \Delta x_i \tag{3.4a}$$

$$y_i = y'_i + \Delta y_i \tag{3.4b}$$

式 (3.3) を近似値近傍で展開すると，式 (3.5) となる．

$$\min \sum_{ij \in L} \left(t_{ij} - d'_{ij} - \frac{x'_{ij}}{d'_{ij}} \left(\Delta x_j - \Delta x_i \right) - \frac{y'_{ij}}{d'_{ij}} \left(\Delta y_j - \Delta y_i \right) \right)^2 \tag{3.5}$$

但し，$x'_{ij} = x'_j - x'_i$，　$y'_{ij} = y'_j - y'_i$，　$d'_{ij} = \sqrt{x'^2_{ij} + y'^2_{ij}}$ である．

次に，正則化条件として，座標値変化量のノルム最小化を導入する．

$$\begin{aligned} \min \Bigg[\sum_{ij \in L} \left(t_{ij} - d'_{ij} - \frac{x'_{ij}}{d'_{ij}} \left(\Delta x_j - \Delta x_i \right) - \frac{y'_{ij}}{d'_{ij}} \left(\Delta y_j - \Delta y_i \right) \right)^2 \\ + \alpha \sum_{i \in N} \left((\Delta x_i)^2 + (\Delta y_i)^2 \right) \Bigg] \end{aligned} \tag{3.6}$$

ここで，N は対象地点の集合，α は正則化項に対する重みである．

更に，式 (3.4) を用いて Δx_i, Δy_i を消去すると，

$$\min \left[\sum_{ij \in L} \left(t_{ij} - \frac{x'_{ij}}{d'_{ij}} \left(x_j - x_i \right) - \frac{y'_{ij}}{d'_{ij}} \left(y_j - y_i \right) \right)^2 + \alpha \sum_{i \in N} \left(\left(x_i - x'_i \right)^2 + \left(y_i - y'_i \right)^2 \right) \right] \tag{3.7}$$

となる．すなわち，式 (3.7) の線形最小二乗問題を，対象地点の x, y 座標を更新しながら，繰り返し計算によって，地点配置を求めることになる (図 3.1)．

以後本論文では，この解法を L-M 解法と呼ぶ．

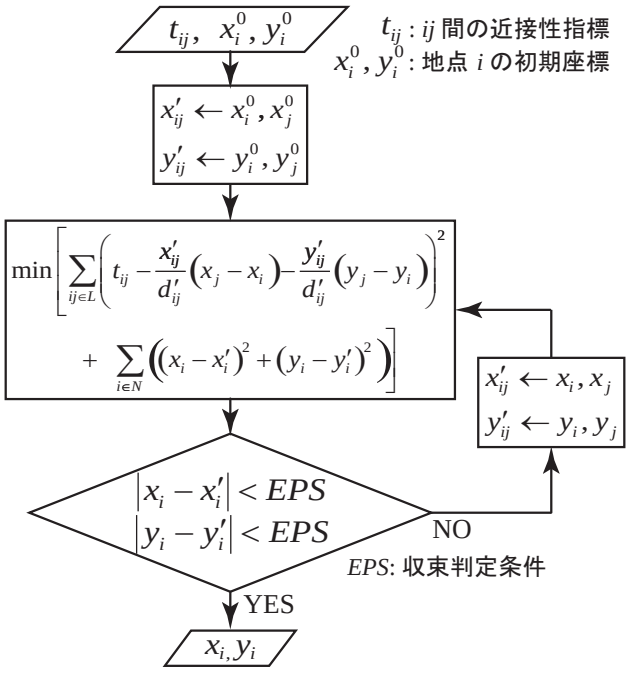


図 3.1: L-M 解法のアルゴリズム

3.2.3　L-M 解法の問題点

L-M 解法は，ディスタンスカルトグラム作成のための解法としては種々の問題を抱えている．L-M 解法は，解の一意性を保証するために，座標近似値からの変化量のノルムを最小化する項を与えて解く方法である．このため，近似値に対してきわめて依存性の高い方法となる．座標近似値が適切な事前情報であれば，この方法は十分に有効であろう．しかし，近接性指標の観測値から地点の配置を求めるディスタンスカルトグラム作成問題では，地点の初期座標の与え方は幾通りも考えられ，どれが適当な初期値であるかは，ユーザー側の試行錯誤の問題となる．

この試行錯誤を，近接性指標の再現精度の最大化だけを目標に行うのであれば，大きな問題ではない．あらかじめ，近似座標を設定するための幾つかの方法を用意しておき，再現精度を最大化するものを出力すればよい．しかし，このようにして得られたディスタンスカルトグラムが，実地図との比較対照に適したディスタンスカルトグラムである保証は何一つない．L-M解法のように初期座標に対して依存性の高い解法は，結局のところ多くの試行錯誤をユーザーに強いることになる．

また，このような試行錯誤を実際上可能ならしめるためには，計算時間が十分に短い解法であることが必要である．長時間の計算を要する方法では，試行錯誤の実用性はない．L-M 解法では，(リンク数) + (対象地点数)×2 の観測方程式を用いて，問題を解くことになる．線形最小二乗問題を LU 分解で解く場合，観測方程式数を M とすると，計算量は $O(M^3)$ となる．従って，対象地点数やリンク数が多い場合，実用性に乏しい方法となる可能性がある．

なお，初期値依存性や計算時間に関しては，第 3.4 節にて検証を行う．

3.3　本論文で提案する汎用解法

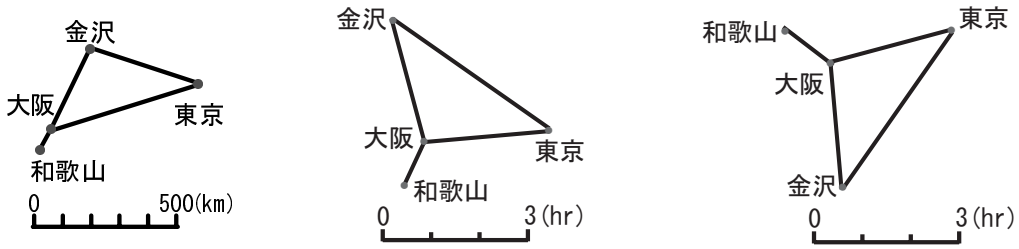
本節では，前節で述べた L-M解法の問題点を踏まえ，筆者による提案解法を示す．

まずは，初期値依存性の問題をどう解消するか，また，実地図との比較対照を容易にするにはどうすればよいかという観点から解法を導出する．そして，計算時間を短縮するべく，この解法をさらに簡便な解法へ改善し，これをもって提案解法とする．

3.3.1　基本的な考え方

L-M 解法の初期値依存性の問題は，地点の座標に対して適切な事前情報を与えるのが難しい点である．換言すれば，ディスタンスカルトグラム作成問題における適切な事前情報とは何かを定めこの事前情報を用いた正則化を行えば，問題は大きく軽減する．

では，ディスタンスカルトグラム作成問題における，適切な事前情報とは何であろうか．ディスタンスカルトグラムは，地点間の近接性の違いを実地図との比較により視覚化する手法である．そのためには，近接性指標の再現とは無関係な，実地図とディスタンスカルトグラムの相違を極力排除した方がよい．例えば，東京，大阪，金沢，和歌山の 4 地点間の鉄道所要時間を表現する部分ネット型ディスタンスカルトグラム (鉄道所要時間ディスタンスカルトグラム)(図 3.2) を考えよう．図 3.2(b)(c) は，時間距離の再現精度は等しい．この時，実地図との比較が容易なディスタンスカルトグラムは，言うまでも無く図 3.2(b) であろう．多くの人は，各地点の地理的位置に対して，図 3.2(a) に近い先験的知識を有しているため，図 3.2(c) のようなディスタンスカルトグラム



(a) 地理的配置　　(b) ディスタンスカルトグラム (1)　　(c) ディスタンスカルトグラム (2)

図 3.2: 1995 年 部分ネット型鉄道所要時間 ディスタンスカルトグラム (例)

は非常に混乱を与える．近接性データの再現を目的としない，実地図のいたずらな変形を避けるということは，ディスタンスカルトグラム作成にとって重要な目標である．

この目標を事前情報による正則化によって達成するため，本論文ではディスタンスカルトグラム上の地点間の方向に関する事前情報として，実地図上の地点間の方向を与えるという考え方をとる．これにより，ディスタンスカルトグラム作成問題を適切化し，かつ，ディスタンスカルトグラムと実地図の相違を視覚的に分かりやすくすることを目指す．

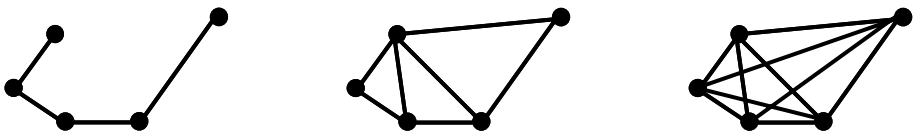
3.3.2　汎用解法の提案

ディスタンスカルトグラム上の地点間の方向に関する事前情報を与えるため，式 (3.7) の正則化項を，地点座標値変化量のノルム最小化から，リンク方位角変化量のノルム最小化へと変更する．

$$\min \sum_{ij\in L} \left[\left(t_{ij} - \frac{x'_{ij}}{d'_{ij}} (x_j - x_i) - \frac{y'_{ij}}{d'_{ij}} (y_j - y_i) \right)^2 + \alpha \left(\frac{y'_{ij}}{d'_{ij}} (x_j - x_i) - \frac{x'_{ij}}{d'_{ij}} (y_j - y_i) \right)^2 \right] \quad (3.8)$$

なお，この解法では，地点間の方向を拘束するだけであるので，ディスタンスカルトグラムを座標系に固定するために，任意の 1 点の座標を固定する必要がある．

まず，式 (3.8) でディスタンスカルトグラム作成が可能であることを確認しておく．対象地点数が n のとき，$n-1$ 地点の x, y 座標が未知変数となり，その数は $2(n-1)$ となる．一方，観測方程式数はネットワーク形状によって異なるが，例えば，図 3.3(a) のような，リンク数が最小のネットワークにおいても，未知変数と同数の観測方程式が存在するため，完全ネット型，部分ネット型に関わらず，常に解を一意に定めることができる (表 3.1)．



(a) 最小　　(b) 三角網　　(c) 最大

図 3.3: ネットワークの形状によるリンク数の違い

表 3.1: ネットワークの形状によるリンク数の違い

地点	未知変数	観測方程式数		
		(a) 最小	(b) 三角網	(c) 最大
n	$2(n-1)$	$2(n-1)$	$4n-6$	$n(n-1)$
5	8	8	14	20
10	18	18	34	90
50	98	98	194	2450

実は，式 (3.8) による解法は非常に便利な特徴を有している．それは，式 (3.8) の x'_{ij}/d'_{ij}, y'_{ij}/d'_{ij} はそれぞれ，ディスタンスカルトグラム上のリンク ij の座北方位角の正弦，余弦の近似値であるということである．すなわち，ディスタンスカルトグラム上のリンクの方向を実地図上のそれと可能な限り合わせたいという正則化の方針を踏まえれば，実地図上の座北方位角をこれらの初期近似値として計算を開始すればよい．また，ディスタンスカルトグラム作成の第一の目的は，実地図との比較であることを考えると，実地図上の対象地点の座標は計算機に入力されていることを前提としてもよいだろう．このとき，式 (3.8) による解法は，初期近似値の設定を自動化できることを意味している．

ここでは，ディスタンスカルトグラム上のリンク ij の座北方位角の近似値を θ'_{ij} として，式 (3.8) を以下のように書き改める．

$$\begin{aligned}\min \sum_{ij\in L} \Big[&\left\{ t_{ij} - (x_j - x_i)\sin\theta'_{ij} - (y_j - y_i)\cos\theta'_{ij} \right\}^2 \\ &+\alpha \left\{ (x_j - x_i)\cos\theta'_{ij} - (y_j - y_i)\sin\theta'_{ij} \right\}^2 \Big]\end{aligned} \tag{3.9}$$

なお，L-M 解法も式 (3.8) による解法も，事前情報との折衷による適切化である．このとき，一般的には α 値は，$\alpha > 0$ の領域で解の安定性を損なわない範囲で小さく設定し，収束速度の高速化を図る．しかし，ここでは簡便化のため，$\alpha = 1$ とし，式 (3.9) を以下のように展開する．

$$\min \sum_{ij\in L} \left[\left(t_{ij}\sin\theta'_{ij} - (x_j - x_i) \right)^2 + \left(t_{ij}\cos\theta'_{ij} - (y_j - y_i) \right)^2 \right] \tag{3.10}$$

これにより，x, y 軸方向に独立させた線形最小二乗問題に帰着させることができる．式 (3.10) では，独立に求めた対象地点の x, y 座標を用いて θ_{ij} の近似値 θ'_{ij} を更新しながら，繰り返し計算にてディスタンスカルトグラム上の地点配置を求めることになる (図 3.4).

なお，$\alpha = 1$ としても，L-M 解法と比較して十分な計算速度を確保できることを第 3.3.3 節に記す.

本論文では，式 (3.10) に基づく解法を筆者の提案解法とする．第 3.5 節では式 (3.10) による解法を重み付け最小二乗法に拡張する方法を示すが，これは提案解法の応用と位置づけられる.

なお，ディスタンスカルトグラム作成プログラムのソースコードを付録 B.2 に記す.

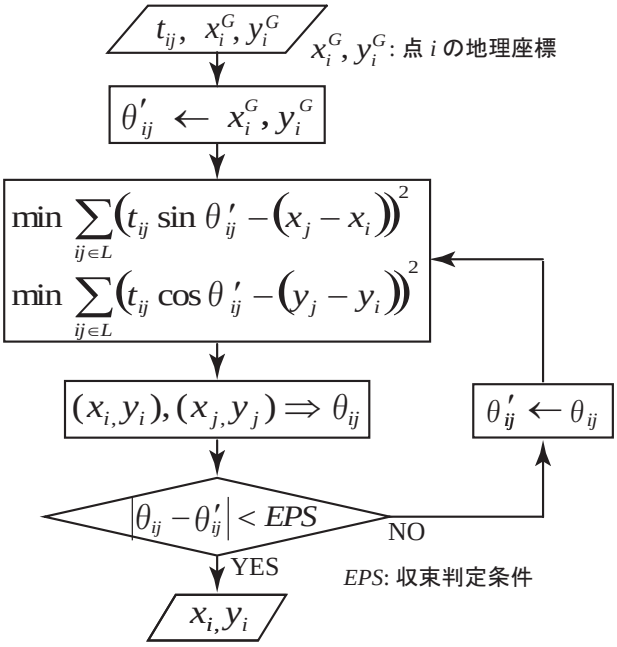


図 3.4: ディスタンスカルトグラム 提案解法のアルゴリズム

3.3.3　提案解法の特徴

提案解法は以下のような利点を有する．

まず，提案解法では，初期値は各リンクの実地図上の座北方位角を用いて与えられる．座北方位角は，対象地点の実地図座標から計算でき，初期値設定を自動化できる．また，ディスタンスカルトグラム上のリンクの方向が，実地図上のそれに近い方向で表現されるため，実地図形状からのいたずらな変形を排除することが可能であり，L-M 解法と比較して実地図との比較が行いやすいディスタンスカルトグラムを作成することができる．

また，提案解法は計算時間の点においても L-M 解法と比較して優れている．先に述べた通り，LU 分解を用いる場合，線形最小二乗問題は，観測方程式数を M とすると $O(M^3)$ の計算量を要する．収束計算 1 回当たり，L-M 解法では，(リンク数) + 2×(対象地点数) の観測方程式の問題を解くのに対して，提案解法では，(リンク数) の観測方程式の問題を x 軸方向，y 軸方向で 2 度解くことになる．したがって，対象地点数が多い場合には，提案解法が圧倒的に有利な解法となる．加えて，線形最小二乗問題を LU 分解で解く場合，説明変数行列は同じで被説明変数ベクトルのみを変更して再計算を行うのであれば，その計算量は $O(M^2)$ しか要しない (Press *et al.*, 1988)．式 (3.7) と式 (3.10) を比較すれば明らかなように，提案解法は L-M 解法とは異なって，この利点を享受できる解法である．提案解法が L-M 解法と比べて，計算時間の面でいかに優れた解法であるかが分かる．

このように，提案解法は，初期値設定を自動化でき，計算時間が短く，かつ，実地図との比較が容易な地図作成が可能であり，実用性が高い．これらの特徴について，第 3.4・3.5 節で確認する．

3.4　L-M 解法と提案解法の適用

L-M 解法および提案解法を用いて，完全ネット型ディスタンスカルトグラムと部分ネット型ディスタンスカルトグラムを作成し，これらの解法の適用可能性を検討する．

国土交通省整備のデータベース TRANET から 1965・75・85・95 年の都市間 (生活圏間) 鉄道所要時間データ (以後，時間距離) を用いた．

対象地点は 81 都市とし，完全ネット型鉄道所要時間ディスタンスカルトグラムでは 3240 リンク，部分ネット型鉄道所要時間ディスタンスカルトグラムでは図 3.5 上に示される 109 リンクの時間距離を表現する．

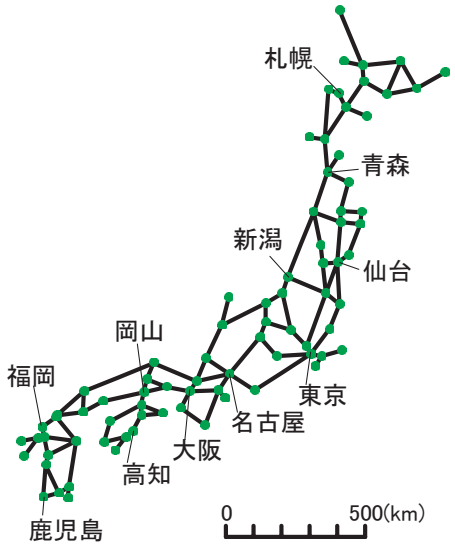


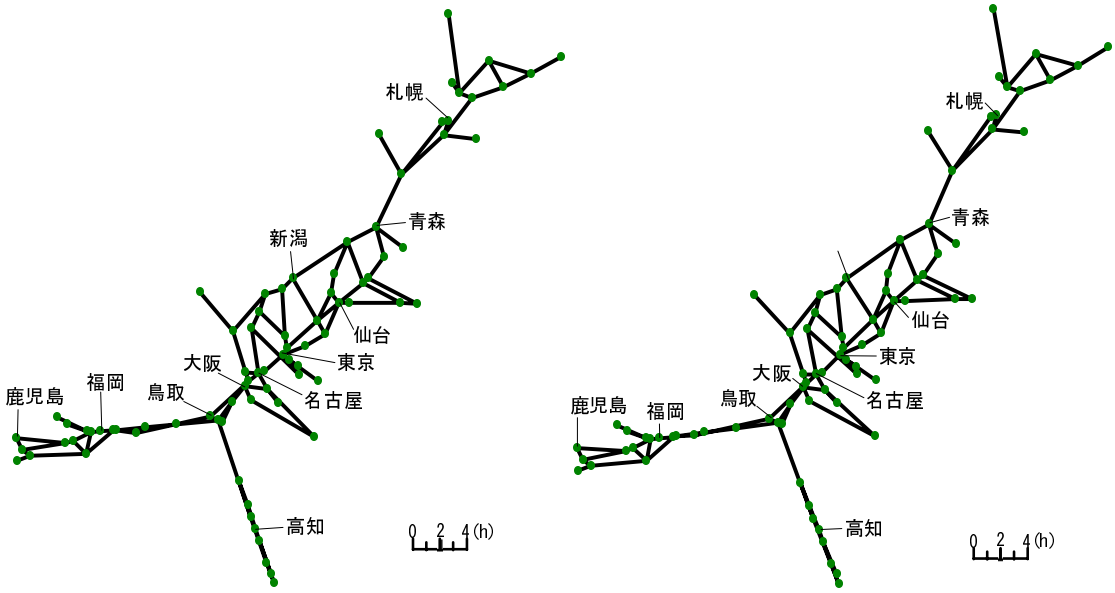
図 3.5: 日本 主要鉄道網

3.4.1　適用と評価

初期値については，L-M 解法では，対象とするリンクの距離の総和が時間距離の総和と等しくなるように，実地図のスケールを変更して対象地点の座標値を与えた．提案解法では，各リンクの実座北方位角を与えた．収束計算では，L-M 解法では，全対象地点の x, y 座標値の変化が 0.01 時間以下を，提案解法では全リンクの角度変化が $0.01rad$ 以下をそれぞれ収束判定条件とした．

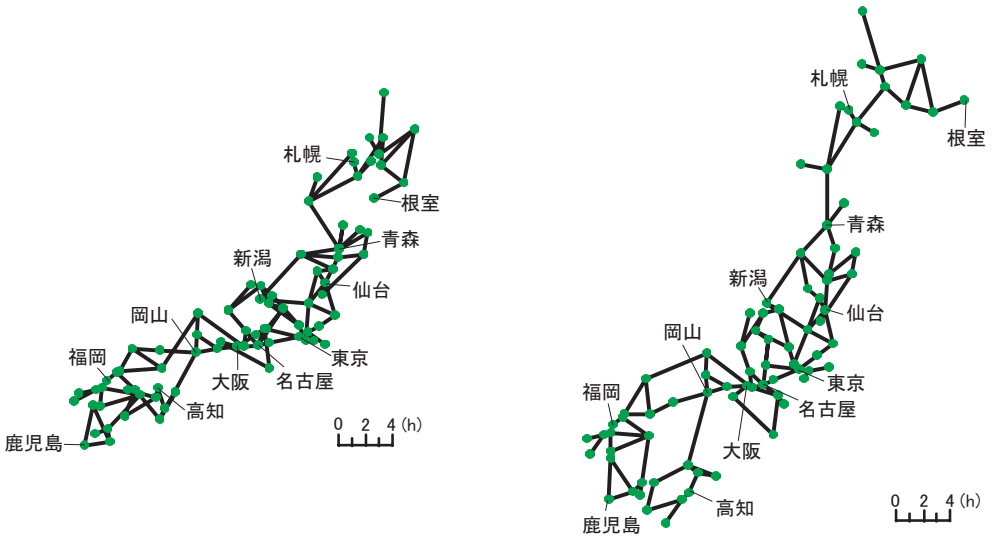
両解法による 1965 年の完全ネット型ディスタンスカルトグラムの作成結果を図 3.6，部分ネット型ディスタンスカルトグラムの作成結果を図 3.7 に示す．

データの表現精度の評価指標として時間距離データとディスタンスカルトグラム上の距離の相関係数・Kruskal の STRESS2(Cox and Cox, 2001) を用い，表 3.2 に記載する．但し，部分ネット型ディスタンスカルトグラムでは，図 3.5 に示すリンクの時間距離・ディスタンスカルトグラム上距離の適合度を表す．また，ディスタンスカルトグラム上地点配置と地理的地点配置の形状類似度を評価するために，ヘルマート変換の自由度修正済み決定係数 (Dryden and Mardia, 1998) とリン



(a) L-M 解法　　(b) 提案解法

図 3.6: 1965 年 完全ネット型 鉄道所要時間ディスタンスカルトグラム



(a) L-M 解法　　(b) 提案解法

図 3.7: 1965 年 部分ネット型 鉄道所要時間ディスタンスカルトグラム

表 3.2: ディスタンスカルトグラム データ表現精度

	L-M 解法		提案解法	
	相関係数	STRESS2	相関係数	STRESS2
完全ネット型	0.995	0.101	0.995	0.101
部分ネット型	1.000	0.026	1.000	0.007

表 3.3: ディスタンスカルトグラム・地理的地点配置 形状比較
ヘルマート変換 自由度修正済み決定係数・方位角 RMSE

	L-M 解法		提案解法	
	Helmert 変換 R^2	方位角 RMSE	Helmert 変換 R^2	方位角 RMSE
完全ネット型	0.8707	32.1	0.8711	31.9
部分ネット型	0.9687	59.1	0.9457	27.4

(方位角 RMSE: 度)

表 3.4: 完全ネット型ディスタンスカルトグラム作成計算の比較

西暦	L-M 解法		提案解法	
	収束計算 (回)	計算時間 (秒)	収束計算 (回)	計算時間 (秒)
1965	22	9.87	97	0.59
1975	22	9.86	144	0.77
1985	48	21.37	81	0.56
1995	17	7.67	69	0.53

(CPU 2.8GHz の計算機使用)

表 3.5: 部分ネット型ディスタンスカルトグラム作成計算の比較

西暦	L-M 解法		提案解法	
	収束計算 (回)	計算時間 (秒)	収束計算 (回)	計算時間 (秒)
1965	64	2.22	41	0.08
1975	96	4.17	42	0.08
1985	108	3.31	53	0.08
1995	95	3.30	51	0.08

(CPU 2.8GHz の計算機使用)

クの方位角の RMSE を用いて表 3.3 に示す．また，1965・1975・1985・1995 年データによるディスタンスカルトグラム作成に要する収束計算回数と計算時間を表 3.4・3.5 に示す．

まず，両解法により，完全ネット型・部分ネット型ディスタンスカルトグラムの双方を作成できることを確認した．

完全ネット型ディスタンスカルトグラム (図 3.6) の比較を行うと，L-M 解法・提案解法の結果の間に顕著な違いは見られない．どちらも，時間距離とディスタンスカルトグラム上の距離の相関係数は 0.995，STRESS2 は 0.101 であり，時間距離を十分に再現している (表 3.2)．また，地理的形状とカルトグラム形状の類似度でも，ヘルマート変換の決定係数・方位角変化量ともに顕著な差は見られない (表 3.3)．収束計算回数の比較 (表 3.4) では，提案解法の方が多くなる．これは，観測方程式数が，L-M 解法 (式 (3.7)) は主問題 3240・正則化項 160 となるのに対し，提案解法 (式 (3.8)) では主問題 3240・正則化項 3240 となるため，提案解法では，正則化項が相対的に大きくなり，収束速度が低下するためである．しかし，計算時間の比較では，提案解法が明らかに優れている．

部分ネット型ディスタンスカルトグラム (図 3.7) では，両解法とも相関係数は 1.000，STRESS2 も小さく，時間距離をほぼ正確に再現している．しかし，L-M 解法による結果は提案解法に比べて形状の崩れが大きい．L-M 解法によるディスタンスカルトグラムは，折りたたまれて複雑にリンクが重なり合っており視認性に劣る．例えば，釧路・根室間のリンクは，実地図上の方向と大きく異なりとても違和感を覚える．これは，初期値として与えた座標近似値の近傍の解が求められたことに原因がある．L-M 解法の初期値設定の困難さが顕在化した例である．一方，提案解法では，ディスタンスカルトグラム上のリンクの方向を実地図上の方向の近傍で表現しており方位角変化も小さい (表 3.3) ため，実地図に比較的近い形状を持ったディスタンスカルトグラムを作成できた．しかし，見た目では提案解法によるディスタンスカルトグラムが明らかに分かりやすいにも関わらず，ヘルマート変換の自由度修正済み決定係数では L-M 解法の結果の方が良好な結果を示す．これは，L-M 解法が地理的地点配置からの座標変化を抑制する解法であるため，求まる解が地理的形状に近いものとなり必然的にヘルマート変換の適合度が高くなるためである．ヘルマート変換の適合度は地点配置の類似度を計測する手法の一つではあるが，ディスタンスカルトグラム形状の優劣を判断する手法としては適さないことが分かる．収束計算回数・計算時間に関しては，提案解法が優れていることが確認でき (表 3.5)，提案解法は初期値設定に調整を要さず簡潔・高速に作図が可能な手法であるといえる．

3.4.2　ディスタンスカルトグラムの限界

完全ネット型・部分ネット型ディスタンスカルトグラムの作成例を示したが，これらは決して分かりやすいとは言えない部分がある．ここでは，2 種類のディスタンスカルトグラムが必然的に持つ限界について述べる．

完全ネット型ディスタンスカルトグラムでは，全ての地点間の時間距離を高い精度で表現しているものの，局所的に見ると分かりにくいところが多い．北海道・四国・九州等の周辺部では，大きく変形し，見づらい地図になってしまう．特に，四国は，本州・九州などの東西両方向からの時間距離を表現しようとするために，ほとんど一直線上に配置されてしまう．この結果，四国内の時間距離の表現精度は低下している．四国内の時間距離とディスタンスカルトグラム上の距離の相関係数は 0.666 であり，全体 (相関係数 0.995) に比べて極端に低い．このように，完全ネット型ディス

タンスカルトグラムでは，全体の傾向を示すことは可能だが，局所的には距離指標データを十分に再現できない場合がある (図 3.8).

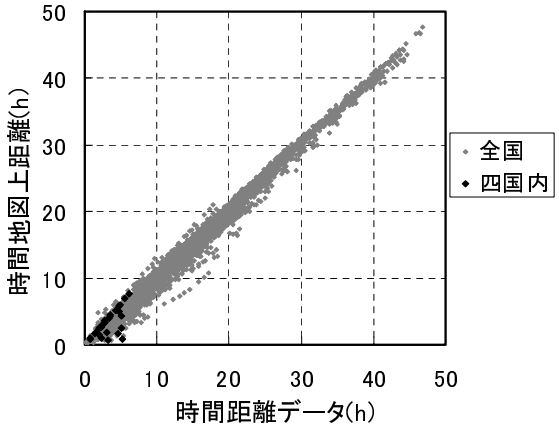


図 3.8: 提案解法による 1965 年 鉄道完全ネット型ディスタンスカルトグラム データ表現精度

一方，部分ネット型ディスタンスカルトグラムは，実地図上の形状を大まかに維持しており，全体としては視覚的に理解しやすい．また，時間距離を正確に再現している．しかし，直接リンクが設定されていない地点間の時間距離が無視されるため，これに伴う問題が生じている．例えば，四国・九州間は時間距離が長いにも関わらず，L-M 解法の結果 (図 3.7(a)) では四国と九州が重なって配置され，提案解法の結果 (図 3.7(b)) においても四国と九州は非常に近接して描かれている．このように，部分ネット型ディスタンスカルトグラムは，交通整備がなされていない海峡部や広域の閉鎖湾域が対象地域に含まれる場合には，適用に十分な注意を要する．

なお，ここで述べた問題点は，完全ネット型・部分ネット型ディスタンスカルトグラムが一般的に持つ限界であって，提案解法の限界ではないことに留意されたい．提案解法は，日本列島の鉄道所要時間ディスタンスカルトグラムの作成、首都圏における道路所要時間ディスタンスカルトグラムの作成といった個別問題に対して柔軟に対応できる方法である．このことについて，第 3.5 節で具体的に述べる．

3.5　提案解法の個別問題への応用

3.5.1　基本的な考え方

筆者による提案解法は，完全ネット型・部分ネット型の双方に適用可能な解法であり，両者の折衷法的な応用を可能にする．

前節における完全ネット型ディスタンスカルトグラムへの適用では，四国に見られるように局所的に再現精度が極端に低下するという問題が生じた．このような場合，時間距離が極端に大きなリンクを削除し，完全ネット型ディスタンスカルトグラムの問題を，リンク数が多いながらも部分ネット型ディスタンスカルトグラムの問題に置き換えて再計算するといった対応が可能である．また，部分ネット型ディスタンスカルトグラムへの適用では，四国と九州がディスタンスカルトグラム上で極端に近く配置されてしまうという問題が生じた．このような場合には，四国・九州間に仮想的なリンクを設定し，そのリンクに完全ネット型の時間距離 (宇高連絡船，関門トンネルを経由する時間距離) を与え，よりリンク数の多い部分ネット型ディスタンスカルトグラムの問題に置き換えるといった対応が可能である．

また，提案解法は線形最小二乗問題の繰り返し計算による解法であるため，重み付き線形最小二乗問題による解法に拡張することが容易である．この拡張により，地点間の時間距離の再現精度に重みを与えることができ，上記のような折衷法的な対応をさらに柔軟に適用することが可能になる．なお，言うまでも無く，完全ネット型ディスタンスカルトグラムを重み付き最小二乗問題で解く場合，重みをゼロとするリンクを設定することにより，完全ネット型ディスタンスカルトグラムを部分ネット型ディスタンスカルトグラムに変更することができる．

以下では，提案解法の個別問題への応用例を提示し，その有効性を確認することにより，筆者による提案解法の意義をより明確に示すこととする．

3.5.2　個別問題への応用例

完全ネット型，部分ネット型ディスタンスカルトグラムの限界を踏まえ，以下に示すような折衷法的な応用により，1965 年の鉄道所要時間ディスタンスカルトグラムの作成を再実行した．

重み付き最小二乗法を用いてディスタンスカルトグラムを作成する．まず，時間距離が 20 時間を超えるリンクの時間距離再現の重みをゼロとした．すなわち，これらのリンクを削除し，部分ネット型ディスタンスカルトグラムの問題に変更した．なお，東京起点のリンクに関しては，終点が札幌から鹿児島までのリンクの時間距離が 20 時間より短い．完全ネット型の 3240 リンクのうち，20 時間を超える 955 リンクが削除されたが，日本国内主要リンクの大半が計算対象に含まれている．残されたリンクのうち，図 3.5 のネットワークに示される各リンクの時間距離の重みを 1 に，それ以外のリンクの時間距離の重みを 0.02 として計算を実行した．なお，この重みの大きさは，分かりやすいディスタンスカルトグラムとなるよう試行錯誤の上，決定した．

簡単に言えば，基本的には図 3.5 のリンクの時間距離を精度良く再現することを目指すが，これらのリンク以外の地点間時間距離についても，極端に大きくない限り，少しは考慮するという方針をとったということである．これにより，局所的な (例えば，四国内の) 再現精度と広域的な (例えば，四国・九州間の) 再現精度をバランスさせることを目指した．

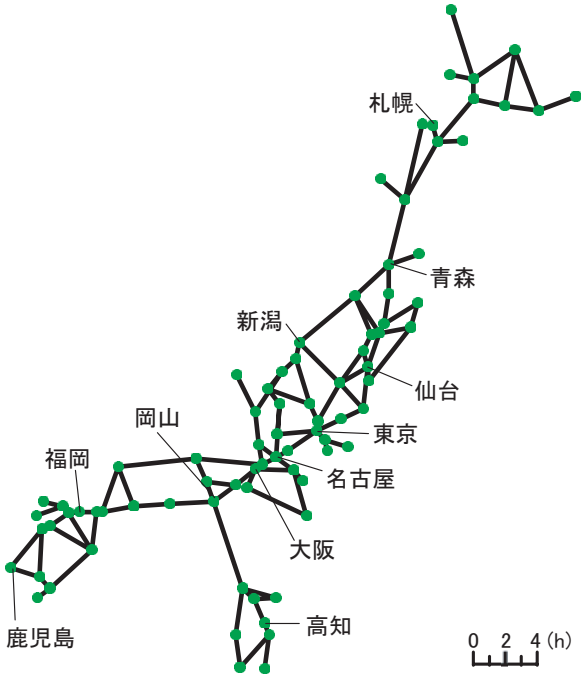


図 3.9: 折衷法による 1965 年 部分ネット型鉄道所要時間ディスタンスカルトグラム

以上の折衷法的応用による計算結果を示したのが図 3.9 である．図 3.7 で示した部分ネット型ディスタンスカルトグラムとの比較のため，これと同じネットワーク (図 3.5) だけを表示している．重みを 1 とした図 3.5 のネットワーク上のリンクは，時間距離とディスタンスカルトグラム上距離の相関係数が 0.999 とほぼ正確に表現されている．また，全ての地点間では，相関係数 0.964 と全体の傾向を示している．また，局所的に時間距離再現精度が著しく低い地域も見られず，視覚的にも理解しやすいカルトグラムが得られた．

このように，筆者の提案解法は，リンクの削除や重視するリンクの指定が自由に行えるため，ユーザーのディスタンスカルトグラム作成の目的・対象に応じて，柔軟に対応が可能な視覚化手法である．

3.5.3　ディスタンスカルトグラムによる交通所要時間変遷の視覚化

第 3.5.3 節では，ディスタンスカルトグラムを用いて交通所要時間の変遷を視覚化した例を示す．

ディスタンスカルトグラムを用いて時系列比較を行う場合には，前後の (t 期と $t+1$ 期の) ディスタンスカルトグラムとの比較を容易にする必要がある．そこで，$t+1$ 期のディスタンスカルトグラムの作成にあたっては，t 期のディスタンスカルトグラム上での対象地点間の座北方位角を初期値として与えるのが適当である．これにより，交通条件の変化以外のディスタンスカルトグラム間の歪みを抑え，時系列的な比較が容易になる．

日本 鉄道所要時間の変遷

1965・75・85・95 年の鉄道時間距離データを使用し，上記第 3.5.2 節の手法で作成した結果を図 3.10 に示す．但し，1965 年時点で時間距離が 20 時間以下の 2285 リンクを計算対象にしている．初期値には，1965 年は実座北方位角を，1975 年以降は，前期のディスタンスカルトグラム上の座北方位角を与えた．計算時間を表 3.8 に，時間距離とディスタンスカルトグラム上の距離の相関係数を表 3.6 に示す．

図 3.10 では，鉄道網整備が進むにつれて，日本列島全体の時間距離が縮んだ様子が表現されている．ディスタンスカルトグラム間のリンクの角度変化が小さいため，各リンクの時間距離変化が見やすく，ディスタンスカルトグラム間の比較が容易である．また，いずれのディスタンスカルトグラムも，図 3.5 のネットワークのリンクでは，ほぼ完全に時間距離を再現しており，また計算の対象とした地点間全体においても高い再現性を得た (表 3.6)．

表 3.6: 日本 鉄道所要時間 表現精度
(相関係数)

西暦	全　　体	ネットワーク
1965	0.964	0.999
1975	0.954	0.999
1985	0.944	0.999
1995	0.932	1.000

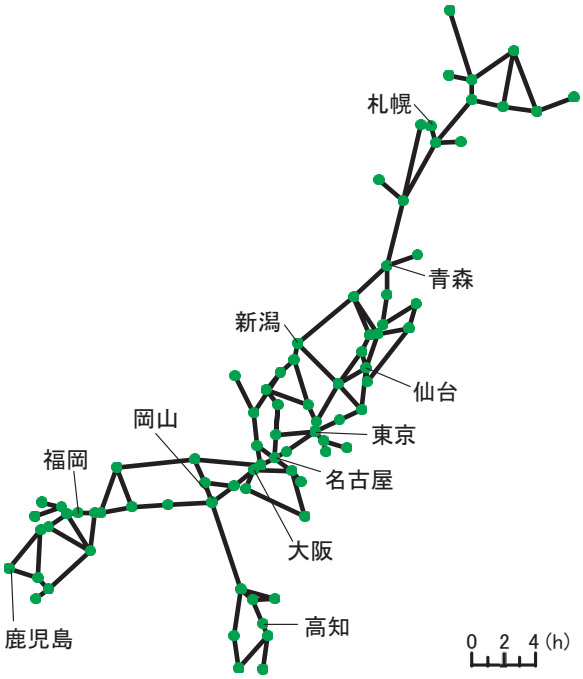
表 3.7: 日本 鉄道所要時間 表現精度
(RMSE: 時間)

西暦	全　　体	ネットワーク
1965	1.550	0.054
1975	1.269	0.046
1985	1.245	0.042
1995	1.082	0.033

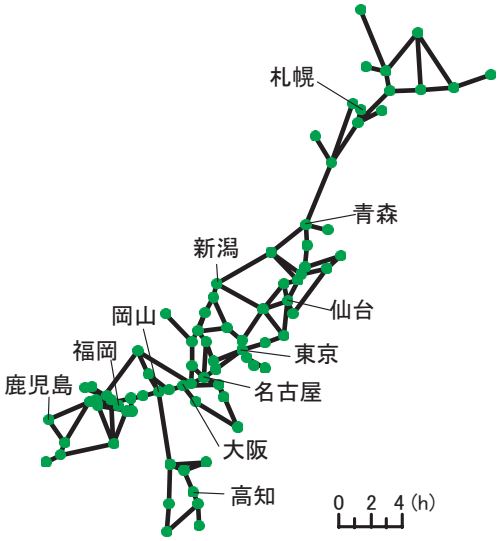
表 3.8: 日本 鉄道所要時間 作成計算
収束計算回数・計算時間

西暦	収束計算 (回)	計算時間 (秒)
1965	78	0.64
1975	210	1.23
1985	49	0.55
1995	91	0.73

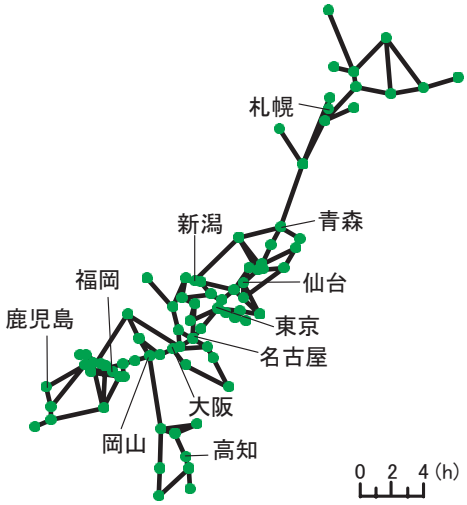
(CPU 2.8GHz の計算機使用)



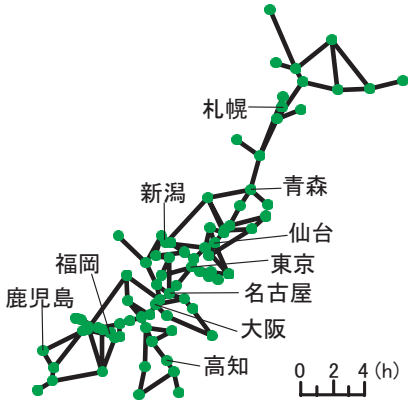
(a) 1965 年



(b) 1975 年



(c) 1985 年



(d) 1995 年

図 3.10: 日本 鉄道所要時間変遷の視覚化

日本 道路所要時間の変遷

国土交通省より提供して頂いた 1955 年・1980 年・2001 年・高規格幹線道路整備後の道路所要時間データを使用し，道路所要時間の変化を視覚化した例を図 3.14 に示す．視覚化対象の都市は 41，リンク数は 63 で，道路網は図 3.13 である．各年の高速道路網は，1955 年は未整備，1980・2001 年はそれぞれ図 3.11，将来は図 3.12 全てが完成した場合としている．また，各年の道路所要時間は，高速道路未整備区間は一般道経由，高速道路整備区間は高速道路経由とし，国土交通省整備のデータベース NAVINET に収録されている現在の所要時間を用いて算出している．

表 3.9: 日本 道路所要時間 表現精度
(相関係数)

西暦	全　　体	ネットワーク
1955	0.997	0.998
1980	0.997	0.985
2001	0.996	0.992
2030	0.998	0.994

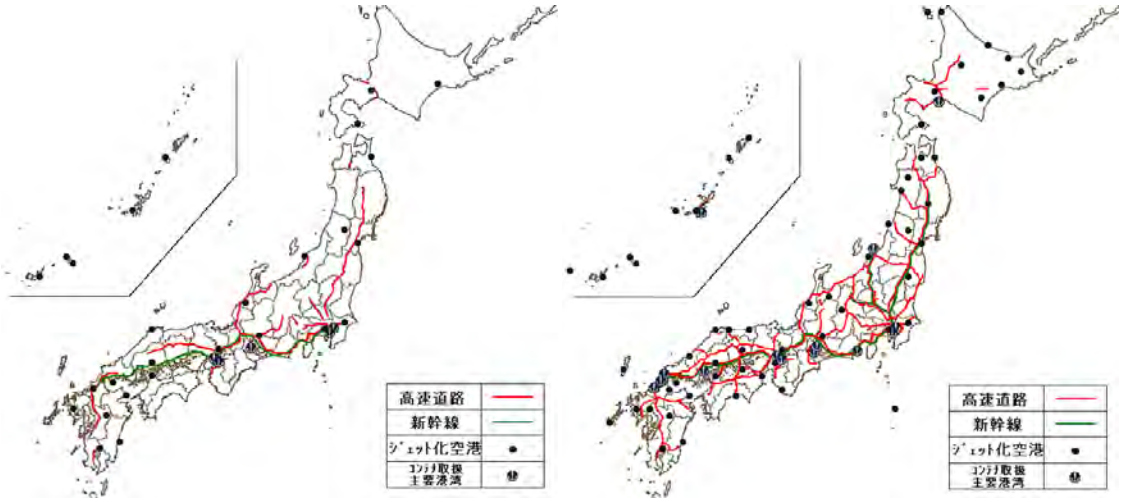
表 3.10: 日本 道路所要時間 表現精度
(RMSE: 時間)

西暦	全　　体	ネットワーク
1955	0.877	0.093
1980	0.890	0.215
2001	0.830	0.143
2030	0.507	0.090

表 3.11: 日本 道路所要時間 作成計算
収束計算回数・計算時間

西暦	収束計算 (回)	計算時間 (秒)
1955	51	0.08
1980	29	0.07
2001	108	0.12
2030	62	0.09

(CPU 2.8GHz の計算機使用)



(a) 1980 年　　(b) 2001 年

図 3.11: 高速道路網 (出典：国土交通省資料)

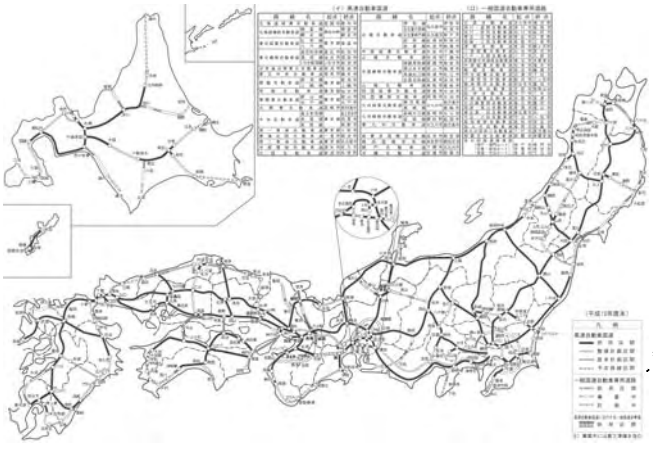


図 3.12: 高規格幹線道路網 (出典：国土交通省資料)

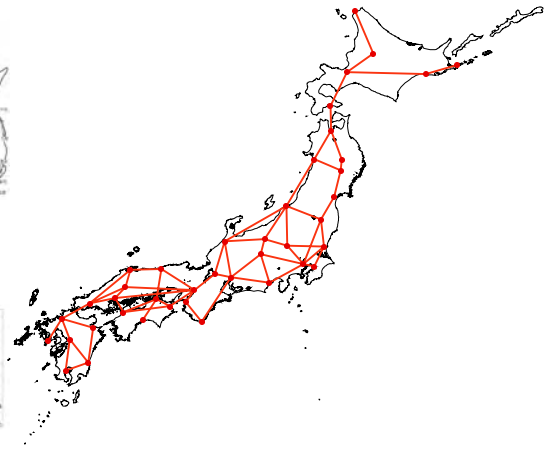
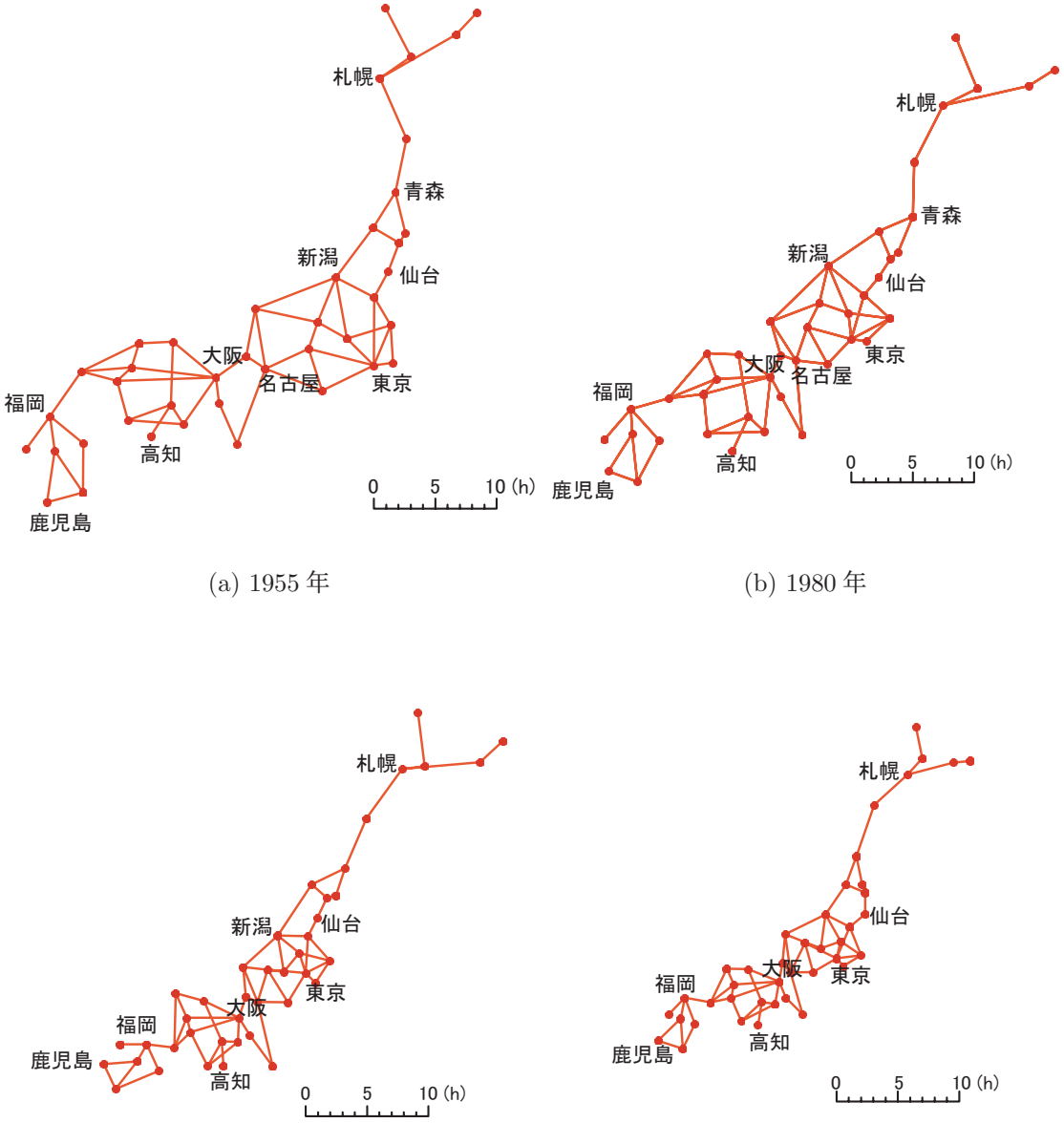


図 3.13: 視覚化対象道路網



(a) 1955 年　(b) 1980 年

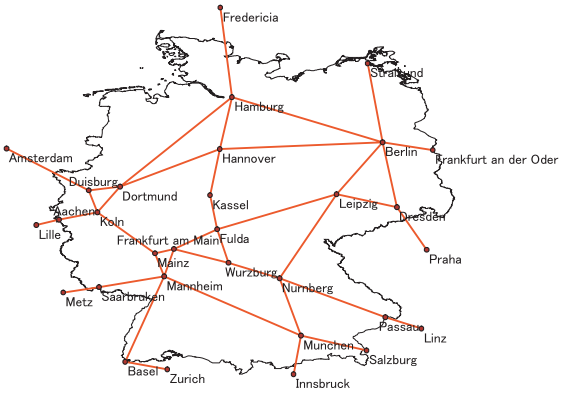
(c) 2001 年　(d) 将来

図 3.14: 日本 道路所要時間変遷の視覚化

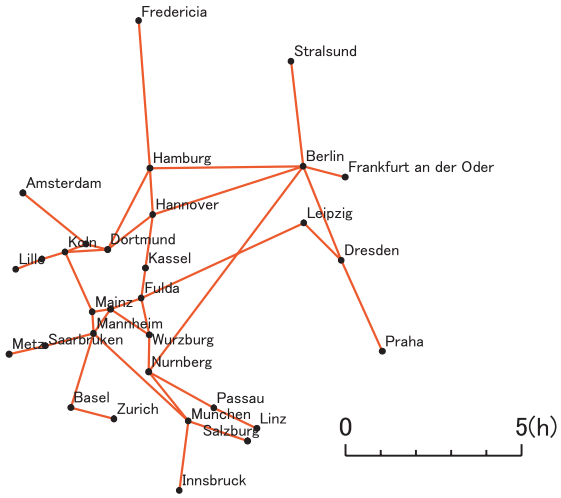
ドイツ 鉄道所要時間の変遷

Spiekermann and Wegener (1993) に付属のデータを用いて，1985・1993・2010 年のドイツ鉄道所要時間をディスタンスカルトグラムを用いて視覚化する．視覚化対象の都市は 31，リンクは 40(但し，1985 年は 39)(図 3.15(a)) であり，鉄道所要時間はリンク上にのみ与えられている．

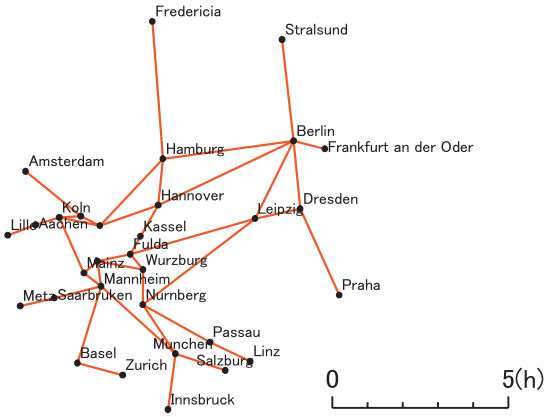
作成結果を図 3.15(b)(c)(d) に記す．計算時間は CPU2.8GHz の計算機で 0.05 秒未満である．



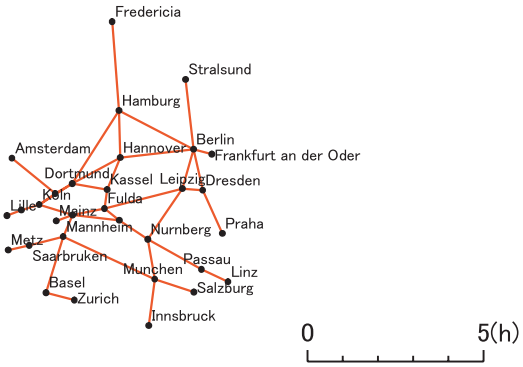
(a) 視覚化対象鉄道ネットワーク配置



(b) 1985 年 所要時間
(相関係数: 0.9998，RMSE: 1.7 分)



(c) 1993 年 所要時間
(相関係数: 0.9995，RMSE: 2.1 分)



(d) 2010 年 所要時間
(相関係数: 0.9993，RMSE: 1.4 分)

図 3.15: ドイツ 鉄道所要時間変遷の視覚化

3.6　まとめ

本論文で提案したディスタンスカルトグラム作成問題の汎用解法の特徴をまとめる．

1. 完全ネット型ディスタンスカルトグラム，部分ネット型ディスタンスカルトグラムの双方に適用可能な汎用解法である．
2. ディスタンスカルトグラム作成問題を適切化するために，ディスタンスカルトグラム上の地点間の方向に関する事前情報として，その地点間の実地図上の方向を与える解法である．近接性指標の再現を目的としない，実地図からディスタンスカルトグラムへの無意味な変形を極力排除することができ，結果として実地図との比較対照が容易になる．
3. 簡単な線形最小二乗問題の繰り返し計算によって，きわめて高速に解を得ることができる．
4. 計算に必要な初期値は実地図上の地点座標から自動的に計算される座北方位角だけであり，複雑な初期値設定を必要としない．
5. 重み付き最小二乗問題への拡張により，完全ネット型，部分ネット型の折衷法的な応用が可能であり，ユーザーが対象とする個々のディスタンスカルトグラム作成問題に柔軟に対応することができる．

本章では，方位角変化を抑制することで視覚的に分かりやすいディスタンスカルトグラムの作図が可能な作成手法を提案した．しかし，ディスタンスカルトグラム形状の視覚的分かりやすさについては十分に定量的な評価・分析はできていないのが実情である．これを可能とするためには，アンケート等の手法を用いて人間の形状認知について分析を行う必要がある．

また本章では，都市間の時間距離を表現する例を用いてディスタンスカルトグラム作成手法について示した．もちろん本手法は地点間所要時間以外にも，認知距離指標に基づく認知空間の視覚化，統計データ等から定義される都市間の非類似度指標に基づく都市類型の視覚化など，多様な分野への応用を可能にする．

以上のように，ディスタンスカルトグラムの形状評価手法の構築と作成手法のより広範囲への応用とを今後の課題とする．

第4章　連続エリアカルトグラム

連続エリアカルトグラム作成問題とは，地図上の面積が表現するデータに一致するように地域形状を変形する問題である．第 2.2 節で述べたように，既存の作成手法は不明快なアルゴリズム，複雑な初期値設定，大きな地域形状変形等の問題を抱えている．ソフトウェアとして実装される例はほとんど見られず，その結果，これらの作成手法を利用した連続エリアカルトグラム作成は稀であった．しかし，連続エリアカルトグラムは有力なデータ視覚化手法であると認識されており，ユーザーフレンドリーな作成ツールの開発を行えば，その利用は増大すると予想される．

そこで，第 4 章では，簡潔な操作を通して連続エリアカルトグラム作成可能なツール構築に向けて作成手法を提案する．具体的には，数学的に明快かつ簡潔な手法の構築を行うため，任意形状の地域を三角網分割することを前提とし，三角形の面積を統計データに合わせるよう変形するアルゴリズムを提案する．また，正則化項として地理的地図からの変形を抑制する条件を導入し，視覚化された統計データの解釈が容易な連続エリアカルトグラム作成を行うことを目指す．

4.1　基本的な考え方

本論文では，連続エリアカルトグラム作成ツール構築を視野に入れ，連続エリアカルトグラム作成に関して以下の基本的考え方をもって手法の構築を行う．

第一に，作成手法は数学的に明快でなければならない．連続エリアカルトグラム作成手法をソフトウェアとして社会一般に提供する場合には，パラメータ設定等に多くの試行錯誤をソフトウェア利用者に強いる手法や長時間の計算を要する手法は実用的ではない．特に，複雑なアルゴリズムを用いた作成手法では，利用者にとってはブラックボックスとなってしまい，パラメータの意味解釈ができないため合理的なパラメータ設定が行えない．このような作成手法は操作性に乏しいと言わざるを得ない．真にユーザーフレンドリーな手法とは，作成アルゴリズムが数学的に明快でパラメータの意味解釈が容易な手法であると考える．

第二に，データ視認性の高い連続エリアカルトグラムを作成しなければならない．連続エリアカルトグラムを視覚化手法として利用する際には，通常，地理的地図との対比を通して地域形状の変形を観察し，連続エリアカルトグラム上に表現された統計データを解釈することになる．その時，統計データの表現には不必要な変形が含まれていると，連続エリアカルトグラム上に表現された統計データの解釈を妨げることになる．そのため，連続エリアカルトグラム作成に当たっては，統計データを面積で表現するように地域形状を変形すると同時に，地理的地図からの地域形状の変形を抑える必要がある．

これら 2 つの基本的考え方を踏まえた連続エリアカルトグラム作成手法の構築を行う．ここで，連続エリアカルトグラム作成問題の問題点の整理を行う．

連続エリアカルトグラム作成問題における問題点は，　「適用対象の地域形状が多様」，　「連続エリアカルトグラム作成問題は不良設定問題」，　「地域形状変形の抑制が必要」の 3 つである．そこで本論文では,「適用対象地域の三角網分割」と「三角網の辺の方位角変化を抑制する正則化項の導入」によって上記の問題点に対応する．

まず,「対象地域の三角網分割」によって，問題点　「適用対象の地域形状が多様」を解消する．もともと適用対象の地域形状は多種多様であるが，複雑な形状の面積計算は容易ではなく，あらゆる形状に適用可能な手法の構築は容易ではない．しかし，適用対象地域を三角網分割することにより，三角形をデータに合わせて変形する手法を構築すればよい．三角網分割を用いる利点は，三角形が最も単純な形状で面積計算が容易な点である．このため，面積をデータに合わせる目的関数を簡潔に記述できる．

次に,「正則化項の導入」により，問題点　「連続エリアカルトグラム作成問題は不良設定問題」を解決する．連続エリアカルトグラム作成問題は，そもそもカルトグラム上の地域の面積に関する情報だけではその地域形状を定めることができない不良設定問題であり，その解決には正則化条件の導入が必要である．また,「三角網の辺の方位角変化を抑制する正則化項の導入」により問題点「地域形状変形の抑制が必要」に対応し，表現する統計データに因らない不必要な形状の変形を排除し，地理的地図との比較が容易な視覚的に優れた連続エリアカルトグラムを作成する．この正則化条件により三角網の変形，つまり地域形状の変形，を抑えることが可能になる．例えば，方位角変化の抑制に対して極端に大きな重み付けを行った場合，通常の地理的地図から拡大・縮小のみを許容することになり，形状を完全に維持した変形を行うことができる．

以上から，本論文では地域を三角網分割し，三角網を構成する辺の方位角変化に対する正則化項を導入し，三角形の面積を与えられたデータに合わせる手法の構築を行う．

第 4.2 節では，地域形状の三角網分割を用いた連続エリアカルトグラム作成手法を構築する．まず，連続エリアカルトグラム作成問題の定式化・三角網の辺の方位角変化を抑制する正則化項の導入・解法の線形化を行い，数学的に明快かつ簡便な作成手法の構築を行う．さらに，正則化項の重み設定・三角網の自動作成を加えて，高速に計算可能な作成手法を提案する．

第 4.3 節では，アメリカ合衆国の州人口データを用いて提案手法の適用を行い，提案手法の評価を行う．また，日本の都道府県人口や県民所得を表す連続エリアカルトグラムを作成し，提案手法の適用可能性を示すと同時に，連続エリアカルトグラムの視覚化手法としての有効性を示す．

4.2　三角網分割を用いた作成手法の提案

4.2.1　連続エリアカルトグラム作成問題の定式化

頂点 i, j, k からなる三角形 t_{ijk} の連続エリアカルトグラム上の面積 A_{ijk} をデータ D_{ijk} に合わせる目的関数は，T を三角網に含まれる三角形の集合とすると，式 (4.1) と書ける．

$$\min \sum_{t_{ijk} \in T} (D_{ijk} - A_{ijk})^2 \tag{4.1}$$

もちろん，式 (4.1) では解が無数に存在するため，正則化項を導入する必要がある．正則化項として，地域形状変化を抑えるため三角網の辺の方位角を拘束することとする．頂点 mn を結ぶ辺を

e_{mn}，e_{mn} の地理的配置上の座北方位角を θ^G_{mn}，連続エリアカルトグラム上の座北方位角を θ^C_{mn}(図 4.1)，三角網に含まれる辺の集合を E とすると，辺 mn の方向変化を抑制する正則化項は式 (4.2) となる．

$$+ \sum_{e_{mn} \in E} \left(\theta^C_{mn} - \theta^G_{mn}\right)^2 \tag{4.2}$$

主問題 (式 (4.1)) と正則化項 (式 (4.5)) より，三角網を用いた連続エリアカルトグラム作成問題は，正則化項の重みを μ とすると，式 (4.3) となる．

$$\min \left[\sum_{t_{ijk} \in T} \left(D_{ijk} - A_{ijk}\right)^2 + \mu \sum_{e_{mn} \in E} \left(\theta^C_{mn} - \theta^G_{mn}\right)^2 \right] \tag{4.3}$$

しかし式 (4.3) では，主問題と正則化項の次元が異なり μ の設定が難しい．また，より大きなデータが与えられた三角形の面積をより正確に合わせることを重視した式となっている．エリアカルトグラム作成では，データの大小に関わらすデータを面積で正確に表現するように三角形の変形を行うことが望ましい．そこで，式 (4.4) のようにエリアカルトグラム作成の主問題を無次元化する．

$$\min \left[\sum_{t_{ijk} \in T} \left(1 - \frac{A_{ijk}}{D_{ijk}}\right)^2 + \mu \sum_{e_{mn} \in E} \left(\theta^C_{mn} - \theta^G_{mn}\right)^2 \right] \tag{4.4}$$

ここで，式 (4.4) を三角網の頂点座標を用いた表現に改める．以降では，頂点 m の連続エリアカルトグラム上の座標を $(x_m,\ y_m)$，地理的地点配置上の座標を $(x^G_m,\ y^G_m)$ と表記することとする．

まず，連続エリアカルトグラム上の三角形の面積 A_{ijk} はその頂点 $i,\ j,\ k$ の座標を用いると，式 (4.5) となる．

$$A_{ijk} = \frac{1}{2} \left| \left(x_j - x_i\right)\left(y_k - y_i\right) - \left(x_k - x_i\right)\left(y_j - y_i\right) \right| \tag{4.5}$$

主問題 (式 (4.4) 第一項) に式 (4.5) を代入する際に絶対値記号を取り除くため，主問題を改め式 (4.6) とする．

$$\min \sum_{t_{ijk} \in T} \left(1 - \frac{A^2_{ijk}}{D^2_{ijk}}\right)^2 \tag{4.6}$$

(a) 地理的配置上　　(b) 連続エリアカルトグラム上

図 4.1: 辺 mn の座北方位角

式 (4.6) は式 (4.4) 第一項と比べ面積 A_{ijk} のデータ D_{ijk} からのずれをより大きく評価するが，エリアカルトグラム作成の主問題として不適切ではない．そこで，式 (4.6) に式 (4.7) を代入する．

$$\min \sum_{t_{ijk}\in T} \left[1 - \frac{1}{4D_{ijk}^2} \left\{ (x_j - x_i)(y_k - y_i) - (x_k - x_i)(y_j - y_i) \right\}^2 \right]^2 \tag{4.7}$$

また，正則化項は，

$$+\mu \sum_{e_{mn}\in E} \left\{ \arctan\left(\frac{x_n - x_m}{y_n - y_m} \right) - \arctan\left(\frac{x_n^G - x_m^G}{y_n^G - y_m^G} \right) \right\}^2 \tag{4.8}$$

と書けることから，

$$+\mu \sum_{e_{mn}\in E} \left\{ \frac{x_n - x_m}{y_n - y_m} - \frac{x_n^G - x_m^G}{y_n^G - y_m^G} \right\}^2 \tag{4.9}$$

とほぼ同値となる．更に，計算上分母が 0 となるのを避けるため，式 (4.10) とする．但し，頂点 mn 間距離を d_{mn} と記述する．

$$+\mu \sum_{e_{mn}\in E} \left\{ \frac{(x_n - x_m)(y_n^G - y_m^G) - (x_n^G - x_m^G)(y_n - y_m)}{d_{mn} d_{mn}^G} \right\}^2 \tag{4.10}$$

以上より，連続エリアカルトグラム作成問題は式 (4.11) と書ける．

$$\begin{aligned} \min \Bigg[& \sum_{t_{ijk}\in T} \left\{ 1 - \frac{\{(x_j - x_i)(y_k - y_i) - (x_k - x_i)(y_j - y_i)\}^2}{4D_{ijk}^2} \right\}^2 \\ & +\mu \sum_{e_{mn}\in E} \left\{ \frac{(x_n - x_m)(y_n^G - y_m^G) - (x_n^G - x_m^G)(y_n - y_m)}{d_{mn} d_{mn}^G} \right\}^2 \Bigg] \end{aligned} \tag{4.11}$$

式 (4.11) の非線形最小二乗問題を解くことにより，連続エリアカルトグラムを作成することができる．但し，位相関係を保った連続エリアカルトグラム作成のためには，正則化項の重み μ を適切に設定し，三角網の反転を防ぐ必要がある．ここで，式 (4.11) で一意に解を求められることを確認する．未知変数の数は，1 点を座標系に固定するため，(頂点数-1)×2 となる．一方，観測方程式数は，(三角形数) + (辺数) となる．三角網では常に，(未知変数数)≤(観測方程式数) が成り立つので，式 (4.11) から解を一意に求めることができる．

4.2.2　解法の線形化

連続エリアカルトグラム上の座標 x_i, y_i に近似値 x_i', y_i' を与え，式 (4.11) を近似値近傍で線形化する．なお，表記上の都合より，以後

$$x_{ij}' = x_j' - x_i' \tag{4.12a}$$

$$y_{ij}' = y_j' - y_i' \tag{4.12b}$$

と表し，符号付きの三角形 t_{ijk} の面積の近似値を

$$A'_{ijk} = \frac{1}{2}\left(x'_{ij}y'_{ik} - x'_{ik}y'_{ij}\right) \tag{4.13}$$

と表記する．

式 (4.11) の主問題を，近似値近傍で線形化すると

$$\sum_{t_{ijk}\in T}\left\{1 + \frac{3A'^{2}_{ijk}}{D^2_{ijk}} + \frac{A'_{ijk}}{D^2_{ijk}}\left(y'_{jk}x_i - y'_{ik}x_j + y'_{ij}x_k - x'_{jk}y_i + x'_{ik}y_j - x'_{ij}y_k\right)\right\}^2 \tag{4.14}$$

となる．

また，式 (4.11) の正則化項は，繰り返し計算時に近似値を用いて正則化条件を更新すると，

$$+\mu\sum_{e_{mn}\in E}\left\{\frac{\left(x_n - x_m\right)y'_{mn} - x'_{mn}\left(y_n - y_m\right)}{d'^2_{mn}}\right\}^2 \tag{4.15}$$

と表せる． 式 (4.14)(4.15) より，式 (4.11) は

$$\begin{aligned}\min\Bigg[\sum_{t_{ijk}\in T}\left\{1 + \frac{3A'^{2}_{ijk}}{D^2_{ijk}} + \frac{A'_{ijk}}{D^2_{ijk}}\left(y'_{jk}x_i - y'_{ik}x_j + y'_{ij}x_k - x'_{jk}y_i + x'_{ik}y_j - x'_{ij}y_k\right)\right\}^2 \\ +\mu\sum_{e_{mn}\in E}\left\{\frac{\left(x_n - x_m\right)y'_{mn} - x'_{mn}\left(y_n - y_m\right)}{d'^2_{mn}}\right\}^2\Bigg]\end{aligned} \tag{4.16}$$

のように線形化できる．この式 (4.16) の線形最小二乗問題の繰り返し計算により連続エリアカルトグラム作成ができる．

4.2.3　提案解法に基づく連続エリアカルトグラム作成手法

式 (4.16) の解法を用いて連続エリアカルトグラム作成を行う際には，次の 2 点を考慮する必要がある．

- 全ての三角形の反転を防ぐ．
- 実用的な計算時間で図を出力するため，解法を高速化する．

これら 2 点は，正則化項に対する重み μ の大きさを制御することで，解決することができる．その詳細を以下に示す．

まず，μ に小さな値を設定した場合，三角網の辺の方向変化が大きくなるため，三角形の反転が生じる可能性が高くなる．三角形の反転が生じると，図 4.2 のように頂点の位相が破壊される．三角形の反転を防ぐためには，μ の値を大きく設定し三角網を構成する各辺の方向変化に対する制約を大きくすることが望ましい．

一方，μ に大きな値を設定すると，収束速度の低下をもたらし，収束計算回数・計算時間の増加を招く．また，収束計算過程において μ を固定していると，式 (4.16) の主問題 (第一項) の値は収束計算毎に小さくなるため，相対的に正則化条件 (第二項) が大きくなり，収束速度が低下する．

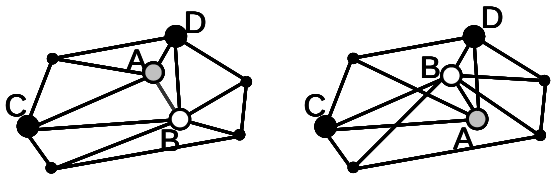


図 4.2: 三角形の反転と位相の破壊

式 (4.16) は，1 次収束する手法であるため，最適解近傍では収束速度が低下し，収束計算に時間を要する．この方法を超 1 次収束させるためには，最適解近傍で $\mu \to 0$ となるよう制御する必要がある．

三角形の反転を防ぎ，かつ，計算時間の短い連続エリアカルトグラム作成を行うためには，収束計算 1 回毎に，三角形の反転が起こらない最小の μ 値を設定することが望ましい．しかし，この値を求めることは困難である．

そこで，μ の値を

$$\left\{ \sum_{t_{ijk} \in T} \left(D_{ijk} - A_{ijk} \right)^2 \Big/ \sum_{t_{ijk} \in T} \left(D_{ijk} - A'_{ijk} \right)^2 \right\} \mu \tag{4.17}$$

に基づき減少させながら収束計算を行い，三角形の反転が観測された場合には μ を増加させて再計算を行い，三角形反転の防止，および，計算時間の短縮を図る．

なお，三角形反転は，頂角の正弦の符号変化により判断することができる．

4.2.4　三角網の自動作成

提案アルゴリズムを用いて連続エリアカルトグラム作成を行うためには，三角網を作成する必要がある．そのため，三角網の自動作成を行う．

連続エリアカルトグラム作成上望ましい三角網とは，三角形の反転が起こりにくいものである．そのため，できる限り極端な鋭角を持たない三角形を作成することが望ましい．そこで，全ての三角形分割の中で，三角網の最小角が最大となる Delaunay 三角分割 (岸本, 1978) を用いる．

三角網自動作成の実装に当たっては，まず各地域のポリゴンをその頂点を用いて Delaunay 三角網を作成し，その後，内点の追加・ラプラシアン法による内点座標の修正 (谷口, 1992) を用いて三角網を作成する．

また，連続エリアカルトグラム作成の高速化のため，提案アルゴリズム内で一定回以上三角形の反転が生じた場合，三角網分割を再実行することとする．

本提案手法による連続エリアカルトグラム作成アルゴリズムの手順は図 4.3 の通りである．また，連続エリアカルトグラム作成アルゴリズムのソースコードを，付録 B.3 に記す．

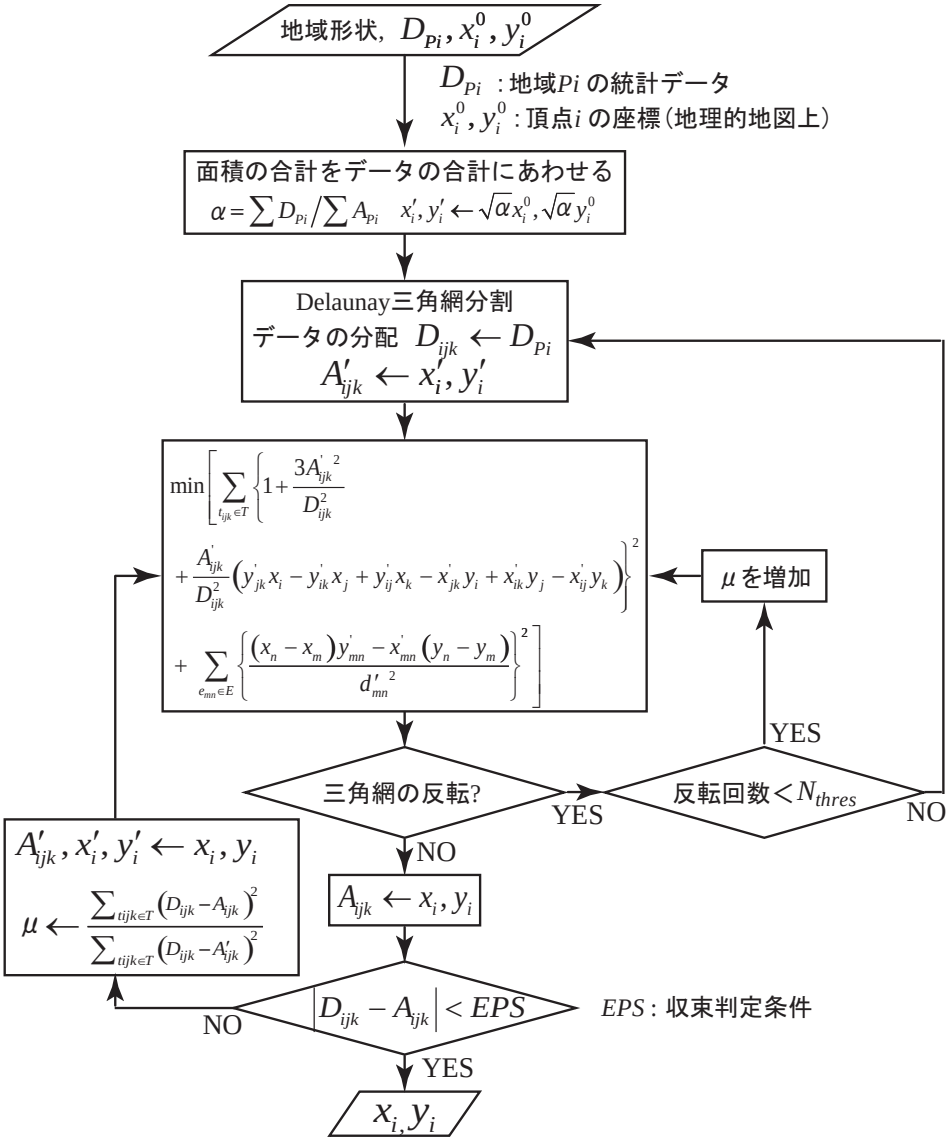


図 4.3: 連続エリアカルトグラム 提案手法のアルゴリズム

4.3　提案手法の適用

提案手法をアメリカ合衆国の人口データに適用し，その適用可能性を検証する．また，日本の都道府県人口・県民所得を表現する連続エリアカルトグラムを作成し，提案手法の適用可能性を示すと同時に連続エリアカルトグラムの視覚化手法としての有効性を示す．

4.3.1　適用と評価

U.S. Census Bureau, United States Department of Commerce による 1980 年アメリカ合衆国州人口データを用いて連続エリアカルトグラムを作成する．本提案手法適用に際して，簡略化したアメリカ合衆国地図 (図 4.4) を用いた．頂点数は 185 である．但し，本提案手法では全計算対象地域は連続していなければ計算できないため，島がある場合には海峡に仮想的なポリゴンを作り仮想的な値を与えている．また，湾を挟む両岸の地域は交差する可能性があるため，湾にも仮想的なポリゴンを設定している．また，地域が複数のポリゴンに分かれている場合，人口データを地理的な面積に基づいて按分している．

正則化項に対する重み μ の初期値を 1，$\sqrt{10}$，10 と設定し，連続エリアカルトグラム作成計算を行った．その出力結果を図 4.5 に，収束計算回数・計算時間・RMSE を表 4.1 に示す．また，μ の初期値 $\sqrt{10}$ の場合の計算過程を図 4.6 に示す．

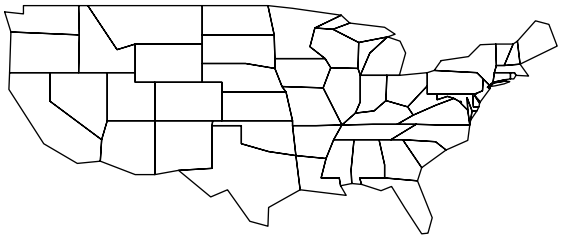


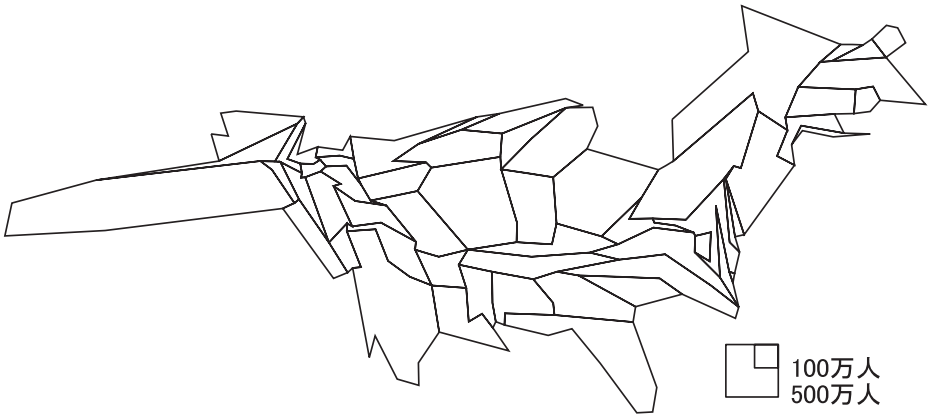
図 4.4: アメリカ合衆国 州形状 入力データ

表 4.1: 1980 年 アメリカ合衆国 州人口連続エリアカルトグラム作成計算
収束計算回数・計算時間・RMSE

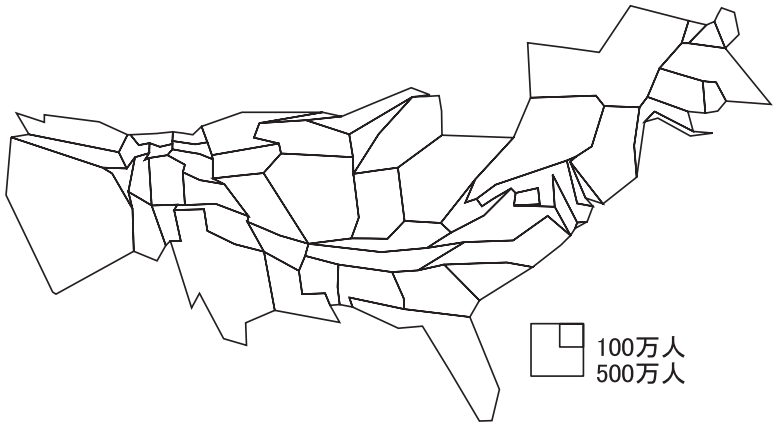
μ 初期値	収束計算 (回)	計算時間 (秒)	RMSE (人)
1	19	5.0	25.5
$\sqrt{10}$	28	7.5	2.1
10	101	23.8	8.3

(CPU2.8GHz の計算機使用)

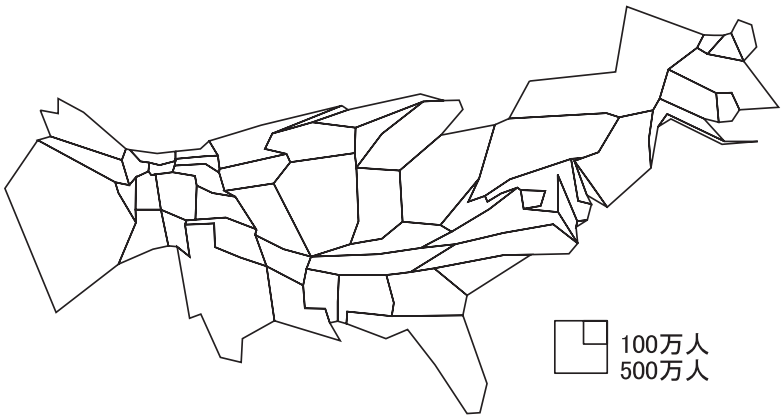
まず，本提案手法により，連続エリアカルトグラム作成が実用的な計算時間で計算可能であることが確認できた．1 州の人口が平均で約 460 万人であることを考えると RMSE は非常に小さく，本提案手法によって得られた図 4.5 は人口データを完全に表現していることが分かる．



(a) 初期値 $\mu = 1$



(b) 初期値 $\mu = \sqrt{10}$



(c) 初期値 $\mu = 10$

図 4.5: 1980 年 アメリカ合衆国 州人口 連続エリアカルトグラム 作成結果

表 4.2: 1980 年 アメリカ合衆国 州人口 連続エリアカルトグラム 国形状評価
地理的国形状 ヘルマート変換 自由度修正済み決定係数

$\mu = 1$	$\mu = \sqrt{10}$	$\mu = 10$
0.9073	0.9166	0.9084

表 4.3: 1980 年 アメリカ合衆国 州人口 連続エリアカルトグラム 州形状評価
地理的州形状 ヘルマート変換 自由度修正済み決定係数

州	$\mu = 1$	$\mu = \sqrt{10}$	$\mu = 10$
AL	0.9991	0.9994	0.9985
AZ	0.9681	0.9941	0.9919
AR	0.9993	0.9987	0.9984
CA	0.1699	0.8949	0.9103
CO	0.9773	0.9947	0.9936
CT	0.9999	0.9995	0.9997
DE	0.9991	0.9999	0.9998
DC	0.9998	0.9997	0.9997
FL	0.9998	0.9999	0.9999
GA	0.9984	0.9991	0.9992
ID	0.9824	0.9981	0.9969
IL	0.9957	0.9988	0.9969
IN	0.9988	0.9998	0.9998
IA	0.9984	0.9984	0.9983
KS	0.9908	0.9976	0.9971
KY	0.9986	0.9994	0.9991
LA	0.9994	0.9997	0.9996
ME	0.9999	0.9997	0.9997
MD	0.9984	0.9994	0.9993
MA	0.9998	0.9995	0.9993
MI	0.9954	0.9942	0.9961
MN	0.9808	0.9847	0.9895
MO	0.9962	0.9972	0.9968
MT	0.9790	0.9993	0.9995
NE	0.9959	0.9983	0.9990
NV	0.9820	0.9972	0.9951

州	$\mu = 1$	$\mu = \sqrt{10}$	$\mu = 10$
NH	0.9980	0.9996	0.9996
NJ	0.9989	0.9985	0.9983
NM	0.9968	0.9975	0.9978
NY	0.9955	0.9963	0.9978
NC	0.9981	0.9989	0.9996
ND	0.9967	0.9987	0.9993
OH	0.9986	0.9969	0.9969
OK	0.9964	0.9994	0.9991
OR	0.4552	0.9112	0.8856
PA	0.9951	0.9971	0.9938
RI	1.0000	1.0000	0.9999
SC	0.9985	0.9986	0.9996
SD	0.9963	0.9991	0.9994
TN	0.9992	0.9997	0.9994
TX	0.9903	0.9951	0.9952
UT	0.9628	0.9975	0.9933
VT	0.9995	0.9999	1.0000
VA	0.9988	0.9990	0.9986
WA	0.8345	0.9162	0.9813
WV	0.9996	0.9997	0.9997
WI	0.9860	0.9944	0.9933
WY	0.9990	0.9997	0.9992
平均値	0.9632	0.9925	0.9935
中央値	0.9981	0.9988	0.9990
最小値	0.1699	0.8949	0.8856
標準偏差	0.1408	0.0222	0.0203

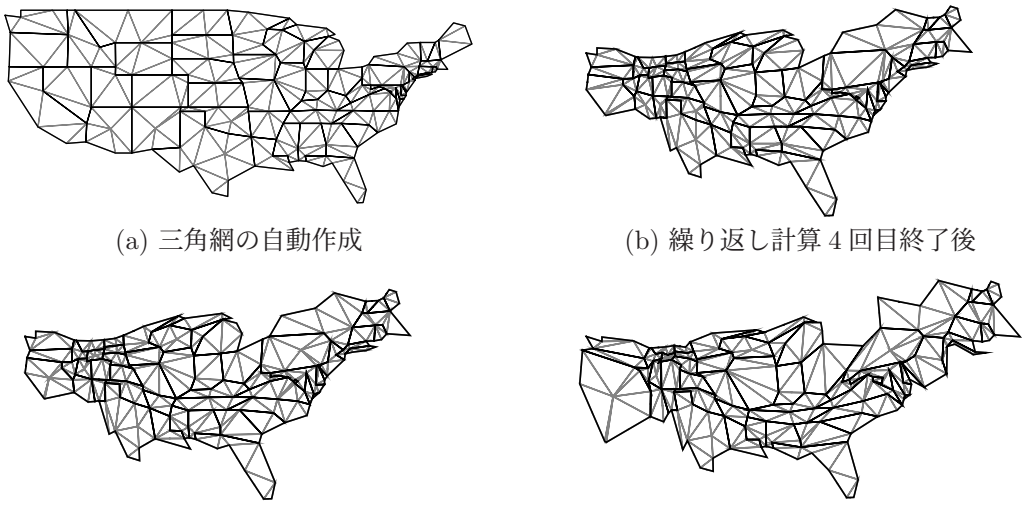
次に，μ 初期値の影響に注目する．μ 初期値が小さいと，収束速度が速く計算時間が短いが (表 4.1)，形状変形は大きくなり分かりにくいカルトグラムを出力する (図 4.5).

ここで，カルトグラム上形状と地理的形状間の類似度を定量的に評価することを試みる．各形状の頂点を合わせるようにヘルマート変換を行い，その自由度修正済み決定係数を 2 形状の類似度指標として用いる (Dryden and Mardia (1998))．まず国全体の形状を評価した場合 (表 4.2)，大きな値の違いは見られない．この要因の一つに，アメリカ合衆国は東海岸に小さな州が集中し頂点が多いため，東海岸の形状が大きく評価され，西海岸の形状の変形を反映できていないことが挙げられる．そのため，形状を評価する際には頂点の分布などに注意を払う必要がある．そこで州毎に形状比較を行い，μ 初期値 1 の場合，西海岸 3 州 (カリフォルニア・オレゴン・ワシントン) で著しい変形を起こすことを確認した (表 4.3)．一方，μ 初期値が $\sqrt{10}\cdot 10$ では，値に顕著な違いは見られず，ほぼ同等の形状を維持していることが確認できる．

以上のように，μ に大きな初期値を設定することにより辺の方位角変化を抑えることが可能になり，分かりやすい連続エリアカルトグラムを作成できることを確認した．また図 4.5(b)(c) には目立った差異は見られないが，計算時間では約 3 倍の差がある．このことから，正則化項に対する過剰な重み付けは出力結果には影響を与えないが，計算時間の増加をもたらし好ましくない．しかし，形状変化を抑えかつ高速に計算できる μ の最適な値を求めることは現状では困難であり，試行錯誤に頼らざるを得ない．

なお，計算過程は図 4.6 に示す通りである．まず各地域を三角網分割する (図 4.6(a))．連続エリアカルトグラム作成計算をの 4 回繰り返した後 (図 4.6(b))，三角形反転が生じるため三角網の修正を行う (図 4.6(c))．更に，修正済みの三角網を用いて繰り返し計算を 12 回行い (図 4.6(d))，三角網の再修正後 12 回繰り返し計算を行い図 4.5(b) のエリアカルトグラムを出力する．

このように，提案手法は実用的な時間で計算が可能であり，また，既存手法とは異なり初期値設定に試行錯誤的作業をほとんど要しない連続エリアカルトグラム作成手法であることが確認できた．次節に他の視覚化例を示す.



(a) 三角網の自動作成　(b) 繰り返し計算 4 回目終了後

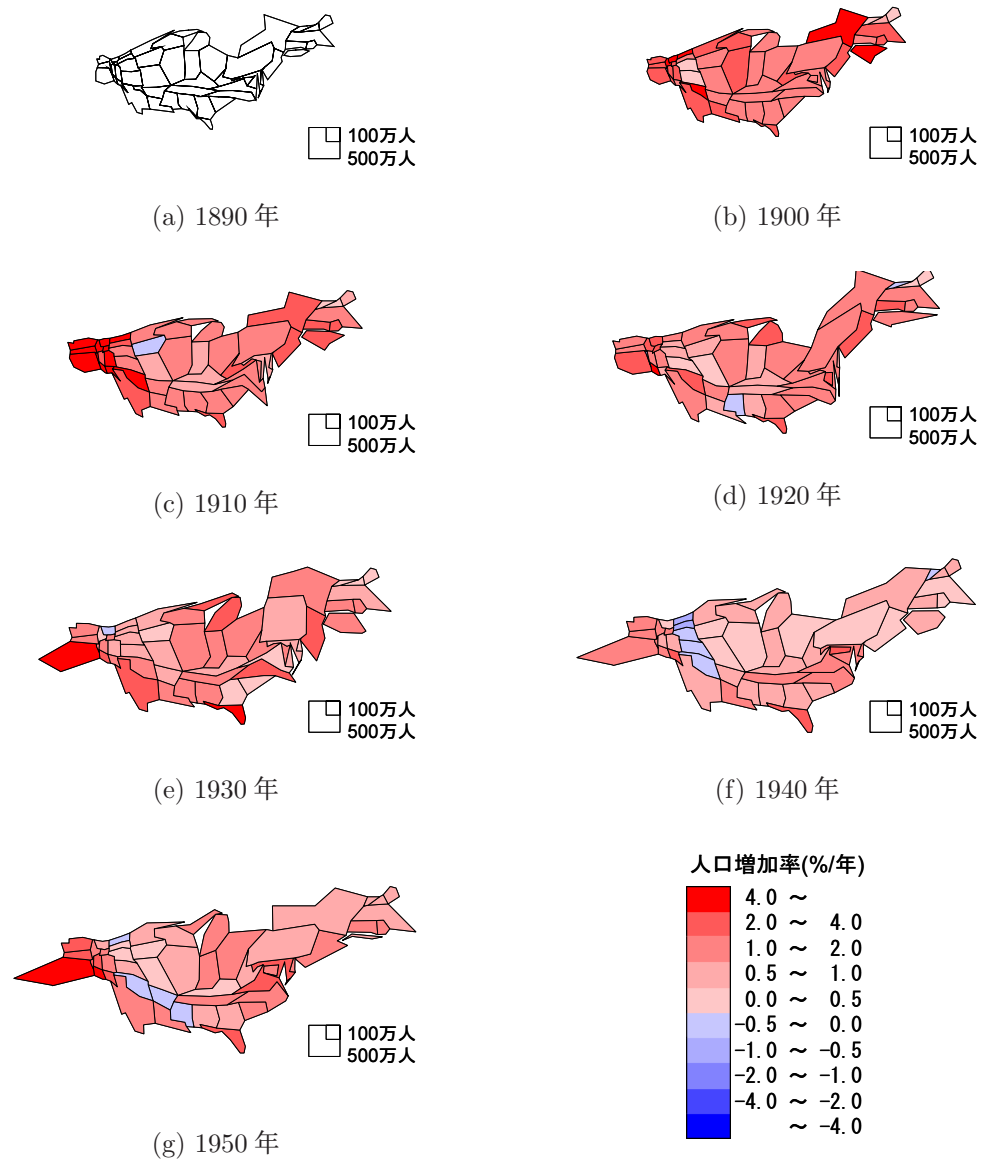
(c) 三角網の修正　(d) 繰り返し計算 16 回目終了後

図 4.6: 1980 年 アメリカ合衆国 州人口 連続エリアカルトグラム 計算経過 ($\mu = \sqrt{10}$)

4.3.2　連続エリアカルトグラムによる視覚化例

アメリカ合衆国 州人口の変遷

U.S. Census Bureau, United States Department of Commerce による 1890〜2000 年まで 10 年毎の州人口データを用いて連続エリアカルトグラムを作成し，その時系列変化を視覚化する (図 4.7〜4.8)．入力州形状データは図 4.4 である．



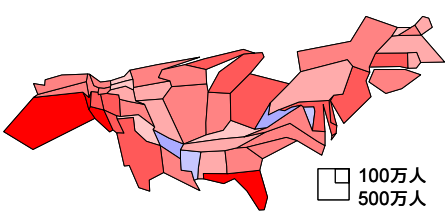
(a) 1890 年　(b) 1900 年

(c) 1910 年　(d) 1920 年

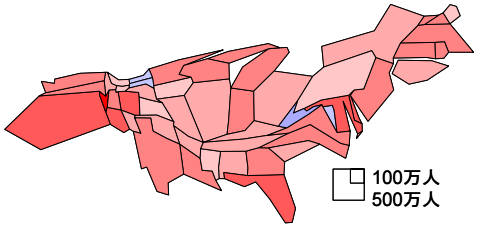
(e) 1930 年　(f) 1940 年

(g) 1950 年

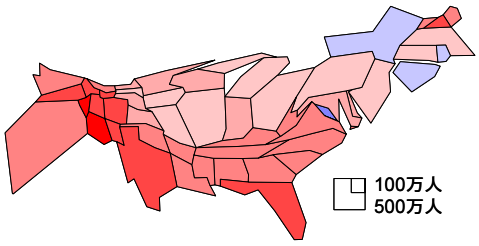
図 4.7: アメリカ合衆国 州人口 連続エリアカルトグラム 1890〜1950 年



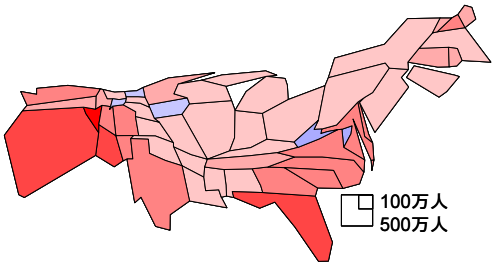
(a) 1960 年



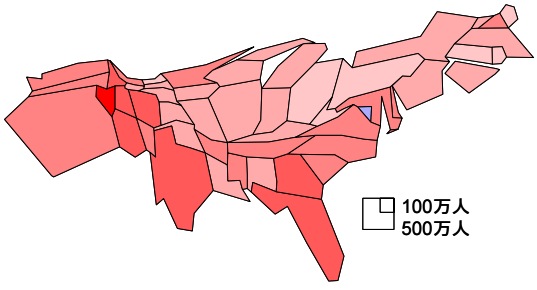
(b) 1970 年



(c) 1980 年



(d) 1990 年



(e) 2000 年

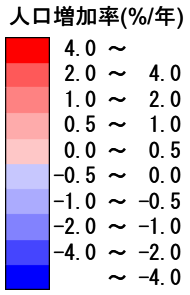


図 4.8: アメリカ合衆国 州人口 連続エリアカルトグラム 1960～2000 年

都道府県人口の変遷

総務省統計局の国勢調査による都道府県人口データ (1920〜2000 年: 5 年毎) に対して提案手法を適用し，その変遷を視覚化した例を示す．

連続エリアカルトグラム作成に当たっては，図 4.9 に示す簡略化した都道府県形状データを用いた．北海道・本州・四国・九州は海峡部にダミーのポリゴンを配置し，331 の頂点座標を一度に計算したが，沖縄県は別に計算を行い配置している．

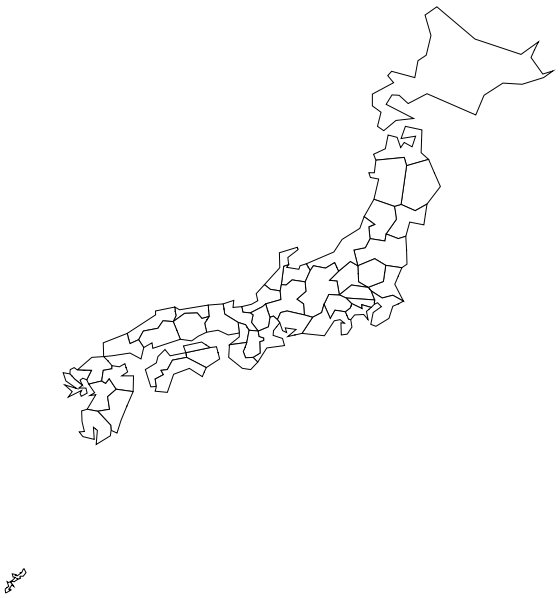
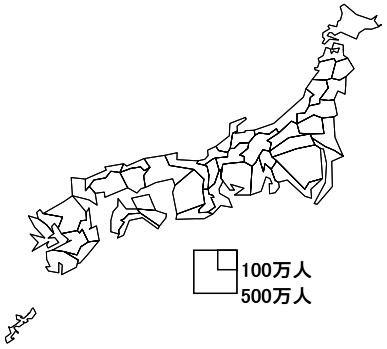


図 4.9: 日本 都道府県形状 入力データ

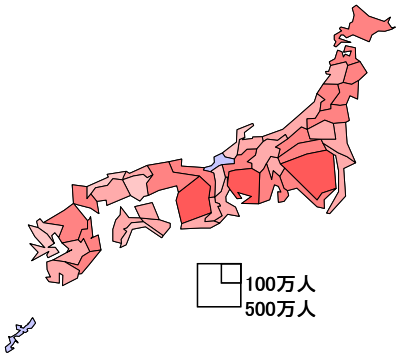
都道府県人口の時系列変化を分かりやすく表現するためには，人口の増減に起因しない地域形状の変形を排除することが望ましい．そこで，まず 1920 年の人口エリアカルトグラムを作成した (図 4.10(a))．その後，作成したエリアカルトグラムの地域形状を初期値として入力し，5 年後の人口エリアカルトグラムを逐次作成した．

図 4.10〜4.13 に 1920〜2000 年の 5 年毎の都道府県人口エリアカルトグラムを示す．但し，過去 5 年間の 1 年あたり人口増加率に応じて各都道府県を色分けしている．μ 初期値は 1，計算時間は地理的地図を初期値として計算した 1920 年エリアカルトグラム作成は CPU1.8GHz の計算機で 9.5 秒，一時点前のエリアカルトグラム上の地域形状を入力した他時点のエリアカルトグラム作成は全て 2 秒以内に完了している．

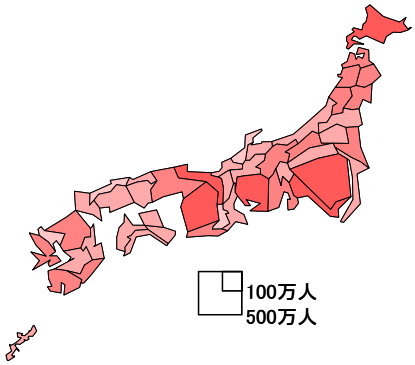
図 4.10〜4.13 のように，三大都市圏に人口が集中していく様子を視覚的に表現することができる．



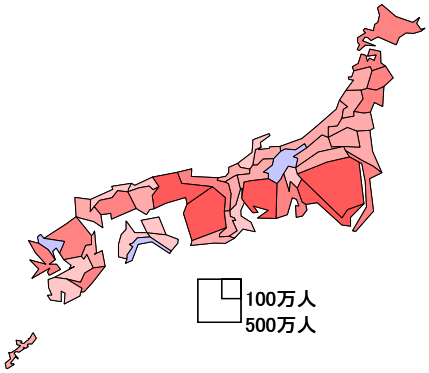
(a) 1920 年



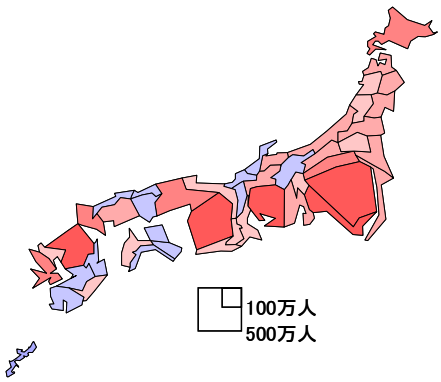
(b) 1925 年



(c) 1930 年



(d) 1935 年



(e) 1940 年

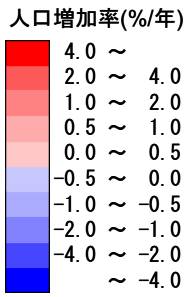
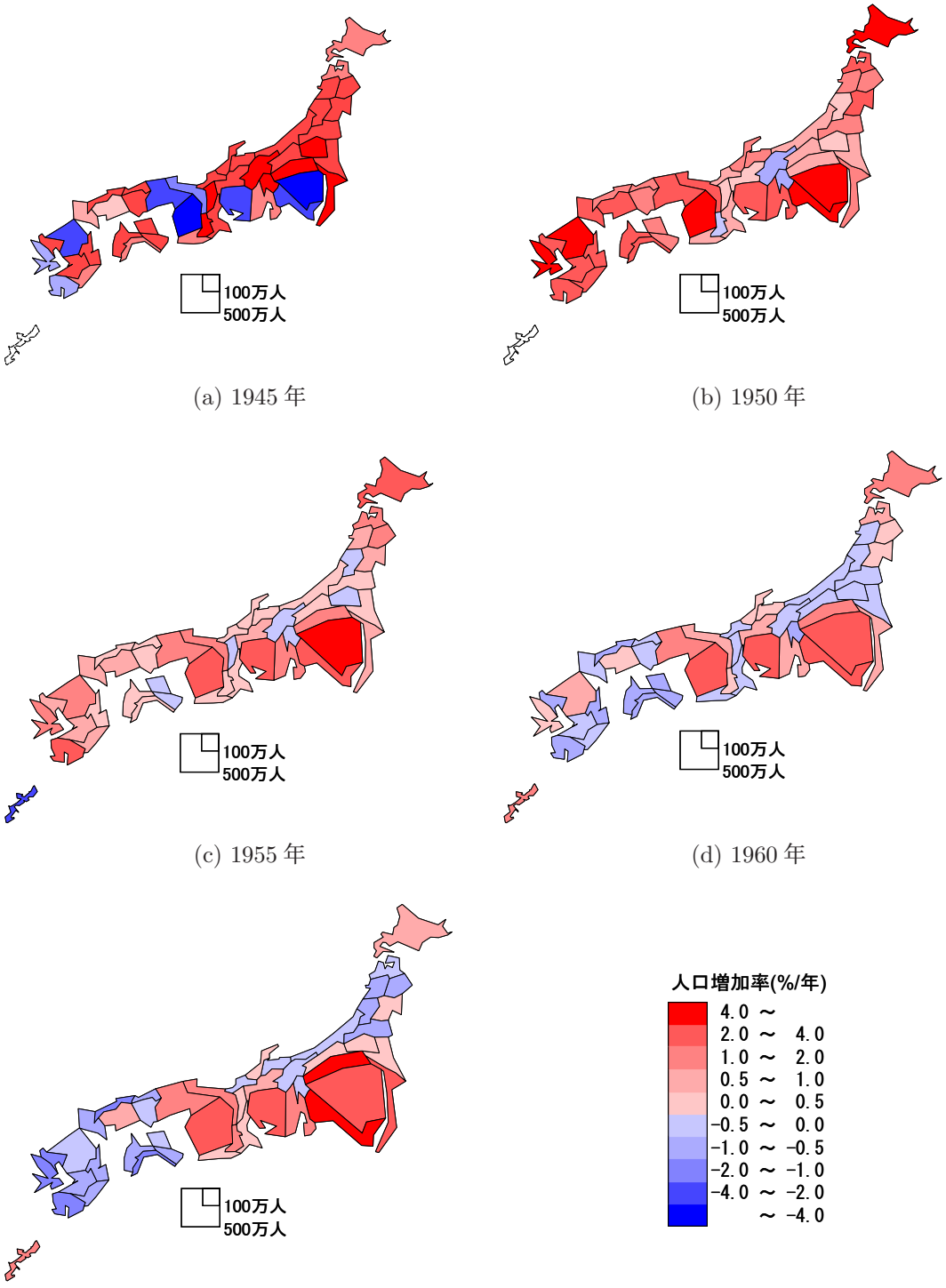


図 4.10: 日本 都道府県人口 連続エリアカルトグラム 1920〜1940 年

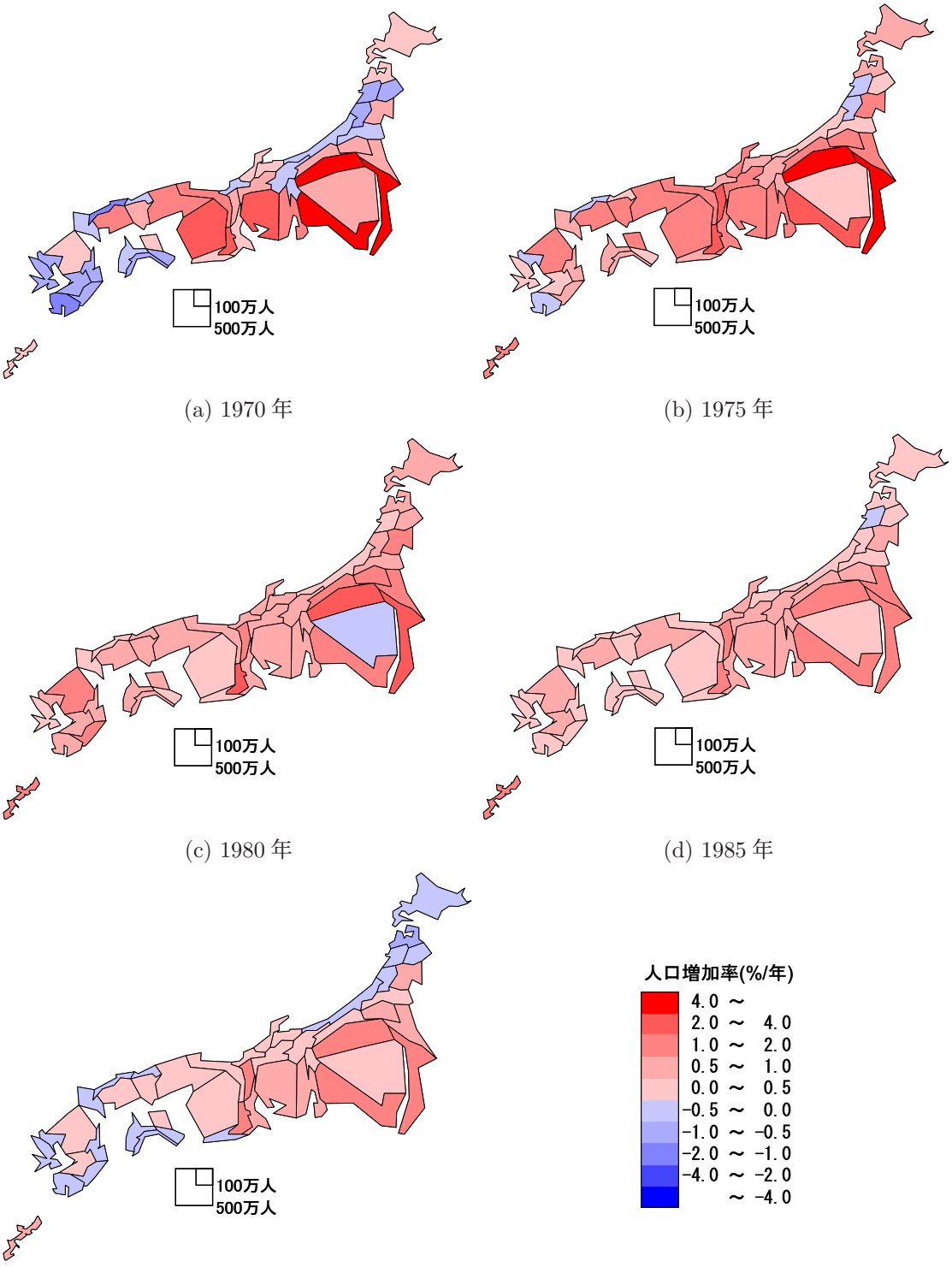


(a) 1945 年　(b) 1950 年

(c) 1955 年　(d) 1960 年

(e) 1965 年

図 4.11: 日本 都道府県人口 連続エリアカルトグラム 1945〜1965 年

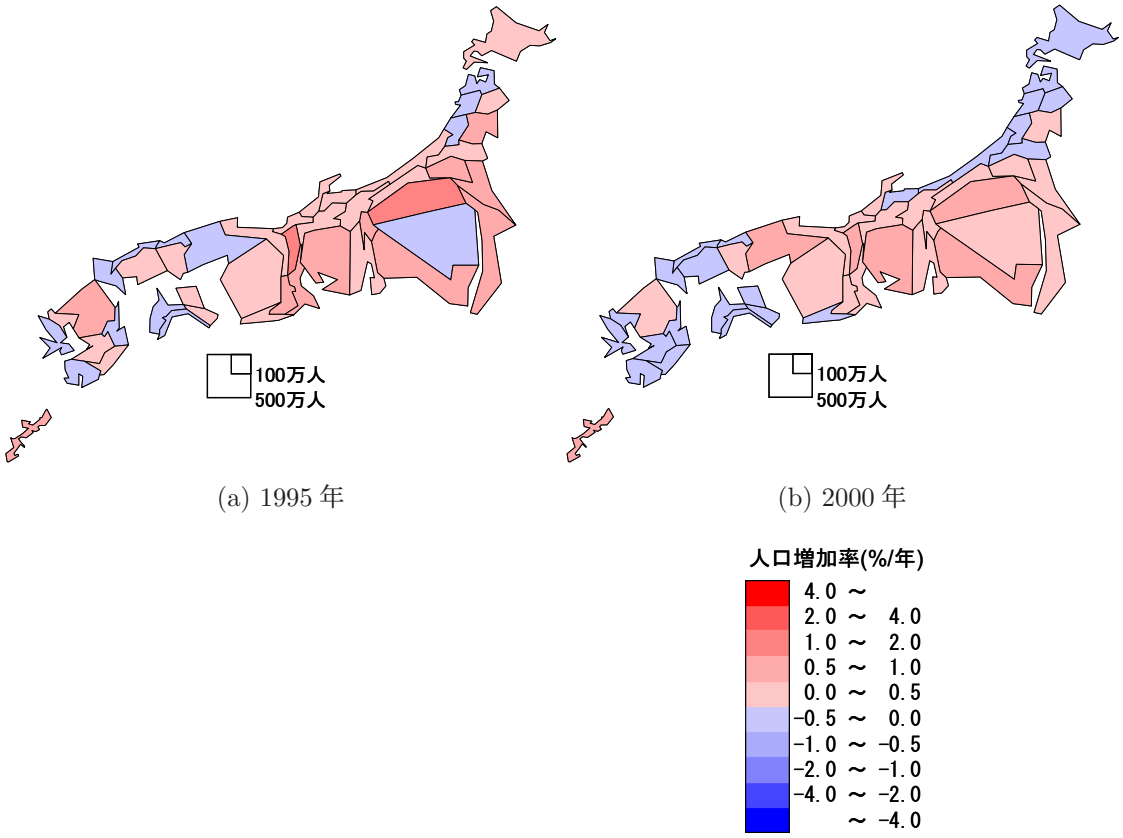


(a) 1970 年　(b) 1975 年

(c) 1980 年　(d) 1985 年

(e) 1990 年

図 4.12: 日本 都道府県人口 連続エリアカルトグラム 1970〜1990 年



(a) 1995 年　　(b) 2000 年

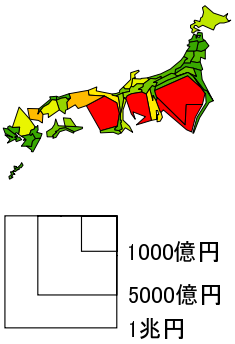
図 4.13: 日本 都道府県人口 連続エリアカルトグラム 1995～2000 年

県民所得の変遷

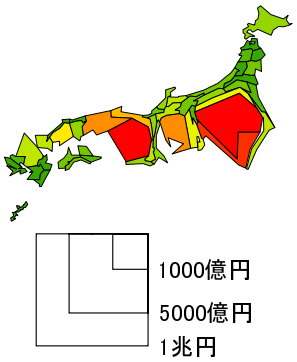
内閣府 経済社会総合研究所の県民経済計算より 1960 年から 2000 年の 5 年毎の県民所得データを利用し，県民所得連続エリアカルトグラムの作成を行った．県民所得を面積で表現し，一人あたり県民所得の一人あたり国民所得からの偏差を色で表現した連続エリアカルトグラムを作成した例を図 4.14〜4.16 に示す．なお，総務省統計局の消費者物価指数データから全国の「持家の帰属家賃を除く総合指数」を用いて，2000 年水準で金額を表現している．

県民所得連続エリアカルトグラムの作成には，県民所得分布が人口分布より偏在しているため計算に時間を要した．1960 年の県民所得エリアカルトグラム作成には CPU1.8GHz の計算機で 199 秒を要している．また，各年次の県民所得分布の変化も大きいため．前年次の連続エリアカルトグラムから変形させて新たなカルトグラム作成計算を行う場合にも時間を要しており，最大で 34 秒 (1990 年→ 1995 年) かかっている．

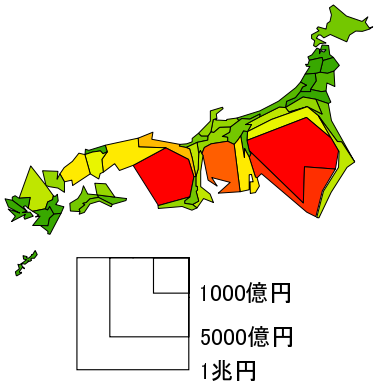
図 4.14〜4.16 のように，所得額が急速に増加するとともに，当初三大都市圏に集中していた県民所得が地方へ拡がり，所得格差が小さくなってきている様子を表現することができる．

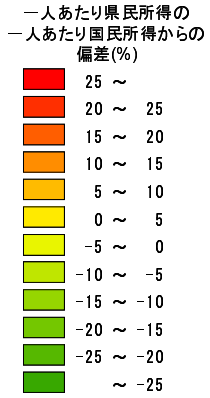


(a) 1960 年



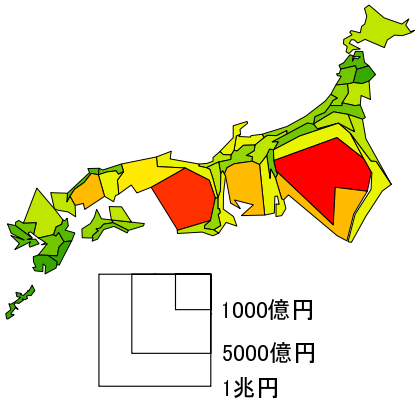
(b) 1965 年



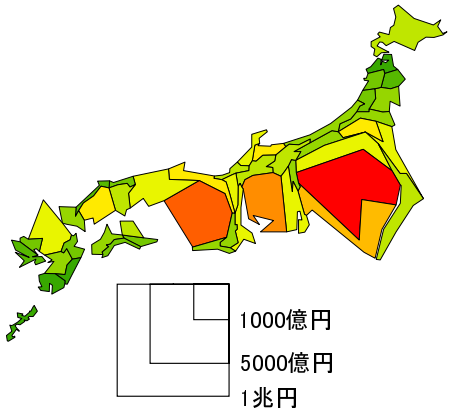


(c) 1970 年

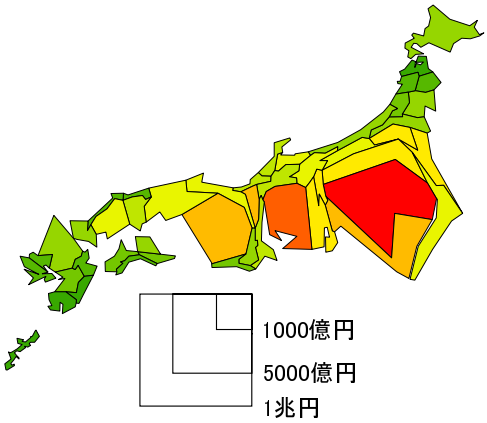
図 4.14: 県民所得 連続エリアカルトグラム 1960〜1970 年



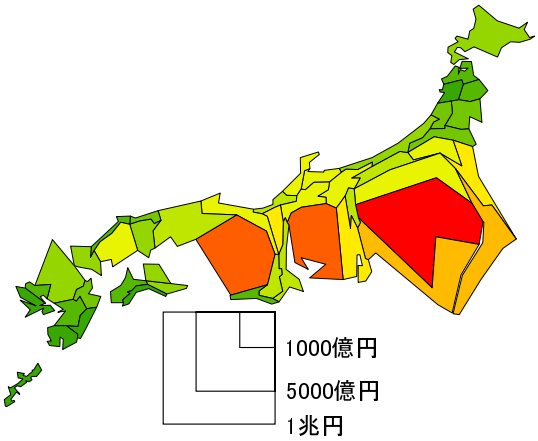
(a) 1975 年



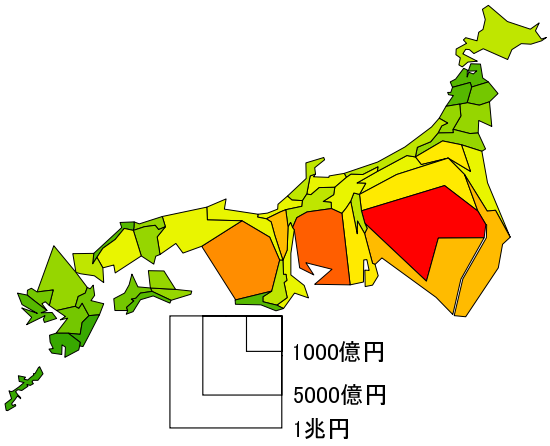
(b) 1980 年

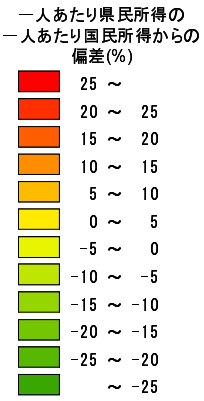


(c) 1985 年



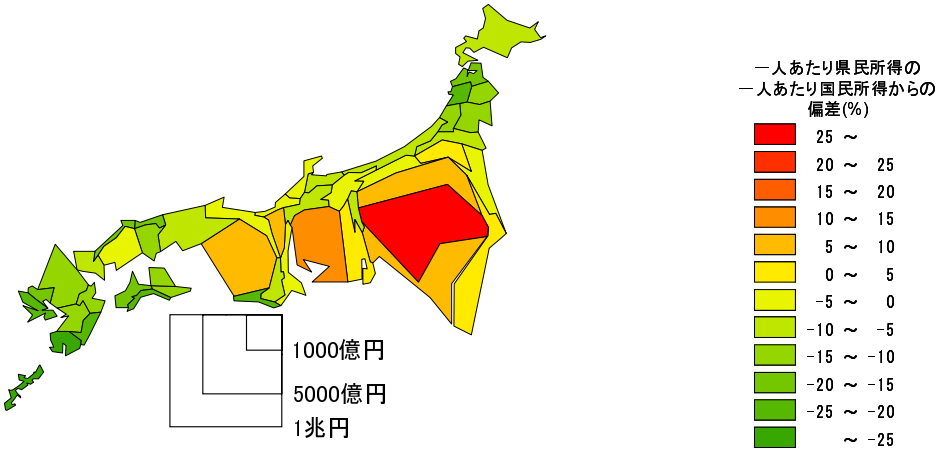
(d) 1990 年





(e) 1995 年

図 4.15: 県民所得 連続エリアカルトグラム 1975～1995 年



(a) 2000 年

図 4.16: 県民所得 連続エリアカルトグラム 2000 年

アニメーション表現による統計データ変遷の視覚化

アニメーション表現では，統計データの変遷をより印象的に視覚化することが可能である (高阪, 2002)．アメリカ合衆国州人口および日本都道府県人口を連続エリアカルトグラムのアニメーションを用いて視覚化した例を下記の Web アドレスにて公開する．

http://planner.t.u-tokyo.ac.jp/gallery/carto/area.html

4.4　まとめ

本論文では新たな連続エリアカルトグラムの作成手法を提案した．連続エリアカルトグラム作成問題を簡略化するため，地域を三角網に分割することを前提とし，各三角形の面積を統計データに合わせるように三角網を変形する問題を設定した．その上で，統計データの表現に不必要な変形を抑え分かりやすい連続エリアカルトグラム作成を行うために，三角網を形成する各辺の方位角変化を抑える条件を設定し，不良設定問題である連続エリアカルトグラム作成問題を正則化した．この解法の線形化を通して，線形最小二乗問題の繰り返しにより連続エリアカルトグラムを作成するアルゴリズムを構築した．また，三角網分割の方法として Delaunay 三角網を採用し，実データを用いて本提案手法の適用可能性を検討した．

本提案手法の特徴は，目的関数が簡潔かつ数学的に明快に記述されており，設定項目が少なく操作性に優れており，また短時間で計算できるという点である．これは，従来手法の問題点を解決した実用的な手法である．本提案手法は，GIS の拡張機能として容易に実装できる手法であり，将来的に一般の GIS ユーザーが統計データを簡単に視覚化できる環境を提供することができる．

本論文では，カルトグラム上の地域形状と地理的地域形状の類似度を，ヘルマート変換の決定係数を指標として評価している．但し，この指標が人間の形状認知の容易さと関連があるかについては検証を行っていない．今後,「分かりやすい」カルトグラムについてより深く議論を行うためには，アンケート調査等を利用して人間の形状認知について調査を行い，形状の類似度指標を構築する必要がある．

また，本論文では従来手法との比較は行えていないのが現状である．連続エリアカルトグラム研究では共通の基準で表現精度・形状・計算時間を比較することが難しい．その要因は，例えば使用した地域形状のデータが異なる点，論文により表現精度の指標が異なる点等が挙げられる．また，連続エリアカルトグラム上の統計データの視認性の評価基準や地理的地図上と連続エリアカルトグラム上の地域形状の類似度の評価基準が論文によって異なる点も作成手法の比較が困難な理由の一つである．また，既往研究では作成アルゴリズムの過程が明快に記載されていないため，論文だけからその作成手法の再現を行うのが難しく，比較実験が行えないという問題もある．既往の作成手法を共通の基準で比較することは，連続エリアカルトグラム研究において今後の重要な課題である．

第5章　サークルエリアカルトグラム

サークルエリアカルトグラムは，エリアカルトグラムの中でも，地域の隣接関係を捨象してデータを表現する非連続エリアカルトグラムの一種である．各地域を円で表現し円の面積を用いて地域のデータを視覚的に表現する手法である．

第 2.2.2 節で詳述したように，既存のサークルエリアカルトグラム作成手法は必ずしも視覚的に分かりやすい作図を行えない．カルトグラム作成においてはデータ視認性は最重要の課題であり，読図者にとって分かりにくい手法には限界があると言わざるを得ない．既存手法の問題点は，カルトグラム上の円配置が地域の地理的配置と大きく異なる可能性を備えている点である．そこで，第 5 章では，地域の地理的配置を考慮して円のカルトグラム上配置を決定するサークルエリアカルトグラム作成手法の提案を行う．

5.1　連続エリアカルトグラムの限界

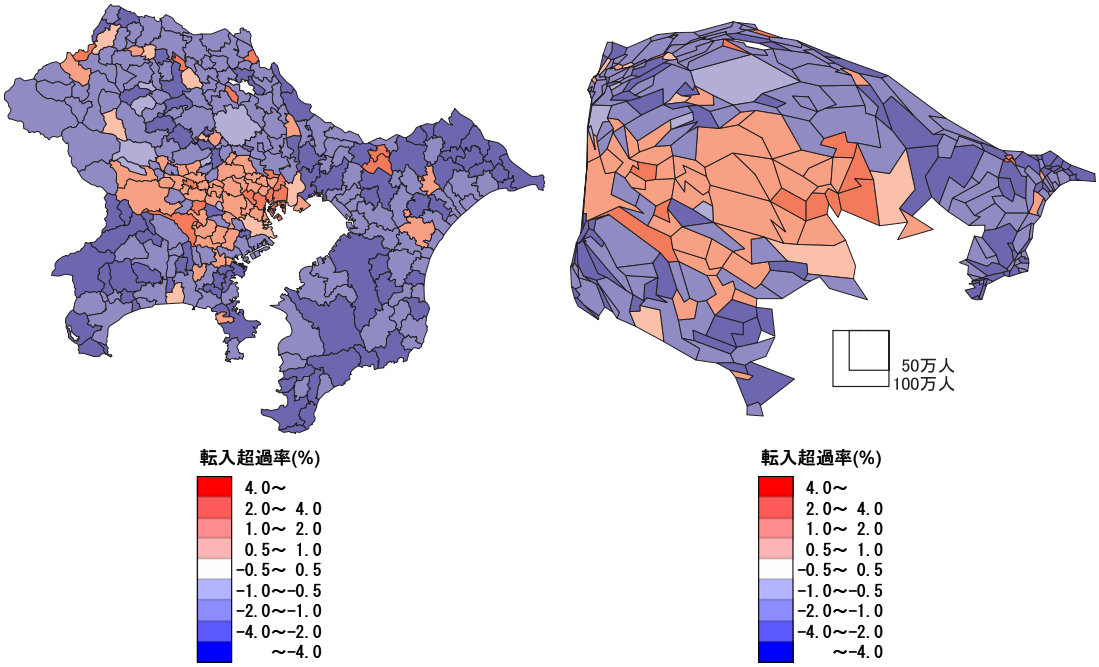
第 4 章で示した連続エリアカルトグラムは地域統計データの有効な視覚化手法である．しかし，連続エリアカルトグラムは万能ではなく，その利用には 3 つの問題が存在する．

第 1 の問題は，読図者が視覚化対象の地域形状に関する先験的な情報を十分に持っていない場合には，連続エリアカルトグラム上の地域形状と地理的な地域形状との比較対照ができないため，データの特徴を把握できない場合がある点である．第 4.3 節に示した，アメリカ合衆国の州形状や日本の都道府県形状を変形した連続エリアカルトグラムでは，地図に馴染みのない人でも国や州・都道府県形状は天気予報などを通して日常的に多くの人が目にしておりそれらの形状を認識しているため，地図上の形状の変形の大小を認識でき，表現されているデータの傾向を正しく解釈・理解することができる．

しかし，対象となる地域形状に関して先験的な知識を持っていない場合，連続エリアカルトグラムによる視覚化は効果的ではない．例えば，総務省統計局による 2000 年南関東 1 都 3 県の市区町村人口・転入超過率データを視覚化する場合を考える．図 5.1(a) は，対象市区町村を転入超過率に基づいて色分けしたコロプレスマップであるが，これらの市区町村人口を表すように連続エリアカルトグラムを第 4 章の手法を元に作成すると図 5.1(b) となる．しかし，読図者はもともと市区町村形状に対する先験的な知識をほとんど持っていないため，図 5.1(b) 上の地域形状の歪みを認識することが難しく，カルトグラム上に示された情報を正しく認識することはできない．

第 2 の問題は，連続エリアカルトグラムでは表現不可能な場合があることである．Keim *et al.* (2004) に示されるように，隣接する地域のデータが大きく異なる場合，隣接関係を保持したままデータを表現することは不可能である．

第 3 の問題は，作成に時間を要する可能性がある点である．例えば，2000 年の 46 都道府県の都道府県人口を表現する連続エリアカルトグラムを地理的地図から作成するには，CPU2.8GHz の計



(a) コロプレスマップ
(転入超過率)

(b) 連続エリアカルトグラム
(面積: 人口, 色: 転入超過率)

図 5.1: 2000 年 南関東市区町村 人口・転入超過率

算機では 20 秒程度の計算を要する．しかし，地域数の多い連続エリアカルトグラム作成にはさらに多くの時間を要する．例えば 288 市区町村の人口を表現する図 5.1(b) の連続エリアカルトグラム作成には，同じ計算機で 5 分弱の計算時間を要する．さらに地域数の多い場合にはさらに長時間を要することが予想される．そもそも，視覚化手法とは個人の思考や集団での議論を助けるために適宜用いられるものである．個人的な興味や遊び感覚での利用も多いため，その作成に長時間の計算を要する可能性がある手法は，視覚化手法として限界があると言わざるをえない．

そこで，エリアカルトグラムの簡便法として，地域の隣接関係の表現を省略する非連続エリアカルトグラムが提案されている (Olson, 1976)．その一種であるサークルエリアカルトグラムは，隣接関係だけでなく地域形状も捨象して各地域を円で表現し，円の面積の大小で統計データの大小を表す手法で，Dorling (1991) によって提案された．この手法は全ての地域を円で表現するため，面積の比較を行うことも容易になる．また，作成法が簡潔で作成操作も容易なため，ソフトウェア Anselin *et al.* (forthcoming) に実装されるなど，広く利用され始めている．

しかし，第 2.2 節で述べたように，既存の作成手法では必ずしも視認性の高い作図を行えるとは限らない．そこで，第 5 章では，より視覚的に分かりやすいサークルエリアカルトグラム作成手法を提案する．

なお，第 5 章では，総務省統計局の住民基本台帳人口移動報告から 2000 年南関東 1 都 3 県の市区町村別人口および転入超過率のデータを用い，人口を面積で，転入超過率を色で表現したサークルエリアカルトグラムを作成し提示する．

5.2　作成手法の提案

既存の手法の問題点より，視認性の高いサークルエリアカルトグラム作成には，　『円の重なりを解消する』と同時に，　『地理的位置関係を保つように配置する』ことが求められる．そこで，これら 2 つを満たす新たな作成手法の提案を行う．

まず，　『円の重なりを解消する』ために，隣接地域を表す円の中心間距離が，それらの円の半径の和となるように配置する．隣接している地域の組み合わせ ij の集合を L，円 i の半径を r_i，円 i, j の中心間距離を d_{ij} とすると，　の目的関数は，式 (5.1) で表される．

$$\min \sum_{ij \in L} (r_i + r_j - d_{ij})^2 \tag{5.1}$$

この式 (5.1) は，式 (3.1) のディスタンスカルトグラム作成問題の目的関数と等価であると言え，サークルエリアカルトグラム作成にはディスタンスカルトグラム作成手法を応用できることが分かる．また，第 3 章の提案解法 (式 (3.10)) は，方位角変化を抑制する正則化項を導入することにより，地理的形状に近いディスタンスカルトグラムを作成することができる手法である．この提案解法をサークルエリアカルトグラム作成に応用することにより，円のカルトグラム上配置が地域の地理的配置に近いサークルエリアカルトグラム作成を行うことが可能になる．

そこで，式 (3.10) をサークルエリアカルトグラム作成に応用した例を図 5.2 に示す．

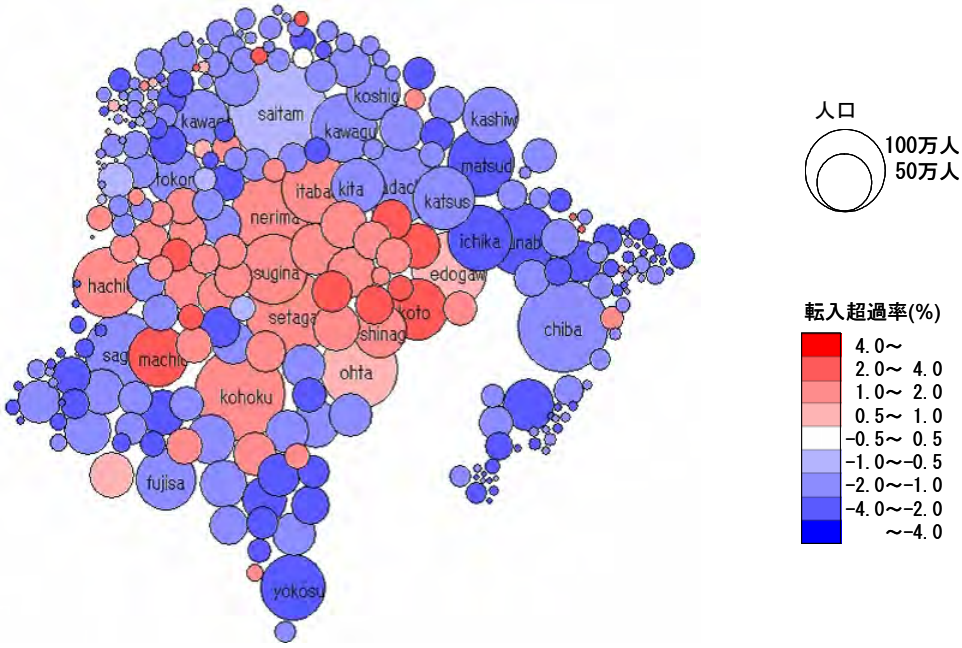
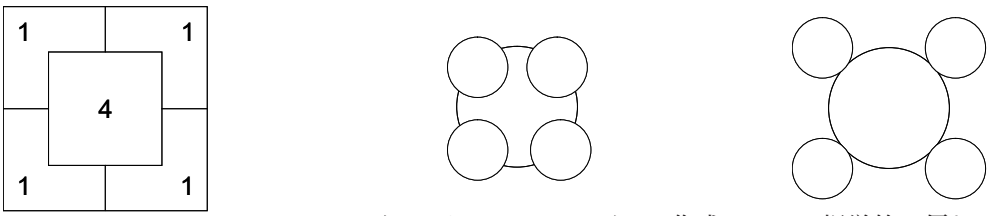


図 5.2: ディスタンスカルトグラム作成手法 応用 適用結果
(2000 年 南関東市区町村 人口・転入超過率)

図 5.2 では，図 2.2(c) よりも地理的位置関係を保持したカルトグラムになっている．しかし，円の重なりは大きく，例えば世田谷区・練馬区・江戸川区等を表す円は周囲の円の下に隠れてしまい，その大きさを図から判読することはできない．



(a) 対象地域形状と表現データ　(b) ディスタンスカルトグラム作成提案手法によるカルトグラム　(c) 視覚的に優れたカルトグラム

図 5.3: サークルエリアカルトグラム模式図

ディスタンスカルトグラム作成手法を応用する場合，全ての隣接する地域を表す円が接するように配置される．例えば，図 5.3(a) のデータを表現するサークルエリアカルトグラムを作成すると，小さな値が与えられた周囲の地域を表す円同士も接するように配置しようとするため，図 5.3(b) のような図が得られ，円の重なりが大きくなりデータを読みとりにくい図となってしまう．このように，隣接している地域を表す円が全て接するように配置することは不可能であり，時間地図作成手法をそのまま利用することはできない．

分かりやすいカルトグラム作成には，隣接して配置すると円が重なってしまう場合には，地理的位置関係を保ちつつ，円を離して表示することが望ましい (図 5.3(c))．そこで，地理的位置関係を保ちつつカルトグラム上の円の中心間距離を広げるため，地域間の地理的距離 (例えば，地域の重心間距離など) を利用し，カルトグラム上の円の中心間距離を与える．地域 ij 間の地理的距離を d_{ij}^0，縮尺変換の変数を $scale$，重み付けの変数を $\alpha\,(0 \leq \alpha \leq 1)$ とし，カルトグラム上円の中心間距離 d_{ij} を

$$\alpha\,(r_i + r_j) + (1-\alpha)\,scale \cdot d_{ij}^0 \tag{5.2}$$

に合わせるように与える．但し，$scale$ は全ての円が重ならなくなる最大の値 $\left(\max\left[(r_i + r_j)\,/d_{ij}^0\right]\right)$ とする．この結果，サークルエリアカルトグラム作成問題の目的関数は式 (5.3) と書ける．

$$\min \sum_{ij \in L} \left[\left\{\alpha\,(r_i + r_j) + (1-\alpha) scale \cdot d_{ij}^0\right\} - d_{ij}\right]^2 \tag{5.3}$$

式 (5.3) の目的関数に，ディスタンスカルトグラムの場合と同様に，リンクの方位角変化を抑制する正則化項を導入して解くことにより，3 つのサークルエリアカルトグラムの要件を満たし，視覚的に分かりやすいカルトグラムを作成することができる．

なお，繰り返し計算中には，もともと地理的には隣接していない地域を表す円が重なるように配置されることが生じる．そこで，新たにカルトグラム上で円が重なるように配置された地域の組み合わせを，隣接している地域の組み合わせの集合 C に加え，円の重なりを防ぐ．また，繰り返し計算中に重なっている円の組み合わせに対しては，次の繰り返し計算時に重み付けを行い，重み付き線形最小二乗問題を解くことにより，円の重なりの解消を図る．

本提案手法を適用した結果を図 5.4 に示す．$\alpha = 0.98$ とし計算している．このように，図 5.4 では，図 2.2(c) と異なり，地理的位置関係がほぼ維持されており，地域と円の対応関係を容易に認識することができる．計算時間は CPU2.8GHz の計算機で 4 秒であり，実用的な時間で計算が可能である．また，円の重なりもほぼ解消されており，情報を読み取ることができる．本提案手法にて，データ視認性の高いサークルエリアカルトグラムを作成できることが確認できた．

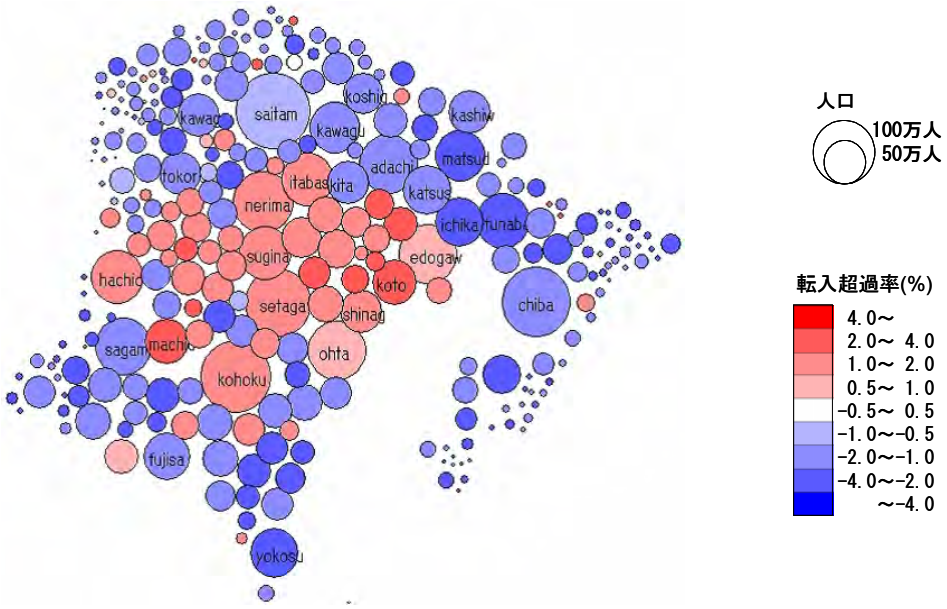


図 5.4: サークルエリアカルトグラム適用結果
(2000 年 南関東市区町村 人口・転入超過率)

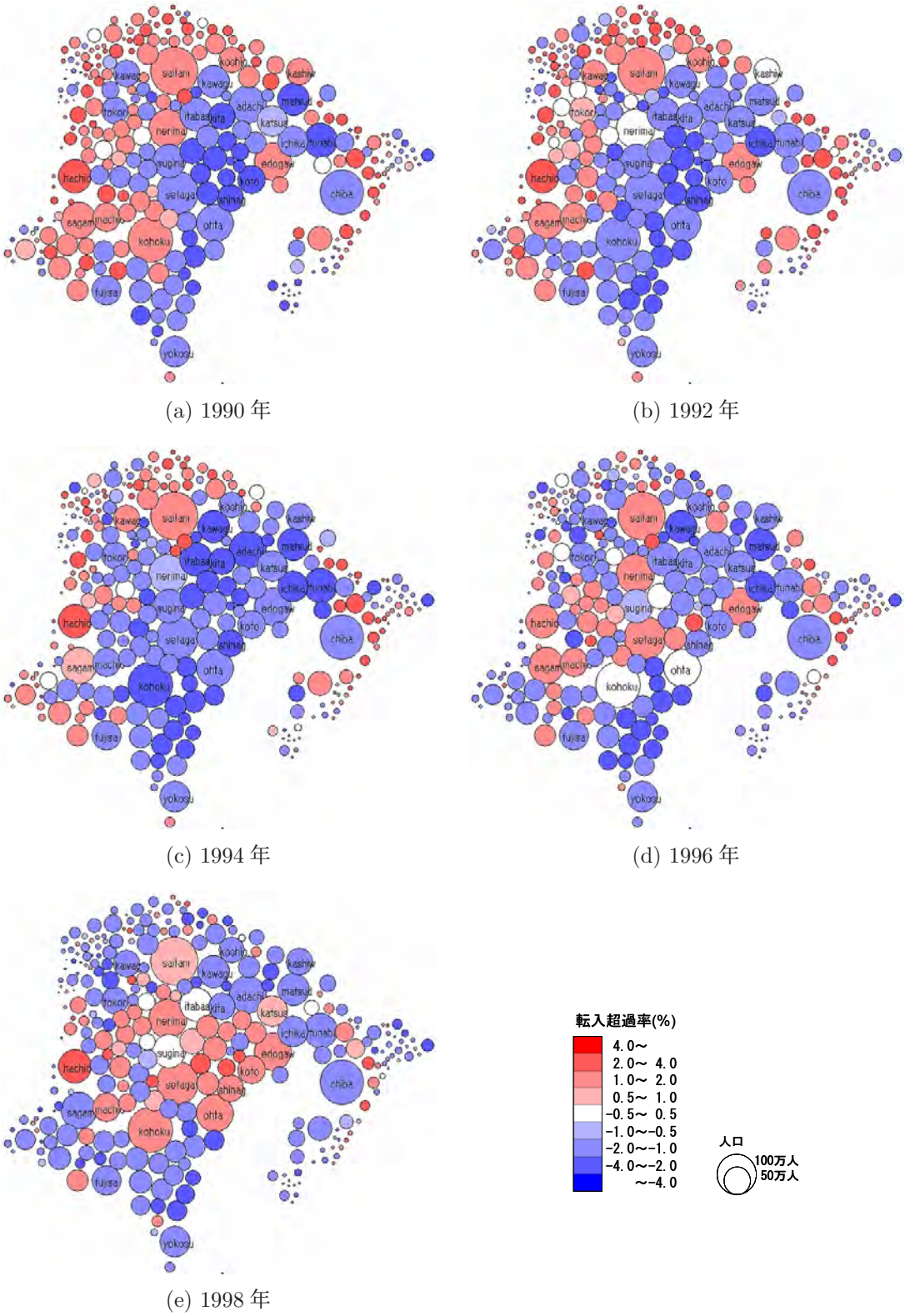
5.3　提案手法の適用

総務省統計局整備の，1990 年から 2000 年まで 2 年ごとの南関東 1 都 3 県の市区町村別人口および転入超過率のデータに対して，本提案手法を適用する．

本提案手法を適用した結果を図 5.5・5.6 に示す．このように，図 5.5・5.6 では，図 2.2 と異なり，地理的位置関係がほぼ維持されており，地域と円の対応関係を容易に認識することができる．また，円の重なりもほぼ解消されている．本提案手法にて，データ視認性の高いサークルエリアカルトグラムを作成することが可能なことを確認できた．

1994 年では，都心部では転出超過，郊外部では転入超過である．しかし，1996 年では都心部の内，人口の多い練馬区・世田谷区・江戸川区等で転入超過へと転換しはじめ，更に，1998 年では大半の都心部で転入超過となっている．一方，郊外部では，次第に転出超過となる市区町村が増え，2000 年では人口規模の小さい市区町村では，ほとんど転出超過となっていることが読みとれる．また，関東 1 都 3 県全体として人口がほぼ均衡している様子も視覚的に読みとることができる．

このように，サークルエリアカルトグラムの時系列の比較により,「都心回帰」と呼ばれる近年の人口移動の様子を視覚的に表現することができる．

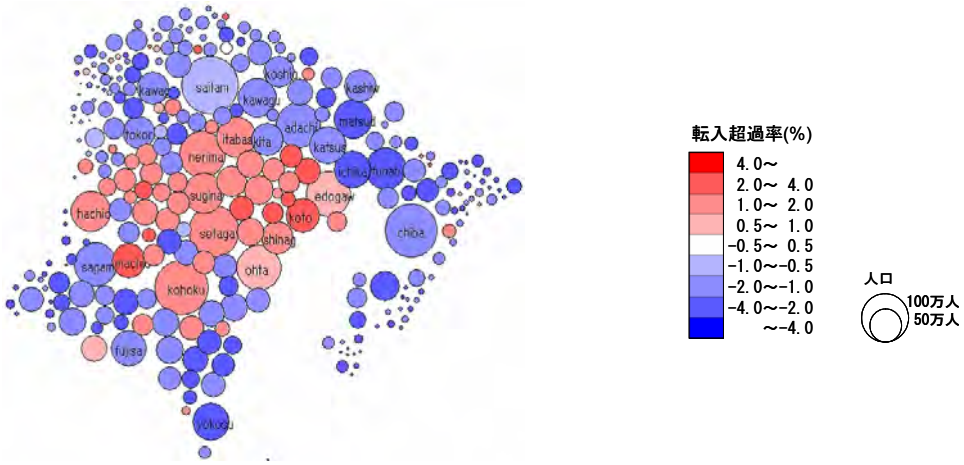


(a) 1990 年　(b) 1992 年

(c) 1994 年　(d) 1996 年

(e) 1998 年

図 5.5: 南関東市区町村 人口・転入超過率 サークルエリアカルトグラム 1990〜1998 年



(a) 2000 年

図 5.6: 南関東市区町村 人口・転入超過率 サークルエリアカルトグラム 2000 年

5.4　まとめ

第 5 章では，エリアカルトグラムの簡略手法であるサークルエリアカルトグラムについて新しい作成手法を提案した．提案手法は，円の重なりを排除すると同時に地理的位置関係を維持することを重視しながら，円を配置を決定する手法である．その結果，読図者にとってデータを読み取りやすいサークルエリアカルトグラムの作成を行うことができる．計算に当たっては，第 3 章で提案したディスタンスカルトグラム作成手法の提案解法を応用し，短時間で計算可能な方法を構築した．

第6章　カルトグラム上への内挿

第3・4章で示した提案手法を用いると，カルトグラムを通して様々なデータを視覚化することが可能になる．さらに，カルトグラム上に他の空間データを内挿することによって，より印象的なデータの視覚化が可能になる．

特に，ディスタンスカルトグラムでは地域境界を内挿する効果は大きい．第3章で示したネットワークを線で表示したディスタンスカルトグラムでは，視覚化対象の地点間を明示的に示すことはできるが，近接性データの全体的な傾向を把握することが難しい可能性も有している．しかし，海岸線等の地域境界をディスタンスカルトグラム上に内挿すると，例えば鉄道所要時間を縮尺とする特徴空間上で日本列島形状が大きく歪んでいる様子を表現することができ，近接性データの全体的特徴を理解しやすくなる．また，エリアカルトグラムの場合と同様に，アニメーション表現を用いて特徴空間の変化を視覚化することが可能になる．

一方，連続エリアカルトグラムでは，他の点データや線データを内挿することにより，エリアカルトグラム上に表現されたデータの等密度特徴空間上における空間分布を視覚化・分析することができる．既往の研究では，等人口密度空間上におけるサンフランシスコの疾病分布の分析 (Shaw *et al.*, 1988; Selvin *et al.*, 1988, 1992, 1993, 1998; Selvin and Merrill, 2002) に連続エリアカルトグラムが応用されている．また，例えば等人口密度空間上での道路ネットワーク配置など，ネットワーク配置の視覚化・分析にも利用できると期待される．また，ディスタンスカルトグラムと同様により詳細な地域境界を内挿することにより，より印象的なエリアカルトグラムを作成できる可能性も備えている．

第6章では，カルトグラム上への空間データの内挿法を構築し，その適用例を示す．

6.1　カルトグラムへの内挿に関する既往研究

ディスタンスカルトグラム上への内挿は，今まで多く検討されてきた．Muller (1978) では，都市の中心の一地点からの所要時間を表現するように極座標上で角度を変更せず半径のみを変え，カルトグラム上の地点座標を決定する．その後，極座標格子点のカルトグラム上における座標を推定し，格子点座標を基準に double linear interpolation により街路網や河川を内挿している．また，Ewing and Wolfe (1977) では，MDS を用いてカルトグラム上の地点配置を決定し，逆距離加重法を用いて内挿を行っている．しかし，Muller (1978); Ewing and Wolfe (1977) はこれらの手法では必ずしも位相は保持されないため，データによっては内挿による表現ができない場合があることを指摘している．そこで，清水 (1992); Shimizu (1992) は，同相写像となるアフィン変換や射影変換を用いてカルトグラム上の地点座標推定・内挿を行う手法を提案している．また，Spiekermann and Wegener (1993, 1994) は，位相破壊が起こらないようデータの一部分のみを用いて MDS を繰

り返しカルトグラム上の地点座標を決定した後，その地点座標を頂点として三角網分割を行いアフィン変換 (affine transformation) を用いた内挿を行っている．

しかし，第 3 章で提案したディスタンスカルトグラム作成手法は，必ずしも位相を保持しない手法であるため，内挿時に基準点となる地点の位相が反転している可能性がある．このような反転した基準点を元に内挿を行うと一対一の写像とならないため，視覚的に分かりやすいカルトグラムとならない．もちろん，ディスタンスカルトグラム上で位相が反転してしまうということは，その表現されているデータからなる特徴空間が大きく歪んでいることを示しており重要な情報である．しかし，内挿を用いた視覚化では，位相の反転など部分的な特徴空間の歪みを表現することは難しく，それらを捨象せざるを得ない．ディスタンスカルトグラム上に空間データを内挿する場合には，全ての地点の情報をカルトグラムに忠実に表現するのではなく，データの全体的な傾向を表現することを目的とすべきである．そのため，位相を保持していない基準点を探索・省略し，内挿を行う手法を構築する必要がある．

6.2　内挿手法の検討

カルトグラムに対して内挿を行う場合，地理的地図上における地点の位相がカルトグラム上で保たれることが必要である．清水 (1992) によると，位相関係を保持する写像 (同相写像) となるためには以下の条件が必要である．

(i)　1 対 1 の連続写像である．

(ii) その写像の逆写像も連続である．

この条件を満たす関数は無数に存在するが，本論文ではアフィン変換を元にした内挿手法を採用する．アフィン変換は，ユークリッド幾何学的な線型変換と平行移動の組み合わせによる図形の平行移動・回転・拡大縮小・斜交軸変換などを行う変換である．写像:$(x, y) \rightarrow (u, v)$ は式 (6.1) で表される．

$$\begin{aligned} u &= \beta_{u0} + \beta_{ux}x + \beta_{uy}y \\ v &= \beta_{v0} + \beta_{vx}x + \beta_{vy}y \end{aligned} \qquad \left| \begin{array}{cc} \beta_{ux} & \beta_{uy} \\ \beta_{vx} & \beta_{vy} \end{array} \right| \neq 0 \tag{6.1}$$

(但し，$\beta_{ux}, \beta_{uy}, \beta_{u0}, \beta_{vx}, \beta_{vy}, \beta_{v0}$ は未知パラメータ)

アフィン変換には 6 個の未知変数があるが，基準点一点あたり 2 本の観測方程式があるモデルなので 3 点以上の基準点があればパラメータ推定ができる．アフィン変換による内挿を行う場合，三角網 (TIN: triangulated irregular network) に分割し，各三角形内の点を異なるアフィン変換で内挿する手法が用いられることが多い．また，基準点全てを用いてアフィン変換のパラメータ推定を行い，パラメータ推定時の残差を普遍クリギング (universal kriging) を用いて空間的に配分すると，基準点を合わせた内挿を行うことができる．

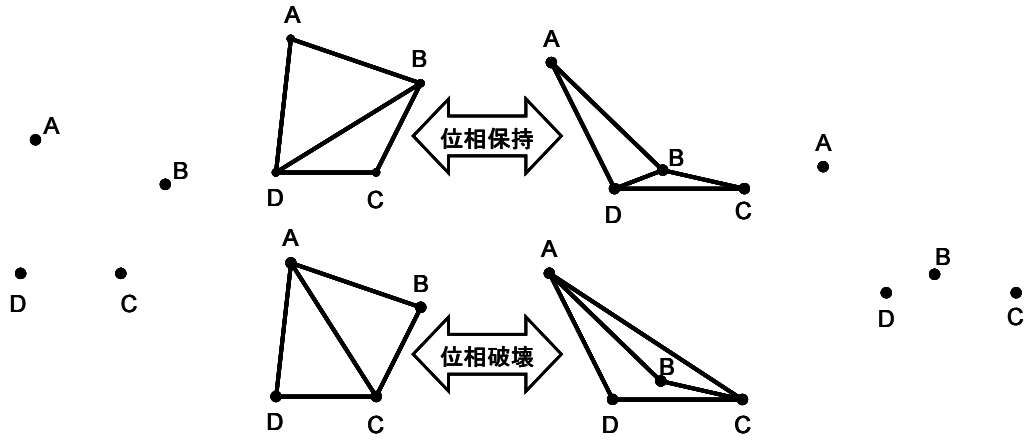
ここで，2 種類の内挿手法の長所・短所を整理する．

6.2.1　三角網分割・アフィン変換を用いた内挿

三角網分割を用いた内挿 (Wiebel and Heller, 1991; 張, 2001; 高阪, 2002) は，各三角形内の点を三角形の頂点座標を用いてアフィン変換する方法であり，簡潔な計算で実行可能である．各三角形

毎に異なった変換を行うため，三角網の辺上では 1 階微分が不連続になる．三角網の辺と交差する線がある場合，その交点で折れ曲がり新たな頂点を追加する必要が生じる．また，ラスターデータでも同様に，三角網の辺上でカルトグラム上の地域形状が折れ曲がるため，統計データの表現とは無関係な歪みがカルトグラムに生じてしまう．このような変形はカルトグラム上で示される統計データの理解を妨げるため，カルトグラムの表現には不適切である．

また，ディスタンスカルトグラムでは必ずしも基準点の位相が保持されている訳ではないため，内挿時には位相が保持されていない基準点を探索し除去する必要がある．しかし，三角網形状を用いて基準点の位相関係を判断すると一意な結果を得ることはできない．例えば，地理的地点配置 (図 6.1(a)) を変形して得られたカルトグラム上地点配置 (図 6.1(c)) が位置関係を保持しているかを判定する場合を考える．地点の地理的配置から三角網を作成し，同じ三角網をカルトグラム上の地点配置で作成し比較する．すると，図 6.1(b) のように，三角網の作成方法によって異なる結果を得る場合があることが分かる．このように，三角網を用いて地点の位相関係が保たれているかを判定する場合，一意な結果を得ることができない．



（a）地理的配置　　（b）三角網作成・比較　　（c）カルトグラム上配置

図 6.1: 三角網による位相判定の失敗

6.2.2　アフィン変換・普遍クリギングを用いた内挿

空間統計学の一手法であるクリギングは，空間現象を連続空間確率場でモデル化しその確率場上で観測されたデータから任意の位置での確率場の値を予測する手法である (例えば Cressie, 1993; 間瀬・武田, 2001; Wackernagel, 1998)．

空間統計学では，データは確率場からの実現値と見なす．まず領域 $D \subset \mathbb{R}^d$ 上の確率場 $Z := Z(\mathbf{s}); \mathbf{s} \in D$ を考える．この確率場に対して 1 次および 2 次モーメントの定常性，二次定常性とい

う強い仮定を置き確率場をモデル化する (式 (6.2)(6.3)).

$$E\{Z(\mathbf{s})\} = \mu. \tag{6.2}$$

$$Cov\{Z(\mathbf{s_1}), Z(\mathbf{s_2})\} = C(\mathbf{s_1} - \mathbf{s_2}). \tag{6.3}$$

この相対位置 $\mathbf{s_1} - \mathbf{s_2}$ のみに依存する関数 C をコバリオグラム (covariogram) と呼ぶ．さらに，等方性を仮定すれば，共分散を距離 h の関数として構造化することができ，距離によって減衰する関数を設定する (図 6.2)．観測データの分散を $Sill$ とし，距離 0 の観測点における観測データの共分散が分散と等しくならないことを $Nugget$ を用いて表す．また，共分散が存在する範囲を $Range$ で表し，$Range$ を超えた観測点間の観測データは共分散がないと仮定する．

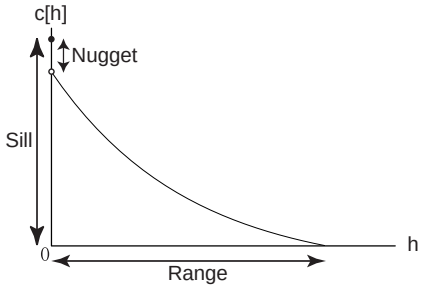


図 6.2: コバリオグラム 模式図

普遍クリギングでは，線形回帰モデル

$$y(\mathbf{s}) = \mathbf{x'}(\mathbf{s})\boldsymbol{\beta} + \epsilon(\mathbf{s}) \tag{6.4}$$

(但し，$y(\mathbf{s})$: 位置 $\mathbf{s}$ の観測データ，$\mathbf{x}(\mathbf{s})$: 位置 $\mathbf{s}$ の説明変数ベクトル，
$\boldsymbol{\beta}$: 線形回帰モデルのパラメータ，$\epsilon(\mathbf{s})$: 位置 $\mathbf{s}$ の誤差)

の誤差 $\epsilon(s)$ に二次定常性を仮定してコバリオグラム C の推定を行い，任意の位置 $\boldsymbol{s_0}$ におけるデータ $y(\boldsymbol{s_0})$ を予測する．予測誤差分散が最小となる線形最良不偏予測量 $\hat{y}(\boldsymbol{s_0})$ は

$$\begin{aligned}\hat{y}(\mathbf{s_0}) =& \mathbf{x'}(\mathbf{s_0})\left(\mathbf{X'V}^{-1}\mathbf{X}\right)^{-1}\mathbf{X'V}^{-1}\mathbf{y} \\ &+ \mathbf{c_0}'\mathbf{V}^{-1}\mathbf{y} - \mathbf{c_0}'\mathbf{V}^{-1}\mathbf{X}\left(\mathbf{X'V}^{-1}\mathbf{X}\right)^{-1}\mathbf{X'V}^{-1}\mathbf{y} \\ =& \mathbf{x'}(\mathbf{s_0})\hat{\boldsymbol{\beta}}_{\mathbf{GLS}} + \mathbf{c_0}'\mathbf{V}^{-1}\left(\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}_{\mathbf{GLS}}\right)\end{aligned} \tag{6.5}$$

(但し，$\mathbf{y} = [y(\mathbf{s_0}), \ldots, y(\mathbf{s_n})]'$: 観測データベクトル，$\mathbf{X} = [\mathbf{x}(\mathbf{s_1}), \ldots, \mathbf{x}(\mathbf{s_n})]'$: 説明変数行列，
$\mathbf{V} = \{C(\mathbf{s_i} - \mathbf{s_j})\}_{ij}$: 観測点誤差の分散共分散行列，
$\mathbf{c_0} = [C(\mathbf{s_1} - \mathbf{s_0}), \ldots, C(\mathbf{s_n} - \mathbf{s_0})]'$: 観測点と予測点の誤差共分散ベクトル，
$\hat{\boldsymbol{\beta}}_{\mathbf{GLS}} = \left(\mathbf{X'V}^{-1}\mathbf{X}\right)^{-1}\mathbf{X'V}^{-1}\mathbf{y}$: 一般化最小二乗推定量)

と表される．この普遍クリギングは，観測点の位置で予測する場合には必ず観測データと一致する，厳密な補間法である．

カルトグラムに対する内挿では，アフィン変換の誤差に対して二次定常性を仮定し普遍クリギングを行う．xy 座標から u 軸の座標値を求めるアフィン変換のモデルは，

$$u = \beta_{u0} + \beta_{ux}x + \beta_{uy}y + \epsilon_u \tag{6.6}$$

である．ここで，普遍クリギングによる内挿を行うため．誤差 ϵ_u に対して xy 平面上での距離に対する二次定常性を仮定し，コバリオグラム C を用いて共分散を距離に対して構造化する．誤差の共分散をコバリオグラム C で表すと，基準点の誤差の分散共分散行列 $\mathbf{V}$ は

$$\mathbf{V} = \begin{bmatrix} C(0) & C(d_{12}) & \cdots & C(d_{1n}) \\ C(d_{12}) & C(0) & \cdots & C(d_{2n}) \\ \vdots & \vdots & \ddots & \vdots \\ C(d_{1n}) & C(d_{2n}) & \cdots & C(0) \end{bmatrix} \tag{6.7}$$

(但し，d_{ij}: xy 平面上における観測点 ij 間距離)

と表される．コバリオグラムに対して関数形を仮定し，そのもとでパラメータを推定する．

次に，基準点座標を用いたパラメータ推定結果をもとに，xy 平面上の点 (x_0, y_0) を uv 平面上に写像する場合を考える．このとき，推定したコバリオグラムより，基準点の誤差と点 (x_0, y_0) の誤差の共分散ベクトルの推定値 $\hat{\mathbf{c}}$ は

$$\hat{\mathbf{c}} = \begin{bmatrix} \hat{C}(d_{01}) \\ \hat{C}(d_{02}) \\ \vdots \\ \hat{C}(d_{0n}) \end{bmatrix} \tag{6.8}$$

と表される．このとき，アフィン変換・普遍クリギングによる写像後の u 座標は

$$u_0 = \hat{\beta}_{u0} + \hat{\beta}_{ux}x_0 + \hat{\beta}_{uy}y_0 + \hat{\mathbf{c}}\hat{\mathbf{V}}^{-1}\mathbf{e_u} \tag{6.9}$$

(但し，$\mathbf{e_u}$: パラメータ推定時の残差ベクトル)

と表される．同様に v 座標も求めることができる．

この手法では，基準点を必ず一致させた内挿を行うことができる．また，コバリオグラムの設定によっては連続的な変換を行うことができ，この変換によって位相が破壊される可能性は低い．これらの特徴を活かして，本論文ではアフィン変換・普遍クリギングを用いた内挿手法を構築する．

但し，この手法では必ずしも同相写像とはならないため，新たに条件を設定する必要がある．また，前述のように，ディスタンスカルトグラム作成結果は必ずしも位相を保持するとは限らないため，位相が反転している基準点を除去した内挿を実行する必要性がある．

そこで，第 6.3 節で，アフィン変換・普遍クリギングを用いた内挿手法が同相写像となる条件を整理し，位相が反転している基準点の探索方法を提案する．その結果を踏まえて，カルトグラム上への内挿手法を構築する．

6.3　アフィン変換・普遍クリギングを用いた内挿手法の構築

6.3.1　同相写像となる十分条件

アフィン変換・普遍クリギングを用いた内挿手法が同相写像となるための十分条件を整理する．$(x,\, y) \in R_{xy}$, $(u,\, v) \in R_{uv}$ の領域で

$$\frac{\partial u}{\partial x}\frac{\partial v}{\partial y} - \frac{\partial u}{\partial y}\frac{\partial v}{\partial x} > 0 \tag{6.10}$$

$$\frac{\partial u}{\partial x},\ \frac{\partial u}{\partial y},\ \frac{\partial v}{\partial x},\ \frac{\partial v}{\partial y}\text{が連続} \tag{6.11}$$

が成り立てば，この写像は一対一変換となり，またその逆写像も一対一変換となるため，同相写像となる．また，アフィン変換・普遍クリギングでは，$\partial u/\partial x$, $\partial u/\partial y$, $\partial v/\partial x$, $\partial v/\partial y$ は基準点で極大値をとることから，基準点で式 (6.10) が満たされていることを確認すれば，一対一変換が保証される．

また，式 (6.10) によって，位相破壊が生じている基準点を探索・削除することができる．各基準点において式 (6.10) が成り立つか否かを検証すれば，同相写像を行うことができない基準点を検出することができる．しかし，式 (6.10) を満たさないことは，必ずしも基準点の位相が破壊されていることを意味しない．アフィン変換の誤差に対する共分散の構造化手法を変更すれば，式 (6.10) を満たさない基準点が変わる可能性がある．つまり，アフィン変換の誤差の共分散に対して，ある特定の構造化を行った場合に同相写像が可能か否かを判定しているだけにすぎず，本質的に基準点の位相が破壊されているかを判断するものではない．そのため，できる限り基準点の位相が反転していると判定されない共分散の構造化を行うことが望ましい．しかし，全ての基準点で式 (6.10) を満たせば，同相写像となる内挿を行うことができることは明らかで，式 (6.10) を満たさない基準点を除去して内挿を行うと，全体的な傾向を表現する内挿を行うことが可能である．

6.3.2　Gaussian 型共分散関数を用いた内挿

本節では，式 (6.11) を満たすコバリオグラムとして $Nugget = 0$ の Gaussian 型の関数を用いる場合について記す．

Gaussian 型共分散関数では，内挿点 $(x,\, y)$ と基準点 $1(x_1,\, y_1)$ の誤差の共分散 cov_1 は，式 (6.12) で定義される．

$$cov_1(x,\, y) = \sigma^2\, e^{-(d_1/\theta)^2} \tag{6.12}$$

(但し $d_1 = \sqrt{(x-x_1)^2 + (y-y_1)^2}$: 内挿点と基準点 1 の距離，$\theta$: $Range$)

なお，$Range$ に極端に小さい値を与えると，内挿点の位置は近傍の基準点のみの影響を受けて配置されることになり，直線が蛇行する曲線に変換される可能性があり，分かりづらい図となる可能性が高い．そこで，カルトグラム作成上では，$Range$ には例えば計算対象領域全てをカバーする値を設定するなど，大きい値を設定する方が望ましい．

この偏微分 $\partial cov_1/\partial x$ は

$$\frac{\partial cov_1(x,\, y)}{\partial x} = \frac{-2(x - x_1)}{\theta^2} e^{-(d_1/\theta)^2} \tag{6.13}$$

となる．この条件の下では

$$\frac{\partial u}{\partial x} = \hat{\beta}_{ux} - \frac{2}{\theta^2} \begin{bmatrix} (x - x_1)\ e^{-(d_1/\theta)^2} \\ (x - x_2)\ e^{-(d_2/\theta)^2} \\ \vdots \\ (x - x_n)\ e^{-(d_n/\theta)^2} \end{bmatrix}' \hat{\mathbf{V}}^{-1}\mathbf{e} \tag{6.14}$$

$$\frac{\partial u}{\partial y} = \hat{\beta}_{uy} - \frac{2}{\theta^2} \begin{bmatrix} (y - y_1)\ e^{-(d_1/\theta)^2} \\ (y - y_2)\ e^{-(d_2/\theta)^2} \\ \vdots \\ (y - y_n)\ e^{-(d_n/\theta)^2} \end{bmatrix}' \hat{\mathbf{V}}^{-1}\mathbf{e} \tag{6.15}$$

(但し，$\hat{\mathbf{V}}$: 基準点誤差の分散共分散行列推定値，$\mathbf{e}$: パラメータ推定時の残差ベクトル)

と表される．また同様に $\partial v/\partial x$, $\partial v/\partial y$ も求めることができる．

これらの値が基準点で式 (6.10) を満たしておれば，一対一の変換が可能である．また，満たしていない基準点があれば，除去して再計算を行う．

6.4　内挿の適用例

6.4.1　ディスタンスカルトグラムに対する適用

ディスタンスカルトグラムに海岸線の内挿を行い，全体の傾向を表現した例を示す．

日本 鉄道所要時間ディスタンスカルトグラム

1965 年鉄道所要時間ディスタンスカルトグラム (図 6.4) に対して，日本列島画像 (図 6.3) を内挿し視覚化した例を示す．

図 6.5 は，鉄道所要時間が地域によって異なり，日本列島が鉄道所要時間の特徴空間上で歪んでいる様子を視覚化している．ディスタンスカルトグラム上の全ての都市の位置は表現できていないが，全体的な傾向を把握することができる．しかし，図 6.5 は決して分かりやすい図ではない．日本列島全体を一度に内挿しようとすると，本来不連続な空間である本州・四国・九州の間の海峡部が連続的に変形されてしまい，海峡付近の地点が伸びて表現されるため，見やすい地図とならない．このような場合には，海峡部を挟んだ不連続な空間は別々の計算で内挿を行う必要がある．

そこで，図 3.10 の鉄道所要時間ディスタンスカルトグラムに，北海道・本州・四国・九州を別けて内挿計算を行った結果を図 6.6 に示す．図 6.6(a) では，図 6.5 とは異なって四国が本州・九州から遠く離れて配置されるようになり，図 3.10 の結果をより効果的に視覚化できる．また，時間の経過とともに日本列島が縮み，海峡部のトンネル・橋の整備によって北海道・四国が本州に近づく様子もはっきり視覚化することができる．

図 6.3: 日本 地図画像

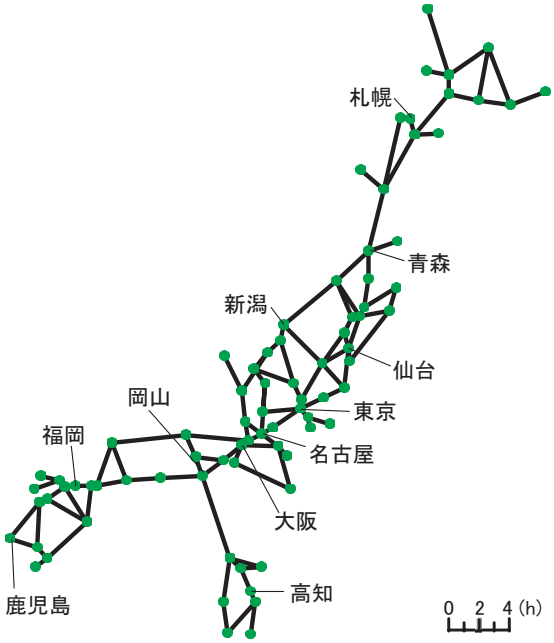


図 6.4: 日本 1965 年鉄道所要時間
ディスタンスカルトグラム
(図 3.10(a) 再掲)

図 6.5: 日本 1965 年鉄道所要時間 地図画像内挿

(a) 1965 年

(b) 1975 年

(c) 1985 年

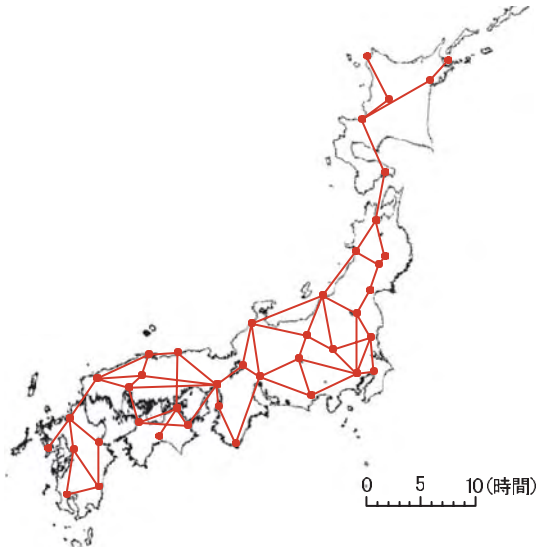
(d) 1995 年

図 6.6: 日本 鉄道所要時間変遷の視覚化 海岸線内挿

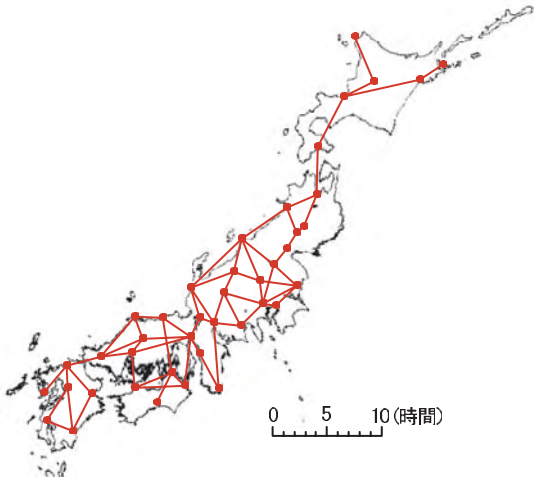
日本 道路所要時間ディスタンスカルトグラム

次に，図 3.14 に示した日本の道路所要時間ディスタンスカルトグラムに，図 3.13 に示す海岸線の内挿を行った例を図 6.7 に示す．

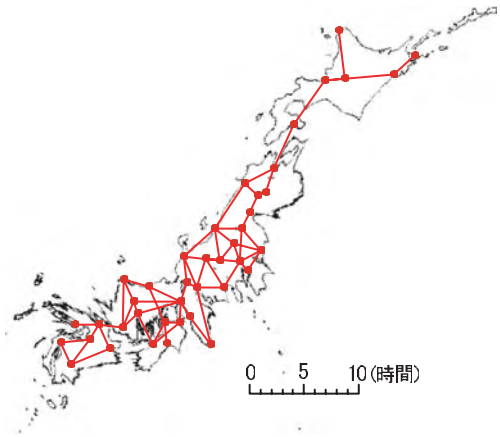
道路所要時間は鉄道所要時間よりも地域格差が小さいため，ディスタンスカルトグラムの形状の変化は大きくない．そのため，同時に全体を内挿を実行しても分かりやすい視覚化が可能である．



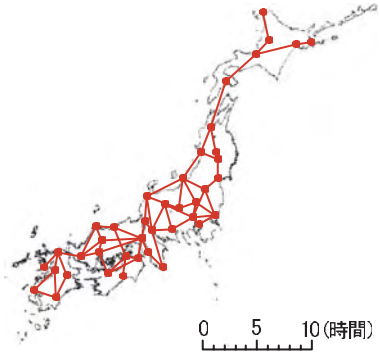
(a) 1955 年



(b) 1980 年



(c) 2001 年



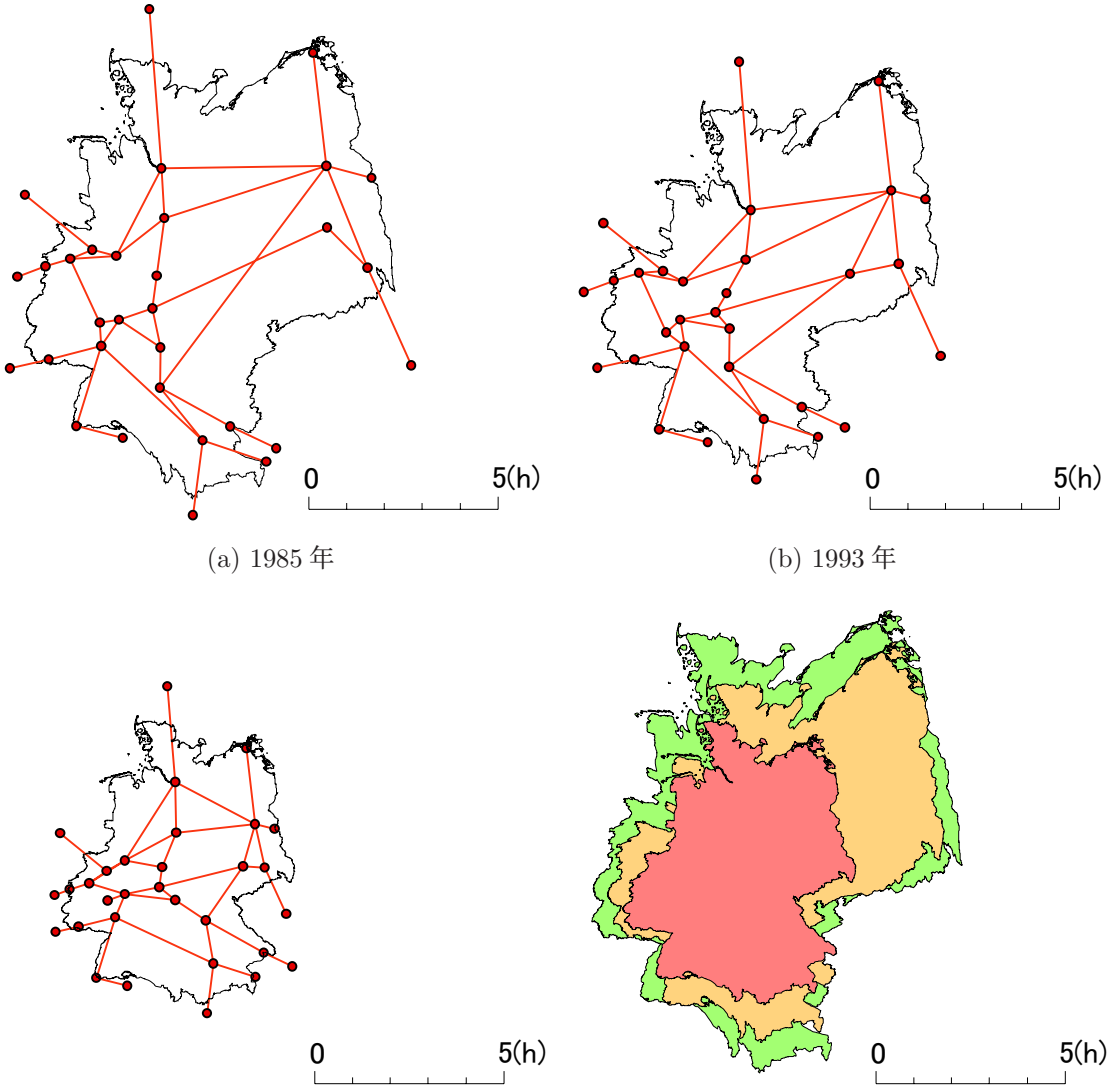
(d) 将来

図 6.7: 日本 道路所要時間変遷の視覚化 海岸線内挿

ドイツ 鉄道所要時間ディスタンスカルトグラム

図 3.15 のドイツ鉄道所要時間ディスタンスカルトグラムに，国境線を内挿した例を図 6.8 に示す．

1985・1993 年では旧東ドイツの鉄道整備水準が低いため大きく東に延びて歪んでいる．しかし，2010 年段階では国土の均衡ある発展を目指して全国的に高速鉄道網整備を実施する計画であることを読み取ることができる．



(a) 1985 年　(b) 1993 年

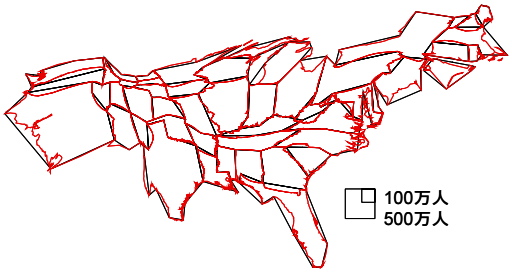
(c) 2010 年　(d) 1985〜2010 年

図 6.8: ドイツ 鉄道所要時間変遷の視覚化 国境線内挿

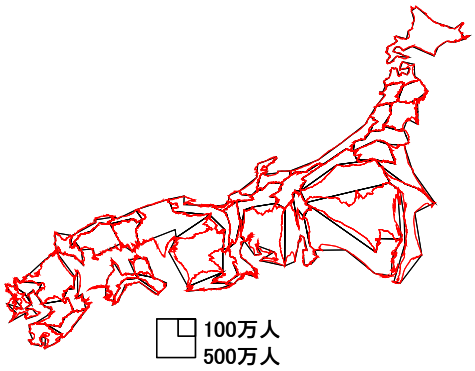
6.4.2　連続エリアカルトグラムに対する適用

まず，アメリカ合衆国州・日本都道府県・南関東市区町村の人口エリアカルトグラムに対して，地域境界の内挿を行った例を示す．

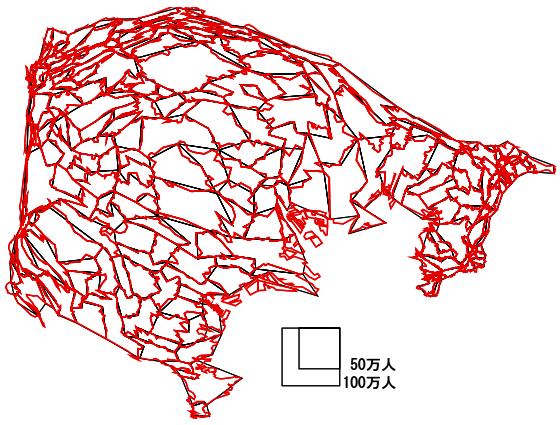
2000 年の人口を表すカルトグラムに対して内挿した例を図 6.9 に示す．黒線が内挿前の地域形状を表し，赤線が内挿後の地域形状を表す．3 種類とも基準点は全て一致するように内挿されており，より詳細な地域形状を表現する連続エリアカルトグラムを作成できた．内挿後のカルトグラムも概ねデータを正しく表現しているが，その精度は低下している．また，図 6.9(b) の大阪府に見られるように，内挿前後の地域面積が大きく異なり，データの表現精度が著しく低下する場合もある．このような場合，連続エリアカルトグラムに入力する地域形状をより詳細にするか，あるいは，内挿後の面積を元に連続エリアカルトグラム作成計算をやり直し修正する必要がある．



(a) アメリカ合衆国 2000 年 州人口



(b) 日本 2000 年 都道府県人口

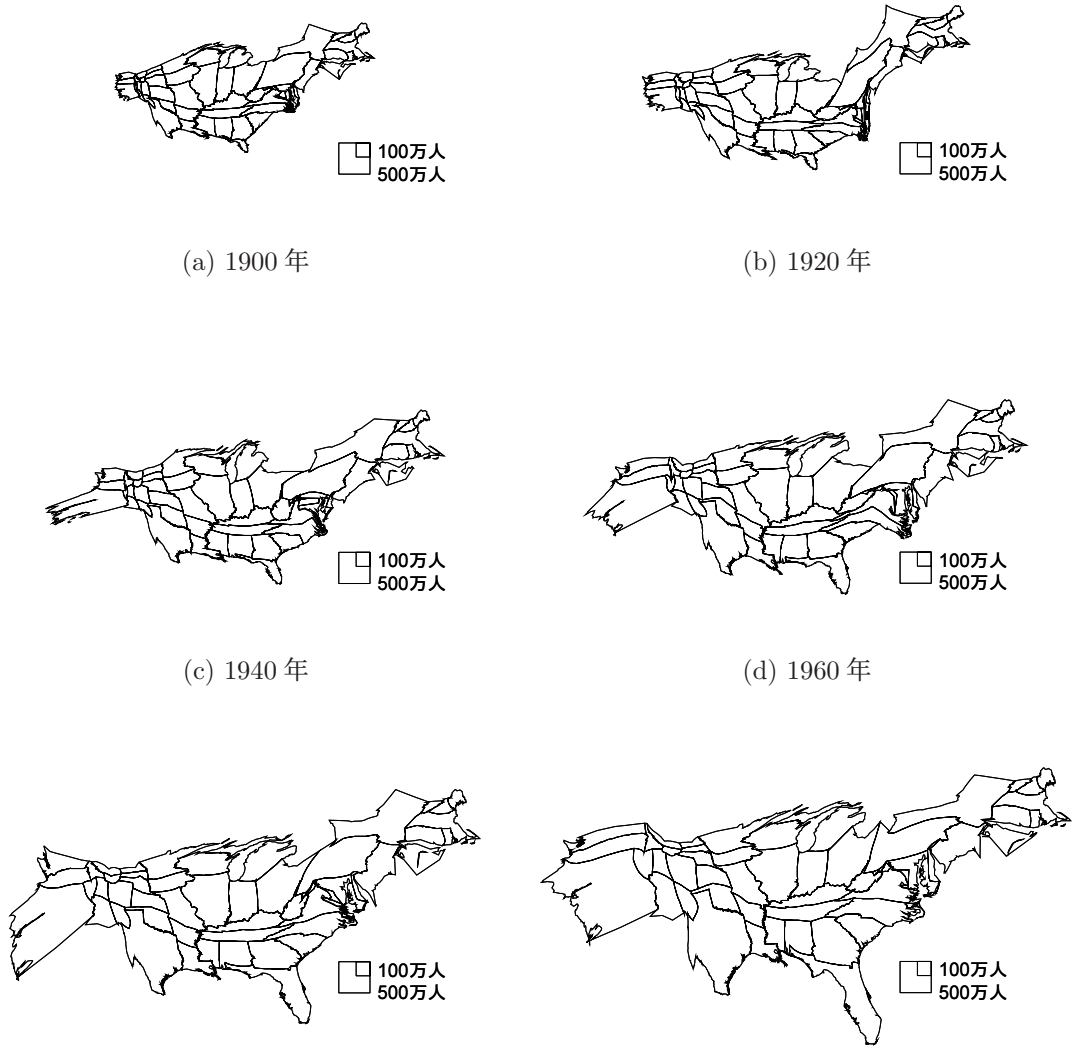


(c) 南関東 2000 年 市区町村人口

図 6.9: 連続エリアカルトグラム 地域境界内挿 評価

アメリカ合衆国 州人口 連続エリアカルトグラム

図 6.10 に，1900〜2000 年 (20 年毎) のアメリカ合衆国州人口連続エリアカルトグラムにより詳細な州境界を内挿した例を示す.



(a) 1900 年　(b) 1920 年

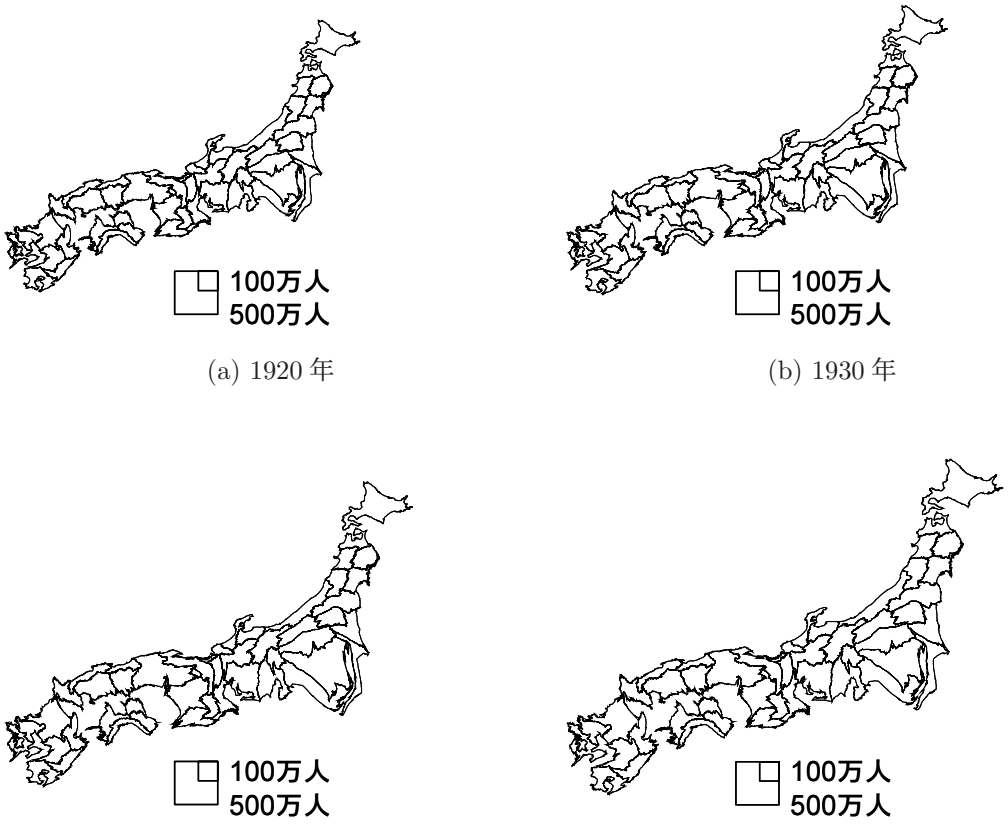
(c) 1940 年　(d) 1960 年

(e) 1980 年　(f) 2000 年

図 6.10: アメリカ合衆国 州人口連続エリアカルトグラム 州境界内挿 1900〜2000 年

日本 都道府県人口 連続エリアカルトグラム

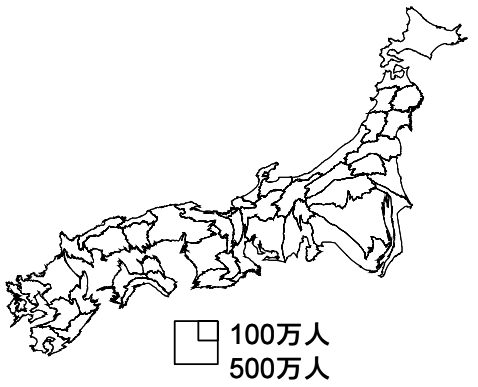
図 6.11・6.12 に，1920〜2000 年 (10 年毎) の日本の都道府県人口連続エリアカルトグラムに詳細な都道府県界を内挿した例を示す.



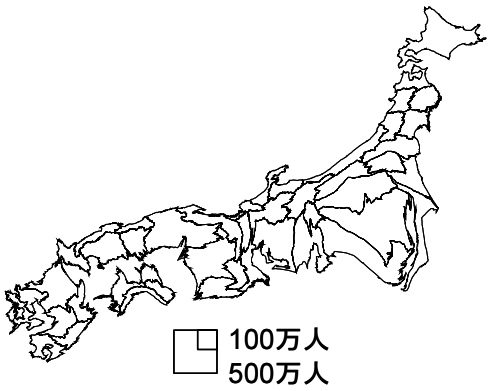
(a) 1920 年　(b) 1930 年

(c) 1940 年　(d) 1950 年

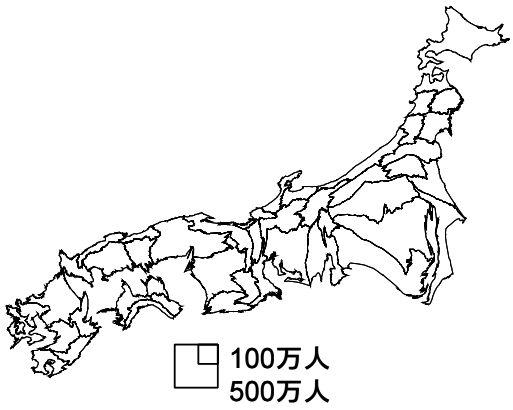
図 6.11: 日本 都道府県人口 連続エリアカルトグラム 都道府県界内挿 1920〜1950 年



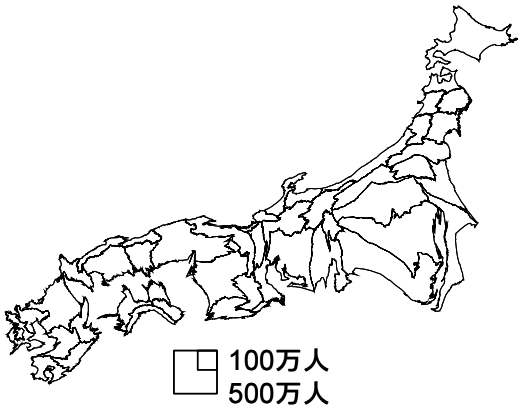
(a) 1960 年



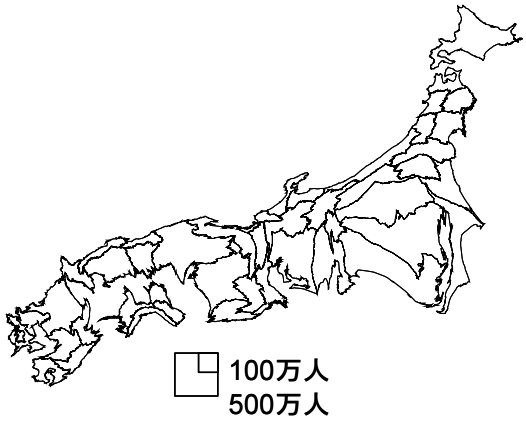
(b) 1970 年



(c) 1980 年



(d) 1990 年



(e) 2000 年

図 6.12: 日本 都道府県人口 連続エリアカルトグラム 都道府県界内挿 1960〜2000 年

南関東 市区町村人口 連続エリアカルトグラム

図 5.1(b) の南関東市区町村人口連続エリアカルトグラムに市区町村界を内挿した例を図 6.13 にしめす.

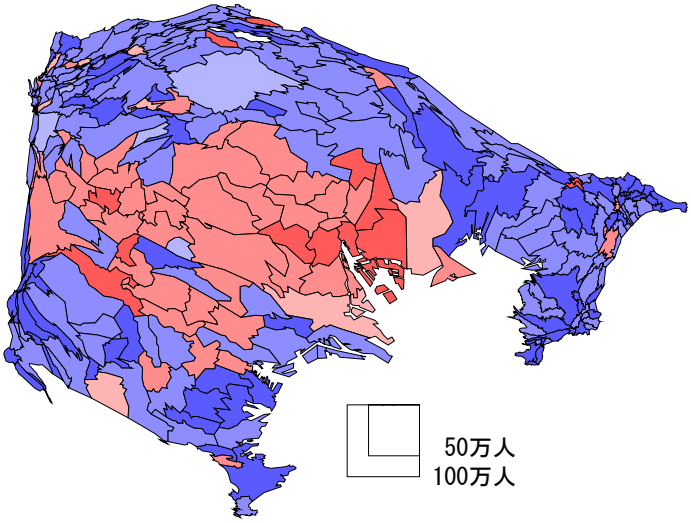
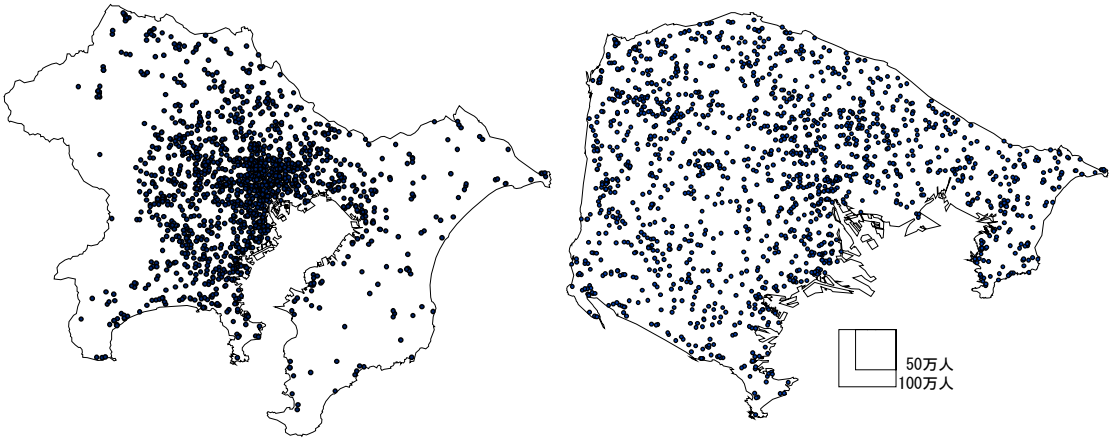


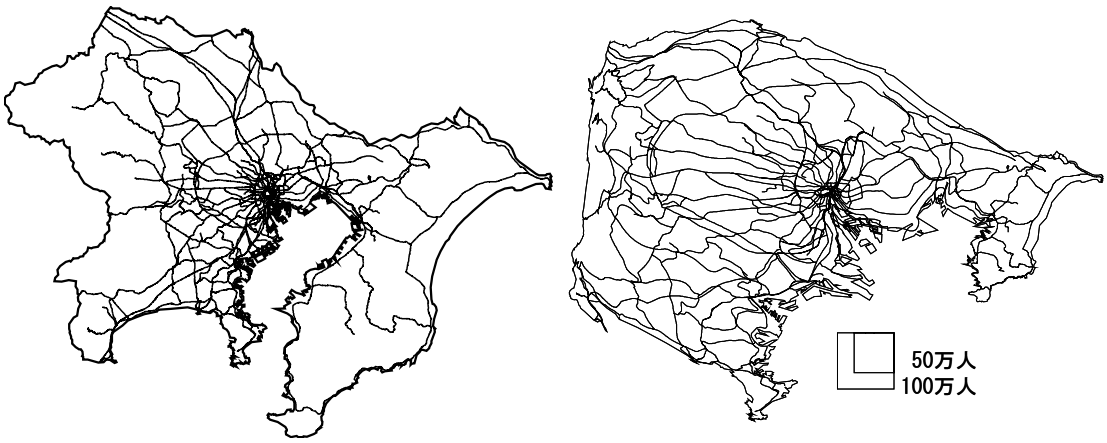
図 6.13: 南関東 2000 年市区町村人口 連続エリアカルトグラム 市区町村界内挿

また，他の空間データをカルトグラム上に内挿することにより，例えば等人口密度空間のような特徴空間上での空間分布を視覚化することが可能になる．ここでは，南関東市区町村人口連続エリアカルトグラム上に，ポイントデータの例として数値地図 25000(地名・公共施設) の「一般病院」(中分類コード:17，小分類コード 01，図 6.14(a)) を，ラインデータの例として数値地図 25000 の「鉄道区間」(但し旅客鉄道のみを抽出，図 6.15(a)) を内挿した例を示す (図 6.14(b)・6.15(b)).



(a) 病院分布　　(b) 人口エリアカルトグラム 病院位置内挿

図 6.14: 南関東 2000 年市区町村人口 連続エリアカルトグラム 病院内挿



(a) 鉄道路線網　　(b) 人口エリアカルトグラム 鉄道網内挿

図 6.15: 南関東 市区町村人口 連続エリアカルトグラム 鉄道網内挿 2000 年

6.5　まとめ

第 6 章では，カルトグラム上に空間データを内挿する手法を構築した．この手法は，アフィン変換と普遍クリギングを利用し，同相写像を行うことができる．また，基準点の位相が破壊されているデータを入力した場合，その原因となる基準点を探索することも可能である．

本手法の適用によって，カルトグラムに空間データを内挿し，データの表す特徴空間の歪みやその変遷をより視覚的に表現したり，特徴空間上における事象の空間分布を視覚化・分析することが可能になる．

第7章　カルトグラム作成ツールの開発

第 7 章では，第 3・4 章で提案したカルトグラム作成手法を多くの GIS ユーザーの利用に供するため，GIS 上で動作するカルトグラム作成ツールの開発を行う．具体的には，最も一般に普及している GIS ソフトウェアの一つである ArcGIS にカルトグラム作成手法を拡張機能として組み込む．この結果，GIS 上で管理されている様々な統計データを元に簡潔な操作でカルトグラムを作成し視覚化することが可能になる．

なお，カルトグラム作成ツールは ArcGIS 8.3 の拡張機能として整備した．初期値・データの入力等を行う GUI (Graphical User Interface) を作成し，ArcGIS でデータの管理を行う．カルトグラム作成結果の画面表示・保存・印刷等の出力は，ArcGIS の機能を用いて行う．カルトグラム作成アルゴリズムは C++で実装し，ArcGIS との接続は Visual Basic で開発した．

7.1　ディスタンスカルトグラム作成ツール

ディスタンスカルトグラム作成ツールの主要機能は，カルトグラム作成に使用するデータ入力機能，作成計算機能，作成済みカルトグラムデータの管理・出力機能である．それぞれに対して，GUI が実装されており，ユーザーは直感的かつ簡潔な操作でディスタンスカルトグラム作成を行うことができるようになっている．

まず，ArcGIS 上からディスタンスカルトグラム作成ツールを起動するとメニュー画面を表示する (図 7.1)．このメニュー画面から，ディスタンスカルトグラム作成に関わる様々な操作を行う画面へ移動することができる．

第 7.1 節では，ディスタンスカルトグラム作成の流れに沿って，作成ツールの機能について記す．

7.1.1　データ入力

第 3 章で示したように，提案解法によるディスタンスカルトグラム作成に必要な入力データは，視覚化する地点間リンク設定・近接性指標と配置対象地点の地理的座標である．また，完全ネット型・部分ネット型の折衷解法を用いる際にはリンクに対する重みのデータを入力する必要がある．

まず，カルトグラム上配置対象地点の地理的座標を入力する (図 7.2)．入力画面内に経度・緯度の数値による入力と，地図上のクリックによる入力が用意されている．他に，“外部データ読み込み” ボタンから，Tab あるいはカンマ区切りのテキストファイルからの座標入力も可能である．また，各地点には，3 種類の属性が与えられるようになっている．

次に，地点間リンク設定と近接性指標の入力を行う．地点間リンクの設定は図 7.3 で行う．まずリンク設定に使用する地点座標データを選択すると，ArcGIS のマップ上に地点がプロットされる．

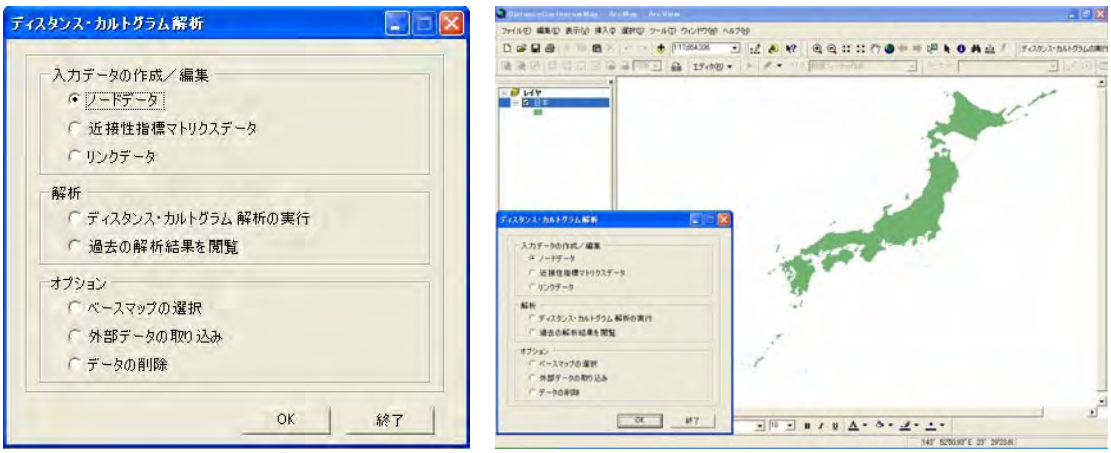


図 7.1: ディスタンスカルトグラム作成ツール メニュー画面

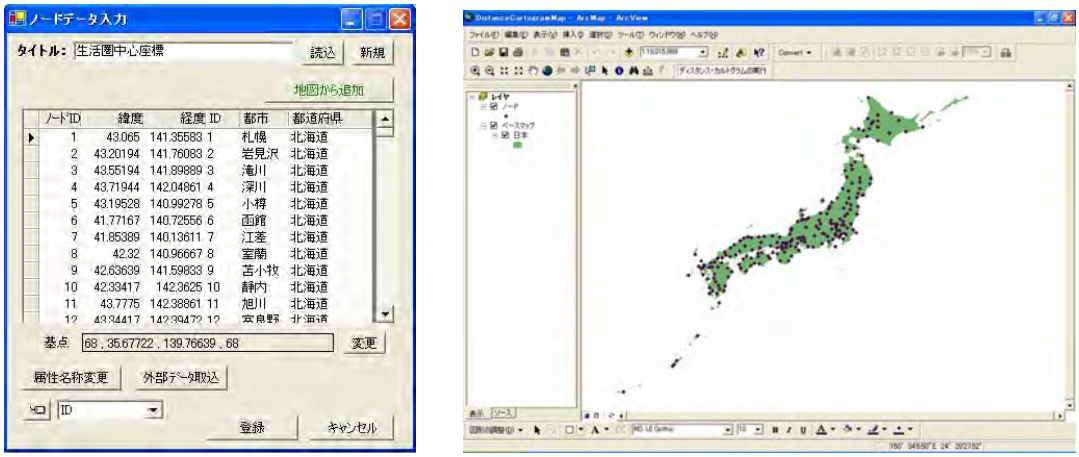


図 7.2: ディスタンスカルトグラム作成ツール 地点座標 入力画面

その後，ドロップダウンメニューからリンクの両端の地点を選択すると，マップ上に二地点を結ぶリンクが表示される．この時，ディスタンスカルトグラムを座標に固定するため，原点に設定する地点を選択する必要がある．近接性指標の入力は，リンク設定時に同時に行えると同時に，近接性行列による入力 (図 7.4) も用意されている．近接性行列からリンクに近接性データを与えることも可能で，またリンク設定画面から近接性行列のデータを更新することも可能である．また，地点座標入力と同様に，テキストファイルからデータのインポートをすることも可能である．

7.1.2　ディスタンスカルトグラム出力

データの入力後に，メニュー画面の “ディスタンスカルトグラム 解析の実行” からディスタンスカルトグラム作成計算を行う．ArcGIS 上のマップ上に新しいレイヤ “解析結果” が作成され，ディスタンスカルトグラムが出力される (図 7.5)．図 7.5 は，1965 年鉄道所要時間ディスタンスカルトグラム (原点: 東京) の出力例である．

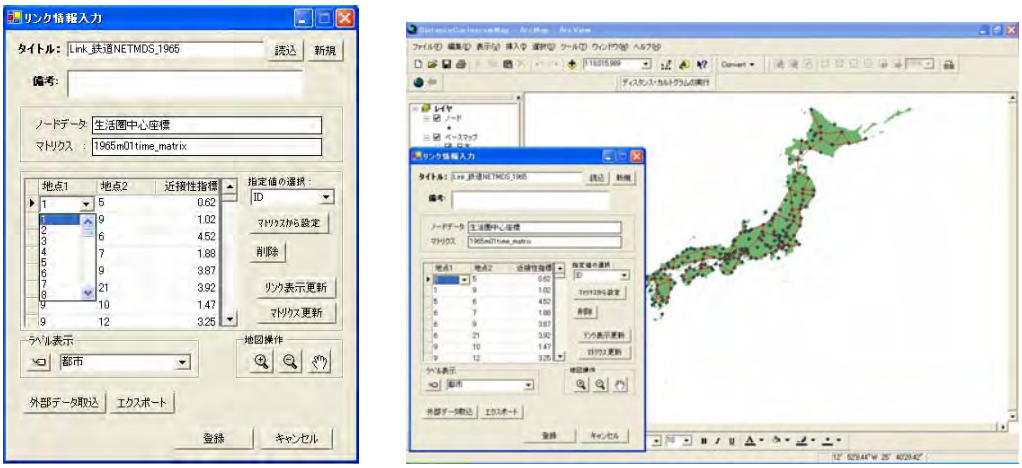


図 7.3: ディスタンスカルトグラム作成ツール 地点間リンク 入力画面

図 7.4: ディスタンスカルトグラム作成ツール 地点間近接性行列 入力画面

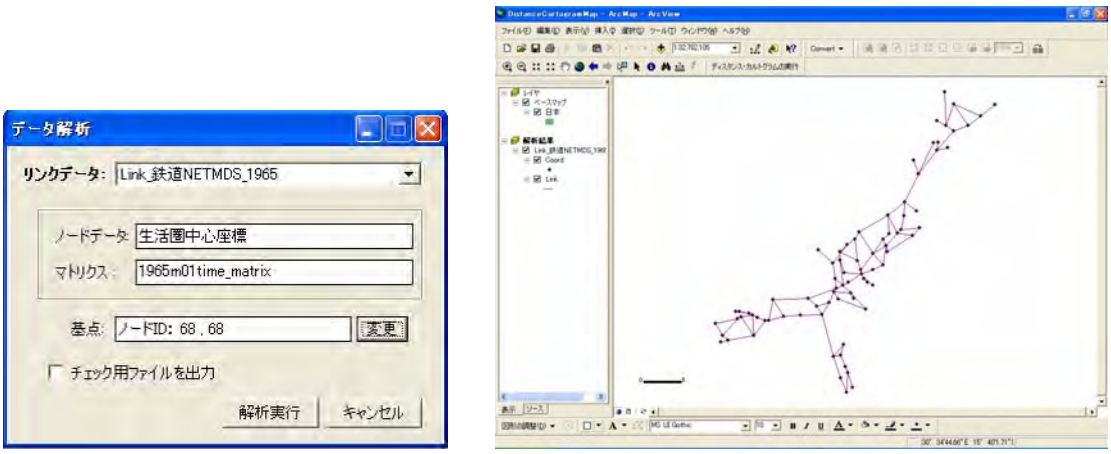


図 7.5: ディスタンスカルトグラム作成ツール 作成計算

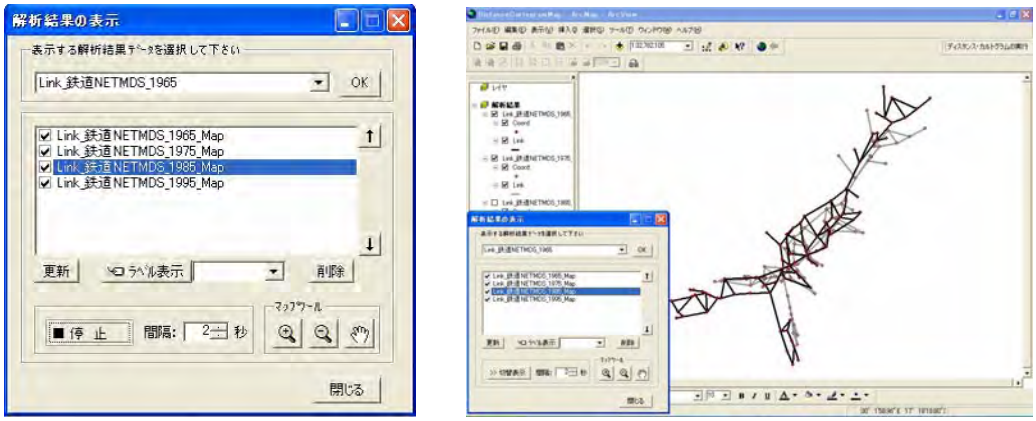


図 7.6: ディスタンスカルトグラム作成ツール 結果出力画面

マップ上には，ディスタンスカルトグラムとその縮尺が表示される．図 7.5 では左下に 5 時間の縮尺が示されている．

また，既に作成したカルトグラムとの比較を行いたい場合には，メニュー画面の “過去の解析結果を閲覧” からディスタンスカルトグラムを読み込む．図 7.6 の “切替表示” を用いると，数枚のディスタンスカルトグラムを交互に表示して統計データの変遷を視覚的に表示することができる．例えば，図 7.6 左図の設定では，1965〜1995 年の鉄道所要時間ディスタンスカルトグラムを順番に表示する．また図 7.6 右図では 1965・1975 年二時点のディスタンスカルトグラムを表示しており，鉄道所要時間の変化を容易に比較することができる．また，これらの作成結果は ArcGIS の機能を用いて，シェープファイルやパーソナルジオデータベース フィーチャークラスとして保存することができる．

このように，本ツールを利用すると，GUI を通して簡単にデータ入力・カルトグラム作成が可能である．

7.2　連続エリアカルトグラム作成ツール

連続エリアカルトグラム作成ツールの主要機能は，カルトグラム作成に使用するデータ入力機能，カルトグラム作成・出力機能と複数のカルトグラムを補間して表示するアニメーション作成機能である．

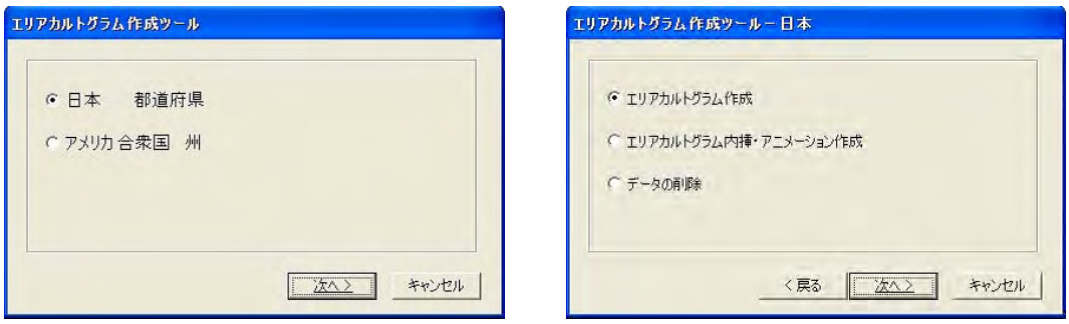
本作成ツールでは，日本都道府県とアメリカ合衆国州の形状データをあらかじめ用意し，これらを基にしたカルトグラムのみ作成できるようになっており，ユーザーが自らの地域形状データを入力して計算できるものとはしていない．この原因は，連続エリアカルトグラム作成手法の入力データにはいくつかの制約があるためである．第一の制約は，全地域が連続していなければならない点である．例えば，日本の連続エリアカルトグラム作成を行う場合には，北海道・四国・九州等が本州と連続していないためそのまま適用することができない．そこで，海峡部等にダミーのデータを設定する必要がある．第二の制約は，エリアカルトグラム計算対象内にデータがない地域がある場合，作図できないという点である．例えば人口を表現するエリアカルトグラムを作成する場合，湖沼が対象地域に含まれていると，人口データがないためにつぶれてしまい，カルトグラムに無用な

変形を生じる．そのため，ダミーの人口データを湖沼に与えるか，あるいは地域形状データから湖沼を削除する必要がある．また，東京湾などの閉鎖湾域を含んだ地域では，対岸に位置するポリゴンの位置関係を直接には定めることができないため，データによっては湾を挟んだ地域がカルトグラム上で重なった結果が得られる場合がある．このような状態を避けるため，湾などにもダミーのデータを与えなければならない．第三の制約として，地域データのトポロジーが正確でなければならない．すなわち，隣接する地域の同一境界上の頂点座標は全て等しくなければならないが，GIS データによっては，この条件が満たされず，エラーを発生してしまう．その他にも，例えばマルチポリゴンがある場合にはデータを按分しなければならないなど，ユーザーが自由に地域形状データを入力して連続エリアカルトグラムを作成できるようにするには支障が多い．そこで，作成ツールではあらかじめ地図データを内部で保持し，ユーザーはデータ入力のみを行う用に設定している．また，海峡や内湾部には，入力されたデータをもとに自動でダミーのデータを作成するが，ユーザーがカルトグラム形状を見て自由に調整することも可能にしている．

第 7.2 節では，の流れに沿って，開発したツールの機能について記す．

7.2.1　連続エリアカルトグラム作成

本作成ツールを起動すると，“日本 都道府県”・“アメリカ合衆国 州” のいずれの連続エリアカルトグラムを作成するかを選択する画面が表示される (図 7.7(a))．“日本 都道府県” を選択すると，次にエリアカルトグラム作成のメニュー画面へと移動する (図 7.7(b))．この画面から “エリアカルトグラム作成”・“アニメーション作成”・“データ管理” の各機能へ移動することできる．



(a) 作成対象選択　　(b) 作成ツール メニュー

図 7.7: 連続エリアカルトグラム作成ツール メニュー画面

“エリアカルトグラム作成” では，まず初期地域形状選択を行う (図 7.8(a))．もちろん，地理的地域形状を選択することもできるが，データの時系列変化を視覚化するためにアニメーション作成を行う場合には，前時点のエリアカルトグラムを初期地域形状として入力すると，データの変化に無関係な地域形状変形を避けることができるため望ましい．

次に，視覚化するデータの入力画面 (図 7.8(b)) に移動する．ここでは視覚化するデータを数値でセルに入力することもできるが，ArcGIS にあらかじめ読み込まれている他のテーブルのデータを結合することも可能である．

データ入力後，正則化項の重みに関する初期値設定を行う．第 4.3 節で述べたように，正則化項の重みを小さくすると計算時間は短くなるがカルトグラム上の地域形状変化は大きくなる．この重

み設定を簡便化するため，図 7.8(c) のようにつまみを用意した．ここで，“作成” を実行すれば図 7.8(d) のようにエリアカルトグラムをマップ上に出力する．

7.2.2　連続エリアカルトグラム アニメーション作成

連続エリアカルトグラムをアニメーション表示すると，データの変化を印象的に視覚化することができる．そこで，複数の連続エリアカルトグラムをもとに中間画像を自動作成し，スムーズなアニメーションを作成するツールを用意した．

まず，メニュー画面から “アニメーション作成” を選択すると，アニメーション作成データ選択へと移動する (図 7.9(a))．ここで，アニメーション作成時に基準となるエリアカルトグラムを 2 枚以上選択し，表示させる順序に並べ替える．

次に，中間フレームに属性を与える方法を選択する (図 7.9(b))．前後のカルトグラムの属性データを与える方法や属性データを線形補間する方法を用意している．人口データなど各時点で計測されたデータを配分する場合には線形補間して与えることもできるが，人口増加率のようにある一定期間の状態を表す属性データの場合は前後いずれかのデータを与えなければならない．

その後，フレーム数・シンボル設定画面に移動する．まず，各エリアカルトグラム間に挿入する中間フレームの枚数を選択し，“マップの表示” ボタンを押すと全フレームをマップ上に生成する．また，一枚目のフレームを表すマップにシンボル設定を行い “シンボル一括変更” を押すと，全てのフレームを同じシンボル設定で表示できる．

次に，アニメーションファイル保存先設定画面で，フレーム一枚あたりの表示時間を設定し，保存先を指定すると AVI ファイルが作成される．

このように，簡単な操作でアニメーションファイルを作成することできるため，データの空間分布の違いや変遷を視覚化するツールとして連続エリアカルトグラムの有用性を高めることができる．

7.3　まとめ

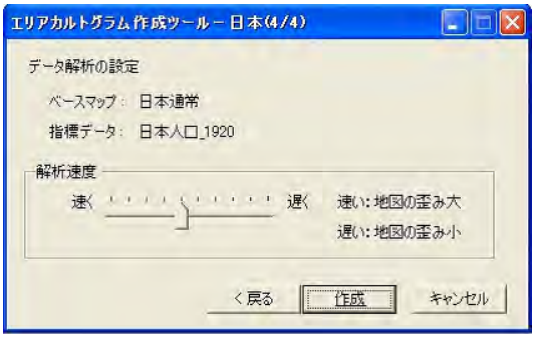
GIS 上で動作するディスタンスカルトグラムと連続エリアカルトグラム作成ツールの開発を行い，その機能について示した．本作成ツールでは，第 3・4 章で述べた提案手法の数学的明快さという特徴を活用し，複雑な初期値設定なしに 2 種類のカルトグラムの作成が可能である．また，GUI を通したデータ入出力・設定によって，簡単な操作でカルトグラム作成を実行できる．

前述のように，カルトグラムは有効な視覚化手法として注目されてきたにも関わらず，その作成の困難さから今まで利用が妨げられてきた．本作成ツールはカルトグラム作成の困難さを解消でき，多くの GIS ユーザーがカルトグラムという統計データ視覚化手法の新たな選択肢を手に入れることができるという点で，大変意義深いと考える．

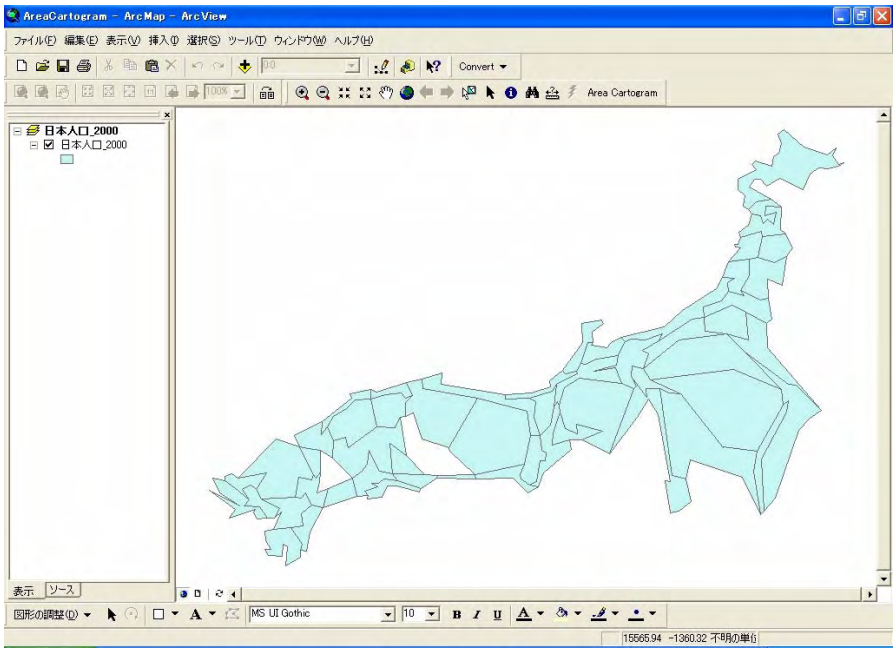
但し，連続エリアカルトグラム作成ツールでは，入力地域形状データに関する制約から日本の都道府県とアメリカ合衆国の州の二つのカルトグラム作成しか行えないようになっている．今後，ユーザーの自由度を高めた作成ツールの開発が不可欠である．

(a) 初期地域形状選択

(b) データ入力

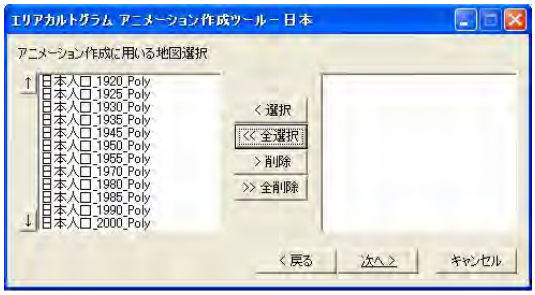


(c) 正則化項 初期値設定

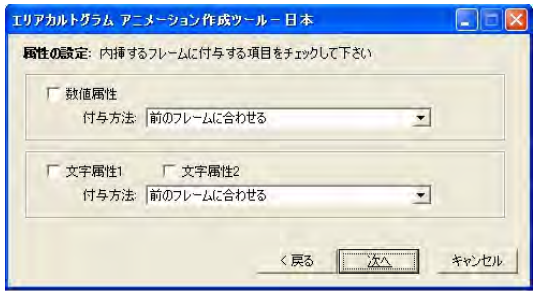


(d) 作成結果出力

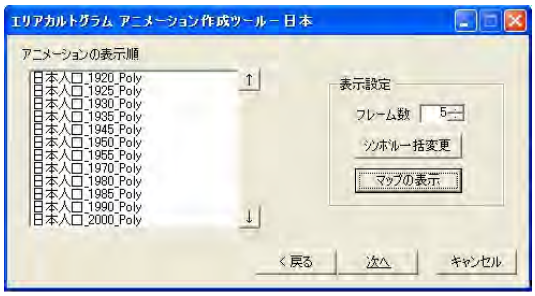
図 7.8: 連続エリアカルトグラム作成ツール カルトグラム作成画面



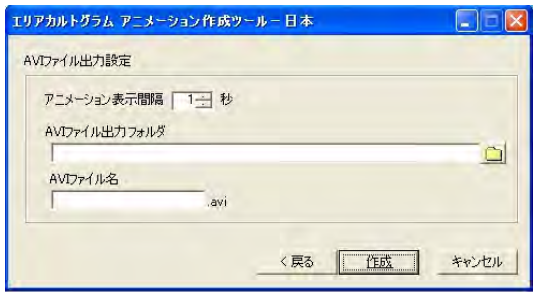
(a) アニメーション作成データ選択



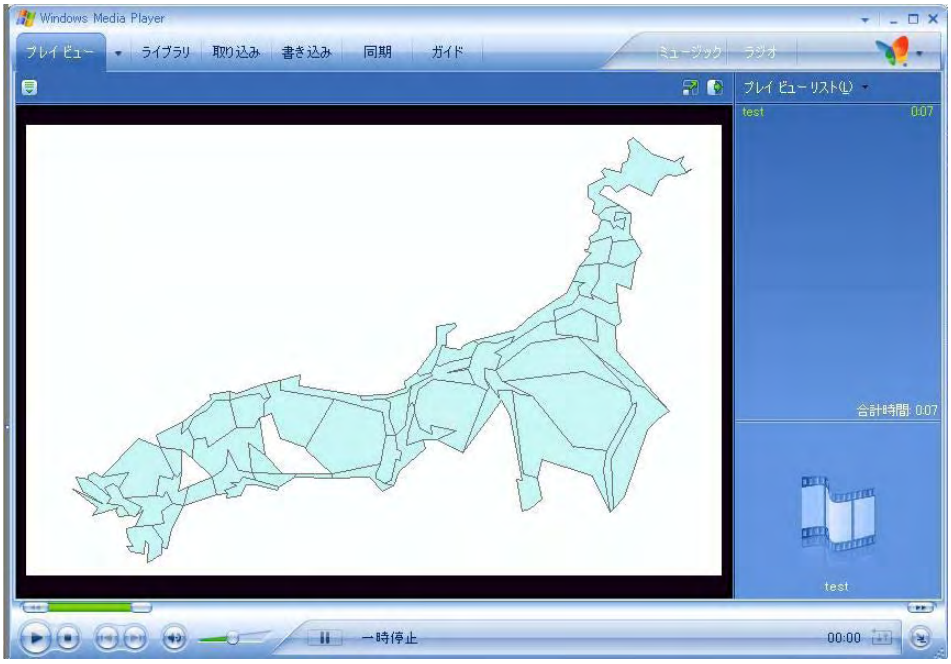
(b) 中間フレーム 属性設定



(c) フレーム数・シンボル設定



(d) アニメーションファイル保存先設定



(e) アニメーション作成結果出力

図 7.9: 連続エリアカルトグラム作成ツール アニメーション作成画面

第8章　結論

本論文では，有効な統計データの視覚化手法であるカルトグラムに着目し，カルトグラムを利用した視覚化を一般の人々が簡単に利用できる環境を整備することを最終目的とし，その目的に沿ったカルトグラム作成手法を提案し，GIS ソフトウェア上で動作する作成ツールを構築した．

まず，カルトグラムの一般利用を可能にするためには，視認性の高いカルトグラムを作成できる操作性の高い作成ツールを整備しなければならない．そのことを踏まえて,「過剰な変形の排除」および「作成手法の数学的明快さ」という基本的方針を挙げ，この方針を基に，ディスタンスカルトグラムおよびエリアカルトグラム双方の作成手法を構築した．

まず，カルトグラム作成問題の目的関数を数式で表現した上で，カルトグラム作成問題は解の一意性を持たない不良設定問題であり，正則化条件が必要であることを指摘した．その正則化条件の設定に当たり,「過剰な変形の排除」を排除するため，カルトグラム上の辺の方位角変化を抑制する条件を導入する手法を提案した．更に，カルトグラム作成問題を非線形最小二乗問題で記述した後，一次の線形化を行い線形最小二乗問題の繰り返し計算で解く手法を提案し,「数学的明快さ」を備えたカルトグラム作成手法を構築した．

提案したカルトグラム作成手法の特徴は以下の通りである．

1. 目的関数・正則化項が数学的に明快に記述されており変数の初期値設定の数学的意味が明らかであり，その設定に多くの試行錯誤を必要としない．

2. カルトグラム上の辺の方位角変化を抑制する正則化項の導入により，実地図からの変形が小さく比較対照が容易なカルトグラム作成が可能である．

3. 線形最小二乗問題の繰り返しで計算することができ，短時間でカルトグラム作成が可能である．

これらの特徴により，本論文による提案手法は操作性に優れており，一般的な利用が可能になっている．

更に，カルトグラムを更に魅力的な視覚化手法とするため，カルトグラム上に内挿する手法の開発を行った．

また，提案手法をユーザーインターフェースを追加し，GIS ソフトウェアの一つである ArcGIS 上で動作するカルトグラム作成ツールの開発を行い，簡単操作でカルトグラムを利用した統計データの視覚化が可能な状況を用意した．

但し，本論文では以下の事項に関して議論を行えていない．今後の研究課題について最後に記す．

1. 本論文では，視覚的に分かりやすいカルトグラム形状とは地理的形状に近い形状であるとの考えのもと，地点間の方位角変化の抑制を採用しカルトグラム作成手法を提案した．形状類

似度の指標としてヘルマート変換の決定係数を用いて分析を行ったが，その結果は人間の形状認知を反映するものとは言えず，形状の定量的評価には課題がある．そもそも人間が形状の類似度を何をもって判断しているのかについては検討が行えていないのが現状である．今後，アンケート等を利用して形状の類似度指標を構築し，分かりやすいカルトグラム形状の条件を定量的評価できるようにする必要がある．

2. ディスタンスカルトグラムに関しては，第 3.6 節に示したように，交通所要時間以外の入力データを用いたディスタンスカルトグラム作成とその適用可能性の検証が課題である．もちろん，ディスタンスカルトグラムは，例えば地点間の移動費用や交易量など地点間近接性を表す統計データに適用することが可能である．また，認知地図の解析など，統計データ以外の地点間データの分析にも利用ができると考えられる．これらデータへの適用を行い，ディスタンスカルトグラムによる視覚化の適用可能性とその限界を明確化する必要がある．

3. 連続エリアカルトグラムに関しては，第 4.4 節に示したように，既往の連続エリアカルトグラム作成手法との比較は十分に行えていない状況である．既往作成手法のプログラムが公開されていないこともあり，同一の入力データに基づくカルトグラム作成を行うことができないため，作図結果・データ表現精度・計算時間等の比較を行うことに限界がある．今後，同一の基準による手法の比較は課題である．

4. カルトグラム作成ツール開発では，第 7.3 節に記したように，現状では連続エリアカルトグラム作成ツールについては，入力データに対する制約から，日本とアメリカの 2 種類の地域形状データを基にした連続エリアカルトグラム作成が行えないようになっている．今後，ユーザーの自由度をより高めた作成ツールの開発が必要である．また，サークルエリアカルトグラムについても同様の作成ツールを構築・公開する必要があると考えている．

付 録A　球面ディスタンスカルトグラム

A.1　球面ディスタンスカルトグラム作成問題の解法

第 3 章と同様に，球面上の距離をデータに合わせて表示するカルトグラム作成手法を構築する．

ディスタンスカルトグラム上でデータを表現するリンクの集合を L，リンク ij 間に与えられた近接性指標を t_{ij}，リンク ij のディスタンスカルトグラム球面上距離を d_{ij} と表現すると，ディスタンスカルトグラム作成問題は，式 (A.1) と表される．

$$\min \sum_{ij \in L} \left(t_{ij} - d_{ij}\right)^2 \tag{A.1}$$

次に，正則化条件としてリンクの方位角拘束を導入し，その式を記述する．リンク ij の地点 $i,\ j$ における座北方位角 $\xi_{i,ij},\ \xi_{j,ij}$ の変化を抑制する目的関数を導入する．地点 i における地理的座標上の座北方位角を $\xi^G_{i,ij}$，カルトグラム上の座北方位角を $\xi^C_{i,ij}$，正則化項の重みを μ とし，方位角拘束式を式 (A.2) と表す．

$$\min \sum_{ij \in L} \left[\mu \left\{ \left(\xi^G_{i,ij} - \xi^C_{i,ij}\right)^2 + \left(\xi^G_{j,ij} - \xi^C_{j,ij}\right)^2 \right\}\right] \tag{A.2}$$

つまり，球面上のディスタンスカルトグラム作成問題は，

$$\min \sum_{ij \in L} \left[\left(t_{ij} - d_{ij}\right)^2 + \mu \left(\xi^G_{i,ij} - \xi^C_{i,ij}\right)^2 + \mu \left(\xi^G_{j,ij} - \xi^C_{j,ij}\right)^2\right] \tag{A.3}$$

と記述できる．

ここで，球の半径を r，地点 i の球面座標を $(r,\ \theta_i,\ \phi_i)$ と表し，式 (A.3) を地点座標の関数として記述する．

まず，地点 ij 間の球面上距離 d_{ij} は式 (A.4) で表される．

$$d_{ij} = r \cos^{-1} \left\{\sin \phi_i \sin \phi_j + \cos \phi_i \cos \phi_j \cos \left(\theta_j - \theta_i\right)\right\} \tag{A.4}$$

また，点 i における座北方位角 $\xi^C_{i,ij}$ を地点の座標を用いて表現する．$\xi^C_{i,ij}$ は，原点 O(0, 0, 0) と点 $i,\ j$ を通る平面と，原点 O と二つの極点 $\mathrm{P_N}(r,\ 0,\ \pi/2)$，$\mathrm{P_S}(r,\ 0,\ -\pi/2)$ を通る平面の点 i における角度である．この角度を求める際には，極点を結ぶ直線を回転軸として回転しても解には影響しない．つまり，$\theta_i = 0$ となるように回転したとしても一般性は失われない．

そこで，地点 i の座標を $(r,\ 0,\ \phi_i)$，地点 j の座標を $(r,\ \theta_j - \theta_i,\ \phi_i)$ とする．原点 O・地点 i・地点 j を通る平面の法線ベクトルは

$$\left(-\frac{\sin \phi_i}{\cos \phi_i},\ \frac{\cos(\theta_j - \theta_i) \sin \phi_i}{\sin(\theta_j - \theta_i) \cos \phi_i} - \frac{\sin \phi_j}{\sin(\theta_j - \theta_i) \cos \phi_j},\ 1\right) \tag{A.5}$$

と表される．$\xi^C_{i,ij}$ は，この平面が原点・2 極点を通る平面 (法線ベクトル $(0,\ 1,\ 0)$) と成す角なので，

$$\xi^C_{i,ij} = \cos^{-1}\left(\frac{\sin(\theta_j-\theta_i)\cos\phi_j}{\cos(\theta_j-\theta_i)\sin\phi_i\cos\phi_j-\sin\phi_j\cos\phi_i}\right) \tag{A.6}$$

と表すことができる．

式 (A.4)(A.6) を用いて式 (A.3) を書き直すと，球面ディスタンスカルトグラム作成問題は

$$\begin{aligned}\min\sum_{ij\in L}\Bigg[&\left\{t_{ij}-r\cos^{-1}\left(\sin\phi_i\sin\phi_j+\cos\phi_i\cos\phi_j\cos\left(\theta_j-\theta_i\right)\right)\right\}^2\\&+\mu\left\{\xi^G_{i,ij}-\cos^{-1}\left(\frac{\sin(\theta_j-\theta_i)\cos\phi_j}{\cos(\theta_j-\theta_i)\sin\phi_i\cos\phi_j-\sin\phi_j\cos\phi_i}\right)\right\}^2\\&+\mu\left\{\xi^G_{j,ij}-\cos^{-1}\left(\frac{\sin(\theta_j-\theta_i)\cos\phi_i}{\cos(\theta_j-\theta_i)\sin\phi_j\cos\phi_i-\sin\phi_i\cos\phi_j}\right)\right\}^2\Bigg]\end{aligned} \tag{A.7}$$

と表すことができ，未知変数が $r,\ \theta,\ \phi$ の非線形最小二乗問題で記述できる．

なお，正則化項簡略化のため，

$$\begin{aligned}\min\sum_{ij\in L}\Bigg[&\left\{t_{ij}-r\cos^{-1}\left(\sin\phi_i\sin\phi_j+\cos\phi_i\cos\phi_j\cos\left(\theta_j-\theta_i\right)\right)\right\}^2\\&+\mu\left\{\cos\xi^G_{i,ij}-\frac{\sin(\theta_j-\theta_i)\cos\phi_j}{\cos(\theta_j-\theta_i)\sin\phi_i\cos\phi_j-\sin\phi_j\cos\phi_i}\right\}^2\\&+\mu\left\{\cos\xi^G_{j,ij}-\frac{\sin(\theta_j-\theta_i)\cos\phi_i}{\cos(\theta_j-\theta_i)\sin\phi_j\cos\phi_i-\sin\phi_i\cos\phi_j}\right\}^2\Bigg]\end{aligned} \tag{A.8}$$

とする．

A.2　解法の線形化

第 3 章と同様に，式 (A.8) を線形化して解くこととする．まず，変数 $r,\ \theta_i,\ \phi_i$ に初期値 $r',\ \theta'_i,\ \phi'_i$ を与える．

$$r = r' + \Delta r \tag{A.9a}$$

$$\theta_i = \theta'_i + \Delta\theta_i \tag{A.9b}$$

$$\phi_i = \phi'_i + \Delta\phi_i \tag{A.9c}$$

これにより球面上距離 d_{ij} は

$$\begin{aligned}d_{ij} =&(r'+\Delta r)\cos^{-1}\big\{\sin(\phi'_i+\Delta\phi_i)\sin(\phi'_j+\Delta\phi_j)\\&+\cos(\phi'_i+\Delta\phi_i)\cos(\phi'_j+\Delta\phi_j\cos\big((\theta'_j+\Delta\theta_j)-(\theta'_i+\Delta\theta_i)\big)\big\}\end{aligned} \tag{A.10}$$

となり，$\theta' = \theta'_j - \theta'_i$, $A_{ij} = \sin\phi'_i \sin\phi'_j + \cos\phi'_i \cos\phi'_j \cos\theta'$ と表記すると

$$\begin{aligned} d_{ij} = &(r' + \Delta r)\cos^{-1} A_{ij} \\ &- \Delta\theta_i \frac{r'}{1 - A_{ij}^2} \cos\phi'_i \cos\phi'_j \sin\theta' \\ &+ \Delta\theta_j \frac{r'}{1 - A_{ij}^2} \cos\phi'_i \cos\phi'_j \sin\theta' \\ &- \Delta\phi_i \frac{r'}{1 - A_{ij}^2} \left(\sin\phi'_j \cos\phi'_i - \sin\phi'_i \cos\phi'_j \cos\theta'\right) \\ &- \Delta\phi_j \frac{r'}{1 - A_{ij}^2} \left(\sin\phi'_i \cos\phi'_j - \sin\phi'_j \cos\phi'_i \cos\theta'\right) \end{aligned} \tag{A.11}$$

となる．

従って，主問題の線形化式は

$$\begin{aligned} \sum_{ij\in L} \big[&\left(1 - A_{ij}^2\right) t_{ij} - r\left(1 - A_{ij}^2\right)\cos^{-1} A_{ij} \\ &+(\theta_i - \theta'_i) r' B_{ij} - (\theta_j - \theta'_j) r' B_{ij} \\ &+(\phi_i - \phi'_i) r' C_{i,ij} + (\phi_i - \phi'_i) r' C_{j,ij}\big]^2 \end{aligned} \tag{A.12}$$

但し $B_{ij} = \cos\phi'_i \cos\phi'_j \sin\theta'$, $C_{i,ij} = \sin\phi'_j \cos\phi'_i - \sin\phi'_i \cos\phi'_j \cos\theta'$

$C_{j,ij} = \sin\phi'_i \cos\phi'_j - \sin\phi'_j \cos\phi'_i \cos\theta'$

となる．

正則化項は，地理的座標による座北方位角からの変化を抑制するのではなく，近似座標による座北方位角からの変化を抑える式とする．

$$\begin{aligned} &+\mu\left(\frac{\sin\theta'\cos\phi'_j}{\cos\theta'\sin\phi'_i\cos\phi'_j - \sin\phi'_j\cos\phi'_i} - \frac{\sin(\theta_j - \theta_i)\cos\phi_j}{\cos(\theta_j - \theta_i)\sin\phi_i\cos\phi_j - \sin\phi_j\cos\phi_i}\right)^2 \\ &+\mu\left(\frac{\sin\theta'\cos\phi'_i}{\cos\theta'\sin\phi'_j\cos\phi'_i - \sin\phi'_i\cos\phi'_j} - \frac{\sin(\theta_j - \theta_i)\cos\phi_i}{\cos(\theta_j - \theta_i)\sin\phi_j\cos\phi_i - \sin\phi_i\cos\phi_j}\right)^2 \end{aligned} \tag{A.13}$$

ここで計算上の理由から，式から分母をなくすように式変形を行う．

$$\begin{aligned} &+\mu\left\{\sin\theta'\cos\phi'_j\left(\cos(\theta_j - \theta_i)\sin\phi_i\cos\phi_j - \sin\phi_j\cos\phi_i\right)\right. \\ &\quad\left. -\sin(\theta_j - \theta_i)\cos\phi_j\left(\cos\theta'\sin\phi'_i\cos\phi'_j - \sin\phi'_j\cos\phi'_i\right)\right\}^2 \\ &+\mu\left\{\sin\theta'\cos\phi'_i\left(\cos(\theta_j - \theta_i)\sin\phi_j\cos\phi_i - \sin\phi_i\cos\phi_j\right)\right. \\ &\quad\left. -\sin(\theta_j - \theta_i)\cos\phi_i\left(\cos\theta'\sin\phi'_j\cos\phi'_i - \sin\phi'_i\cos\phi'_j\right)\right\}^2 \end{aligned} \tag{A.14}$$

この式を線形化すると，

$$\begin{aligned} &+\mu\left\{-(\theta_i - \theta'_i)C_{j,ij}\cos\phi'_j + (\theta_j - \theta'_j)C_{j,ij}\cos\phi'_j\right. \\ &\quad\left. -(\phi_i - \phi'_i)\sin\theta'\cos\phi'_j A_{ij} + (\phi_j - \phi'_j)\sin\theta'\cos\phi'_i\right\}^2 \\ &+\mu\left\{(\theta_i - \theta'_i)C_{i,ij}\cos\phi'_i - (\theta_j - \theta'_j)C_{i,ij}\cos\phi'_i\right. \\ &\quad\left. -(\phi_i - \phi'_i)\sin\theta'\cos\phi'_j + (\phi_j - \phi'_j)\sin\theta'\cos\phi'_i A_{ij}\right\}^2 \end{aligned} \tag{A.15}$$

と表される．

すなわち，球面上ディスタンスカルトグラム作成問題は

$$
\begin{aligned}
\min \sum_{ij\in L} \Big\{ & \left(1-A_{ij}^2\right) t_{ij} - r\left(1-A_{ij}^2\right) \cos^{-1} A_{ij} \\
& +(\theta_i-\theta_i')r' B_{ij} - (\theta_j-\theta_j')r' B_{ij} \\
& +(\phi_i-\phi_i')r' C_{i,ij} + (\phi_i-\phi_i')r' C_{j,ij} \Big\}^2 \\
& + \mu \left\{ -(\theta_i-\theta_i')C_j \cos\phi_j' + (\theta_j-\theta_j')C_j \cos\phi_j' \right. \\
& \left. -(\phi_i-\phi_i')\sin\theta' \cos\phi_j' A_{ij} + (\phi_j-\phi_j')\sin\theta' \cos\phi_i' \right\}^2 \\
& + \mu \left\{ (\theta_i-\theta_i')C_i \cos\phi_i' - (\theta_j-\theta_j')C_i \cos\phi_i' \right. \\
& \left. -\ (\phi_i-\phi_i')\sin\theta' \cos\phi_j' + (\phi_j-\phi_j')\sin\theta' \cos\phi_i' A_{ij} \right\}^2
\end{aligned}
\tag{A.16}
$$

と表され，変数が r, θ, ϕ の線形最小二乗問題の繰り返し計算で解くことができる．

但し，r は変数の中で唯一長さの次元を持つ変数であり，θ, ϕ と次元が異なる．そのため，r 値の変化は他の変数の値に大きく影響し，解を不安定にする．そこで計算時には，まず表現データの総和がリンクの総距離となるように縮尺変換したのち，r を固定して計算するのがよい．

付 録B　作成プログラム ソースコード

付録 B では，各カルトグラム作成に用いたコンソールベースのプログラムのソースコードを記載する．

プログラミングの環境は，Microsoft 社 VisualStudio.Net 2003 を用いた．なお，線形代数の計算に関しては，Intel 社の “Math Kernel Library” の中から，数値計算用ライブラリ “BLAS” および “LAPACK” を使用した．

B.1　共通ファイル

CartogramClass.h

```
class coord2D{
public:
    double x;
    double y;
    coord2D(void){};
    ~coord2D(void){};
    coord2D(double in_x, double in_y){
        x = in_x;
        y = in_y;
    }
    coord2D operator- (const coord2D &in)const{return coord2D(x - in.x, y - in.y);}
    coord2D operator+ (const coord2D &in)const{return coord2D(x + in.x, y + in.y);}
    coord2D operator* (const double in)const{return coord2D(in * x, in * y);}
    coord2D operator* (const coord2D &in)const{return coord2D(x * in.x, y * in.y);}
    coord2D operator/ (const double in)const{return coord2D(x / in, y / in);}
    operator==(const coord2D &in)const{return x == in.x && y == in.y;}
    operator<(const coord2D &in)const{return x < in.x;}
};

class Edge_Info{
public:
    int V[2];
    int P[2];
    Edge_Info(void){};
    ~Edge_Info(void){};
    Edge_Info(const int in_V_0, const int in_V_1){
        V[0]=std::min(in_V_0, in_V_1);
        V[1]=std::max(in_V_0, in_V_1);
    }
    operator==(const Edge_Info &a)const{return (V[0] == a.V[0] && V[1] == a.V[1]);}
};

class V_Info{
public:
    bool use;
    int ID;
    V_Info(void){};
    ~V_Info(void){};
    V_Info(int in_ID):use(false){ID = in_ID;}
};
```

CartogramUtility.h

```
#define M_PI 3.1415926535
double sqrt_norm(const coord2D &in);
double norm(const coord2D &in);
double inner_product(const coord2D a, const coord2D b);
typedef std::vector< int > i_vector;
typedef std::vector< double > d_vector;
typedef std::vector< coord2D > co_vector;
typedef std::vector< i_vector > i_matrix;
typedef std::vector< Edge_Info > Edge_vector;
typedef std::map< int, int > i_map;
typedef std::vector< V_Info > V_vector;
```

CartogramUtility.cpp

```
#include "stdafx.h"
double sqrt_norm(const coord2D &in){return hypot(in.x, in.y);}
double norm(const coord2D &in){return(in.x * in.x + in.y * in.y);}
double inner_product(const coord2D a, const coord2D b){return(a.x * b.x + a.y * b.y);}
```

B.2　ディスタンスカルトグラム

ファイル構成

ディスタンスカルトグラム作成では，以下のプログラムを使用する．

1. 共通ファイル (CartogramClass.h; CartogramUtility.h; CartogramUtility.cpp)
2. stdafx.h
3. DC.h
4. DistanceCartogram.cpp
5. DC_Algorithm.cpp
6. DC_IO.cpp

地点の地理座標およびリンク・距離指標のカンマ区切りファイルを入力し，地点のカルトグラム上座標を始め，距離指標データの表現精度等を出力する (図 B.1).

NodeID	N	E
1	55.5682	9.7448
3	54.3090	13.0792
4	53.5453	10.0079
5	52.3788	4.8977
6	52.3678	9.7288
8	52.5229	13.4170
9	52.3431	14.5538
10	51.4329	6.7630
11	51.5151	7.4701
12	51.3188	9.5086
13	50.6455	5.5735
14	50.7682	6.0863
15	50.9367	6.9593
16	50.5492	9.6675
17	51.3452	12.3764
18	51.0517	13.7388
19	50.0036	8.2636
20	50.1040	8.6883

(a) 地理的地点座標 入力

地点1	地点2	距離指標
1	4	252
3	8	180
4	6	75
4	8	264
4	11	161
5	10	138
6	8	265
6	11	93
6	12	90
8	9	74
8	18	173
8	22	440
10	11	38
10	15	38
11	15	72
12	16	49
13	14	48
14	15	41
15	19	113

(b) リンク・距離指標 入力

NodeID	x	y
1	47.6451	491.1934
3	307.5096	422.2012
4	67.0619	239.9425
5	-149.3604	197.8918
6	71.8494	161.0383
8	328.0870	243.3812
9	399.7252	224.8349
10	-42.3043	110.8115
11	-5.2536	101.4347
12	59.2880	69.5448
13	-161.7031	67.7704
14	-116.8086	84.7551
15	-77.6061	96.7621
16	52.3790	18.6600
17	329.2573	146.3959
18	392.9871	82.9594
19	-31.0566	-4.9208
20	0.0000	0.0000

(c) カルトグラム座標 出力

図 B.1: ディスタンスカルトグラム 入出力ファイル (例)

stdafx.h

```
#include <iostream>
#include <tchar.h>
#using <mscorlib.dll>
#include <vector>
#include <map>
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <time.h>
#include "mkl.h"
#include "CartogramClass.h"
#include "CartogramUtility.h"
#include "DC.h"
using namespace System;
using namespace std;
```

DC.h

```
class DC_IO{
public:
    void In_NECoord(char *in_NEcoord, int ORIGIN, co_vector &NE_coord, V_vector &Node_List, i_map &Node);
    void In_LinkTime(char *in_link, i_map &Node, Edge_vector &link, d_vector &t, co_vector &NE,
         V_vector &Node_List);
    void Out_ArcView_Link(char *out_link, co_vector &coord, Edge_vector &link);
    void Out_Coordinate(char *out_config, V_vector &Node_List, co_vector &coord);
    void Out_Precision(char *out_precision, V_vector &Node_List, Edge_vector &link, d_vector &t, d_vector &d,
        double *rad_0, double *rad);
    void Set_Rad(Edge_vector &link, double *rad, double *rad_0, co_vector &NE);
};

class DC_Algorithm{
public:
    int DC_Algorithm_main_net(int M, int N, const double EPS_ANGLE, double *rad, d_vector &t, co_vector &coord,
        Edge_vector &link);
    void link_distance(d_vector &d, co_vector &coord, Edge_vector &link);
private:
    void axis_distance(int m, double *t_t, double *t_x, double *t_y, double *si, double *co);
    void renew_angle(int m, double *old_r, double *r, double *t_x_hut, double *t_y_hut, double *s, double *c);
    bool convergence_check(int m, const double EPS, double *r, double *old_r);
};
```

DistanceCartogram.cpp

```
#include "stdafx.h"

//--------------ディスタンスカルトグラム作成メインプログラム--------------
void main()
{
    const int ORIGIN=1;         //基準点のノード番号
    const int PRECISION = -2;    //収束計算の判定基準 (10^(PRECISION))

    int count, N_Node=-1, N_Link;
    double *rad_0, *rad, EPS_ANGLE=pow(10.0, double(PRECISION));
    V_vector Node_List;
    i_map Node;
    d_vector d, t;
    co_vector NE_coord, coord_result;
    Edge_vector link;
    time_t start, last;
    clock_t clock(void);
    DC_Algorithm DC;
    DC_IO IO;

    //ファイル入出力関係
    FILE *outfp;
```

```
    char in_NEcoord[]="Germany_Cities";    //地点座標
    char in_data[]="所要時間データ Germany2010";    //リンク・所要時間
    char out_config[100], out_precision[100], out_link[100];

    start=clock();    //計算時間測定開始

    //ファイル名を与えるための文字列操作
    sprintf(out_config, "Output data/%s_(%2d)_config.csv", in_data, PRECISION);
    sprintf(out_precision, "Output data/%s_(%2d)_precision.csv", in_data, PRECISION);
    sprintf(out_link, "Output data/%s_(%2d)_link.csv", in_data, PRECISION);

    IO.In_NECoord(in_NEcoord, ORIGIN, NE_coord, Node_List, Node);    //座標データ入力
    IO.In_LinkTime(in_data, Node, link, t, NE_coord, Node_List);

    for (V_vector::iterator itr=Node_List.begin(); itr<Node_List.end(); itr++)
        if ((*itr).use) ++N_Node;    //基準点以外のノード数
    N_Link=int(link.size());

    rad = new double[N_Link], rad_0 = new double[N_Link];
    IO.Set_Rad(link, rad, rad_0, NE_coord);

    //Distance Cartogram 作成計算
    count = DC.DC_Algorithm_main_net(N_Link, N_Node, EPS_ANGLE, rad, t, coord_result, link);
    DC.link_distance(d, coord_result, link);

    //結果の出力
    IO.Out_Coordinate(out_config, Node_List, coord_result);
    IO.Out_Precision(out_precision, Node_List, link, t, d, rad_0, rad);
    IO.Out_ArcView_Link(out_link, coord_result, link);

    delete[] rad_0, rad;

    //ファイルの出力 計算時間計測
    last=clock();
    printf("iteration = %d, clock [sec] = %.3f \n",count, (float)(last-start)/(float)CLOCKS_PER_SEC);
    if((outfp= fopen("Output data/計算時間計測結果 DC.csv","a")) != NULL){
        fprintf(outfp,"%s,%2d,%d,%.3f\n",in_data,PRECISION,count,
            (float)(last-start)/(float)CLOCKS_PER_SEC);
        fclose(outfp);
    }
    else cout<<"ERROR----計算時間ファイル出力失敗"<<endl;
}
```

DC_Algorithm.cpp

```
#include "stdafx.h"

//--------------ディスタンスカルトグラム作成--------------
int DC_Algorithm::DC_Algorithm_main_net(int M, int N, const double EPS_ANGLE,
        double *rad, d_vector &t, co_vector &coord, Edge_vector &link)
{
    int i, count=0, lwork, info, *ipiv;
    double *X, *itXX, *titXXtX, *work, *beta_x, *beta_y, *t_x, *t_y;
    double *oldrad, *s, *c, *t_calc;
    char Upper='U';

    t_calc = new double[M];
    copy(t.begin(), t.end(), t_calc);

    X = new double[M*N];
    fill_n(X, M*N, 0.0);

    for (i=0; i<M; i++){
        if (link[i].V[0]!=0) X[i+(link[i].V[0]-1)*M] = -1.0;
        if (link[i].V[1]!=0) X[i+(link[i].V[1]-1)*M] = 1.0;
    }

    itXX=new double[N*N];
    cblas_dgemm(CblasColMajor, CblasTrans, CblasNoTrans, N, N, M, 1.0, X, M, X, M, 0.0, itXX, N);

    ipiv=new int[N];
    lwork=N*64;
```

```
    work=new double[lwork];
    dsytrf(&Upper, &N, itXX, &N, ipiv, work, &lwork, &info);
    dsytri(&Upper, &N, itXX, &N, ipiv, work, &info);
    delete[] ipiv, work;

    titXXtX=new double[M*N];
    cblas_dsymm(CblasColMajor, CblasRight, CblasUpper, M, N, 1.0, itXX, N, X, M, 0.0, titXXtX, M);
    delete[] itXX;

    s = new double[M];
    c = new double[M];
    vdSinCos(M, rad, s, c);

    oldrad = new double[M];
    beta_x = new double[N];
    beta_y = new double[N];
    t_x = new double[M];
    t_y = new double[M];

    //収束判定
    while(convergence_check(M, EPS_ANGLE, rad, oldrad)){
        axis_distance(M, t_calc, t_x, t_y, s, c);

        cblas_dgemv(CblasColMajor, CblasTrans, M, N, 1.0, titXXtX, M, t_x, 1, 0.0, beta_x, 1);
        cblas_dgemv(CblasColMajor, CblasTrans, M, N, 1.0, titXXtX, M, t_y, 1, 0.0, beta_y, 1);

        cblas_dgemv(CblasColMajor, CblasNoTrans, M, N, 1.0, X, M, beta_x, 1, 0.0, t_x, 1);
        cblas_dgemv(CblasColMajor, CblasNoTrans, M, N, 1.0, X, M, beta_y, 1, 0.0, t_y, 1);

        renew_angle(M, oldrad, rad, t_x, t_y, s, c);
        ++ count;
    }
    delete[] X, titXXtX, oldrad, s, c, t_x, t_y, t_calc;

    //ノードに基準点を加え，Distance Cartogram 座標を出力
    coord.push_back(coord2D(0.0, 0.0));
    for (i=0; i<N; i++) coord.push_back(coord2D(beta_x[i], beta_y[i]));

    delete[] beta_x, beta_y;

    return count;
}

//--------------sin,cos を用いて x,y 方向の時間距離を計算--------------
void DC_Algorithm::axis_distance(int m, double *t_t, double *t_x, double *t_y, double *si, double *co)
{
    for (int i=0; i<m; i++){
        t_x[i] = t_t[i] * si[i];
        t_y[i] = t_t[i] * co[i];
    }
}

//--------------方位角の更新--------------
void DC_Algorithm::renew_angle(int m, double *old_r, double *r,
        double *t_x_hut, double *t_y_hut, double *si, double *co)
{
    copy(r, r+m, old_r);
    vdAtan2(m, t_x_hut, t_y_hut, r);
    vdSinCos(m, r, si, co);
}

//--------------角度変化による収束判定--------------
bool DC_Algorithm::convergence_check(int m, const double EPS, double *r, double *old_r)
{
    for (int i=0; i<m; i++) if(fabs(r[i]-old_r[i])>=EPS) return true;
    return false;
}

//--------------リンク長計算--------------
void DC_Algorithm::link_distance(d_vector &d, co_vector &coord, Edge_vector &link)
{
    for (Edge_vector::iterator itr=link.begin(); itr < link.end(); itr++)
        d.push_back(sqrt_norm(coord[(*itr).V[1]] - coord[(*itr).V[0]]));
}
```

DC_IO.cpp

```
#include "stdafx.h"

//--------------ファイル入力 ノード地理座標データ--------------
void DC_IO::In_NECoord(char *in_NEcoord, int ORIGIN, co_vector &NE_coord, V_vector &Node_List, i_map &Node)
{
    int tmp_ID, N_node = 1;
    char tmp_char[100], in_file[100];
    FILE *infp;
    coord2D tmp_coord;

    sprintf(in_file, "Input data/%s.csv", in_NEcoord);
    if ((infp=fopen(in_file,"r")) == NULL){
        cout<<"座標ファイルが開けません"<<endl;
        exit(1);
    }

    //原点の情報
    NE_coord.push_back(coord2D(0.0, 0.0));    //仮の値を入力
    Node.insert(pair<int, int>(ORIGIN, 0));
    Node_List.push_back(V_Info(ORIGIN));

    fgets(tmp_char, 100, infp);
    while(!feof(infp) && !ferror(infp)){
        fscanf(infp,"%d,%lf,%lf\n", &tmp_ID, &tmp_coord.y, &tmp_coord.x);
        if (tmp_ID==ORIGIN)    NE_coord[0] = tmp_coord;
        else{
            NE_coord.push_back(tmp_coord);
            Node.insert(pair<int, int>(tmp_ID, N_node));
            Node_List.push_back(V_Info(tmp_ID));
            ++ N_node;
        }
    }
    fclose(infp);
}

//--------------ファイル入力 表現近接性指標データ--------------
void DC_IO::In_LinkTime(char *in_link, i_map &Node, Edge_vector &link, d_vector &t, co_vector &NE,
    V_vector &Node_List)
{
    int i, tmp[2], N_Link=0;
    double tmp_t;
    char tmp_char[100], in_file[100];
    FILE *infp;
    i_map::iterator itr;
    coord2D d_coord;

    sprintf(in_file, "Input data/%s.csv", in_link);
    //ファイル入力 リンクリスト
    if ((infp=fopen(in_file,"r")) == NULL){
        cout<<"リンクデータファイルが開けません"<<endl;
        exit(1);
    }

    fgets(tmp_char, 100, infp);
    while (!feof(infp) && !ferror(infp)){
        fscanf(infp,"%d,%d,%lf\n",&tmp[0], &tmp[1], &tmp_t);
        for (i=0; i<2; i++){
            itr = Node.find(tmp[i]);
            if (itr == Node.end()){
                cout<<"ノード"<<tmp[i]<<"データなし"<<endl;
                exit(1);
            }
            tmp[i] = itr->second;
            Node_List[itr->second].use = true;
        }
        link.push_back(Edge_Info(tmp[0], tmp[1]));
        t.push_back(tmp_t);
    }
    fclose(infp);
}

//--------------方位角初期設定--------------
```

```
void DC_IO::Set_Rad(Edge_vector &link, double *rad, double *rad_0, co_vector &NE)
{
    int N_Link = int(link.size());
    coord2D d_coord;

    for (int i=0; i<N_Link; i++){
        d_coord.y = NE[link[i].V[1]].y-NE[link[i].V[0]].y;
        d_coord.x = (NE[link[i].V[1]].x-NE[link[i].V[0]].x)*cos(NE[link[i].V[1]].y/180 * M_PI);
        rad[i] = atan2(d_coord.x, d_coord.y);
        rad_0[i]=rad[i];
    }
}

//--------------ファイル出力 カルトグラム座標--------------
void DC_IO::Out_Coordinate(char *out_config, V_vector &Node_List, co_vector &coord)
{
    FILE *outfp;
    unsigned int n=Node_List.size();

    if((outfp = fopen(out_config,"w")) != NULL){
        fprintf(outfp,"ノード番号,x,y\n");
        for (unsigned int i=0; i<n; i++)
            if (Node_List[i].use)
                fprintf(outfp,"%d,%f,%f\n",Node_List[i].ID, coord[i].x, coord[i].y);
        fclose(outfp);
    }
    else cout<<"座標ファイル出力失敗"<<endl;
}

//--------------ファイル出力 距離表現精度検証--------------
void DC_IO::Out_Precision(char *out_precision, V_vector &Node_List, Edge_vector &link, d_vector &t,
    d_vector &d, double *rad_0, double *rad)
{
    int N_link=int(link.size());
    FILE *outfp;

    if((outfp = fopen(out_precision,"w")) != NULL){
        fprintf(outfp,"リンク番号,, ノード番号, 近接性指標,,, 方位角\n");
        fprintf(outfp,", 起点, 終点, 実際, カルトグラム上, カルトグラム上/実際, 収束計算前, 収束時\n");
        for (int i=0; i<N_link; i++){
            fprintf(outfp,"%d,%d,%d,%.2f,%f,%.10f,%f,%f\n",    i, Node_List[link[i].V[0]].ID,
                Node_List[link[i].V[1]].ID, t[i], d[i], d[i]/t[i], rad_0[i], rad[i]);
        }
        fclose(outfp);
    }
    else cout<<"表現精度確認ファイル出力失敗"<<endl;
}

//--------------ファイル出力 リンクデータ--------------
void DC_IO::Out_ArcView_Link(char *out_link, co_vector &coord, Edge_vector &link)
{
    int N_Link=int(link.size());
    FILE *outfp;

    if((outfp= fopen(out_link,"w")) != NULL){
        fprintf(outfp,"GsGroup,GsOrder,x,y\n");
        for (int i=0; i<N_Link; i++)
            for (int j=0; j<2; j++)
                fprintf(outfp,"%d,%d,%f,%f\n",i,j,coord[link[i].V[j]].x,coord[link[i].V[j]].y);
        fclose(outfp);
    }
    else cout<<"ArcView 用リンクデータファイル出力失敗"<<endl;
}
```

B.3　連続エリアカルトグラム

ファイル構成

連続エリアカルトグラム作成では，以下のプログラムを使用する．

1. 共通ファイル (CartogramClass.h; CartogramUtility.h; CartogramUtility.cpp)
2. stdafx.h
3. ContinuousAreaCartogramClass.h
4. ContinuousAreaCartogram.h
5. ContinuousAreaCartogram.cpp
6. ContinuousAreaCartogram_IO.cpp
7. ContinuousAreaCartogramUtility.cpp
8. TIN_Area.cpp
9. TIN_Make.cpp

連続エリアカルトグラム作成プログラムでの地域形状入力では，頂点 ID と座標を記録したファイルと，地域を構成する頂点 ID のリストを記録したファイルを使用する．表現する統計データを入力して計算を実行し，各地域の頂点のカルトグラム上座標や統計データの表現精度を出力する．(図 B.2)

VertexID	X	Y
0	129.57	33.37604
1	129.627	32.97795
2	129.6658	33.10157
3	129.7389	32.56945
4	129.7908	33.34042
5	129.8065	33.48567
6	129.8661	32.75057
7	129.9278	33.5501
8	129.9445	33.1647
9	130.0073	32.84891
10	130.0402	33.47021
11	130.0874	32.86388
12	130.1079	31.40903
13	130.1227	33.12325
14	130.1269	32.68929
15	130.1699	32.5908
16	130.1761	32.10623
17	130.1792	32.79451
18	130.1889	31.74674

(a) 頂点座標リスト入力

PolygonID	Order	Vertice_ID
1	0	356
1	1	355
1	2	357
1	3	359
1	4	358
1	5	354
1	6	352
1	7	351
1	8	350
1	9	343
1	10	330
1	11	318
1	12	307
1	13	300
1	14	333
1	15	314
1	16	296
1	17	290
1	18	292

(b) 地域頂点リスト入力

PolygonID	Populagion
1	5683062
2	1475728
3	1416180
4	2365320
5	1189279
6	1244147
7	2126935
8	2985676
9	2004817
10	2024852
11	6938006
12	5926285
13	12064101
14	8489974
15	2475733
16	1120851
17	1180977
18	828944
19	888172

(c) 表現統計データ入力

PolygonID	Order	VertexID	X	Y
1	0	356	13014.2	8352.1
1	1	355	12819.6	7937.8
1	2	357	13127.7	7504.1
1	3	359	13399.8	7593.9
1	4	358	13145.7	7414.8
1	5	354	12584.2	7240.6
1	6	352	12079.7	7271.2
1	7	351	11588.4	6952.3
1	8	350	11371.2	6439.8
1	9	343	10092.6	6995.6
1	10	330	9594.58	6737
1	11	318	9365.24	6952.5
1	12	307	9177.39	6965.3
1	13	300	9005.37	6714.8
1	14	333	9864.87	6404.8
1	15	314	9227.04	6392
1	16	296	8834.44	6050.2
1	17	290	8662.4	6149
1	18	292	8822.04	6474.7

(d) カルトグラム上座標出力

図 B.2: 連続エリアカルトグラム 入出力ファイル (例)

stdafx.h

```
#include <iostream>
#include <tchar.h>
#include <cmath>
#include <cstring>
#include <cstdlib>
#include <vector>
#include <list>
#include <map>
#include <algorithm>
#include "time.h"
```

```
#include "mkl.h"
#include "CartogramClass.h"
#include "ContinuousAreaCartogramClass.h"
#include "CartogramUtility.h"
#include "ContinuousAreaCartogram.h"
using namespace System;
using namespace std;
```

ContinuousAreaCartogramClass.h

```
class Triangle_Info{
public:
    int V[3];
    int E[3];
    int Poly_ID;
    double Data;
    double Area;
    coord2D Centroid;
    Triangle_Info(void){};
    ~Triangle_Info(void){};
    Triangle_Info(int *in_V){std::copy(in_V, in_V + 3, V);}
    Triangle_Info(int in_V_0, int in_V_1, int in_V_2){
        V[0] = in_V_0;
        V[1] = in_V_1;
        V[2] = in_V_2;
    }
};

class Polygon_Info{
public:
    int ID;
    int N_Vert;
    std::vector<int> V_ID;
    std::vector<int> C_Poly;
    double Data;
    double Area;
    double Density;
    coord2D Centroid;
    Polygon_Info(void){};
    ~Polygon_Info(void){};
    Polygon_Info(int in_ID):N_Vert(0), Data(0.0){ID = in_ID;}
};

class ID_Value{
public:
    int ID;
    double Value;
    ID_Value(void){};
    ~ID_Value(void){};
    ID_Value(int in_ID, double in_Value){
        ID = in_ID;
        Value = in_Value;
    }
    bool operator<(const ID_Value &a)const{return Value < a.Value;}
    bool operator>(const ID_Value &a)const{return Value > a.Value;}
    operator==(const ID_Value &a)const{return Value == a.Value;}
};

typedef std::vector< Triangle_Info > Tri_vector;
typedef std::vector< Polygon_Info > Poly_vector;
```

ContinuousAreaCartogram.h

```
class TIN_MAKE{
public:
    void TIN_Make_in_Poly(Tri_vector &Tri, co_vector &coord, Poly_vector &Poly);
private:
    void Adjust_TIN_Flip(const int N_Tri_first, Tri_vector &Tri, co_vector &coord);
    void Circumcircle(int *Point_ID, coord2D *center, double *radius_sq, co_vector &coord);
```

```
    void Distribute_Data2TIN(Tri_vector &Tri, co_vector &coord, Poly_vector &Poly);
    void Laplacian_Smoothing(const int N_Tri_First, Tri_vector &Tri, co_vector &coord, i_vector &Added_V_ID);
    void Out_Tri_Info(int *Tri_V, Tri_vector &Tri);
    void Out_Tri_Info(const int Tri_V_0, const int Tri_V_1, const int Tri_V_2, Tri_vector &Tri);
    void Steiner(const int N_Tri_first, co_vector &coord, Tri_vector &Tri, i_vector &Added_V_ID);
    void Tin_Make_wo_Centroid(Tri_vector &Tri, i_vector &tmp_Poly_V, co_vector &coord);
    bool Flip(int *tmp_Poly_V, co_vector &coord, Tri_vector &Tri);
    bool Overturn_Triangle(const int Target, coord2D coord_COM, Triangle_Info Tri, co_vector &coord);
    bool Point_in_Circle(const int Point_ID, double radius_sq, coord2D circle, co_vector &coord);
    bool Tri_Share_Edge(const int v_ID_1, const int v_ID_2, const int Tri_ID, const int min_Tri_ID,
        const int max_Tri_ID, Tri_vector &Tri, int *found_tri_ID, int *not_shared_v_ID);
};

class TIN_AREA{
public:
    double TIN_Area(const double Error_desired, double *myu, co_vector &coord, Tri_vector &Tri,
        Poly_vector &Poly);
private:
    void AreaCarto_Expl_Matrix(double *X, double *Y, co_vector &Edge_Vec, Edge_vector &Edge,
        const int X_col_half, const double myu, Tri_vector &Tri);
    void calc_Edge_vector(co_vector &coord, co_vector &Edge_vec, Edge_vector &Edge);
    void Expand_Area(double *Total_A, co_vector &Edge_vec, co_vector &coord, Tri_vector &Tri);
    void Renew_coord(int N, co_vector &new_coord, double *Y);
    void Set_New_Edge_ID(Edge_vector &Edge, Triangle_Info *tmp_Tri, const int E_Order,
        const int V_Order0, const int V_Order1);
    void Triangle_Area(co_vector &Edge_vec, Tri_vector &Tri);
    bool TIN_Topology(double *myu, co_vector &new_coord, co_vector &old_coord, co_vector &Edge_vec,
        Edge_vector &Edge, Tri_vector &Tri);
    int Set_Edge_ID(Edge_vector &Edge, Tri_vector &Tri);
    double Error_Ratio(const double Total_Data, Tri_vector &Tri);
    double Squared_Errors(Tri_vector &Tri);
    double TIN_Area_main(int *iter, const double Error_desired, double *myu, co_vector &calc, Tri_vector &Tri);
};

class ContinuousAreaCartogram_IO{
public:
    bool Input_Data(char *in_Data, char *in_Area, co_vector &coord, Poly_vector &Poly, V_vector &V);
    void Output_Poly_V_ID(char *in_Area, co_vector &coord, Poly_vector &Poly, V_vector &V);
    void Output_TIN(char *in_Area, co_vector &coord, Tri_vector &Tri, Poly_vector &Poly);
private:
    bool Input_Area(char *in_Area, i_map &Poly_ID, Poly_vector &Poly);
    bool Input_Vert_coord(char *in_Data, i_map &in_Vert, co_vector &coord, V_vector &V);
    bool Input_Polygon_Vert(char *in_Data, i_map &in_Poly, Poly_vector &Poly, i_map &in_Vert, V_vector &V);
};

class ContinuousAreaCartogramUtility{
public:
    void calc_angle(const int i, const int N_Vert, co_vector &Edge, d_vector &e_d, d_vector &c_v,
        d_vector &s_v);
    void calc_edge_length(const int i, i_vector &Poly_V, const int N_Vert, co_vector &coord, co_vector &Edge,
        d_vector &e_d);
    void calc_sincos_ijk(const int i, const int j, const int k, co_vector &coord, double *sin, double *cos);
    void Search_Vert(const int V1, const int V2, int *found_V1, int *found_V2, Tri_vector &Tri);
    void Set_Edge_ID(Poly_vector &Poly, Edge_vector &Edge);
    bool intersect(const int S0, const int E0, const int S1, const int E1, co_vector &coord);
    bool Point_In_ThreePoints(const coord2D Point, const coord2D P0, const coord2D P1, const coord2D P2);
    int ID_check(const int ID, const int N_Vert);
    int Search_Same_Edge(Edge_Info tmp_ID, Edge_vector &Edge);
    double calc_edge_ij_length(const int i, const int j, co_vector &coord);
    double calc_tri_area(co_vector &coord, const Triangle_Info Tri);
    double tri_area_w_sign(const int p1, const int p2, const int p3, co_vector &coord);
    double tri_area_w_sign(const coord2D p1, const coord2D p2, const coord2D p3);
};
bool large_ID_Value(const ID_Value* &a, const ID_Value* &b);
```

ContinuousAreaCartogram.cpp

```
#include "stdafx.h"
#using <mscorlib.dll>

//--------------連続エリアカルトグラム作成メインプログラム--------------
void _tmain()
```

```
{
    char in_Data[]="Data and Results/USA/USA";
    char in_Area[]="Data and Results/USA/USA_Pop2000";

    const double Error_desired = 1.0;
    double myu = 1.0;
    double Error;

    V_vector V;
    co_vector coord;
    Tri_vector Tri;
    Poly_vector Poly;
    Edge_vector Edge;

    time_t start, last, current;
    struct tm  *local;
    clock_t clock(void);

    TIN_IO * IO = new TIN_IO;
    TIN_AREA * Area = new TIN_AREA;
    TIN_MAKE * Make = new TIN_MAKE;
    ContinuousAreaCartogramUtility Util;
    Poly_vector::iterator itr_P;

    start = clock();

    if (!IO->Input_Data(in_Data, in_Area, coord, Poly, V)) exit(1);  //データ入力
    Util.Set_Edge_ID(Poly, Edge);  //Edge リストの設定

//接している Polygon のリスト作成
    for (Edge_vector::iterator itr_E = Edge.begin(); itr_E < Edge.end(); itr_E ++){
        if ((*itr_E).P[1]!=-1){
            Poly[(*itr_E).P[0]].C_Poly.push_back((*itr_E).P[1]);
            Poly[(*itr_E).P[1]].C_Poly.push_back((*itr_E).P[0]);
        }
    }
    for (itr_P = Poly.begin(); itr_P < Poly.end(); itr_P ++)
        unique((*itr_P).C_Poly.begin(), (*itr_P).C_Poly.end());

//エリアカルトグラム作成計算
    do {
        Make -> TIN_Make_in_Poly(Tri, coord, Poly);
        Error = Area->TIN_Area(Error_desired, &myu, coord, Tri, Poly);
    } while (Error > Error_desired);

    itr_i;
    coord2D min_coord(0.0, 0.0), max_coord(0.0, 0.0);
    for (itr_P = Poly.begin(); itr_P < Poly.end(); itr_P ++){
        for (i_vector::iterator itr_i = (*itr_P).V_ID.begin(); itr_i < (*itr_P).V_ID.end(); itr_i ++){
            if (coord[*itr_i].x > max_coord.x) max_coord.x = coord[*itr_i].x;
            else if (coord[*itr_i].x < min_coord.x) min_coord.x = coord[*itr_i].x;
            if (coord[*itr_i].y > max_coord.y) max_coord.y = coord[*itr_i].y;
            else if (coord[*itr_i].y < min_coord.y) min_coord.y = coord[*itr_i].y;
         }
    }

    min_coord = (min_coord + max_coord)/2;

    for (co_vector::iterator itr_co = coord.begin(); itr_co < coord.end(); itr_co ++)
        *itr_co = *itr_co - min_coord;
    IO->Output_Poly_V_ID(in_Area, coord, Poly, V);

    last = clock();
    cout<<"clock [sec] = "<<(float)(last-start)/(float)CLOCKS_PER_SEC<<endl;
    FILE *outfp;
    outfp= fopen("計算時間計測結果 Area Cartogram.csv","a");
    time(&current);                         //  時刻を取得
    local = localtime(&current);            //  地方時の構造体に変換
    fprintf(outfp,"%s,%.3f,%.3f,%04d/%02d/%02d,%02d:%02d:%02d\n", in_Area, Error,
        (float)(last-start)/(float)CLOCKS_PER_SEC, local->tm_year+1900, local->tm_mon+1, local->tm_mday,
        local->tm_hour, local->tm_min, local->tm_sec);
    fclose(outfp);
}
```

ContinuousAreaCartogram_IO.cpp

```
#include "stdafx.h"

//--------------ファイル入力--------------
bool TIN_IO::Input_Data(char *in_Data, char *in_Area, co_vector &coord, Poly_vector &Poly, V_vector &V)
{
    i_map in_Vert, Poly_ID;

    if ((!Input_Vert_coord(in_Data, in_Vert, coord, V))
        ||(!Input_Polygon_Vert(in_Data, Poly_ID, Poly, in_Vert, V))
        ||(!Input_Area(in_Area, Poly_ID, Poly))) return false;
    return true;
}

//--------------ファイル入力 頂点座標--------------
bool TIN_IO::Input_Vert_coord(char *in_Data, i_map &in_Vert, co_vector &coord, V_vector &V)
{
    int tmp_V_ID, V_ID = 0;
    coord2D tmp_coord;
    char input[100], tmp_char[80];
    FILE *infp;

    sprintf(input, "%s_Vertices.csv", in_Data);
    if ((infp=fopen(input, "r")) == NULL){
        cout<<"-----頂点座標ファイルが開けません-----"<<endl;
        return false;
    }

    fgets(tmp_char, 80, infp);
    while(!feof(infp) && !ferror(infp)){
        fscanf(infp,"%d,%lf,%lf\n", &tmp_V_ID, &tmp_coord.x, &tmp_coord.y);
        coord.push_back(tmp_coord);
        in_Vert.insert(pair<int, int>(tmp_V_ID, V_ID));
        V.push_back(V_Info(tmp_V_ID));
        ++ V_ID;
    }

    fclose(infp);
    return true;
}

//--------------ファイル入力 ポリゴン頂点データ--------------
bool TIN_IO::Input_Polygon_Vert(char *in_Data, i_map &in_Poly, Poly_vector &Poly, i_map &in_Vert,  V_vector &V)
{
    int tmp_Poly_ID, Order, Vert_ID, P_ID = 0;
    char input[100], tmp_char[80];
    i_map::iterator p;
    FILE *infp;

    sprintf(input, "%s_Polygon.csv", in_Data);
    if ((infp = fopen(input, "r")) == NULL){
        cout<<"-----Polygon 頂点情報ファイルが開けません-----"<<endl;
        return false;
    }

    fgets(tmp_char, 80, infp);
    while(!feof(infp) && !ferror(infp)){
        fscanf(infp,"%d,%d,%d\n", &tmp_Poly_ID, &Order, &Vert_ID);
        if (Order == 0){
            in_Poly.insert(pair<int, int>(tmp_Poly_ID , P_ID));

            p = in_Poly.find(tmp_Poly_ID);
            if (p == in_Poly.end()){
                cout<<"Polygon ID にエラーあり Polygon->"<< tmp_Poly_ID << endl;
                return false;
            }

            Poly.push_back(Polygon_Info(tmp_Poly_ID));
            ++ P_ID;
        }

        p = in_Vert.find(Vert_ID);
        if (p == in_Vert.end()){
```

```
            cout<<"頂点 ID にエラーあり Polygon->"<< tmp_Poly_ID << " Order->"
                << Order << " Vertice->" << Vert_ID << endl;
            return false;
        }

        Poly.back().V_ID.push_back(p->second);
        V[p->second].use = true;
    }

    for (Poly_vector::iterator itr_P=Poly.begin(); itr_P<Poly.end(); itr_P++)
        (*itr_P).N_Vert = (*itr_P).V_ID.size();

    fclose(infp);
    return true;
}

//--------------ファイル入力 表現データ--------------
bool TIN_IO::Input_Area(char *in_Poly_Area, i_map &Poly_ID, Poly_vector &Poly)
{
    int in_ID;
    double in_Area;
    char tmp_char[80], in_file[100];
    FILE *infp;
    i_map::iterator p;

    sprintf(in_file, "%s.csv", in_Poly_Area);
    if ((infp = fopen(in_file,"r")) == NULL){
        cout<<"-----Polygon 表現データ (面積) ファイルが開けません-----"<<endl;
        return false;
    }

    fgets(tmp_char, 80, infp);
    while(!feof(infp) && !ferror(infp)){
        fscanf(infp,"%d,%lf\n", &in_ID, &in_Area);
        p = Poly_ID.find(in_ID);
        if (p == Poly_ID.end()) cout<<"座標データ無し Polygon "<<in_ID<<endl;
        else Poly[p->second].Data = in_Area;
    }
    fclose(infp);

    //面積の入力されていない Polygon がある場合, 終了
    for (Poly_vector::iterator itr_P = Poly.begin(); itr_P < Poly.end(); itr_P ++){
        if ((*itr_P).Data == 0.0){
            cout<<"面積データ無し Polygon "<< (*itr_P).ID <<endl;
            return false;
        }
    }
    return true;
}

//--------------ファイル出力 ポリゴンの頂点データ--------------
void TIN_IO::Output_Poly_V_ID(char *in_Area, co_vector &coord, Poly_vector &Poly, V_vector &V)
{
    char out_file[100];
    FILE *outfp;
    Poly_vector::iterator itr_P;
    i_vector::iterator itr_i;

    sprintf(out_file, "%s_Polygon_Vertex.csv", in_Area);
    outfp=fopen(out_file, "w");
    fprintf(outfp, "gsgroup,gsorder,Vertex_ID,X,Y\n");

    for (itr_P = Poly.begin(); itr_P < Poly.end(); itr_P ++){
        for (itr_i = (*itr_P).V_ID.begin(); itr_i < (*itr_P).V_ID.end(); itr_i ++)
            fprintf(outfp, "%d,%d,%d,%.8f,%.8f\n", (*itr_P).ID, itr_i - (*itr_P).V_ID.begin(),
                *itr_i, coord[*itr_i].x, coord[*itr_i].y);

        itr_i = (*itr_P).V_ID.begin();
        fprintf(outfp, "%d,%d,%d,%.8f,%.8f\n",
            (*itr_P).ID, (*itr_P).V_ID.size(), *itr_i, coord[*itr_i].x, coord[*itr_i].y);
    }
    fclose(outfp);

    //座標ファイル
```

```
    sprintf(out_file, "%s_Vertices.csv", in_Area);
    outfp = fopen(out_file, "w");
    fprintf(outfp,"Vertex_ID,X,Y\n");
    for (int i = 0; i < int(V.size()); i ++)
        if (V[i].use) fprintf(outfp,"%d,%.10f,%.10f\n", V[i].ID, coord[i].x, coord[i].y);
    fclose(outfp);
}

//--------------ファイル出力 三角網データ--------------
void TIN_IO::Output_TIN(char *in_Area, co_vector &coord, Tri_vector &Tri, Poly_vector &Poly)
{
    int i, Tri_ID;
    char out_filename[100];
    FILE *outfp, *outfp2;
    Tri_vector::iterator itr_Tri;
    co_vector::iterator itr_co;

    //ArcGIS TIN 表示用ファイル
    sprintf(out_filename, "%s_Triangle_Vertices.csv", in_Area);
    outfp = fopen(out_filename,"w");
    fprintf(outfp,"X,Y,gsgroup,gsorder,Vertex,Polygon_ID\n");

    //三角網面積データファイル
    sprintf(out_filename, "%s_Triangle_Area.csv", in_Area);
    outfp2 = fopen(out_filename,"w");
    fprintf(outfp2,"Triangle_ID,Data,Area,Vertex1,Vertex2,Vertex3,Polygon\n");

    for (itr_Tri = Tri.begin(); itr_Tri < Tri.end(); itr_Tri ++){
        Tri_ID = itr_Tri - Tri.begin();
        for (i = 0; i < 3;  i ++)
            fprintf(outfp,"%.5f,%.5f,%d,%d,%d,%d\n", coord[(*itr_Tri).V[i]].x, coord[(*itr_Tri).V[i]].y,
                Tri_ID+1, i+1, (*itr_Tri).V[i], (*itr_Tri).Poly_ID);

        fprintf(outfp2,"%d,%.5f,%.5f,%d,%d,%d,%d\n", Tri_ID, (*itr_Tri).Data, calc_tri_area(coord, Tri[Tri_ID]),
            (*itr_Tri).V[0], (*itr_Tri).V[1], (*itr_Tri).V[2], (*itr_Tri).Poly_ID);
    }

    fclose(outfp);
    fclose(outfp2);
}
```

ContinuousAreaCartogramUtility.cpp

```
#include "stdafx.h"

//--------------頂点の ID を確認--------------
int ContinuousAreaCartogramUtility::ID_check(const int ID, const int N_Vert)
{
    if (ID < 0) return (ID + N_Vert);
    if (ID >= N_Vert) return (ID - N_Vert);
    return (ID);
}

//--------------ある辺を含む三角形を検索し，辺に含まれない頂点の ID を探索--------------
void ContinuousAreaCartogramUtility::Search_Vert(const int V1, const int V2, int *found_V1, int *found_V2,
    Tri_vector &Tri)
{
    int j, k, m, in_V[2];
    Tri_vector::iterator itr;

    in_V[0] = V1;
    in_V[1] = V2;
    sort(in_V, in_V + 2);

    *found_V1 = -1;
    *found_V2 = -1;

    for (itr = Tri.begin(); itr < Tri.end(); itr++){
        for (j=0; j<2; j++){
            if (in_V[0] == (*itr).V[j]){
                for (k=j+1; k<3; k++){
                    if (in_V[1] == (*itr).V[k]){
```

```
                        m = 0;
                        while ((m==j)||(m==k)) ++ m;
                        if (*found_V1==-1) *found_V1 = (*itr).V[m];
                        else{
                            *found_V2 = min(*found_V1, (*itr).V[m]);
                            *found_V1 = max(*found_V1, (*itr).V[m]);
                            return;
                        }
                    }
                }
            }
        }
    }
}

//--------------三角形の面積を計算--------------
double ContinuousAreaCartogramUtility::calc_tri_area(co_vector &coord, const Triangle_Info Tri)
{   return (fabs(tri_area_w_sign(Tri.V[0], Tri.V[1], Tri.V[2], coord)));}

//--------------三角形の符号付き面積を計算--------------
double ContinuousAreaCartogramUtility::tri_area_w_sign(const int p1, const int p2, const int p3, co_vector &coord)
{
    return ((coord[p1].x - coord[p3].x) * (coord[p2].y - coord[p3].y)
            + (coord[p2].x - coord[p3].x) * (coord[p3].y - coord[p1].y)) / 2;
}

//--------------三角形の符号付き面積を計算--------------
double ContinuousAreaCartogramUtility::tri_area_w_sign(const coord2D p1, const coord2D p2, const coord2D p3)
{   return ((p1.x - p3.x) * (p2.y - p3.y) + (p2.x - p3.x) * (p3.y - p1.y)) / 2;}

//--------------辺の距離を計算--------------
double ContinuousAreaCartogramUtility::calc_edge_ij_length(const int i, const int j, co_vector &coord)
{   return sqrt_norm(coord[j]-coord[i]);}

//--------------2 本の線分が交差しているか判定--------------
bool ContinuousAreaCartogramUtility::intersect(const int S0, const int E0, const int S1, const int E1,
    co_vector &coord)
{
    //端点を共有している場合は交差していないと判定
    if ((coord[S0]==coord[S1])||(coord[E0]==coord[S1])
        ||(coord[S0]==coord[E1])||(coord[E0]==coord[E1])) return false;

    //交差する可能性の無い場合を外す
    if ((max(coord[S0].x, coord[E0].x) <= min(coord[S1].x, coord[E1].x))
        || (max(coord[S0].y, coord[E0].y) <= min(coord[S1].y, coord[E1].y))
        || (min(coord[S0].x, coord[E0].x) >= max(coord[S1].x, coord[E1].x))
        || (min(coord[S0].y, coord[E0].y) >= max(coord[S1].y, coord[E1].y))) return false;

    return
      ((tri_area_w_sign(S0, E0, S1, coord) * tri_area_w_sign(S0, E0, E1, coord) < 0.0)
     &&(tri_area_w_sign(S1, E1, S0, coord) * tri_area_w_sign(S1, E1, E0, coord) < 0.0));
}

//--------------sin・cos を計算--------------
void ContinuousAreaCartogramUtility::calc_sincos_ijk(const int i, const int j, const int k, co_vector &coord,
    double *sin, double *cos)
{
    coord2D Edge_Vec[2];

    Edge_Vec[0] = coord[i] - coord[j];
    Edge_Vec[1] = coord[k] - coord[j];

    double tmp = sqrt(norm(Edge_Vec[0]) * norm(Edge_Vec[1]));
    *sin = (Edge_Vec[0].x * Edge_Vec[1].y - Edge_Vec[0].y * Edge_Vec[1].x) / tmp;
    *cos = (Edge_Vec[0].x * Edge_Vec[1].x + Edge_Vec[0].y * Edge_Vec[1].y) / tmp;
}

//--------------頂点 i → i+1 の距離を計算--------------
void ContinuousAreaCartogramUtility::calc_edge_length(const int i, i_vector &Poly_V, const int N_Vert,
     co_vector &coord, co_vector &Edge_Vec, d_vector &e_d)
{
    Edge_Vec[i] = coord[Poly_V[ID_check(i+1, N_Vert)]] - coord[Poly_V[i]];
    e_d[i] = sqrt_norm(Edge_Vec[i]);
}
```

```
//--------------角度を計算--------------
void ContinuousAreaCartogramUtility::calc_angle(const int i, const int N_Vert, co_vector &Edge_Vec,
    d_vector &e_d, d_vector &c_v, d_vector &s_v)
{
    int tmp_i = ID_check(i-1, N_Vert);
    double tmp = e_d[i]*e_d[tmp_i];    //辺長の積
    c_v[i] = -(Edge_Vec[i].x * Edge_Vec[tmp_i].x + Edge_Vec[i].y * Edge_Vec[tmp_i].y) / tmp;
    s_v[i] = (Edge_Vec[i].x * Edge_Vec[tmp_i].y - Edge_Vec[i].y * Edge_Vec[tmp_i].x) / tmp;
}

//--------------三角形内に点があるかを判定--------------
bool ContinuousAreaCartogramUtility::Point_In_ThreePoints(const coord2D Point, const coord2D P0,
    const coord2D P1, const coord2D P2)
{
    int unity = 1, two = 2, work_size = 66, info;
    double Y[2], X[4], work[66];
    char trans_N='N';

    Y[0] = Point.x - P0.x;
    Y[1] = Point.y - P0.y;
    X[0] = P1.x - P0.x;
    X[1] = P1.y - P0.y;
    X[2] = P2.x - P0.x;
    X[3] = P2.y - P0.y;

    dgels(&trans_N, &two, &two, &unity, X, &two, Y, &two, work, &work_size, &info);
    return ((Y[0]>=0)&&(Y[1]>=0)&&(Y[0]+Y[1]<=1));
}

//--------------辺の ID を設定--------------
void ContinuousAreaCartogramUtility::Set_Edge_ID(Poly_vector &Poly, Edge_vector &Edge)
{
    int i, same_ID;
    Edge_Info tmp_E;

    for (Poly_vector::iterator itr_P = Poly.begin(); itr_P < Poly.end(); itr_P ++){
        for (i=0; i<(*itr_P).N_Vert; i++){
            tmp_E = Edge_Info((*itr_P).V_ID[i], (*itr_P).V_ID[ID_check(i+1, (*itr_P).N_Vert)]);

            same_ID = Search_Same_Edge(tmp_E, Edge);
            if (same_ID==-1){//新しい辺
                tmp_E.P[0] = itr_P - Poly.begin();
                tmp_E.P[1] = -1;
                Edge.push_back(tmp_E);
            }
            else Edge[same_ID].P[1] = itr_P - Poly.begin();
        }
    }
}

//--------------同じ辺を検索--------------
int ContinuousAreaCartogramUtility::Search_Same_Edge(Edge_Info tmp_E, Edge_vector &Edge)
{
    for (Edge_vector::iterator itr = Edge.begin(); itr < Edge.end(); itr ++)
        if (tmp_E==(*itr)) return itr - Edge.begin();
    return -1;
}

//--------------ID の大小を判定--------------
bool large_ID_Value(const ID_Value* &a, const ID_Value* &b){return *a > *b;}
```

TIN_Area.cpp

```
#include "stdafx.h"

//--------------エリアカルトグラム作成計算--------------
double TIN_AREA::TIN_Area(const double Error_desired, double *myu, co_vector &coord, Tri_vector &Tri,
    Poly_vector &Poly)
{
    int i, tmp, iter = 0;
    double A_Error;
```

```
    Tri_vector calc_Tri(Tri.size());
    Tri_vector::iterator itr;
    coord2D origin;
    co_vector calc;
    i_vector calc_V_ID;

    std::copy(Tri.begin(), Tri.end(), calc_Tri.begin());
    vector<bool> Vert_ID_use(coord.size(), false);

    for (itr = calc_Tri.begin(); itr < calc_Tri.end() ; itr++)
        for (i=0; i<3; i++) Vert_ID_use[(*itr).V[i]] = true;

    for (i=0; i<int(coord.size()); i++){
        if (Vert_ID_use[i]){
            origin = coord[i];
            tmp = i;
            break;
        }
    }

    for (i=tmp; i<int(coord.size()); i++){
        if (Vert_ID_use[i]){
            calc.push_back(coord[i] - origin);
            calc_V_ID.push_back(i);
        }
    }

    for (itr = calc_Tri.begin(); itr < calc_Tri.end() ; itr++)
        for (i=0; i<3; i++)
            (*itr).V[i] = lower_bound(calc_V_ID.begin(), calc_V_ID.end(), (*itr).V[i]) - calc_V_ID.begin();

    A_Error = TIN_Area_main(&iter, Error_desired, myu, calc, calc_Tri);

    if (iter!=0) for (i=0; i<int(calc.size()); i++) coord[calc_V_ID[i]] = calc[i];

    return A_Error;
}

//--------------エリアカルトグラム作成計算--------------
double TIN_AREA::TIN_Area_main(int *iter, const double Error_desired, double *myu, co_vector &calc,
     Tri_vector &Tri)
{
    int effective_iter = 0, N_Tri, N_Vert, N_Edge;
    int X_row, X_col, X_element, work_size, info, unity=1;
    double Error = Error_desired * 2, Total_Data = 0.0, new_Error, old_Error, *X, *Y, *work;
    char trans_N='N';

    N_Tri = Tri.size();
    N_Vert = calc.size();

    //辺の ID を設定
    Edge_vector Edge;
    Edge.reserve(2*N_Tri+1);
    N_Edge = Set_Edge_ID(Edge, Tri);

    co_vector Edge_vec(N_Edge);
    calc_Edge_vector(calc, Edge_vec, Edge);  //三角網の辺長を計算
    Triangle_Area(Edge_vec, Tri);  //面積を計算
    Expand_Area(&Total_Data, Edge_vec, calc, Tri);  //  総面積に応じて拡大

    //計算用行列の設定
    X_row = N_Tri+N_Edge;
    X_col = 2*(N_Vert-1);
    X_element = X_row*X_col;
    work_size = 64*X_row+X_col;
    X = new double[X_element];
    Y = new double[X_row];
    work = new double[work_size];

    co_vector tmp_coord(N_Vert);
    std::copy(calc.begin(), calc.end(), tmp_coord.begin());

    //収束計算開始
    while ((Error>Error_desired)&&(*iter - effective_iter < 1)){
```

```
        ++ *iter;

        //初期化
        fill_n(X, X_element, 0.0);
        fill_n(Y, X_row, 0.0);

        AreaCarto_Expl_Matrix(X, Y, Edge_vec, Edge, N_Vert-1, *myu, Tri);

        dgels(&trans_N, &X_row, &X_col, &unity, X, &X_row, Y, &X_row, work, &work_size, &info);
        work_size = int(work[0]);

        Renew_coord(N_Vert-1, tmp_coord, Y);

        calc_Edge_vector(tmp_coord, Edge_vec, Edge);
        Triangle_Area(Edge_vec, Tri);

    //位相が破壊されていないか確認
        if (TIN_Topology(myu, tmp_coord, calc, Edge_vec, Edge, Tri)){
            //位相が破壊されていない場合
            Error = Error_Ratio(Total_Data, Tri);
            cout<<"iteration "<<*iter<<"  Error 比率  "<<Error <<"  myu="<<*myu<<endl;

            std::copy(tmp_coord.begin(), tmp_coord.end(), calc.begin());

            new_Error = Squared_Errors(Tri);
            if (effective_iter != 0) *myu *= new_Error / old_Error * 0.99;
            old_Error = new_Error;
            ++ effective_iter;
        }
        else{
            if (*iter == 1){
                *iter = 0;
                *myu *= 1.5;
                printf("重みを増加 myu=%.5f\n", *myu);
            }
        }
    }
    delete[] work, X, Y;

    copy(tmp_coord.begin(), tmp_coord.end(), calc.begin());
    return Error;
}

//--------------エリアカルトグラム作成の説明変数行列設定--------------
void TIN_AREA::AreaCarto_Expl_Matrix(double *X, double *Y, co_vector &Edge_Vec, Edge_vector &Edge,
    const int X_col_half, const double myu, Tri_vector &Tri)
{
    int X_row, tmp_zone, N_Tri, T_ID;
    double tmp;
    coord2D tmp_byd;
    Tri_vector::iterator itr_Tri;

    N_Tri = Tri.size();
    X_row = N_Tri + Edge.size();

    for (itr_Tri = Tri.begin(); itr_Tri < Tri.end(); itr_Tri ++){
        T_ID = itr_Tri - Tri.begin();
        tmp = (*itr_Tri).Area / (*itr_Tri).Data / (*itr_Tri).Data;

        if ((*itr_Tri).V[0] != 0){
            tmp_zone = (*itr_Tri).V[0];
            X[T_ID + (tmp_zone-1)*X_row] = - Edge_Vec[(*itr_Tri).E[2]].y * tmp;
            X[T_ID + (X_col_half+tmp_zone-1)*X_row] = Edge_Vec[(*itr_Tri).E[2]].x * tmp;
        }

        if ((*itr_Tri).V[1]!=0){
            tmp_zone = (*itr_Tri).V[1];
            X[T_ID + (tmp_zone-1)*X_row] = Edge_Vec[(*itr_Tri).E[1]].y * tmp;
            X[T_ID + (X_col_half+tmp_zone-1)*X_row] = - Edge_Vec[(*itr_Tri).E[1]].x * tmp;
      }

        if ((*itr_Tri).V[2]!=0){
            tmp_zone = (*itr_Tri).V[2];
            X[T_ID + (tmp_zone-1)*X_row] = - Edge_Vec[(*itr_Tri).E[0]].y * tmp;
```

```
            X[T_ID + (X_col_half+tmp_zone-1)*X_row] = Edge_Vec[(*itr_Tri).E[0]].x * tmp;
        }

        Y[T_ID] = 1 + 3 * pow((*itr_Tri).Area / (*itr_Tri).Data, 2);
    }

    for (unsigned int i=0; i<Edge.size(); i++){
        tmp_byd = Edge_Vec[i] * myu / norm(Edge_Vec[i]);

        if ( Edge[i].V[0]!=0 ){
            X[i + N_Tri +( Edge[i].V[0] - 1)*X_row] = - tmp_byd.y;
            X[i + N_Tri +( X_col_half+ Edge[i].V[0] -1)*X_row] = tmp_byd.x;
        }

        if ( Edge[i].V[1]!=0 ){
            X[i + N_Tri +( Edge[i].V[1] -1)*X_row] = tmp_byd.y;
            X[i + N_Tri +( X_col_half+ Edge[i].V[1] -1)*X_row] = - tmp_byd.x;
        }
    }
}

//--------------総面積を計算し，初期座標を拡大・位相破壊判定のため入力値 A に符号を付ける--------------
void TIN_AREA::Expand_Area(double *Total_Data, co_vector &Edge_vec, co_vector &coord, Tri_vector &Tri)
{
    co_vector::iterator itr_co;
    Tri_vector::iterator itr_Tri;
    double Expand, Expand_sq, Total_Area = 0.0;

    for (itr_Tri=Tri.begin(); itr_Tri<Tri.end(); itr_Tri++){
        Total_Area += fabs((*itr_Tri).Area);
        *Total_Data += (*itr_Tri).Data;
    }

    Expand_sq = *Total_Data/Total_Area;
    Expand = sqrt(Expand_sq);

    for (itr_co=Edge_vec.begin(); itr_co<Edge_vec.end(); itr_co++) *itr_co = *itr_co * Expand;
    for (itr_co=coord.begin(); itr_co<coord.end(); itr_co++) *itr_co = *itr_co * Expand;
    for (itr_Tri=Tri.begin(); itr_Tri<Tri.end(); itr_Tri++){
        (*itr_Tri).Area *= Expand_sq;
        if ((*itr_Tri).Area < 0) (*itr_Tri).Data *= -1;
    }
}

//--------------三角形の面積を計算--------------
void TIN_AREA::Triangle_Area(co_vector &Edge_vec, Tri_vector &Tri)
{
    for (Tri_vector::iterator itr=Tri.begin(); itr<Tri.end(); itr++)
        (*itr).Area = (Edge_vec[(*itr).E[0]].x * Edge_vec[(*itr).E[1]].y
            - Edge_vec[(*itr).E[1]].x * Edge_vec[(*itr).E[0]].y)/2;
}

//--------------三角網の位相が破壊されていないか判定--------------
//--------------破壊されているときには重みを増加・計算前の座標にもどす--------------
bool TIN_AREA::TIN_Topology(double *myu, co_vector &new_coord, co_vector &old_coord, co_vector &Edge_vec,
    Edge_vector &Edge, Tri_vector &Tri)
{
    Edge_vector::iterator itr_E_0, itr_E_1;
    ContinuousAreaCartogramUtility Util;

    for (Tri_vector::iterator itr_T = Tri.begin(); itr_T < Tri.end(); itr_T++){
        if (((*itr_T).Data * (*itr_T).Area)<0){  //  三角形の反転は面積の符号の変化で判定
            cout<<"三角網 位相破壊"<<endl;
            copy(old_coord.begin(), old_coord.end(), new_coord.begin());

            calc_Edge_vector(new_coord, Edge_vec, Edge);
            Triangle_Area(Edge_vec, Tri);
            return false;
        }
    }

    for (itr_E_0 = Edge.begin(); itr_E_0 < Edge.end()-1; itr_E_0++){
        for (itr_E_1 = Edge.begin()+1; itr_E_1 < Edge.end(); itr_E_1++){
            if (Util.intersect((*itr_E_0).V[0], (*itr_E_0).V[1], (*itr_E_1).V[0],
```

```
                (*itr_E_1).V[1], new_coord)){
                cout<<"三角網 交差"<<endl;
                copy(old_coord.begin(), old_coord.end(), new_coord.begin());

                calc_Edge_vector(new_coord, Edge_vec, Edge);
                Triangle_Area(Edge_vec, Tri);
                return false;
            }
        }
    }
    return true;
}

//--------------辺の ID を設定--------------
int TIN_AREA::Set_Edge_ID(Edge_vector &Edge, Tri_vector &Tri)
{
    for (Tri_vector::iterator itr_T = Tri.begin();
        itr_T < Tri.end(); itr_T++){
        Set_New_Edge_ID(Edge, &(*itr_T), 0, 0, 1);
        Set_New_Edge_ID(Edge, &(*itr_T), 1, 0, 2);
        Set_New_Edge_ID(Edge, &(*itr_T), 2, 1, 2);
    }
    return Edge.size();
}

//--------------新しい辺の ID を設定--------------
void TIN_AREA::Set_New_Edge_ID(Edge_vector &Edge, Triangle_Info *tmp_Tri, const int E_Order, const int V_0,
    const int V_1)
{
    for (Edge_vector::iterator itr = Edge.begin(); itr < Edge.end(); itr++){
        if (((*tmp_Tri).V[V_0] == (*itr).V[0])&&((*tmp_Tri).V[V_1] == (*itr).V[1])){
            (*tmp_Tri).E[E_Order] = itr - Edge.begin();    //辺の ID 既設定
            return;
        }
    }
    (*tmp_Tri).E[E_Order] = Edge.size();    //ID を設定
    Edge.push_back(Edge_Info((*tmp_Tri).V[V_0], (*tmp_Tri).V[V_1]));
}

//--------------辺の方向ベクトルを計算--------------
void TIN_AREA::calc_Edge_vector(co_vector &coord, co_vector &Edge_vec, Edge_vector &Edge)
{   for (unsigned int i = 0; i < Edge.size(); i ++) Edge_vec[i] = coodEdge[i].V[1]] - coord[Edge[i].V[0]];}

//--------------表現精度の残差二乗和を計算--------------
double TIN_AREA::Squared_Errors(Tri_vector &Tri)
{
    double sq_error = 0.0;
    for (Tri_vector::iterator itr = Tri.begin(); itr < Tri.end(); itr ++)
        sq_error += pow((*itr).Data - (*itr).Area, 2);
    return sq_error;
}

//--------------表現精度の残差の総和を計算--------------
double TIN_AREA::Error_Ratio(const double Total_Data, Tri_vector &Tri)
{
    double error = 0.0;
    for (Tri_vector::iterator itr = Tri.begin(); itr < Tri.end(); itr ++)
        error += fabs((*itr).Data - (*itr).Area);
    return (error/Total_Data*100);
}

//--------------計算結果を用いて修正--------------
void TIN_AREA::Renew_coord(const int N, co_vector &new_coord, double *Y)
{
    new_coord[0] = coord2D(0.0, 0.0);
    for (int i=0; i<N; i++) new_coord[i+1] = coord2D(Y[i], Y[i+N]);
}
```

TIN_Make.cpp

```
#include "stdafx.h"
```

```
//--------------TIN 作成--------------
void TIN_MAKE::TIN_Make_in_Poly(Tri_vector &Tri, co_vector &coord, Poly_vector &Poly)
{
    int N_Tri_first;
    coord2D tmp;
    i_vector tmp_Poly_V, Added_V_ID;
    ContinuousAreaCartogramUtility Util;

    Tri.clear();  //  三角形データの初期化

    for (Poly_vector::iterator itr_P = Poly.begin(); itr_P < Poly.end(); itr_P ++){
        tmp = coord2D(0.0, 0.0);
        (*itr_P).Area = 0.0;
        Added_V_ID.clear();
        N_Tri_first = Tri.size();

        tmp_Poly_V.resize((*itr_P).N_Vert);
        copy((*itr_P).V_ID.begin(), (*itr_P).V_ID.end(), tmp_Poly_V.begin());

        Tin_Make_wo_Centroid(Tri, tmp_Poly_V, coord);
        Adjust_TIN_Flip(N_Tri_first, Tri, coord);

        Steiner(N_Tri_first, coord, Tri, Added_V_ID);
        Adjust_TIN_Flip(N_Tri_first, Tri, coord);

        for (int i=0; i<3; i++){
            Laplacian_Smoothing(N_Tri_first, Tri, coord, Added_V_ID);
            Adjust_TIN_Flip(N_Tri_first, Tri, coord);
        }

        //三角形の面積・重心を計算し，ポリゴンの重心を計算する
        for (Tri_vector::iterator itr_T = Tri.begin() + N_Tri_first;
            itr_T < Tri.end(); itr_T ++){
            (*itr_T).Area = Util.calc_tri_area(coord, *itr_T);
            (*itr_T).Poly_ID = (*itr_P).ID;
            (*itr_T).Centroid = (coord[(*itr_T).V[0]] + coord[(*itr_T).V[1]] + coord[(*itr_T).V[2]]) / 3;
            tmp = tmp + (*itr_T).Centroid * (*itr_T).Area;
            (*itr_P).Area += (*itr_T).Area;
        }

        (*itr_P).Centroid = tmp / (*itr_P).Area;
        tmp_Poly_V.clear();
        (*itr_P).Density = (*itr_P).Data / (*itr_P).Area;    //ポリゴンの密度 (データ/面積) を計算
    }

    //各ポリゴンごとに隣接するポリゴンの重心までのベクトルを計算し，データを割り振る
    Distribute_Data2TIN(Tri, coord, Poly);
}

//--------------三角形の面積に応じて表現データを分配--------------
void TIN_MAKE::Distribute_Data2TIN(Tri_vector &Tri, co_vector &coord, Poly_vector &Poly)
{
    int i_P, i_T = 0,  N_P = Poly.size(), N_T = Tri.size(), X_row = N_T + N_P;
    int X_element = X_row * X_row, unity = 1, info, i, max_P, Row, N_Contig, *ipiv;
    char trans_N='N';
    double *X, *Y, tmp_d, max_d, d_Contig_P, ratio;
    bool Suc = true;
    co_vector coord_c_Poly;
    i_vector::iterator itr;
    coord2D tmp_coord;

    //---計算用行列の設定
    X = new double[X_element];
    Y = new double[X_row];
    fill_n(X, X_element, 0.0);
    fill_n(Y, X_row, 0.0);

    for (Poly_vector::iterator itr_P = Poly.begin(); itr_P < Poly.end(); itr_P ++){
        coord_c_Poly.clear();
        i_P = itr_P - Poly.begin();

        //Y にポリゴンに与えられたデータを入力
        Y[i_P] = (*itr_P).Data;
```

```
        for (itr = (*itr_P).C_Poly.begin(); itr < (*itr_P).C_Poly.end(); itr++)
            coord_c_Poly.push_back(Poly[*itr].Centroid - (*itr_P).Centroid);

        N_Contig = (*itr_P).C_Poly.size();

        while (Tri[i_T].Poly_ID == (*itr_P).ID){
            X[i_P + i_T * X_row] = 1.0;

            tmp_coord = Tri[i_T].Centroid - (*itr_P).Centroid;

            max_d = 0.0;
            for (i=0; i<N_Contig; i++){
                tmp_d = inner_product(tmp_coord, coord_c_Poly[i]) / sqrt_norm(coord_c_Poly[i]);
                 if (tmp_d > max_d){
                    max_d = tmp_d;
                    max_P = (*itr_P).C_Poly[i];
                    d_Contig_P = sqrt_norm(coord_c_Poly[i]);
                }
            }

            Row = N_P + i_T;

            if (max_d > 0.0){
                if (max_d < d_Contig_P){
                    X[Row + i_T * X_row] = - d_Contig_P / Tri[i_T].Area;
                    X[Row + (N_T + i_P) * X_row] = (*itr_P).Density * (d_Contig_P - max_d);
                    X[Row + (N_T + max_P) * X_row] = Poly[max_P].Density * max_d;
                }
                else{
                    X[Row + i_T * X_row] = - 1 / Tri[i_T].Area;
                    X[Row + (N_T + max_P) * X_row] = Poly[max_P].Density;
                }
            }
            else{
                X[Row + i_T * X_row] = - 1 / Tri[i_T].Area;
                X[Row + (N_T + i_P) * X_row] = (*itr_P).Density;
            }

            ++ i_T;
        }
    }

    ipiv = new int[X_row];
    dgetrf(&X_row, &X_row, X, &X_row, ipiv, &info);
    dgetrs(&trans_N, &X_row, &unity, X, &X_row, ipiv, Y, &X_row, &info);
    delete[] X, ipiv;

    //三角形に与えられる面積があまりに小さくなる Polygon の場合は，面積に応じて配分する
    for (i_P=0; i_P<N_P; i_P++){
        if (Y[i_P+N_T] < 0.1){
            Suc = false;
            break;
        }
    }

    if (Suc) for (i_T=0; i_T<N_T; i_T++) Tri[i_T].Data = Y[i_T];
    else{
        for (i_P=0; i_P<N_P; i_P++){
            ratio = Poly[i_P].Data / Poly[i_P].Area;
            for (i_T=0; i_T<N_T; i_T++)
                if (Tri[i_T].Poly_ID == Poly[i_P].ID) Tri[i_T].Data = ratio * Tri[i_T].Area;
        }
    }
    delete[] Y;
}

//--------------辺を共有する三角形を検索--------------
bool TIN_MAKE::Tri_Share_Edge(const int V_ID_1, const int V_ID_2, const int Tri_ID, const int min_Tri_ID,
    const int max_Tri_ID, Tri_vector &Tri, int *found_Tri_ID, int *not_shared_v_ID)
{
    int i, j, k, m, V_ID_s, V_ID_l;

    V_ID_s = min(V_ID_1, V_ID_2);
    V_ID_l = max(V_ID_1, V_ID_2);
```

```
    for (i = min_Tri_ID; i < max_Tri_ID; i++){
        if (Tri_ID!=i){//異なる三角形を探す
            for (j=0; j<2; j++){
                if (V_ID_s==Tri[i].V[j]){
                    for (k=j+1; k<3; k++){
                        if (V_ID_l==Tri[i].V[k]){
                            m = 0;
                            while ((m==j)||(m==k)) ++ m;
                            *not_shared_v_ID = Tri[i].V[m];
                            *found_Tri_ID = i;
                            return true;
                        }
                    }
                }
            }
        }
    }
    return false;
}

//--------------Delaunay 三角網となるように Flip 操作を用いて三角網を修正--------------
void TIN_MAKE::Adjust_TIN_Flip(const int N_Tri_first, Tri_vector &Tri, co_vector &coord)
{
    bool Flipped;
    int i, j, found_Tri_ID, not_shared_v_ID, N_Tri, tmp_V[4];
    ContinuousAreaCartogramUtility Util;

    N_Tri = Tri.size();

    do{
        Flipped = false;
        for (i = N_Tri_first; i < N_Tri; i ++){
            for (j=0; j<3; j++){
                tmp_V[0] = Tri[i].V[j];
                tmp_V[1] = Tri[i].V[Util.ID_check(j+1, 3)];
                tmp_V[2] = Tri[i].V[Util.ID_check(j+2, 3)];
                if (Tri_Share_Edge(tmp_V[0], tmp_V[2], i, N_Tri_first, N_Tri, Tri,
                    &found_Tri_ID, &not_shared_v_ID)){
                    tmp_V[3] = not_shared_v_ID;

                    if (Flip(tmp_V, coord, Tri)){
                        Tri.erase(Tri.begin()+max(i,found_Tri_ID));
                        Tri.erase(Tri.begin()+min(i,found_Tri_ID));
                        Flipped = true;
                    }
                }
            }
        }
    } while (Flipped);
}

//--------------三角形の頂点 ID を記録--------------
void TIN_MAKE::Out_Tri_Info(int *Tri_V, Tri_vector &Tri)
{
    int tmp_Tri_V[3];
    std::copy(Tri_V, Tri_V+3, tmp_Tri_V);
    std::sort(tmp_Tri_V, tmp_Tri_V+3);
    Tri.push_back(Triangle_Info(tmp_Tri_V));
}

//--------------三角形の頂点 ID を記録--------------
void TIN_MAKE::Out_Tri_Info(const int Tri_V_0, const int Tri_V_1, const int Tri_V_2, Tri_vector &Tri)
{
    int tmp_Tri_V[3];
    tmp_Tri_V[0] = Tri_V_0;
    tmp_Tri_V[1] = Tri_V_1;
    tmp_Tri_V[2] = Tri_V_2;
    std::sort(tmp_Tri_V, tmp_Tri_V+3);
    Tri.push_back(Triangle_Info(tmp_Tri_V));
}

//--------------TIN 作成--------------
void TIN_MAKE::Tin_Make_wo_Centroid(Tri_vector &Tri, i_vector &tmp_Poly_V, co_vector &coord)
```

```
{
    bool hantei;
    int i, tmp_V[3], tmp_O[3], tmp_N_Vert = tmp_Poly_V.size();
    d_vector e_d(tmp_N_Vert), s_v(tmp_N_Vert), c_v(tmp_N_Vert);
    co_vector Edge_Vec(tmp_N_Vert);
    std::vector<ID_Value*> V_pos;
    vector<ID_Value*>::iterator itr;
    ContinuousAreaCartogramUtility Util;

    for (i=0; i<tmp_N_Vert; i++) Util.calc_edge_length(i, tmp_Poly_V, tmp_N_Vert, coord, Edge_Vec, e_d);
    for (i=0; i<tmp_N_Vert; i++) Util.calc_angle(i, tmp_N_Vert, Edge_Vec, e_d, c_v, s_v);

    while (tmp_N_Vert>3){
        //角度が 180 °以下を抽出する
        for (i=0; i<tmp_N_Vert; i++) if (s_v[i] > 0.0) V_pos.push_back(new ID_Value(i, c_v[i]));
        sort(V_pos.begin(), V_pos.end(), large_ID_Value);

        //小さな角度から順に三角形を作る
        //但し他の点が内部にあったときは次の候補を探す
        for (itr  = V_pos.begin(); itr < V_pos.end(); itr++){
            for (i=0; i<3; i++){
                tmp_O[i] = Util.ID_check((**itr).ID - 1 + i, tmp_N_Vert);
                tmp_V[i] = tmp_Poly_V[tmp_O[i]];
            }

            //中に点がないか探し，無ければ三角形を作る頂点を決定
            hantei = true;
            for (i=0; i<tmp_N_Vert; i++){
                if ((s_v[i] <= 0.0)&&(tmp_Poly_V[i] != tmp_V[0])&&(tmp_Poly_V[i] != tmp_V[2])){
                    if (Util.Point_In_ThreePoints(coord[tmp_Poly_V[i]],
                        coord[tmp_V[0]], coord[tmp_V[1]], coord[tmp_V[2]])){
                        hantei = false;
                        break;
                    }
                }
            }

            if (hantei){
                //三角形ファイルの出力
                Out_Tri_Info(tmp_V, Tri);
                V_pos.clear();

                //tmp_O[0] の辺長を頂点 tmp_O[0] tmp_O[2] 間の辺長に更新
                e_d[tmp_O[0]] = calc_edge_ij_length(tmp_V[2], tmp_V[0], coord);

                //tmp_O[0]・tmp_O[2] の頂角の cos・sin を更新
                Util.calc_sincos_ijk(tmp_Poly_V[Util.ID_check(tmp_O[0]-1, tmp_N_Vert)],
                    tmp_V[0], tmp_V[2], coord, &s_v[tmp_O[0]], &c_v[tmp_O[0]]);
                Util.calc_sincos_ijk(tmp_V[0], tmp_V[2], Tmp_Poly_V[Util.ID_check(tmp_O[2]+1, tmp_N_Vert)], coord,
                   &s_v[tmp_O[2]], &c_v[tmp_O[2]]);

                //頂点・cos・sin・辺長のリストから消去
                tmp_Poly_V.erase(tmp_Poly_V.begin() + tmp_O[1]);
                e_d.erase(e_d.begin() + tmp_O[1]);
                s_v.erase(s_v.begin() + tmp_O[1]);
                c_v.erase(c_v.begin() + tmp_O[1]);
                -- tmp_N_Vert;

                break;
            }
        }
    }

    //三角形のノード番号を記録
    for (i=0; i<3; i++) tmp_V[i] = tmp_Poly_V[i];
    Out_Tri_Info(tmp_V, Tri);
}

//--------------フリップ操作--------------
bool TIN_MAKE::Flip(int *tmp_Poly_V, co_vector &coord, Tri_vector &Tri)
{
    double radius_sq;
    coord2D circle;
    ContinuousAreaCartogramUtility Util;
```

```
    //二つの三角形から作られる四辺形が凸かどうか判定←対角線が交差する
    if (Util.intersect(tmp_Poly_V[0], tmp_Poly_V[2], tmp_Poly_V[1], tmp_Poly_V[3], coord)){
        Circumcircle(tmp_Poly_V, &circle, &radius_sq, coord);  //外接円を求める

        //外接円内に tmp_Poly_V[3] がある場合
        if (Point_in_Circle(tmp_Poly_V[3], radius_sq, circle, coord)){
            Out_Tri_Info(tmp_Poly_V[0], tmp_Poly_V[1], tmp_Poly_V[3], Tri);
            Out_Tri_Info(tmp_Poly_V[1], tmp_Poly_V[2], tmp_Poly_V[3], Tri);
            return true;
        }
    }
    return false;
}

//--------------三角形の外接円の中心座標と半径を算出--------------
void TIN_MAKE::Circumcircle(int *Point_ID, coord2D *center, double *radius_sq, co_vector &coord)
{
    int i;
    double tmp_U_norm[2], tmp_L;
    coord2D Edge_Vec[2];

    for (i=0; i<2; i++) Edge_Vec[i] = coord[Point_ID[i+1]] - coord[Point_ID[0]];

    tmp_L = (Edge_Vec[0].y * Edge_Vec[1].x - Edge_Vec[0].x * Edge_Vec[1].y) * 2;
    for (i=0; i<2; i++) tmp_U_norm[i] = norm(Edge_Vec[i]);

    *center = coord2D((-tmp_U_norm[0] * Edge_Vec[1].y + tmp_U_norm[1] * Edge_Vec[0].y) / tmp_L,
       (tmp_U_norm[0] * Edge_Vec[1].x - tmp_U_norm[1] * Edge_Vec[0].x) / tmp_L);
    *radius_sq = norm(*center);
    *center = coord[Point_ID[0]] + *center;
}

//--------------外接円内に Point_ID がないか探す--------------
bool TIN_MAKE::Point_in_Circle(const int Point_ID, const double radius_sq, coord2D circle, co_vector &coord)
{
    if (norm(coord[Point_ID]-circle) < radius_sq) return true;
    else return false;
}

//--------------Steiner 点の算出--------------
void TIN_MAKE::Steiner(const int N_Tri_first, co_vector &coord, Tri_vector &Tri, i_vector &Added_V_ID)
{
    bool Add;
    int i, j, N_tmp_V = 0;
    double tmp_radius, dist, length;
    const double alpha = 0.65, beta = 0.5;
    coord2D coord_center;
    co_vector tmp_coord, add_coord;
    i_map tmp_V;
    pair<i_map::iterator, bool> res;
    Tri_vector::iterator itr_T, itr_T_First;

    itr_T_First = Tri.begin() + N_Tri_first;

    //三角形の頂点 ID を格納
    for (itr_T = itr_T_First; itr_T < Tri.end(); itr_T ++){
        for (j=0; j<3; j++){
            res = tmp_V.insert(pair<int, int>((*itr_T).V[j], N_tmp_V));
            if (res.second){
                tmp_coord.push_back(coord[(*itr_T).V[j]]);
                ++ N_tmp_V;
            }
        }
    }

    vector<bool> Edge_list(N_tmp_V * N_tmp_V, false);
    i_vector count_Edge(N_tmp_V, 0);
    d_vector radius(N_tmp_V), Sum_Edge_length(N_tmp_V, 0.0);

    //三角形の辺の情報を格納
    for (itr_T = itr_T_First; itr_T < Tri.end(); itr_T ++){
        for (i=0; i<2; i++){
            for (j=i+1; j<3; j++){
```

```
                if (tmp_V[(*itr_T).V[i]] < tmp_V[(*itr_T).V[j]])
                    Edge_list[tmp_V[(*itr_T).V[i]] N_tmp_V] + tmp_V[(*itr_T).V[j]] * N_tmp_V] = true;
                else
                    Edge_list[tmp_V[(*itr_T).V[j]] N_tmp_V] + tmp_V[(*itr_T).V[i]] * N_tmp_V] = true;
            }
        }
    }

    //頂点につながる辺の平均長を計算
    for (i=0; i<N_tmp_V-1; i++){
        for (j=i+1; j<N_tmp_V; j++){
            if (Edge_list[i+j*N_tmp_V]){
                length = sqrt_norm(tmp_coord[i]-tmp_coord[j]);
                ++ count_Edge[i];
                ++ count_Edge[j];
                Sum_Edge_length[i] += length;
                Sum_Edge_length[j] += length;
            }
        }
    }

    for (i=0; i<N_tmp_V; i++) radius[i] = Sum_Edge_length[i] / count_Edge[i];

    //Steiner 点の追加基準を作成 各頂点より radius 以上離れること
    for (i = N_Tri_first; i < int(Tri.size()); i ++){
        tmp_radius = (radius[tmp_V[Tri[i].V[0]]] + radius[tmp_V[Tri[i].V[1]]] + radius[tmp_V[Tri[i].V[2]]])/3;
        coord_center = (coord[Tri[i].V[0]] + coord[Tri[i].V[1]] + coord[Tri[i].V[2]])/3;
        dist = min(sqrt_norm(coord[Tri[i].V[0]]-coord_center),
            min(sqrt_norm(coord[Tri[i].V[1]]-coord_center), sqrt_norm(coord[Tri[i].V[2]]-coord_center)));

        if (dist > tmp_radius * alpha){
            Add = true;
            for (j=0; j<int(add_coord.size()); j++){
                if (sqrt_norm(add_coord[j] - coord_center) < tmp_radius * beta){
                    Add = false;
                    break;
                }
            }

            if (Add){
                Out_Tri_Info(Tri[i].V[0], Tri[i].V[2], coord.size(), Tri);
                Out_Tri_Info(Tri[i].V[1], Tri[i].V[2], coord.size(), Tri);

                Tri[i].V[2] = coord.size();
                Added_V_ID.push_back(coord.size());

                coord.push_back(coord_center);
                add_coord.push_back(coord_center);
            }
        }
    }
}

//--------------Laplacian_Smoothing で三角網形状を改善--------------
void TIN_MAKE::Laplacian_Smoothing(const int N_Tri_First, Tri_vector &Tri, co_vector &coord,
    i_vector &Added_V_ID)
{
    bool Inside, Overturn;
    int k;
    double Tri_Area, Total_Area;
    coord2D coord_COM, coord_Tri;
    i_vector::iterator itr_i;
    Tri_vector::iterator itr_T, itr_T2, itr_T_First = Tri.begin() + N_Tri_First;
    ContinuousAreaCartogramUtility Util;

    //追加された点に関して
    for (itr_i = Added_V_ID.begin(); itr_i < Added_V_ID.end(); itr_i ++){
        coord_COM = coord2D(0.0, 0.0);
        Total_Area = 0.0;

        //点を頂点とする多角形の重心を求める
        for (itr_T = itr_T_First; itr_T < Tri.end(); itr_T ++){
            for (k=0; k<3; k++){
                if ( (*itr_i) == (*itr_T).V[k]){
```

```
                coord_Tri = (coord[(*itr_T).V[0]] + coord[(*itr_T).V[1]] + coord[(*itr_T).V[2]])/3;
                Tri_Area = Util.calc_tri_area(coord, (*itr_T));
                coord_COM = coord_COM + coord_Tri * Tri_Area;
                Total_Area += Tri_Area;
                break;
            }
        }
    }

    if (Total_Area == 0.0) printf("Error");
    coord_COM = coord_COM / Total_Area;

    Inside = false;
    Overturn = false;

    //多角形の重心が多角形内に含まれているとき
    for (itr_T = itr_T_First; itr_T < Tri.end(); itr_T ++){
        if (Util.Point_In_ThreePoints(coord_COM, coord[(*itr_T).V[0]], coord[(*itr_T).V[1]],
            coord[(*itr_T).V[2]])){
            Inside = true;

            for (itr_T2 = itr_T_First; itr_T2 < Tri.end(); itr_T2 ++){
                //頂点の移動による三角形の反転の有無を確認
                if (Overturn_Triangle(*itr_i, coord_COM, *itr_T2, coord)){
                    Overturn = true;
                    break;
                }
            }
            break;
        }
    }

    if ((!Overturn)&&(Inside)) coord[*itr_i] = coord_COM;
  }
}

//--------------Target の座標を coord_COM に変更したら三角形が反転するかを判定--------------
bool TIN_MAKE::Overturn_Triangle(const int Target, coord2D coord_COM, Triangle_Info Tri, co_vector &coord)
{
    bool included = false;
    coord2D Edge_Vec[4], co_Target_p, co_Target_m;
    ContinuousAreaCartogramUtility Util;

    for (int i=0; i<3; i++){
        if (Tri.V[i] == Target){
            co_Target_p = coord[Tri.V[Util.ID_check(i+1, 3)]];
            co_Target_m = coord[Tri.V[Util.ID_check(i-1, 3)]];
            included = true;
            break;
        }
    }

    if (!included) return false;

    Edge_Vec[0] = co_Target_p - coord[Target];
    Edge_Vec[1] = co_Target_m - coord[Target];
    Edge_Vec[2] = co_Target_p - coord_COM;
    Edge_Vec[3] = co_Target_m - coord_COM;

    //sin の符号が変化しないとき反転なし
    if ((Edge_Vec[0].x * Edge_Vec[1].y - Edge_Vec[0].y * Edge_Vec[1].x)
        *(Edge_Vec[2].x * Edge_Vec[3].y - Edge_Vec[2].y * Edge_Vec[3].x) > 0.0) return false;
    return true;
}
```

B.4　サークルエリアカルトグラム

ファイル構成

サークルエリアカルトグラム作成では，以下のプログラムを使用する．

1. 共通ファイル (CartogramClass.h; CartogramUtility.h; CartogramUtility.cpp)
2. stdafx.h
3. CircleAreaCartogram.h
4. CircleAreaCartogram.cpp
5. CircleAreaCartogramAlgorithm.cpp
6. CircleAreaCartogramIO.cpp

地域の代表点座標と表現データ, および地域の隣接関係に関するデータを入力すると，カルトグラム上の中心座標を出力する (図 B.3).

地域ID	X	Y	データ
11201	12695.0	3990.2	324253
11202	12685.3	4017.6	155940
11203	12718.0	3981.6	459859
11206	12694.4	4016.2	86013
11207	12658.9	3996.6	61384
11208	12693.0	3977.7	327592
11209	12674.0	3987.5	82185
11210	12706.8	4014.8	68753
11211	12669.0	4026.6	59818
11212	12687.0	4003.5	90390
11213	12716.3	3993.8	109817
11214	12720.2	3997.0	204752
11215	12689.7	3984.3	161770
11216	12701.7	4019.2	57397
11217	12698.3	4007.0	83083
11218	12677.2	4022.1	102488
11219	12704.3	3996.7	213980
11221	12724.4	3982.1	225177
11222	12723.3	3989.0	306535

(a) 地域代表点座標・表現データ入力

地域ID1	地域ID2
11201	11208
11201	11215
11201	11219
11201	11235
11201	11236
11201	11239
11201	11241
11201	11242
11201	11244
11201	11322
11201	11324
11201	11346
11202	11206
11202	11212
11202	11218
11202	11304
11202	11341
11202	11401
11202	11402

(b) 地域隣接関係入力

地域ID	X	Y	半径
11201	-4065.2	2433.0	321.3
11202	-4380.5	3348.5	222.8
11203	-2031.3	2242.6	382.6
11206	-3815.0	3667.8	165.5
11207	-5330.0	2047.6	139.8
11208	-4336.3	1590.6	322.9
11209	-4992.5	2026.2	161.7
11210	-3103.6	3678.8	147.9
11211	-5272.1	3130.9	138.0
11212	-4314.9	2944.4	169.6
11213	-2362.8	2918.0	187.0
11214	-1997.6	3104.7	255.3
11215	-4443.9	2053.5	226.9
11216	-3434.7	3839.3	135.2
11217	-3612.3	3422.9	162.6
11218	-4829.6	3306.5	180.6
11219	-3577.2	2875.9	261.0
11221	-1210.9	2350.1	267.7
11222	-1619.1	2746.5	312.4

(c) 円中心座標・半径出力

図 B.3: サークルエリアカルトグラム 入出力ファイル (例)

stdafx.h

```
#include <iostream>
#include <tchar.h>
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include "time.h"
#include <string.h>
#include <vector>
#include <map>
#include "CartogramClass.h"
#include "CartogramUtility.h"
#include "Circle Area Cartogram.h"
#include "mkl.h"
using namespace System;
using namespace std;
```

CircleAreaCartogram.h

```
class Circle_Algorithm
{
public:
    void Circle_Area_Cartogram_algorithm(int N_Zone, int N_Link_max, bool *link_pair, double alpha, double beta,
        double eps, d_vector &d, d_vector &t, d_vector &radius, Edge_vector &link, co_vector &coord,
        co_vector &link_vec);
    void link_distance(d_vector &d, co_vector &link_vec, Edge_vector &link, co_vector &coord, int N_Link);
private:
    void Adjust_scale(int N_Link, int N_Zone, d_vector &t, d_vector &d, d_vector &d_ini, co_vector &coord,
        co_vector &coord_ini);
    void First_Iter(int N_Zone, double alpha,  double *max_overlap, Edge_vector &link, bool *link_pair,
        double *X, double *tmp_X, double *Y, double *work, d_vector &d, d_vector &d_ini, d_vector &t,
        d_vector &radius, co_vector &link_vec, co_vector &coord, co_vector &ini_coord);
    void Iter(int N_Zone, double alpha, double beta, double *max_overlap, Edge_vector &link, bool *link_pair,
        double *X, double *tmp_X, double *Y, double *work, d_vector &d, d_vector &d_ini, d_vector &t,
        d_vector &radius, co_vector &link_vec, co_vector &coord, co_vector &ini_coord);
    void make_X_circle(int N_Link, int N_element, Edge_vector &link, double *X);
    void adjust_XY_circle(int N_Link, Edge_vector &link, d_vector &d, d_vector &d_design, d_vector &d_real,
        co_vector &link_vec, double alpha, double *Y, double *tmp_X, double beta);
    void make_Y_circle(int N_Link, d_vector &d, d_vector &d_design, d_vector &d_real, co_vector &link_vec,
        double alpha, double *Y, int ID);
    void adjust_Y_circle(int n_link, d_vector &d, d_vector &d_design, d_vector &d_real, co_vector &link_vec,
        double alpha, double *Y, double beta);
    double Add_OverLapped_Link(int N_Zone, bool *link_pair, d_vector &t, d_vector &d, d_vector &d_ini,
        d_vector &radius, Edge_vector &link, co_vector &coord, co_vector &ini_coord, co_vector &link_vec);
};

class Circle_IO{
public:
    void In_Data(char *in_data, char *in_contig, int N_zone, i_vector &zone_ID, bool *link_pair,
        d_vector &radius, d_vector &t, co_vector &coord, Edge_vector &link);
    void Out_Coord(char *input_data, char *output_a_b, int N_Zone, i_vector &zone_ID, d_vector &radius,
        co_vector &coord);
    void Out_Precision(char *input_data, char *output_a_b, int N_Link, d_vector &t, d_vector &d,
        Edge_vector &link);
    void Out_Precision_all(char *input_data, char *output_a_b, int N_Zone, d_vector &radius,
        co_vector &coord);
    int In_N_Zone(char *in_data);
private:
    void In_Coord(char *in_data, i_vector &zone_ID, d_vector &radius, co_vector &zone_coord,
        i_map &zone_list);
    void In_Contiguity(char *in_data, int N_zone, bool *link_pair, d_vector &radius, d_vector &t,
        Edge_vector &link, i_map &zone_list);
};
```

CircleAreaCartogram.cpp

```
#include "stdafx.h"
#using <mscorlib.dll>

//--------------メインプログラム--------------
void _tmain()
{
    double alpha=0.98;    // 0<α≦ 1
    double beta=2.5;    //円が重なる場合に与える重み付け
    double eps=0.22;    //収束判定の条件
    char input_data[]="Kanto_in_1990-2000";
    char input_contiguity[]="Kanto_new_zones_list";
    char in_data[50], in_contig[50], output_a_b[40];

    bool *link_pair;
    int i, N_Zone, N_Link, N_Link_max;
    i_vector zone_ID;
    d_vector radius, t, d;
    co_vector zone_coord, dc_coord, link_vec;
    Edge_vector link;
    time_t start, last;
    clock_t clock(void);
    FILE *outfp;
    Circle_IO IO;
```

```
    Circle_Algorithm Algo;

    start=clock();    //計算時間測定開始

    sprintf(in_data, "%s.csv", input_data);
    sprintf(in_contig, "%s.csv", input_contiguity);
    sprintf(output_a_b, " alpha=%.2f beta=%.2f eps=%.2f", alpha, beta, eps);

    N_Zone = IO.In_N_Zone(in_data);    //地域数　入力
    N_Link_max = N_Zone * (N_Zone-1)/2;

    link_pair=new bool[N_Zone*N_Zone];
    fill_n(link_pair, N_Zone*N_Zone, false);

    //データ入力
    IO.In_Data(in_data, in_contig, N_Zone, zone_ID, link_pair, radius, t, zone_coord, link);

    N_Link = int(link.size());

    //最初の点を原点に設定
    for (i=0; i<N_Zone; i++) dc_coord.push_back(zone_coord[i] - zone_coord[0]);

    //リンク長 計算
    d.resize(N_Link);
    link_vec.resize(N_Link);
    Algo.link_distance(d, link_vec, link, dc_coord, N_Link);

    //Circle Area Cartogram 作成計算
    Algo.Circle_Area_Cartogram_algorithm(N_Zone, N_Link_max, link_pair,
        alpha, beta, eps, d, t, radius, link, dc_coord, link_vec);

    delete[] link_pair;

    IO.Out_Coord(input_data, output_a_b, N_Zone, zone_ID, radius, dc_coord);
    IO.Out_Precision(input_data, output_a_b, N_Link, t, d, link);
    IO.Out_Precision_all(input_data, output_a_b, N_Zone, radius, dc_coord);

    //ファイルの出力 計算時間計測
    last=clock();
    printf("clock [sec] = %.3f \n",(float)(last-start)/(float)CLOCKS_PER_SEC);
    if((outfp= fopen("計算時間計測結果.csv","a")) != NULL){
        fprintf(outfp,"%s,%.3f,%.3f,%.3f,%.3f\n",
            in_data,alpha,beta,eps,(float)(last-start)/(float)CLOCKS_PER_SEC);
        fclose(outfp);
    }
    else cout<<"計算時間ファイル出力失敗"<<endl;
}
```

CircleAreaCartogramAlgorithm.cpp

```
#include "stdafx.h"

//--------------サークルエリアカルトグラム作成アルゴリズム--------------
void Circle_Algorithm::Circle_Area_Cartogram_algorithm(int N_Zone, int N_Link_max, bool *link_pair,
    double alpha, double beta, double eps, d_vector &d, d_vector &t, d_vector &radius,
    Edge_vector &link, co_vector &coord, co_vector &link_vec)
{
    const int max_row = N_Link_max, max_work = 4 * N_Link_max, N_Link=int(link.size());
    const int max_element = max_row * (N_Zone-1);
    double max_overlap, *X, *tmp_X, *Y, *work;
    d_vector d_ini(N_Link);
    co_vector coord_ini(N_Zone);

    //縮尺を計算
    Adjust_scale(N_Link, N_Zone, t, d, d_ini, coord, coord_ini);

    X = new double[max_element];
    Y = new double[max_row];
    tmp_X = new double[max_element];
    work = new double[max_work];

    First_Iter(N_Zone, alpha, &max_overlap, link, link_pair, X, tmp_X, Y, work, d, d_ini, t, radius, link_vec,
```

```
        coord, coord_ini);

    while (max_overlap > eps)
        Iter(N_Zone, alpha, beta, &max_overlap, link, link_pair, X, tmp_X, Y, work, d, d_ini, t, radius,
            link_vec, coord, coord_ini);

    delete[] X, Y, tmp_X, work;
}

//--------------縮尺を設定--------------
void Circle_Algorithm::Adjust_scale(int N_Link, int N_Zone, d_vector &t, d_vector &d, d_vector &d_ini,
    co_vector &coord, co_vector &coord_ini)
{
    int i;
    double scale=0.0;

    for (i=0; i<N_Link; i++) scale = max(scale, t[i]/d[i]);
    for (i=0; i<N_Link; i++) d_ini[i] = d[i] * scale;
    for (i=0; i<N_Zone; i++) coord_ini[i] = coord[i] * scale;
}

//--------------第 1 回目計算 接している地域間のみの情報を用いて計算--------------
void Circle_Algorithm::First_Iter(int N_Zone, double alpha, double *max_overlap, Edge_vector &link,
    bool *link_pair, double *X, double *tmp_X, double *Y, double *work, d_vector &d, d_vector &d_ini,
    d_vector &t, d_vector &radius, co_vector &link_vec, co_vector &coord, co_vector &ini_coord)
{
    int N_Link = int(link.size()), N_col = N_Zone - 1, N_work = 2 * N_Link, N_element = N_Link * N_col;
    int i, unity = 1, info;
    char trans_N = 'N';

    make_X_circle(N_Link, N_element, link, X);
    copy(X, X+N_element, tmp_X);

    make_Y_circle(N_Link, d, t, d_ini, link_vec, alpha, Y, 0);
    dgels(&trans_N, &N_Link, &N_col, &unity, tmp_X, &N_Link, Y, &N_Link, work, &N_work, &info);

    coord[0].x = 0.0;
    for (i=0; i<N_Zone-1; i++) coord[i+1].x = Y[i];

    make_Y_circle(N_Link, d, t, d_ini, link_vec, alpha, Y, 1);
    dgels(&trans_N, &N_Link, &N_col, &unity, X, &N_Link, Y, &N_Link, work, &N_work, &info);

    coord[0].y = 0.0;
    for (i=0; i<N_Zone-1; i++) coord[i+1].y = Y[i];

    link_distance(d, link_vec, link, coord, N_Link);

    *max_overlap = Add_OverLapped_Link(N_Zone, link_pair, t, d, d_ini, radius, link, coord,
        ini_coord, link_vec);

    cout<<*max_overlap<<endl;
}

//--------------カルトグラム上で重なる円を計算対象に追加--------------
double Circle_Algorithm::Add_OverLapped_Link(int N_Zone, bool *link_pair, d_vector &t, d_vector &d,
    d_vector &d_ini, d_vector &radius, Edge_vector &link, co_vector &coord, co_vector &ini_coord,
    co_vector &link_vec)
{
    double tmp_d, tmp_r, max_overlap = 0.0;
    coord2D tmp_coord;

    for (int i=0; i<N_Zone-1; i++){
        for (int j=i+1; j<N_Zone; j++){
            tmp_coord = coord[j] - coord[i];
            tmp_d = sqrt_norm(tmp_coord);
            tmp_r = radius[i] + radius[j];
            max_overlap = max(max_overlap, 1 - tmp_d/tmp_r);

            if ((!link_pair[i+j*N_Zone])&&(tmp_d < tmp_r)){
                link.push_back(Edge_Info(i, j));
                t.push_back(tmp_r);
                d.push_back(tmp_d);
                link_vec.push_back(tmp_coord);
                d_ini.push_back(sqrt_norm(ini_coord[j] - ini_coord[i]));
```

```
                link_pair[i+j*N_Zone] = true;
            }
        }
    }
    return max_overlap;
}

//--------------第 2 回目以降計算 カルトグラム上で接する地域間の情報を用いて計算--------------
void Circle_Algorithm::Iter(int N_Zone, double alpha, double beta, double *max_overlap, Edge_vector &link,
    bool *link_pair, double *X, double *tmp_X, double *Y, double *work, d_vector &d, d_vector &d_ini,
    d_vector &t, d_vector &radius, co_vector &link_vec, co_vector &coord, co_vector &ini_coord)
{
    int N_Link=int(link.size()), N_col=N_Zone-1, N_work=2*N_Link, N_element=N_Link*N_col;
    int i, unity=1, info;
    char trans_N='N';

    make_X_circle(N_Link, N_element, link, X);
    adjust_XY_circle(N_Link, link, d, t, d_ini, link_vec, alpha, Y, X, beta);

    copy(X, X+N_element, tmp_X);

    dgels(&trans_N, &N_Link, &N_col, &unity, tmp_X, &N_Link, Y, &N_Link, work, &N_work, &info);

    coord[0].x = 0.0;
    for (i=0; i<N_Zone-1; i++) coord[i+1].x = Y[i];

    adjust_Y_circle(N_Link, d, t, d_ini, link_vec, alpha, Y, beta);
    dgels(&trans_N, &N_Link, &N_col, &unity, X, &N_Link, Y, &N_Link, work, &N_work, &info);

    coord[0].y = 0.0;
    for (i=0; i<N_Zone-1; i++) coord[i+1].y = Y[i];

    link_distance(d, link_vec, link, coord, N_Link);

    *max_overlap = Add_OverLapped_Link(N_Zone, link_pair, t, d, d_ini, radius, link, coord,
        ini_coord, link_vec);

    cout<<*max_overlap<<endl;
}

//--------------説明変数行列の作成--------------
void Circle_Algorithm::make_X_circle(int N_Link, int N_element, Edge_vector &link, double *X)
{
    fill_n(X, N_element, 0.0);
    for (int i=0; i<N_Link; i++){
        if (link[i].V[0]!=0) X[i+(link[i].V[0]-1)*N_Link] = -1.0;
        if (link[i].V[1]!=0) X[i+(link[i].V[1]-1)*N_Link] = 1.0;
    }
}

//--------------被説明変数ベクトルの作成--------------
void Circle_Algorithm::make_Y_circle(int N_Link, d_vector &d, d_vector &d_design, d_vector &d_real,
    co_vector &link_vec, double alpha, double *Y, int ID)
{
    int i;

    //半径の和と実際の距離の scale 倍の加重平均に中心間距離を合わせる
    if (ID==0)
        for (i=0; i<N_Link; i++)
            Y[i] = (alpha * d_design[i] + (1 - alpha) * d_real[i]) * link_vec[i].x / d[i];
    else
        for (i=0; i<N_Link; i++)
            Y[i] = (alpha * d_design[i] + (1 - alpha) * d_real[i]) * link_vec[i].y / d[i];

}

//--------------重なる円に対して重み付け (説明変数行列)--------------
void Circle_Algorithm::adjust_XY_circle(int N_Link, Edge_vector &link, d_vector &d, d_vector &d_design,
    d_vector &d_real, co_vector &link_vec, double alpha, double *Y, double *tmp_X, double beta)
{
    double a_weight;

    for (int i=0; i<N_Link; i++){
        a_weight=1.0;
```

```
        if (d[i] < d_design[i]) a_weight = pow(d_design[i]/d[i], beta);
        //半径の和に中心間距離を合わせる目的関数
        Y[i] = (alpha  * d_design[i] + (1-alpha) * d_real[i]) * a_weight * link_vec[i].x / d[i];
        if (link[i].V[0]!=0) tmp_X[i+(link[i].V[0]-1)*N_Link] *= a_weight;
        if (link[i].V[1]!=0) tmp_X[i+(link[i].V[1]-1)*N_Link] *= a_weight;
    }
}

//--------------重なる円に対して重み付け (被説明変数ベクトル)--------------
void Circle_Algorithm::adjust_Y_circle(int N_Link, d_vector &d, d_vector &d_design, d_vector &d_real,
    co_vector &link_vec, double alpha, double *Y, double beta)
{
    double a_weight;

    for (int i=0; i<N_Link; i++){
        a_weight = 1.0;
        if (d[i] < d_design[i]) a_weight = pow(d_design[i]/d[i], beta);
        Y[i] = (alpha * d_design[i] + (1-alpha) * d_real[i]) * a_weight * link_vec[i].y / d[i];
    }
}

//--------------リンク長 計算--------------
void Circle_Algorithm::link_distance(d_vector &d, co_vector &link_vec, Edge_vector &link, co_vector &coord,
    int N_Link)
{
    for(int i=0; i<N_Link; i++){
        link_vec[i] = coord[link[i].V[1]] - coord[link[i].V[0]];
        d[i] = sqrt_norm(link_vec[i]);
    }
}
```

CircleAreaCartogramIO.cpp

```
#include "stdafx.h"

//--------------ファイル入力 地域数--------------
int Circle_IO::In_N_Zone(char *in_data)
{
    int N_zone = 0, tmp_i;
    double tmp_d;
    FILE *infp;

    if((infp=fopen(in_data, "r")) == NULL){
        cout<<"座標ファイルが開けません"<<endl;
        exit(1);
    }
    while (!feof(infp) && !ferror(infp)){
        fscanf(infp,"%d,%lf,%lf,%lf\n", &tmp_i, &tmp_d, &tmp_d, &tmp_d);
        ++ N_zone;
    }
    fclose(infp);
    return N_zone;
}

//--------------ファイル入力 表現データ・地域代表点座標--------------
void Circle_IO::In_Data(char *in_data, char *in_contig, int N_zone, i_vector &zone_ID, bool *link_pair,
    d_vector &radius, d_vector &t, co_vector &coord, Edge_vector &link)
{
    i_map zone_list;

    In_Coord(in_data, zone_ID, radius, coord, zone_list);
    In_Contiguity(in_contig, N_zone, link_pair, radius, t, link, zone_list);
}

//--------------ファイル入力 地域代表点座標--------------
void Circle_IO::In_Coord(char *in_data, i_vector &zone_ID, d_vector &radius, co_vector &zone_coord,
    i_map &zone_list)
{
    int N_zone=0, tmp_ID;
    double data;
    FILE *infp;
    coord2D tmp;
```

```
    if ((infp = fopen(in_data, "r"))==NULL){
        cout<<"座標ファイル入力エラー"<<endl;
        exit(1);
    }
    while (!feof(infp) && !ferror(infp)){
        fscanf(infp,"%d,%lf,%lf,%lf\n", &tmp_ID, &tmp.x, &tmp.y, &data);
        zone_ID.push_back(tmp_ID);
        radius.push_back(sqrt(data/M_PI));
        zone_coord.push_back(tmp);
        zone_list.insert(pair<int, int>(tmp_ID, N_zone));
        ++ N_zone;
    }
    fclose(infp);
}

//--------------ファイル入力 地域隣接関係--------------
void Circle_IO::In_Contiguity(char *in_data, int N_zone, bool *link_pair, d_vector &radius, d_vector &t,
    Edge_vector &link, i_map &zone_list)
{
    int i, tmp[2];
    FILE *infp;
    i_map::iterator itr;

    if((infp = fopen(in_data, "r"))==NULL){
        cout<<"隣接関係データ入力エラー"<<endl;
        exit(1);
    }

    while (!feof(infp) && !ferror(infp)){
        fscanf(infp,"%d,%d\n",&tmp[0], &tmp[1]);
        for (i=0; i<2; i++){
            itr = zone_list.find(tmp[i]);
            if (itr == zone_list.end()){
                cout<<"ID Error. Check %d"<<tmp[i]<<endl;
                exit(1);
            }
            tmp[i] = itr->second;
        }

        if (tmp[0]==tmp[1]) cout<<"リンクの両端が等しいデータあり"<<cout;
        if (tmp[0] > tmp[1]) swap(tmp[0], tmp[1]);
        if (!link_pair[tmp[0]+tmp[1]*N_zone]){
            link_pair[tmp[0]+tmp[1]*N_zone] = true;
            link.push_back(Edge_Info(tmp[0], tmp[1]));
            t.push_back(radius[tmp[0]]+radius[tmp[1]]);
        }
    }
    fclose(infp);
}

//--------------ファイル出力 円中心座標--------------
void Circle_IO::Out_Coord(char *input_data, char *output_a_b, int N_Zone, i_vector &zone_ID, d_vector &radius,
    co_vector &coord)
{
    char out_data[100];
    FILE *outfp;

    sprintf(out_data, "%s%s_config.csv", input_data, output_a_b);
    if((outfp = fopen(out_data,"w")) != NULL){
        fprintf(outfp,"ゾーン番号,x,y,r \n");
        for(int i=0; i<N_Zone; i++)
            fprintf(outfp,"%d,%.3f,%.3f,%.3f\n",zone_ID[i], coord[i].x, coord[i].y, radius[i]);
        fclose(outfp);
    }
    else cout<<"座標出力ファイルが開けません"<<endl;
}

//--------------ファイル出力 表現精度検証用--------------
void Circle_IO::Out_Precision(char *input_data, char *output_a_b, int N_Link, d_vector &t, d_vector &d,
    Edge_vector &link)
{
    char out_data[100];
    FILE *outfp;
```

```
    sprintf(out_data, "%s%s_precision.csv",input_data, output_a_b);
    if((outfp = fopen(out_data,"w")) != NULL){
        fprintf(outfp,"リンク番号,, ノード番号, 距離\n");
        fprintf(outfp,", 起点, 終点, 目標とする中心間距離, カルトグラム上, カルトグラム上/実際\n");
        for (int i=0; i<N_Link; i++)
            fprintf(outfp,"%d,%d,%d,%.2f,%f,%.10f\n", i, link[i].V[0], link[i].V[1], t[i], d[i], d[i]/t[i]);
        fclose(outfp);
    }
    else cout<<"精度検証用ファイルが開けません"<<endl;
}

//--------------ファイル出力 表現精度検証用 全地点間--------------
void Circle_IO::Out_Precision_all(char *input_data, char *output_a_b, int N_Zone, d_vector &radius,
    co_vector &coord)
{
    double tmp_d;
    char out_data[100];
    FILE *outfp;

    sprintf(out_data, "%s%s_precision_all.csv",input_data, output_a_b);
    if((outfp = fopen(out_data, "w")) != NULL){
        fprintf(outfp,"ゾーン番号,, 距離 \n");
        fprintf(outfp,"起点, 終点, 半径の和, カルトグラム上, カルトグラム上/実際 \n");
        for (int i=0; i<N_Zone-1; i++){
            for (int j=i+1; j<N_Zone; j++){
                tmp_d = sqrt_norm(coord[i] - coord[j]);
                fprintf(outfp,"%d,%d,%.2f,%.2f,%.5f\n",
                    i, j, radius[i]+radius[j], tmp_d, tmp_d/(radius[i]+radius[j]));
            }
        }
        fclose(outfp);
    }
    else cout<<"精度検証用ファイルが開けません"<<endl;
}
```

参考文献

Angel, S. and Hyman, G. M. (1972) "Transformation and Geographic Theory", *Geographical Analysis*, Vol. 4, pp. 350–367.

Anselin, L., Syabri, I., and Kho, Y. (forthcoming) "GeoDa: An Introduction to Spatial Data Analysis", *Geographical Analysis*.

Anselin, L. (2004) *GeoDaTM 0.9.5-i Release Notes*, Spatial Analysis Laboratory, Department of Agricultural and Consumer Economics, University of Illinois, Urbana-Champaign, Urbana, IL, 61801 USA.

Beniger, J. R. and Robyn, D. L. (1978) "Quantitative Graphics in Statistics: a brief history", *The American Statistician*, Vol. 32, No. 1, pp. 1–11.

Canter, D. and Tagg, S. K. (1975) "Distance Estimation in Cities", *Environment and Behavior*, Vol. 7, No. 1, pp. 59–80.

Clark, J. W. (1977) "Time-Distance Transformations of Transportation Networks", *Geographical Analysis*, Vol. 9, No. 2, pp. 195–205.

Cox, T. F. and Cox, M. A. (2001) *Multidimensional Scaling*, Boca Raton: Chapman Hall/CRC.

Cressie, N. A. C. (1993) *Statistics for Spatial Data*, Wiley Series in Probability and Mathematical Statistics: John Wiley & Sons, revised edition.

Dent, B. D. (1999) *Cartography: thematic map design*, Boston: WCB/McGraw-Hill.

Dorling, D. and Openshaw, S. (1992) "Using Computer Animation to Visualize Spacetime Patterns", *Environment and Planning B*, Vol. 19, No. 6, pp. 639–650.

Dorling, D., Johnston, R. J., and Pattie, C. J. (1996) "Using Triangular Graphs for Representing, Exploring and Analysing Electoral Change Using Triangular Graphs", *Environment and Planning A*, Vol. 28, No. 6, pp. 979–998.

Dorling, D. (1991) "Visualisation of Spatial Social Structure", Ph. D. dissertation, University of Newcastle upon Tyne, Newcastle upon Tyne.

——— (1992a) "Stretching Space and Splicing Time: from cartographic animation to interactive visualization", *Cartography and Geographic Information Systems*, Vol. 19, No. 4, pp. 215–227, 267–270.

——— (1992b) "Visualizing People in Space and Time", *Environment and Planning B*, Vol. 19, No. 6, pp. 613–637.

——— (1993) "Map Design for Census Mapping", *The Cartographic Journal*, Vol. 30, pp. 167–183.

——— (1995a) "The Visualization of Local Urban Change across Britain", *Environment and Planning B*, Vol. 22, No. 3, pp. 269–290.

——— (1995b) "Visualizing Changing Social Structure from a Census", *Environment and Planning A*, Vol. 27, No. 3, pp. 353–378.

——— (1996) *Area Cartograms: their use and creation*, No. 59, Norwich: School of Environmental Sciences, University of East Anglia.

——— (1998) "Human Geography - when it is good to map", *Environment and Planning A*, Vol. 30, No. 2, pp. 277–288.

Dougenik, J. A., Christman, N. R., and Niemeyer, D. R. (1985) "An Algorithm to Construct Continuous Area Cartograms", *The Professional Geographer*, Vol. 37, No. 1, pp. 75–81.

Dryden, I. L. and Mardia, K. V. (1998) *Statistical Shape Analysis*, Wiley Series in Probability and Statistics, Chichester: John Wiley & Sons.

Du, C. and Liu, K. (1999) "Constructing Contiguous Area Cartogram Using ArcView Avenue", *Proceedings of Geoinformatics '99 Conference*, pp. 1–7.

Ewing, G. O. and Wolfe, R. (1977) "Surface Feature Interpolation on Two Dimensional Time-Space Maps", *Environment and Planning A*, Vol. 9, pp. 429–437.

Ewing, G. O. (1974) "Multidimensional Scaling and Time-Space Maps", *The Canadian Geographer*, Vol. 18, No. 2, pp. 161–167.

Gastner, M. T. and Newman, M. E. J. (2004) "Diffusion-Based Method for Producing Density-Equalizing Maps", *Proceedings of the National Academy of Sciences of the United States of America*, Vol. 101, No. 20, pp. 7499–7504, May.

Gusein-Zade, S. M. and Tikunov, V. S. (1993) "A New Technique for Constructing Continuous Cartograms", *Cartography and Geographic Information Systems*, Vol. 20, No. 3, pp. 167–173.

Haining, R. P. (1990) *Spatial Data Analysis in the Social and Environmental Sciences*, Wiley Series in Probability and Mathematical Statistics: Cambridge University Press.

House, D. H. and Kocmoud, C. J. (1998) "Continuous Cartogram Construction", *Proceedings of the IEEE Symposium on Information Visualization*, pp. 197–204.

Kadmon, N. and Shlomi, E. (1978) "A Polyfocal Projection for Statistical Surfaces", *The Cartographic Journal*, Vol. 15, No. 1, pp. 36–41.

Keahey, T. A. and Robertson, E. L. (1996) “Techniques fot Non-Linear Magnification Transformations”, *Proceedings of the IEEE Symposium on Information Visualization*, pp. 38–45.

——— (1997) “Nonlinear Magnification Field”, *Proceedings of the IEEE Symposium on Information Visualization*, pp. 51–58.

Keahey, T. A. (1997) “Nonlinear Magnification”, Ph. D. dissertation, Indiana University.

——— (1999a) “Area-Normalized Thematic Views”, *Proceedings of International Cartography Assembly*.

——— (1999b) “Visualization of High-Dimensional Clusters Using Nonlinear Magnification”, in *SPIE Visual Data Exploration and Analysis VI*, Vol. Proceedings of SPIE Visual Data Exploration and Analysis VI.

Keim, D. A., North, S. C., and Panse, C. (2003a) “Medial Axis Cartograms”, in *IEEE Computer Graphics and Applications*.

Keim, D. A., North, S. C., Panse, C., and Schneidewind, J. (2003b) “Visualizing Geographic Information: VisualPoints vs CartoDraw”, *Information Visualization*, Vol. 2, pp. 58–67.

Keim, D. A., North, S. C., and Panse, C. (2004) “CartoDraw: a fast algorithm for generating contiguous cartograms”, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 10, No. 1, pp. 95–110.

Koch, T. and Denike, K. (2001) “GIS Approaches to the Problem of Disease Clusters: a brief commentary”, *Social Science & Medicine*, Vol. 52, No. 11, pp. 1751–1754.

Kocmoud, C. J. and House, D. H. (1998) “A Constraint-Based Approach to Constructing Continuous Cartograms”, *the Eighth International Symposiums on Spatial Data Handling Proceedings*.

Kocmoud, C. J. (1990) “Constructing Continuous Cartograms: a constraint-based approach”, Master’s thesis, Texas A&M University, College Station, Texas, USA.

Kraak, M. J. and Ormeling, F. (2002) *Cartography: visualization of geospatial data*, New York: Pearson Education.

MacEachren, A. M. (1995) *How Maps Work: representation, visualization, and design*, New York: Guilford Press.

Maguire, D. J., Goodchild, M. F., and Rhind, D. W. eds. (1991) *Geographical Information Systems: Principles and Applications*, Cambridge, MA: Longman.（小方登・小長谷一之・碓井照子・酒井高正訳,『GIS 原典ー地理情報システムの原理と方法ー［I]』，古今書院，1998 年）.

Monmonier, M. S. (1977) *Maps, Distortion, and Meaning*, Washington: Association of American Geographers.

——— (1993) *Mapping It Out: expository cartography for the humanities and social sciences*, Chicago Guides to Writing, Editing, and Publishing, Chicago: University of Chicago Press.

——— (1996) *How to Lie with Maps*, Chicago: University of Chicago Press.

Muller, J. (1978) "The Mapping of Travel Time in Edmonton, Alberta", *The Canadian Geographer*, Vol. 22, No. 3, pp. 195–210.

Okabe, A., Boots, B., Sugihara, K., and Chiu, S. N. (2000) *Spatial Tessellations: concepts and applications of voronoi diagrams*, Chichester, New York: Wiley.

Olson, J. M. (1976) "Noncontiguous Area Cartograms", *The Professional Geographer*, Vol. 28, No. 4, pp. 371–380.

Peterson, M. P. (1995) *Interactive and Animated Cartography*, Prentice Hall Series in Geographic Information Science, Englewood Cliffs, N.J: Prentice Hall.

Peterson, I. (2004) "A Better Distorted View: The physics of diffusion offers a new way of generationg maps", *Science News*, Vol. 166, No. 9, p. 136, Aug.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1988) *Numerical Recipes in C*: Cambridge University Press.（丹慶勝市・奥村晴彦・佐藤俊郎・小林誠訳, 『Numerical Recipes in C [日本語版]』, 技術評論社, 1993 年）.

Selvin, S. and Merrill, D. W. (2002) "Adult Leukemia: a spatial analysis", *Epidemiology*, Vol. 13, No. 2, pp. 151–156.

Selvin, S., Merrill, D. W., Schulman, J., Sacks, S., Bedell, L., and Wong, L. (1988) "Transformations of Maps to Investigate Clusters of Disease", *Social Science & Medicine*, Vol. 26, No. 2, pp. 215–221.

Selvin, S., Schulman, J., and Merrill, D. W. (1992) "Distance and Risk Measures for the Analysis of Spatial Data: a study of childhood cancers", *Social Science & Medicine*, Vol. 34, No. 7, pp. 769–777.

——— (1993) "Interpoint Squared Distance as a Measure of Spatial Clustering", *Social Science & Medicine*, Vol. 36, No. 8, pp. 1011–1016.

Selvin, S., Merrill, D. W., Erdmann, C., White, M., and Ragland, K. (1998) "Breast Cancer Detection: maps of 2 San Francisco Bay area counties", *American Journal of Public Health*, Vol. 88, No. 8, pp. 1186–1192.

Sen, A. K. (1976) "On a Class of Map Transformations", *Geographical Analysis*, Vol. 8, No. 1, pp. 23–37.

Shaw, G. M., Selvin, S., Swan, S. H., Merrill, D. W., and Schulman, J. (1988) "An Examination of Three Spatial Disease Clustering Methodologies", *International Journal of Epidemiology*, Vol. 17, No. 4, pp. 913–919.

Shimizu, E. and Inoue, R. (2003) “A Generalized Solution of Time-Space Mapping”, in *Proceedings of the 8th International Conference on Computers in Urban Planning and Urban Management (CD-ROM)*, Sendai.

Shimizu, E. (1992) “Time-space Mapping Based on Topological Transformation of Physical Map”, in *The Sixth World Conference on Transportation Research*, pp. 219–230: Selected Proceedings of the Sixth World Conference on Transportation Research.

Slocum, T. A. (1999) *Thematic Cartography and Visualization*, Upper Saddle River, NJ: Prentice Hall.

Small, C. G. (1996) *The Statistical Theory of Shape*, Springer Series in Statistics, New York: Springer.

Spiekermann, K. and Wegener, M. (1993) “New Time-Space Maps of Europe”, arbeitspapier 132, Institut für Raumplanung, Fachbereich Raumplanung, Universität Dortmund.

——— (1994) “The Shrinking Continent: new time-space maps of Europe”, *Environment and Planning B*, Vol. 21, pp. 653–673.

Srinivasan, V., Nackman, L. R., Tang, J. M., and Meshkat, S. N. (1992) “Automatic Mesh Generation Using the Symmetric Axis Transformation of Polygonal Domains”, *Proceedings of the IEEE*, Vol. 80, No. 9, pp. 1485–1501.

Tobler, W. R. (1961) “Map Transformations of Geographic Space”, Ph. D. dissertation, University of Washington.

——— (1963) “Geographical Area and Map Projections”, *The Geographical Review*, Vol. 53, No. 1, pp. 59–78.

——— (1973) “A Continuous Transformation Useful for Districting”, *Annals of the New York Academy of Sciences*, Vol. 219, No. 9, pp. 215–220.

——— (1979) “Cartograms and Cartosplines”, *Proceedings of 1976 Workshop on Automated Cartography and Epidemiology*, pp. 53–58.

——— (1986) “Pseudo-Cartograms”, *The American Cartographer*, Vol. 13, pp. 43–50.

——— (2004) “Thirty Five Years of Computer Cartograms”, *Annals of the Association of American Geographer*, Vol. 94, No. 1, pp. 58–73.

Torgerson, W. S. (1952) “Multidimensional Scaling: I. theory and method”, *Psychometrika*, Vol. 17, No. 4, pp. 401–419.

Tufte, E. R. (1990) *Envisioning Information*, Cheshire: Graphics Press.

Wackernagel, H. (1998) *Multivariate Geostatistics*, Berlin Heidelberg: Springer-Verlag.（地球統計学研究委員会訳,『地球統計学』，森北出版，2003 年）.

White, M. S. J. and Griffin, P. (1985) “Piecewise Linear Rubber-Sheet Map Transformation”, *The American Cartographer*, Vol. 12, No. 2, pp. 123–131.

Wiebel, R. and Heller, M. (1991) “Digital Terrain Model”, in David J Maguire, Michael F Goodchild, and David W Rhind eds. *Geographical Information Systems: Principles and Applications*, Cambridge, MA: Longman, Chap. 19.

Wood, D. and Fels, J. (1992) *The Power of Maps*, Mappings: Society/Theory/Space, New York: Guilford Press.

Wood, C. and Keller, C. P. (1996) *Cartographic Design: theoretical and practical perspectives*, International Western Geographical Series, Chichester, New York: Wiley.

Yano, K., Nakaya, T., and Kato, H. (2001) “A New Social Atlas of Kyoto Based on Cartogram Systems for Small Area Units”, in *AsiaGIS2001*.

伊藤悟 (1986)「長野県中信地域におけるバス交通ネットワークの時空間構造とその変化－ 1962〜1982 年－」,『金沢大学教育学部紀要（人文科学・社会科学編)』, 第 35 巻, 21–40 頁.

——— (1997a)「時間地図のデイリーリズム－都市の時空間分析に関わる一つの試み－」,『地理情報システム学会講演論文集』, 第 6 号, 275–280 頁.

——— (1997b)『都市の時空間構造－都市のコスモロジー－』, 古今書院, 東京.

——— (2001a)「金沢都市圏の時空間構造分析－時間距離に基づく各種分析測度の適用と比較－」,『金沢大学教育学部紀要（人文科学・社会科学編)』, 第 50 巻, 55–68 頁.

——— (2001b)「時間距離に基づく各種分析測度からみた金沢都市圏構造とその日変化」,『日本地理学会発表要旨集』, 第 59 巻, 172 頁.

井上亮・清水英範 (2005)「連続エリアカルトグラム作成の新手法－ GIS 時代の統計データの視覚化手法－」,『土木学会論文集』, 第 779/IV-66 号, 1 月.

大場亨 (2003)『ArcGIS8 で地域分析入門』, 成文堂, 東京.

岡本耕平 (1983)「名古屋市における認知距離」,『地理学評論』, 第 56 巻, 第 10 号, 695–713 頁.

加藤十吉 (1990)「空間の数学的イメージ　数のイメージと空間のイメージ」,『数理科学』, 第 35 巻, 21–40 頁.

菅野峰明 (1987)「地図」, 中村和郎・高橋伸夫・菅野峰明・斎藤功・杉浦芳夫・手塚章・野上道男・三上岳彦（編)『地理的情報の分析手法』, 古今書院, 東京, 1–68 頁.

岸本一男 (1978)「領域の最適三角形群への分割アルゴリズム」,『情報処理』, 第 19 巻, 第 3 号, 211–218 頁.

高阪宏行 (2002)『地理情報技術ハンドブック』, 朝倉書店, 東京.

古藤浩 (1995)「時間距離網による都市連関構造の視覚化」,『第 30 回日本都市計画学会学術研究論文集』, 553–558 頁.

——— (1997)「地域構造と視覚化時間距離網」,『GIS －理論と応用』, 第 5 巻, 第 2 号, 1–10 頁.

——— (2000a)「いくつかの新しい時間地図」,『日本オペレーションズ・リサーチ学会秋期研究発表会アブストラクト集 2000』, 72–73 頁.

——— (2000b)「位相構造を保持する時間地図」,『日本オペレーションズ・リサーチ学会春期研究発表会アブストラクト集 2000』, 134–135 頁.

佐藤甚次郎 (1971)『統計図表と分布図』, 古今書院, 東京.

清水英範 (1992)「時間地図の作成手法と応用可能性」,『土木計画学研究・論文集』, 第 10 号, 15–29 頁.

清水英範・井上亮 (2004)「時間地図作成問題の汎用解法」,『土木学会論文集』, 第 765/IV-64 号, 105–114 頁, 7 月.

水津一朗 (1978)「行動空間とトポロジー－位相地理学試論－」,『人文地理』, 第 30 巻, 第 1 号, 1–16 頁.

杉浦芳夫 (1980)「多次元尺度構成法 (MDS) による空間分析とその拡散問題への応用」,『地理学評論』, 第 53 巻, 第 10 号, 617–635 頁.

杉原厚吉 (1989)「計算幾何学的手法と画像解析－ボロノイ図の応用を中心として－」,『情報処理』, 第 30 巻, 第 9 号, 1067–1075 頁.

谷口健男 (1992)『FEM のための要素自動分割デローニー三角分割法の利用』, 森北出版, 東京.

譚学厚・平田富夫 (2001)『計算幾何学入門－幾何アルゴリズムとその応用－』, 森北出版, 東京.

張長平 (2001)『地理情報システムを用いた空間データ分析』, 古今書院, 東京.

徳田良仁 (1990)「2 次元平面に描かれた絵はなぜ 3 次元になるか?」,『数理科学』, 第 327 号, 78–79 頁.

中岡良司・五十嵐日出夫・森弘 (1992)「北海道における時間距離図の歴史的変遷に関する研究」,『土木史研究』, 第 12 巻, 53–64 頁.

中岡良司・今尚之・千葉博正・佐藤馨一 (1995)「鉄道網整備による全国主要都市の相対的時間距離の変遷に関する研究」,『第 30 回日本都市計画学会学術研究論文集』, 589–594 頁.

中川篤史 (1998)「カルトグラムの自動作成手法に関する研究」, 修士論文, 東京大学大学院工学系研究科 都市工学専攻.

藤田宏・今野宏・田辺國士 (1994)『最適化法』, 岩波講座 応用数学方法 7, 岩波書店, 東京.

藤目節夫 (1999) 「時間・費用距離から見た中四国地域の自動車交通空間の変化」,『地理学評論』, 第 72A 巻, 第 4 号, 227–241 頁.

星仰・堀勝也 (1998) 『地理情報システム用語辞典』, 朝日出版社, 東京.

桝谷有三・浦田康滋・田村亨・斎藤和夫 (1997) 「北海道の高規格幹線道路網を対象とした時間距離行列の視覚化」,『高速道路と自動車』, 第 40 巻, 第 6 号, 32–40 頁.

桝谷有三・田村亨・斉藤和夫 (1995) 「道路網を対象とした時間距離行列の視覚化」,『土木計画学研究・論文集』, 第 12 巻, 567–574 頁.

間瀬茂・武田純 (2001) 『空間データモデリング—空間統計学の応用』, データサイエンス・シリーズ, 共立出版.

三好哲也・市橋秀友・黒田由香 (1999) 「スプライン関数とフリーネットワーク解法を用いる時間地図作成法」,『電子情報通信学会論文誌』, 第 J82-D-II 巻, 第 8 号, 1299–1304 頁.

山下英生・中田健二・中前栄八郎 (1981) 「2 次三角形要素による有限要素解析結果の表示—面積座標による—」,『情報処理学会論文誌』, 第 22 巻, 第 2 号, 106–113 頁.

山下英生・中前英八郎 (1980) 「有限要素解析のための三角形要素自動作成の一手法」,『情報処理学会論文誌』, 第 21 巻, 第 3 号, 183–190 頁.

吉本剛典 (1980) 「全国都市間時間距離の地図化」,『地理科学』, 第 34 巻, 46–47 頁.

——— (1981) 「全国主要都市間時間距離の地図化の試み」,『地理学評論』, 第 54 巻, 第 11 号, 605–620 頁.

——— (1982) 「再現された時空間における次元性の検討」,『地理科学』, 第 37 巻, 第 1 号, 25–41 頁.

若林芳樹 (1987) 「時間・空間における広島都市圏の因子生態分析」,『地理学評論』, 第 60A 巻, 第 7 号, 431–454 頁.

——— (1989) 「認知地図の歪みに関する計量的分析」,『情報処理』, 第 62A 巻, 第 5 号, 339–358 頁.

——— (1990) 「札幌における認知地図の相対的歪み」,『地理学評論』, 第 63A 巻, 第 4 号, 255–273 頁.

——— (1999) 『認知地図の空間分析』, 地人書房, 京都.

レリトードニ (1998) 「有限要素解析のためのアダプティブメッシュ生成法」, 修士論文, 東京大学大学院工学系研究科 計数工学専攻.

謝辞

長いようで短かった 3 年間の課程の成果として論文を纏めることができたのは，私の周りの多くの方々による御指導・御助力の御蔭です．ここで，全ての方の御名前を挙げることはできませんが，感謝の意を表します．

まず，指導教官であり本論文の主査である，清水英範教授には大変お世話になりました．修士 2 年の春，進路を決める折に，進学・研究者への道を勧めて頂くことがなければ現在論文を執筆していることもなかったと思うと，感謝の意を尽くすことはできません．また，本論文のカルトグラム作成手法に関する研究は，先生の時間地図研究に端を発するものです．また，測量における誤差配分法を応用したディスタンスカルトグラム作成手法は先生の御指導がなければ構築することは不可能であり，その後の研究も進展しなかったと思います．

また，昨年度本大学を御退官になられた森地茂教授には，研究室ゼミの折り等に私の思い至らない点についてご指摘を頂きました．また，カルトグラムによる視覚化の有効性を認めて頂き，各所でカルトグラムを利用した視覚化について御紹介して頂きました．ありがとうございました．

副査の岡部篤行教授，浅見 泰司教授，貞広幸雄助教授，堀田昌英助教授には，カルトグラム研究に対して御指導・御鞭撻を賜りましてありがとうございました．

また，地域/情報研究室に在籍しておられた他の先生方にもお世話になりました．筑波大学の堤盛人助教授には，作成手法について詳細な御指摘を頂くとともに，発表の場を御紹介頂くなどお世話になりました．清水哲夫助教授には，研究室ゼミにおいて色々御指導頂きました．布施孝志助手には，研究面はもちろん，私の身近な研究者の先輩として公私に渡って御世話になりました．

本論文で使用したデータに関して，TRANET の鉄道所要時間データは，国土交通省の吉田忠司様，財団法人計量計画研究所の毛利雄一様，萩野保克様，名倉俊明様，株式会社ライテックの坂本隆様，NAVINET の道路所要時間データは，国土交通省の戸谷有一様，藪雅行様，山田拓也様のご協力を頂きました．

カルトグラム作成ツールの開発では，株式会社パスコゲオグラフィアの武井克之様，長沢一也様，小室真里様に大変お世話になりました．私の力だけでは，操作性の優れたユーザーインターフェースを持ったツールの作成はできませんでした．

また，地域/情報研究室 (旧 測量/地域計画研究室) および交通地域ラボの皆様にもお世話になりました．その中でも特に，米澤明男君，Mongkol Tawechaitosapol 君，野本卓也君には，カルトグラム作成計算に関していろいろ作業をして頂きありがとうございました．また，修士課程以来 5 年間有意義な研究室生活を送ることができました．

最後に，28 年間未熟な私を支えてくれた両親に感謝の意を表したいと思います．

平成 16 年 12 月

井上　亮