画像処理のためのソフトウェア : (5) 画像中のネットワーク構造上の流れ解析

中野 司*・藤井 直之**

Softwares for Digital Image Processing : (5) Network Flow Analyzer

Tsukasa NAKANO* and Naoyuki FUJII**

Abstract : We explain a procedure to analyze the flow through the network structure on the digital image. The network analysis described in this paper is as follows. First we construct the Pore-Channel (PC) model using the tessellating triangles of the network area. The tessellation procedure was previously explained in Nakano and Fujii (1991b). For the flow percolating through the network, we calculate the resistance due to channel geometry, and construct the resistor network model. Calculating the net resistance of the network, we can obtain the macroscopic permeability and/or electric conductivity of the network structure on the image data. We also explain the computer programs for the above procedure.

Key Words : Image processing, Network analysis, PC model, Resistor network, Permeability, Conductivity

1. はじめに

画像データに含まれる鉱物粒界、割れ目模様、河川や道路網などの「ネットワーク構造」を伝った流体、電気、自動車などの流れに対するネットワーク全体の巨視的な性質（浸透率、巨視的な電気伝導率など）を実際に流れを観測せずに画像データだけから見積る場合、一般に以下のような一連の処理が順番に行われる（第1図; Koplik *et al.*, 1984）。

第1図に示したように、まず最初に原画像を二値化してネットワークをなしている部分（ネットワーク領域）の輪郭線を抽出する（第1図(a)から第1図(b)）。ただしここでいう二値化とは、もともとなった濃淡画像を単に黑白画像へと変換することだけを意味するのではなく、ネットワーク領域とそれ以外の部分を分離してその輪郭線を抽出するためのあらゆる手法（簡単な例として空間的なフィルタリング処理など）を含む。このようにして得られたネット

ワーク領域の輪郭線から、普通は第1図(c)のような Pore（溜り）と Channel（流路）からなる Pore-Channel (PC) モデルを想起することが多い。PC モデルは「ネットワーク」という言葉に対応して思い浮かべるポピュラーなものではあるが、その定義はかなり曖昧で、定量的な取り扱いがむずかしい。このモデリングは電気回路とのアナロジーで、「Pore = コンデンサー」、「Channel = 抵抗」という直感的な対応にしたがって（人間の主観で）なされている場合が多い。

さて、このような Pore や Channel を伝った流体などの流れに対するネットワーク全体の巨視的な性質を議論する場合、普通は流れが時間的に変化しない定常状態だけを考える。このとき、PC モデルで Pore と Channel の名前の由来からもわかるように、Pore 領域が流れにおよぼす影響は無視できる。つまり、先のアナロジーでいうと、定常状態では Pore = コンデンサーは飽和していて、そこに蓄えられる電荷の量は時間的に変化しないと考える。そこで、点と見なした Pore（領域）を結ぶそれぞれの Channel 領域を抵抗で置き換えた、PC モデルとトポロジカルな意味で等価な抵抗網（モデル）を用いてネットワーク上での流れを解析する（第1図(d)）。ただし、ここで置き換えたそれぞれの抵抗の値は、PC モデルの対応する Channel 領域の形状や流れる流体の性質（粘性や電気伝導度；これらは普通は単位の値を持つと考えて処理を行う）によって決められる。抵抗網全体としての抵抗値は、例え

* 神戸大学自然科学研究科。現在、地質調査所。Graduate School of Science and Technology, Kobe University, Nada, Kobe 657, Japan. Now at Geological Survey of Japan, 1-1-3, Higashi, Tsukuba 305, Japan.

** 神戸大学理学部。現在、名古屋大学理学部。Faculty of Sciences, Kobe University, Nada, Kobe 657, Japan. Now at Faculty of Sciences, Nagoya University, Chikusa, Nagoya 464, Japan.

第1図 ネットワーク解析の概念図 (Koplik *et al.*, 1984を改変).

ば Kirchhoff の法則を用いて計算することができる。

もし抵抗網に含まれる抵抗の個数が非常に多ければ、抵抗の両端のポテンシャル（水圧、電圧など）や抵抗を流れる流量をすべて求めるために非常に長い計算時間が必要になる。そこで、繰り込み群の手法や有効媒質近似 (Effective Medium (EM) approximation) などを利用して抵抗網全体の性質を推定する場合も多い (第1図(e); 有効媒質近似した抵抗網だけを示した)。前者は、抵抗網の一部をそれと等価な「超抵抗」で置き換えることを繰り返して抵抗網全体の抵抗を求める方法 (高安, 1986) であり、後者は、抵抗値の出現頻度分布からすべての抵抗が同一の「有効な」抵抗値を持つとしたときの抵抗網全体の抵抗値をセルフコンシステントに計算する近似法である (Kirkpatrick, 1973)。

最終的には、以上のようにして見積られた抵抗網全体の抵抗値を抵抗網の空間サイズなどで規格化する。もとの画像データに映された物体がネットワーク領域とそれ以外が融合した連続体であると見なした場合、ここで得られた値が (たとえば Darcy の法則や Archie の法則などに表れる) 巨視的な浸透率や電気伝導率となる。

我々はこれまで、以上のような画像データをもとにした「ネットワーク解析」を行うためのソフトウェア群を開発してきた (中野・藤井, 1989a, 1991a, 1991b)。特に中野・藤井 (1991b) では、上に述べた (曖昧ではあるが直感的な) PC モデルを人間の手を借りずに画像データから抽出するための、画像中のネットワーク領域の三角形分割を行うためのソフトウェアについて詳しく述べた。本文では、それに続いて、残りの解析のステップそれぞれを行うためのアルゴリズムについて詳しく述べ、また、それらを実際に応用するためのソフトウェア群について説明する。

2. 原理

以下では、与えられた画像データに含まれるネットワーク領域とそうでない部分は適当な方法 (中野・藤井, 1991a) で二値化され (第2図(a)), すでにその輪郭線が抽出されているものとする (第2図(b))。また、中野・藤井 (1991b) で述べられているアルゴリズムにもとづいて、輪郭線に囲まれたネットワーク領域の三角形分割 (第3図) がなされているものとする。

第2図(a) 二値化画像。もともとなったデータは、鉄鉱石焼結体の X 線 CT 画像 (新日本製鉄株式会社, 1989)。

(b) 上図でネットワークをなしているように見える黒い部分 (ネットワーク領域; 焼結した岩石) の輪郭線を抽出したものの。

2.1. 三角形分割と Pore-Channel (PC) モデル

たとえば第3図に示した三角形分割したネットワーク領域の一部を拡大して見ると第4図のようになっている。多数の正方形画素からなる画像データに含まれるネットワーク領域の輪郭線は、第4図の二重線で示したようなデジタルな折れ線である。第4図の点線で示した分割三角形の辺は、ネットワーク領域を隔てて向かい合う輪郭線をなす点 (輪郭線をなす折れ線の端点; 第4図の白丸) のうち一番近い2点を結ぶことを行って得たものである。中野・藤井 (1991b) でも詳しく説明されているように、これらの辺を垂直二等分する線分を結んだ分割三角形と双対な図形

第3図 第2図のネットワーク領域を三角形分割したものの。

第5図 第3図の分割三角形の外心（節点）を結んだスケルトン。

は、輪郭線をなす点を母点とする Voronoi 図形に相当し、それをネットワーク領域の「スケルトン（骨格線）」と見なすことができる。第4図や第5図（第2図、第3図のネットワーク領域に対応している）でも明らかなように、スケルトンはネットワーク領域のトポロジカルな性質（ネットワークのつながりぐあい）だけを抽出した図形である。スケルトンをなす線分のことをスケルトンの「枝」と呼ぶことにする。このとき、それらの交点（第4図の黒丸）は分割三角形と一対一対応していて、対応する三角形の外接円の中心（外心）になっている。このようなスケルトンの枝の交点（もしくはネットワーク領域を分割した三角形の外心）のことを、以下ではネットワークの「節点」と呼ぶことにする。

ネットワークの節点で交わるスケルトンの線分の本数はその節点の「配位数」と呼ばれている。ネットワークの各節点の配位数とその節点に対応する分割三角形の2つの情報をあわせると、画像データから人間の手をかりずに PC モデルを得ることができる。つまり、配位数が 3 以上の節点に対応する分割三角形を Pore 領域、配位数が 2 の

ひとつながりの節点に対応する分割三角形を Channel 領域と見なすことによって、三角形分割されたネットワーク領域を PC モデルに変換することができる。なお、Pore にも Channel 領域にも対応しない配位数が 1 の節点はネットワークの「端点」と呼んで、後の解析で使用する。

ところで、たとえば正方形を対角線で区切って三角形分割したときのように、隣接する 2 つの分割三角形が同じ位置に外心を持つ場合がある。このような 2 つ以上の分割三角形の節点と同じ位置を占める場合は、それらの間のスケルトンの枝の長さが 0 であることになってしまう。これを避けるためには、同一の位置にある分割三角形の節点を「融合」して 1 つの節点であると見なし、またそれと同時に、融合した節点の配位数を調節しなければならない。すなわち、ネットワークの分割三角形情報から PC モデルを作成する場合には、このようにして得られた調整ずみの節点の配位数を用いる必要がある。

2.2.枝刈り

第5図からも明らかなように、ネットワーク領域を分割した三角形の外心を結んだだけのスケルトンには、多数の髭状の短い枝が付随している。これは、スケルトンを得る際にもとになったネットワーク領域の輪郭線がなめらかな曲線でなくデジタルな折れ線であるために生じたものである（スケルトンは輪郭線に囲まれたネットワーク領域の「中央」を通る図形である、としているので、輪郭線の微小な凹凸に対応する髭ができてしまう）。また、人間の目には明らかなのだが、得られたスケルトンに対応する配線の電気回路に電流を流した場合に電流が流れない枝（Channel 領域）もいくつか存在する。このような髭や枝は、PC モデルもしくはスケルトンから作成した抵抗網の全体の抵抗値を求める場合に計算には差し支えないが、その手間を著しく増大させる。そこでこれを減らすために、節点の配位数の情報を利用して（ネットワークの端点から

第4図 ネットワーク領域の分割三角形とスケルトン（骨格線）の関係（詳細は本文を参照のこと）。

始めて) ネットワークの枝刈りを行う。

枝刈りの処理は、前節の最後に述べた節点の融合にともなう節点の配位数の調節の処理とあわせて、具体的には以下のような手順で行う。まず、もとの分割三角形から得られたスケルトン(第5図)のすべての節点の配位数を調べる。そして、隣接する2個の三角形に対応する節点(外心)が同じ位置を占めるか否かを調べ、もし同じ位置にあるなら、それら2つの節点を結ぶスケルトンの枝を取り除いて1つの節点に融合する。具体的には、2節点のうち一方を以下の解析で扱わないように取り除いて、同時に、残りの節点の配位数を調節する。次に、このようにして得た節点から配位数が1のネットワークの端点にあたる節点を少なくとも2個を選び出し、それらをネットワークに電流や流体を流すときの「流入口」と「流出口」とする。その後、すべての節点をそれらが登録されている順番でそれぞれ逆の向きに2回走査して、各パスで一方の端が流入口と流出口以外のネットワークの端点であるスケルトンの枝を取り除いては、その他方の端の(配位数2以上の)節点の配位数を調節する(配位数を1だけ減らす)ことを繰り返す。

上の枝刈りの処理で、ネットワークの流入口と流出口の選択のしかたは任意である。本文では、節点の中から x , y 座標値がそれぞれ最小, 最大なもの4個を捜し出し、これらの座標値から計算した x および y 方向のネットワーク領域の幅のうち広い方の方向に対応する2個の節点を流入口と流出口と見なしている。

以上の枝刈り処理を第5図のデータに対して行って得たスケルトンを第6図に示す。また、このスケルトンに対応する Pore 領域を第7図に示した。なお、これらの図で黒丸で示したものがネットワークの流入口と流出口である。

2.3. チャンネルの抵抗値

枝刈り処理を行った後、得られたスケルトンもしくは

第6図 第5図のスケルトンから余分な枝を取り除いたもの。

第7図 第6図のネットワークの節点(Pore)に対応する分割三角形。

PC モデルから抵抗網モデルを作成する。その際、配位数3以上の節点(以下では単に「Pore」と呼ぶ)には通し番号(Pore 番号)をつけておく。また、配位数が2の節点とそれに対応する三角形は、ひとつながりのものをそれぞれまとめて「Channel」と呼ぶことにする。そして、各Channelには、それに関係する三角形の面積の和 S 、スケルトンの枝の長さの和(軸長) l 、およびその両端のPore 番号(もしくはネットワークの流入口, 流出口の名前)を付随させるようにする。

さて、抵抗網モデルの各々の抵抗値をChannelの形状を考慮して決めるためには、ネットワークを伝って流れる流体の流れとポテンシャル(圧力, 電圧)場の関係式が必要である。本文では、流れとして低レイノルズ数の粘性流体の流れと電流の場合だけを考える。このとき、流体が第8図のような2つのPore P_1 と P_2 をつなぐChannelを流れるとき、 P_1 から P_2 への流量(粘性流体の場合は体積流量, 電気の場合は電流) q_{12} は、

$$q_{12} = -a \cdot b \cdot C_{12} \cdot \Delta p_{12} \quad (1)$$

と表わすことができる。ここで、 Δp_{12} は P_1 と P_2 の間のポテンシャルの差である。また、 a と b はネットワークに含まれるすべての抵抗に対して同じ値をとる定数で、前者は流体の物性(粘性, 電気伝導度)に関連した定数を、後者はそうでないものを表わしている。これらは流体の粘性を μ 、電気伝導度を σ とすると、具体的には以下になる。

第8図 Pore と Channel の概念図。

$$a = \begin{cases} 1/\mu & (\text{粘性流体の場合}) \\ \sigma & (\text{電流の場合}) \end{cases} \quad (2)$$

$$b = \begin{cases} 1/12 & (\text{粘性流体の場合}) \\ 1 & (\text{電流の場合}) \end{cases} \quad (3)$$

ただし、粘性流体に対する b は、低レイノルズ数の放物型速度プロファイルを仮定した場合の値である（詳細は適当な流体力学の教科書、たとえば、Landau and Lifshitz, 1970を参照のこと）。なお、解析終了後にこれら定数の分は容易に補正できるので、本文では a , b がどちらも 1 であるとして話を進める。

(1) 式で、 C_{12} が第 8 図のような Channel を流れる流体の受ける抵抗（もしくは流れに対する Channel 内の流体の伝導度）のうち Channel の形状に依存する部分である。これは具体的には以下のように表わすことができる。

$$C_{12} = \begin{cases} w^3/l & (\text{粘性の流体の場合}) \\ w/l & (\text{電流の場合}) \end{cases} \quad (4)$$

ここで、 w は Channel の平均的な幅（第 8 図）であり、本文では、Channel の面積 S と軸長 l から、

$$w = S/l \quad (5)$$

のようにして計算した値を用いることにする。

2.4. 抵抗網上でのポテンシャル分布と巨視的な浸透率、電気伝導度

はじめに述べたとおり、抵抗網の全体としての抵抗値は前節で得られた抵抗の値の分布から有効媒質近似などで求めることもできるが、本文では、Kirchhoff の法則を用いて実際にすべての Pore でのポテンシャルの値を計算してそれを求めることにする。

これを行うためには、ネットワークの流入口と流出口に適当なポテンシャル差を与えて、Pore 番号 j の Pore に対して (1) 式と Kirchhoff の電流則、

$$\sum_i q_{ij} = 0 \quad (6)$$

を適応する。ただし、(6) 式の和は Pore j に対してそれにつながっているすべての Pore i との間の Channel を伝った流量 q_{ij} についてとる。すべての Pore に対する (6) 式を連立させて解けば、各 Pore での流入口もしくは流出口に対する相対的なポテンシャル値、それぞれの

Channel を伝った流れ、およびネットワークへ流出入する流れ Q が得られる。

これまで例としてあげてきたネットワーク（第 2, 3, 5, 7 図）に、粘性流体もしくは電流が流れたとして計算したポテンシャル場の様子を第 9 図 (a), (b) に示した。なお、

第 9 図 第 2 図(a)のネットワーク領域の両端(黒丸)にポテンシャル差(圧力差, 電位差)を与えて流体を流した場合のポテンシャル場の様子. 等高線はネットワーク両端に与えたポテンシャル差の値の 1/20 ごとに引いてある. (a) 流体が粘性流体の場合. (b) 電流を流した場合.

第10図 ネットワークを含む媒質全体の巨視的な電気伝導度、浸透率の見積りの概念図。

これら図では、上の計算で得られた Pore (第9図のX印)でのポテンシャルの値をもとにして空間的な補間(中野・藤井, 1989b)を行った後に等高線を描いてある(中野・藤井, 1989c)。したがって、もとのネットワーク領域(第2図)ではない部分に対して描かれているポテンシャル場の様子には信頼性はない。

以上のようなネットワーク解析の結果を用いて、画像データのもとになった岩石サンプルなどの巨視的な流体の浸透率や電気伝導度を求める方法は以下のとおりである。ただし、浸透率、電気伝導度のどちらもほぼ同じ手順で求めることができるので、ここでは浸透率の場合だけを述べる。画像データのもとになった岩石サンプルに対して浸透流における Darcy 法則が適用できるならば、その(巨視的な)浸透率 K は、

$$K = \frac{\mu \cdot D}{\Delta P} \frac{Q}{A} \quad (7)$$

と表わされる。ここで、 Q は岩石を通り抜ける巨視的な流れの総体積流量、 A と D はそれぞれ流れの方向に対する岩石の幅と奥行きである。また、 ΔP は流れを駆動する奥行き方向の圧力差である(つまり $\Delta P/D$ が流れを駆動する圧力勾配)。これを、本文のネットワーク(抵抗網)モデルにあてはめると第10図のようになる。(7)式に表れる流体の粘性 μ は、 Q に含まれるもの(2式)とあわせて消去できる。また ΔP は、ネットワークの流入口と流出口の間のポテンシャル差と見なせばよい。このようにして、(7)式のすべての量はここまで行ってきた画像データのネットワーク解析の結果から推定することができる。

3. プログラムの説明と使い方

上で説明した原理にもとづいたネットワーク解析を行うためのプログラム群のC言語で書かれたソースリストを付録に示した。これらは、MS-DOS上のTurbo-C、およびUNIX-Cでコンパイルして正常に動作することを確認してある。ただし、UNIX-Cでは、コンパイル時にソースリスト中のincludeファイルの一部が存在しないというエラーが起こる可能性があるため、その場合は対応するinclude宣言を取り除く必要がある。また、MS-DOSの走るi8086系のコンピュータでTurbo-Cなどのコンパイラにかけける場合には、コンパクトもしくはラージメモリモデルを指定する必要がある。なお、以下で説明するように、これらのソースリストから5個の独立なプログラムが作成できる。これらのプログラムのメインルーチンを含むソースリストの先頭には、UNIX-Cでコンパイルする方法(およびコンパイルに必要なモジュール名)をコメントしておいた。UNIX-C以外のコンパイラを使う場合も、

コンパイラの名前を除けばほぼ同様な指定でコンパイルできるはずである。

本文で紹介するプログラムはどれも、オペレーションシステム(UNIX, MS-DOS)のコマンドラインから処理の際に必要なファイル名などのパラメータを引き数として与えて起動するようになっている(会話的なパラメータの入力はない)。また、得られた結果はすべて標準出力(普通はコンソール)に書き出すようになっているので、それをリダイレクトして適当なファイル名でセーブすることを起動時に指定する必要がある。

3.1. prune

付録にそのメインモジュールのソースリスト"prune.c"を示したプログラムpruneでは、中野・藤井(1991b)で紹介したプログラムtc2によって得られたネットワーク領域の三角形分割の情報を用いて、ネットワークの節点での配位数情報の調査、ネットワークの流入口と流出口の決定、スケルトンの枝刈りやPore, Channelの幾何学的情報の抽出、Poreに対応する分割三角形のグラフィック表示など様々な処理を行う。

ソースリストを解読すればわかるように、ここで行っている処理の大部分はスケルトンをなす節点のつながりを調節する「リスト処理」である。pruneでは、スケルトンの節点の配位数を格納するために用意した配列を使って、節点の融合や枝刈りの巧妙なリスト処理を行っている(たとえば、関数Linkの処理でもわかるように、ある節点に対応する配位数格納用の配列要素の値が負であれば、その節点は別の節点と融合していて、その配列要素の値の絶対値が融合先の節点の名前になるようにしてある)。pruneで行われている処理内容についてのより詳しい説明は省略する。各自でソースリストを解読して頂きたい。また、pruneで利用しているグラフィックスシステム"gg2"のサブルーチンの詳細はすでに説明している(たとえば、中野・藤井, 1991a)ので、ここでは触れない。なお、pruneをコンパイルする際に必要なモジュール"network.c"と"network.h"のソースリストは、すでに中野・藤井(1991b)で示しているので本文では割愛した。

pruneは以下のように指定して走らせる。

```
prune TriangleFile { CoordinationNumber }
```

ここで「TriangleFile」はプログラムtc2(中野・藤井, 1991b)で得られたネットワーク領域の三角形分割情報の入ったファイル(そのフォーマットについては後で説明する)の名前である。また、中括弧で囲んだもの(CoordinationNumber)はオプションで、2以上の整数値

を指定する。この指定がなければ配位数が 3 以上の Pore 領域に対応する三角形が、また、そうでないときは、配位数がここで指定された値より大きい節点に対応する分割三角形だけがグラフィック表示される。

3.2. pcnet, netsolve と netdiv

pcnet は、プログラム prune で得られたそれぞれの Channel の形状（軸長と面積）からそれに対応する抵抗網モデルの抵抗値を計算するためのプログラムである。ただし、実際には pcnet という実行可能なオブジェクトファイルは存在しない（作成しない）。付録に示したソースリスト “pcnet.c” の先頭でもコメントしているように、コンパイル時にサブルーチンモジュール “vfc.c”（粘性流体用）と “ecc.c”（電流用）を使い分けて、1つのメインモジュールからネットワークを流れる流体の種類に応じた2つのオブジェクトファイル、pcnvf（粘性流体用）と pcnec（電流用）を作成できるようにしてある。

pcnvf および pcnec の起動の仕方はどちらも同じで、

```
pcnvf PoreChannelFile
pcnec PoreChannelFile
```

のようにする。ここで、引き数の「PoreChannelFile」は、プログラム prune で得た Pore と Channel の情報の入ったファイルの名前である。なお、同じソースリストから生成されたプログラムであるため、pcnvf, pcnec の出力データのフォーマット（後述）はまったく同じである。

こうして得られた抵抗網モデルのすべての Pore でのポテンシャル、すべての Channel を伝った流れを計算するためのプログラムが netsolve である。netsolve ではネットワークの流入口と流出口に単位ポテンシャル差を与えて、(1)式と(6)式を組み合わせたそれぞれの Pore でのポテンシャルに関する連立方程式を掃き出し法で解いている。

netsolve の起動の仕方は、以下のとおりである。

```
netsolve ResistorFile
```

引き数「ResistorFile」で、pcnvf もしくは pcnec で求めた抵抗網モデルの抵抗値の情報の入ったファイルの名前を指定する。

抵抗網モデルの Pore でのポテンシャル値と Channel を伝った流れがすべて得られた後に、指定した Pore での（ネットワークの流出口に相対的な）ポテンシャルの値を調べたり、また、そこでの流れの湧き出し（divergence）を計算するために、ソースリスト “netdiv.c” で示したプロ

グラム、netdiv を用意した。このプログラムはポテンシャル値を調べる以外に、(6) 式で示される Pore での流れの湧き出しが 0 という条件をもとにして、得られた解の数値的な誤差を評価するために用いることができる。また、ネットワークの流入口と流出口を特別な（配位数 1 の）Pore だと見なしているの、それらに流出入する流れの総量 Q （第10図）を調べるために使うこともできる。

netdiv は以下のようにして使用する。

```
netdiv PotentialFile PoreName
```

ここで、「PotentialFile」はプログラム netsolve が出力したネットワークのすべての Pore でのポテンシャル情報などが入ったファイル（後述）の名前で、「PoreName」はポテンシャル値を調べたり湧き出しの値を計算したい Pore の番号である。ただし、Pore 番号として、ネットワークの流入口と流出口はそれぞれ 1 と 2 を、また、それ以外の普通の Pore は 3 以上の値（後述）を指定する。

このようにして起動した後、netdiv が標準出力に書き出す情報は以下のとおりである。

```
Pore 番号
Pore の x 座標値 と y 座標値
ポテンシャル値
配位数
湧き出し量
```

ただし、Pore の x, y 座標値はその Pore に対応する節点、すなわち、分割三角形の外心の位置の座標値である。

4. データファイルの構造

4.1. ネットワーク領域の三角形分割データファイル

プログラム prune が取り扱うネットワーク領域の三角形分割の情報の入ったファイルの構造についてはすでに中野・藤井（1991b）でも説明しているが、ここでもう一度説明しておく。その構造は以下のようになっている。

```
もとの画像の縦横の画素数
ネットワーク領域の輪郭線の総数 (P)
輪郭線上の点の総数 (N, N)
輪郭線上の点の座標値 (i = 1 ..... N)
    点の x 座標と y 座標 (整数値)
分割三角形の辺の総数 (M)
分割三角形の辺の情報 (j = 1 ..... M)
    辺の始点の名前 (Tj)
    辺の終点の名前 (Hj)
```

辺の左側の点もしくは辺の名前 (L_j)

辺の右側の辺の名前 (R_j)

ネットワーク領域はドーナツ状に多数の穴が開いているので、2 本以上 ($P \geq 2$) の輪郭線を持つのが普通である。

なお、歴史的な理由から、輪郭線の点の総数 N が二度繰り返して書き込まれていることに注意する必要がある。

ネットワーク領域を分割したそれぞれの三角形の辺の情報の格納のされ方は、その分割処理の行い方に深く関わっている。たとえば、第 4 図のような T 字型のネットワーク領域に対して図の下方から処理が行われ、辺 $A_0 A_1$ (第 4 図では輪郭線の番号 i の点を A_i と表わしてある) より下のネットワーク領域はすでに三角形分割されているとする。このとき、次に得られる三角形 $A_0 A_1 A_2$ は、左側の頂点 A_2 を持つ 始点 A_1 、終点 A_0 とした向きを持つ底辺として適当な辺番号 j で登録される。

その後、得られた辺 $A_2 A_0$ と $A_1 A_2$ を新たな底辺として、適当な左側の頂点を捜すことによって次の分割三角形を決める処理が繰り返される。ところが、前者の辺はネットワーク領域の輪郭線なので左側の頂点が存在しない。そこで、辺 $A_2 A_0$ に対して、始点番号 A_0 、終点 A_2 、この辺に左側の頂点が見つからなかったことを意味する左側の辺 $A_1 A_0$ の辺番号に負の符号をつけたもの、そして、この辺が輪郭線をなす辺であることを意味する右側の辺の番号 0 が登録される。他方、左側の頂点 A_3 が存在する辺 $A_1 A_2$ に対しては、始点 A_1 、終点 A_2 、左側の頂点番号 A_3 、右側の辺 $A_1 A_0$ の番号が登録される。

最後に登録された底辺 $A_1 A_2$ に対する三角形の辺 $A_1 A_3$ 、 $A_3 A_2$ をもとにした三角形分割では、まず後者の辺を底辺とした分割処理が第 4 図の右の方へと進んで行く。その後、図には示していない経路で三角形分割が第 4 図の左の方に進んで来て、辺 $A_5 A_1$ を底辺、 A_3 を左側の頂点とする三角形が見つかったとする。このとき、三角形分割の底辺の候補であった辺 $A_1 A_3$ は必要なくなるので、その左側の辺として $A_5 A_1$ の辺番号に負の符号をつけたものが登録される。

以上のように、ネットワーク領域の三角形分割データファイルのそれぞれの辺情報の欄で L_j が負の値なら、その辺は分割三角形の底辺でない辺を意味し、その値の絶対値が隣の三角形の底辺の番号を表わしている。また、 R_j が 0 ならば、その辺はネットワーク領域の輪郭線をなす辺であることがわかる。

4.2. Pore-Channel の幾何学的情報ファイル

プログラム prune によって作成され、プログラム pcnet (pcnvf もしくは pcnec) が利用する Pore や Channel

の様々な情報が入ったファイルの構造は以下のとおりである。

輪郭線上の点の総数

分割三角形の辺の総数

分割三角形の総数

ネットワーク領域の総面積

節点の分布領域の広がり (x_b, y_b, x_n, y_n)

もとのスケルトンの節点数 (N_o)

枝刈りで取り除かれた節点数 (N_p)

枝刈り後の節点数 ($N_n = N_o - N_p$)

配位数ごとに示した節点の情報 ($i \geq 1$)

配位数 (i)

配位数 i の節点数 (F_i)

配位数 i の節点に対応する三角形の面積の和

... ($\sum_i F_i = N_n$ となるまでつづく)

Pore の総数 (M_p)

Channel の総数 (M_c)

Pore の情報 ($j = 1 \dots M_p$)

Pore 番号 (j)

Pore j を構成する三角形の個数

Pore j を構成する三角形の面積の和

Pore j に対応する節点の x 座標と y 座標値

Channel の情報 ($k = 1 \dots M_c$)

Channel 番号 (k)

Channel k を構成する三角形の個数

Channel k の両端の Pore の番号

Channel k を構成する三角形の面積の和 (S_k)

Channel k に対応するスケルトンの軸長 (L_k)

なお、Pore の情報の欄で、Pore 番号 1 と 2 のものとして書かれている情報は、ネットワークの流入口と流出口に対するものである。

4.3. チャンネルの抵抗値情報ファイル

プログラム pcnet (pcnvf もしくは pcnec) によって出力される抵抗網モデルのそれぞれの抵抗の値の情報などが入ったファイルの中味は以下のとおりである。

節点の分布領域の広がり (x_b, y_b, x_n, y_n)

Pore の総数 (M_p)

Pore の情報 ($j = 1 \dots M_p$)

Pore j を構成する三角形の面積の和

Pore j に対応する節点の x 座標と y 座標値

Channel の総数 (M_c)

Channel の情報 ($k = 1 \dots M_c$)

Channel k の出入口の Pore の番号 (I_j, O_j)

Channel k の伝導度 (抵抗値)

第 1 表 節点での配位数情報

配置位置	出現品度 (節点数)	面積 (画素数)
1	2	1
2	2 1 3 8	6 6 9 6 . 5
3	3 3 8	5 0 6 6

ここで、Pore および Channel の情報はそれぞれ、さきに説明した Pore-Channel の幾何学的情報ファイルにおける Pore 番号, Channel 番号の順番で並んでいる.

4.4. ポテンシャル情報ファイル

プログラム netsolve によって出力され, netdiv が入力ファイルとして取り扱うネットワークのすべての Pore でのポテンシャルおよび Channel を流れる流量が格納されたデータファイルの構造は以下のようになっている.

- Pore の総数 (M_p)
- Pore でのポテンシャル情報 ($j = 1 \cdots M_p$)
 - Pore j に対応する節点の x 座標と y 座標値
 - Pore j のポテンシャル値
- Channel の総数 (M_c)
- Channel での流量情報 ($k = 1 \cdots M_c$)
 - Channel k の出入口の Pore の番号 (I_j, O_j)
 - Channel k を伝った流量

ただし、Pore でのポテンシャル情報の欄で、ネットワークの流入口と流出口に対応する 1 番目と 2 番目の Pore に対するポテンシャル値としては、それぞれ 1 と 0 が入っている.

なお、このファイルの前半部の Pore のポテンシャル情報の部分は、我々の作成した二次元分布データの処理システム (中野・藤井, 1989b, c) で取り扱えるデータ構造になっている. つまり、不規則に分布する Pore でのポテンシャルの値から第 9 図のようなポテンシャル場の等高線図を描くとき、我々のシステムの入力としてこのファイルそのものを用いることができる.

5. 使用例

ここまですでに示してきた第 6, 7, および 9 図はすべて本文で紹介したプログラムで得た結果である. これらのもとになった画像データは、新日本製鉄株式会社から提供していただいた縦横 150 画素からなる鉄鉱石焼結体の X 線 CT (Computer Tomography) 画像である. このデータを本文で紹介したプログラムにかけける前に行った前処理 (第 2 図と第 3 図) については、中野・藤井 (1991a, b) を参照して頂きたい.

なお、第 6 図および第 7 図は、起動時に与えるパラメータを変えることによってどちらもプログラム prune を使って得た. すなわち、まず、“sxct20.tri” という名の三角形分割データファイルをもとにして、

```
prune sxct20.tri > sxct20.net
```

として、節点での配位数情報 (第 1 表) を求めると同時に、配位数 3 以上の節点に対応する Pore 領域を塗りつぶした第 7 図を得た. その後、ネットワークのスケルトンだけを描かせるために第 1 表を参考にして、

```
prune sxct20.tri 4
```

のように実際の配位数の最大値 3 より大きい値 (4) を持つ節点に対応する三角形を塗りつぶすように指定して、第 6 図を描かせた.

本文で紹介したプログラムによる解析のうち netsolveで行っている (連立方程式の) 計算には 2 MIPS 程度の速度のエンジニアリングワークステーションで数分かかった. それ以外の解析はパーソナルコンピュータでも 1 分以内に終了した.

6. おわりに

ネットワークとは本来三次元的なものであるから、その解析はここで行った二次元解析でなく、三次元解析によって行うべきであろう. ところが、残念なことに、三次元解析に耐えるほど高精度な画像データ (特に X 線 CT データ) は現時点では手に入らない. 今後、三次元画像処理に耐えうる画像データが手にはいるようになれば、本文で紹介したアルゴリズムを拡張して三次元ネットワーク解析を行う必要がある. そのとき、本文で述べた部分は二次元だけに限らず行えるリスト処理が中心となっているので、中野・藤井 (1991b) で紹介した二次元の Voronoi 分割の双対図形としての三角形による分割のかわりに、三次元の四面体分割を (実用になるくらい高速に) 行うアルゴリズムを開発すれば、ここで行ったのとはほぼ同様な三次元ネットワーク解析を行うことができるはずである.

謝 辞 : 鉄 鉱 石 焼 結 体 の X 線 CT (Computer Tomography) 画像を提供していただいた新日本製鉄株式会社に感謝します.

文 献

- Kirkpatrick, S. (1973) Percolation and conduction. *Rev. Mod. Phys.*, **43**, 574-588.
- Koplik, J., Lin, C. and Vermitte, M. (1984) Conductivity and permeability from microgeometry. *J. Appl. Phys.*, **56**, 3127-3131.
- Landau, L.D. and Lifshitz, E.M. (1970) Fluid Dynamics 1. Tokyo Press (エリ ランダウ・イエ リフシツ, 流体力学 1, 東京図書).
- 中野 司・藤井 直之 (1989a) 画像処理のためのソフトウェア : (1) 領域塗りつぶしと仮想スクリーン処理. 情報地質, 14A, 93-107.
- 中野 司・藤井 直之 (1989b) 二次元分布データの処理システム : (2) 二次元分布データの内挿補間. 情報地質, No.14B, 113-132.
- 中野 司・藤井 直之 (1989c) 二次元分布データの処理システム : (3) 等高線図と鳥瞰図. 情報地質, 14B, 133-150.
- 中野 司・藤井 直之 (1991a) 画像処理のためのソフトウェア : (3) 粒子像の識別と輪郭線の抽出. 情報地質, **2**, 23-44.
- 中野 司・藤井 直之 (1991b) 画像処理のためのソフトウェア : (4) 画像データからのネットワーク構造の抽出. 情報地質, **2**, 45-64.
- 新日本製鉄株式会社 (1989) 粒度偏析制御式鉄鉱石焼結法の開発 (詳細資料).
- 高安 秀樹 (1986) フラクタル, 朝倉書店.

要 旨

画像処理のためのソフトウェア : (5) 画像中のネットワーク構造上の流れ解析

: 中野 司・藤井 直之

画像データを用いて、そこに映されているネットワーク状の領域（ネットワーク領域）を伝って電流や粘性流体の流れの解析を行うためのアルゴリズムを説明する。本論文で述べられているネットワーク解析とは以下のようなものである。まず、画像データ中のネットワーク領域の三角形分割（中野・藤井, 1991 b）の情報をもとにして、ネットワークのモデルとしてしばしば用いられる Pore-Channel (PC) モデルを構築する。次に、流体がそのネットワークを流れる場合に、それぞれの Channel を流れる流体の受ける Channel の形状による抵抗値を求め、それをもとにネットワークを抵抗網モデルで近似する。この抵抗網全体の抵抗値を計算することによって、画像データに映された物体の巨視的な浸透率や電気伝導度を計算することができる。以上の処理を実際に行うためのプログラムを本文では紹介する。

キーワード：画像処理, ネットワーク解析, PC モデル, 抵抗網モデル, 浸透率, 電気伝導度

(1990年10月17日受付, 1991年4月20日受理)

付録：プログラムリスト

prune.c :

```

1:
2: /*
3:     prune
4:     prune skelton and calculate pore-channel information.
5:
6:     compile
7:     cc -o prune prune.c network.c g2.c -lm
8:     usage
9:     prune TriangleFileName {CoordinationNumber}
10: */
11:
12: #include <stdio.h>
13: #include <stdlib.h>
14: #include <alloc.h>
15: #include <math.h>
16: #include "network.h"
17:
18: #define D(i)      (double)(i)
19:
20: static Network network;
21:
22: #define AN        network.an
23: #define AX(a)     network.apex[a].x
24: #define AY(a)     network.apex[a].y
25: #define EN        network.en
26: #define ET(e)     network.edge[e].tail
27: #define EH(e)     network.edge[e].head
28: #define EL(e)     network.edge[e].left
29: #define ER(e)     network.edge[e].right
30:
31: static void      ErrorExit(msg)
32: char             *msg;
33: {
34:     fputs(msg,stderr); fputc('\n',stderr); exit(1);
35: }
36:
37: static double    TriangleArea(e)
38: int              e;
39: {
40:     return(0.5*fabs(D(AX(ET(e))-AX(EL(e)))*D(AY(EH(e))-AY(EL(e)))-
41:                    D(AX(EH(e))-AX(EL(e)))*D(AY(ET(e))-AY(EL(e)))));
42: }
43:
44: static int       SamePoint(ea,eb)
45: int              ea,eb;
46: {
47:     int          ta,ha,la,tb,hb,/* lb, */
48:               a1,a2,a3,a4,
49:               x1,y1,x2,y2,x3,y3,x4,y4;
50:
51:     ta=ET(ea); ha=EH(ea); la=EL(ea);
52:     tb=ET(eb); hb=EH(eb); /* lb=EL(eb); */
53:     if (ta==tb) {
54:         a1=ta /* =tb */; a2=ha /* =hb */; a3=la; a4=hb;
55:     }
56:     else
57:     if (ha==hb) {
58:         a1=ta /* =lb */; a2=ha /* =hb */; a3=la; a4=tb;
59:     }
60:     else
61:     if (la==tb) {
62:         a1=ha /* =tb */; a2=la /* =lb */; a3=ta; a4=hb;
63:     }
64:     else
65:     if (la==tb) {
66:         a1=la /* =tb */; a2=ta /* =lb */; a3=ha; a4=hb;
67:     }
68:     else
69:     if (la==hb) {
70:         a1=ha /* =lb */; a2=la /* =hb */; a3=ta; a4=tb;
71:     }
72:     else
73:     if (ta==hb) {
74:         a1=la /* =lb */; a2=ta /* =hb */; a3=ha; a4=tb;

```

```

75:     }
76:     else
77:         ErrorExit("no connection between two triangles.");
78:
79:     x1=AX(a1); y1=AY(a1);
80:     x2=AX(a2); y2=AY(a2);
81:     x3=AX(a3); y3=AY(a3);
82:     x4=AX(a4); y4=AY(a4);
83:     if ((D(x1-x3)*D(y4-y3)-D(x4-x3)*D(y1-y3))*
84:         (D(x2-x3)*D(x2-x1)+D(y2-y3)*D(y2-y1))==
85:         (D(x1-x3)*D(y2-y3)-D(x2-x3)*D(y1-y3))*
86:         (D(x4-x3)*D(x4-x1)+D(y4-y3)*D(y4-y1)))
87:         return(1);
88:     else
89:         return(0);
90: }
91:
92: static int      Link(node,e)
93: int             *node,e;
94: {
95:     while (node[e]<(-1)) e=(-node[e]-2);
96:     return(e);
97: }
98:
99: static int      Rename(number,name)
100: int             number,*name;
101: {
102:     int         e,n,count,*table;
103:
104:     if ((table=(int *)calloc(number,sizeof(*table)))==NULL)
105:         ErrorExit("can not allocate rename area.");
106:
107:     for (e=0; e<EN; e++) if ((n=name[e])>0) table[n-1]=n;
108:     for (count=n=0; n<number; n++) if (table[n]!=0) table[n]=(++count);
109:     for (e=0; e<EN; e++) if ((n=name[e])>0) name[e]=table[n-1];
110:
111:     free(table); return(count);
112: }
113:
114: #define SWAP(a,b)      work=(a); a=(b); b=work
115:
116: static void          Sort3(x1,y1,x2,y2,x3,y3)
117: int                  *x1,*y1,*x2,*y2,*x3,*y3;
118: {
119:     int              work;
120:
121:     if (*x1>(*x2))
122:         if (*x2>(*x3)) {
123:             SWAP(*x1,*x3); SWAP(*y1,*y3);
124:         }
125:     else
126:         if (*x3>(*x1)) {
127:             SWAP(*x1,*x2); SWAP(*y1,*y2);
128:         }
129:     else {
130:         SWAP(*x1,*x2); SWAP(*y1,*y2); SWAP(*x2,*x3); SWAP(*y2,*y3);
131:     }
132:     else
133:         if (*x2>(*x3))
134:             if (*x3>(*x1)) {
135:                 SWAP(*x2,*x3); SWAP(*y2,*y3);
136:             }
137:         else {
138:             SWAP(*x2,*x3); SWAP(*y2,*y3); SWAP(*x1,*x2); SWAP(*y1,*y2);
139:         }
140: }
141:
142: static void          TrianglePaint(a,b,c,color)
143: int                  a,b,c,color;
144: {
145:     int              x1,y1,x2,y2,x3,y3,dx21,dx31,dx32,dy21,dy31,dy32,x,y;
146:     double            s,t,xx,yy;
147:
148:     movep(D(x1=AX(a)),D(y1=AY(a)));
149:     drawl(D(x2=AX(b)),D(y2=AY(b)),color,-1);
150:     drawl(D(x3=AX(c)),D(y3=AY(c)),color,-1);
151:     drawl(D(x1),D(y1),color,-1);
152:
153:     Sort3(&x1,&y1,&x2,&y2,&x3,&y3);
154:     dx21=x2-x1; dx31=x3-x1; dx32=x3-x2;
155:     for (x=1; x<dx21; x++) {
156:         s=D(x)/D(dx21); t=D(x)/D(dx31); xx=D(x1+x);

```

```

157:         movep(xx,s*D(y2)+(1.0-s)*D(y1));
158:         drawl(xx,t*D(y3)+(1.0-t)*D(y1),color,-1);
159:     }
160:     if (x1!=x2 && x2!=x3) {
161:         t=D(dx21)/D(dx31); xx=D(x2);
162:         movep(xx,D(y2));
163:         drawl(xx,t*D(y3)+(1.0-t)*D(y1),color,-1);
164:     }
165:     for (x=1; x<dx32; x++) {
166:         s=D(x)/D(dx32); t=D(dx21+x)/D(dx31); xx=D(x2+x);
167:         movep(xx,s*D(y3)+(1.0-s)*D(y2));
168:         drawl(xx,t*D(y3)+(1.0-t)*D(y1),color,-1);
169:     }
170:
171:     Sort3(&y1,&x1,&y2,&x2,&y3,&x3);
172:     dy21=y2-y1; dy31=y3-y1; dy32=y3-y2;
173:     for (y=1; y<dy21; y++) {
174:         s=D(y)/D(dy21); t=D(y)/D(dy31); yy=D(y1+y);
175:         movep(s*D(x2)+(1.0-s)*D(x1),yy);
176:         drawl(t*D(x3)+(1.0-t)*D(x1),yy,color,-1);
177:     }
178:     if (y1!=y2 && y2!=y3) {
179:         t=D(dy21)/D(dy31); yy=D(y2);
180:         movep(D(x2),yy);
181:         drawl(t*D(x3)+(1.0-t)*D(x1),yy,color,-1);
182:     }
183:     for (y=1; y<dy32; y++) {
184:         s=D(y)/D(dy32); t=D(dy21+y)/D(dy31); yy=D(y2+y);
185:         movep(s*D(x3)+(1.0-s)*D(x2),yy);
186:         drawl(t*D(x3)+(1.0-t)*D(x1),yy,color,-1);
187:     }
188: }
189:
190: typedef struct {
191:     int      t;
192:     double   x,y,a;
193: } Pore;
194:
195: typedef struct {
196:     int      t,i,o;
197:     double   l,a;
198: } Chan;
199:
200: #define PT(n) (pore[n].t)
201: #define PX(n) (pore[n].x)
202: #define PY(n) (pore[n].y)
203: #define PA(n) (pore[n].a)
204: #define CT(n) (chan[n].t)
205: #define CI(n) (chan[n].i)
206: #define CO(n) (chan[n].o)
207: #define CL(n) (chan[n].l)
208: #define CA(n) (chan[n].a)
209:
210: void main(argc,argv)
211: int  argc;
212: char **argv;
213: {
214:     int      faceno,e,*node,e1,e2,work,nodemax,nodeno,*freq,n,a,
215:             l,r,b,u,i,o,pruned,cn,*pname,poreno,*cname,channo,
216:             xmin,ymin,xmax,ymax;
217:     double   *area,totalarea,xl,xr,yb,yu,tri;
218:     Point    d,p;
219:     Pore     *pore;
220:     Chan     *chan;
221:
222:     if (argc<2 || argc>3)
223:         ErrorExit("usage : prune trianglefile {coordination}");
224:
225:     if (!loadnetwork(&network,argv[1])) ErrorExit("file read error.");
226:
227:     printf("%d\n%d\n",AN,EN);
228:
229:     faceno=0; totalarea=0.0;
230:     for (e=0; e<EN; e++)
231:         if (EL(e)>0) {
232:             ++faceno; totalarea+=TriangleArea(e);
233:         }
234:
235:     printf("%d\t%lf\n",faceno,totalarea);
236:
237:     if ((node=(int *)calloc(EN,sizeof(*node)))==NULL)
238:         ErrorExit("can not allocate node area.");

```

```

239:
240:     for (e=0; e<EN; e++)
241:         if (ER(e)>=0) {
242:             ++(node[ER(e)]); ++(node[(EL(e)>=0)?e:-EL(e)-2]);
243:         }
244:     for (e=0; e<EN; e++)
245:         if (ER(e)>=0) {
246:             e1=(EL(e)>=0)?e:-EL(e)-2; e2=ER(e);
247:             if (SamePoint(e1,e2)) {
248:                 e1=Link(node,e1); e2=Link(node,e2);
249:                 if (e1>e2) {
250:                     SWAP(e1,e2);
251:                 }
252:                 node[e1]+=(node[e2]-2); node[e2]=(-e1-2);
253:             }
254:         }
255:
256:     nodeno=0;
257:     for (e=0; e<EN; e++)
258:         if ((n=node[e])>0) {
259:             p=voronoi(&network,(EL(e)>=0)?e:-EL(e)-2);
260:             if (nodeno==0) {
261:                 l=r=b=u=e; xl=xr=p.x; yb=yu=p.y;
262:             }
263:             else {
264:                 if (p.x<xl || (p.x==xl && n<node[l])) {
265:                     l=e; xl=p.x;
266:                 }
267:                 if (p.x>xr || (p.x==xr && n<node[r])) {
268:                     r=e; xr=p.x;
269:                 }
270:                 if (p.y<yb || (p.y==yb && n<node[b])) {
271:                     b=e; yb=p.y;
272:                 }
273:                 if (p.y>yu || (p.y==yu && n<node[u])) {
274:                     u=e; yu=p.y;
275:                 }
276:             }
277:             ++nodeno;
278:         }
279:
280:     if ((xr-xl)>(yu-yb)) {
281:         i=l; o=r;
282:     }
283:     else {
284:         i=b; o=u;
285:     }
286:
287:     printf("%lf\t%lf\t%lf\t%lf\n",xl,yb,xr,yu);
288:     printf("%d\n",nodeno);
289:
290:     pruned=0;
291:     for (e=EN-1; e>=0; e--)
292:         if (ER(e)>=0) {
293:             e1=Link(node,(EL(e)>=0)?e:-EL(e)-2);
294:             e2=Link(node,ER(e));
295:             if (e1!=e2 && node[e1]==1 && node[e2]>1 && e1!=i && e1!=o) {
296:                 ++pruned; node[e1]=(-1); --(node[e2]);
297:             }
298:         }
299:     for (e=0; e<EN; e++)
300:         if (ER(e)>=0) {
301:             e1=Link(node,(EL(e)>=0)?e:-EL(e)-2);
302:             e2=Link(node,ER(e));
303:             if (e1!=e2 && node[e2]==1 && node[e1]>1 && e2!=i && e2!=o) {
304:                 ++pruned; node[e2]=(-1); --(node[e1]);
305:             }
306:         }
307:
308:     printf("%d\n",pruned);
309:
310:     nodemax=nodeno;
311:     for (e=0; e<EN; e++)
312:         if (node[e]>0) {
313:             ++nodeno; if (node[e]>nodemax) nodemax=node[e];
314:         }
315:
316:     printf("%d\n",nodeno);
317:
318:     if ((freq=(int *)calloc(nodemax,sizeof(*freq)))==NULL ||
319:         (area=(double *)calloc(nodemax,sizeof(*area)))==NULL)
320:         ErrorExit("can not allocate coordination number area.");

```

```

321:
322:     for (n=0; n<nodemax; n++) area[n]=0.0;
323:     for (e=0; e<EN; e++) {
324:         if ((n=node[e]-1)>=0) ++freq[n];
325:         if (EL(e)>=0 && (n=node[Link(node,e)]-1)>=0)
326:             area[n]+=TriangleArea(e);
327:     }
328:     for (n=0; n<nodemax; n++)
329:         printf("%d\t%d\t%lf\n",n+1,freq[n],area[n]);
330:
331:     free(area); free(freq);
332:
333:     if ((pname=(int *)calloc(EN,sizeof(*pname)))==NULL ||
334:         (cname=(int *)calloc(EN,sizeof(*cname)))==NULL)
335:         ErrorExit("can not allocate pore-channel name area.");
336:
337:     if (node[i]<=2) node[i]=3;
338:     if (node[o]<=2) node[o]=3;
339:
340:     pname[i]=1; pname[o]=2;
341:
342:     poreno=2;
343:     for (e=0; e<EN; e++)
344:         if (pname[e]==0 && node[n=Link(node,e)]>2)
345:             if (pname[n]!=0)
346:                 pname[e]=(-n-2);
347:             else
348:                 pname[e]=(++poreno);
349:     for (e=0; e<EN; e++)
350:         if (ER(e)>=0) {
351:             e1=Link(pname,ER(e));
352:             e2=Link(pname,(EL(e)>=0)?e:-EL(e)-2);
353:             if (e1!=e2 && pname[e1]>0 && pname[e2]>0)
354:                 if (pname[e1]<pname[e2])
355:                     pname[e2]=(-e1-2);
356:                 else
357:                     pname[e1]=(-e2-2);
358:         }
359:     poreno=Rename(poreno,pname);
360:
361:     channo=0;
362:     for (e=0; e<EN; e++)
363:         if (pname[e]==0 && node[n=Link(node,e)]=2)
364:             if (cname[n]!=0)
365:                 cname[e]=(-n-2);
366:             else
367:                 cname[e]=(++channo);
368:     for (e=0; e<EN; e++)
369:         if (ER(e)>=0) {
370:             e1=Link(cname,ER(e));
371:             e2=Link(cname,(EL(e)>=0)?e:-EL(e)-2);
372:             if (e1!=e2 && cname[e1]>0 && cname[e2]>0)
373:                 if (cname[e1]<cname[e2])
374:                     cname[e2]=(-e1-2);
375:                 else
376:                     cname[e1]=(-e2-2);
377:         }
378:     channo=Rename(channo,cname);
379:
380:     printf("%d\t%d\n",poreno,channo);
381:
382:     if ((pore=(Pore *)calloc(poreno,sizeof(*pore)))==NULL ||
383:         (chan=(Chan *)calloc(channo,sizeof(*chan)))==NULL)
384:         ErrorExit("can not allocate pore-channel data area.");
385:
386:     for (n=0; n<poreno; n++) {
387:         PT(n)=0; PX(n)=PY(n)=PA(n)=0.0;
388:     }
389:     for (e=0; e<EN; e++)
390:         if (pname[e]!=0) {
391:             ++PT(n=pname[Link(pname,e)]-1);
392:             PA(n)+=(tri=TriangleArea(e));
393:             p=voronoi(&network,e);
394:             PX(n)+=(p.x*tri); PY(n)+=(p.y*tri);
395:         }
396:     for (n=0; n<poreno; n++)
397:         printf("%d\t%d\t%lf\t%lf\t%lf\n",
398:             n+1,PT(n),PA(n),PX(n)/PA(n),PY(n)/PA(n));
399:
400:     for (n=0; n<channo; n++) {
401:         CT(n)=CI(n)=CO(n)=0; CA(n)=CL(n)=0.0;
402:     }

```

```

403:     for (e=0; e<EN; e++)
404:         if (cname[e]!=0) {
405:             ++CT(n=cname[Link(cname,e)]-1);
406:             CA(n)+=TriangleArea(e);
407:             if ((r=ER(e))>0 && (pname[r]!=0 || cname[r]!=0)) {
408:                 d=voronoi(&network,r); p=voronoi(&network,e);
409:                 d.x-=p.x; d.y-=p.y;
410:                 CL(n)+=sqrt(d.x*d.x+d.y*d.y);
411:                 if (pname[r]!=0)
412:                     if (CI(n)==0)
413:                         CI(n)=pname[Link(pname,r)];
414:                     else
415:                         CO(n)=pname[Link(pname,r)];
416:             }
417:         }
418:     else
419:         if ((r=ER(e))>0) {
420:             l=((EL(e)>0)?e:-EL(e)-2);
421:             if ((n=cname[Link(cname,l)]-1)>0) {
422:                 d=voronoi(&network,l); p=voronoi(&network,r);
423:                 d.x-=p.x; d.y-=p.y;
424:                 CL(n)+=sqrt(d.x*d.x+d.y*d.y);
425:                 if (pname[r]!=0)
426:                     if (CI(n)==0)
427:                         CI(n)=pname[Link(pname,r)];
428:                     else
429:                         CO(n)=pname[Link(pname,r)];
430:             }
431:         }
432:     else
433:         if (pname[l]!=0 && (n=cname[Link(cname,x)]-1)>0) {
434:             d=voronoi(&network,l); p=voronoi(&network,r);
435:             d.x-=p.x; d.y-=p.y;
436:             CL(n)+=sqrt(d.x*d.x+d.y*d.y);
437:             if (CI(n)==0)
438:                 CI(n)=pname[Link(pname,l)];
439:             else
440:                 CO(n)=pname[Link(pname,l)];
441:         }
442:     }
443:     for (n=0; n<channo; n++)
444:         printf("%d\t%d\t%d\t%d\t%lf\t%lf\n",
445:             n+1,CT(n),CI(n),CO(n),CA(n),CL(n));
446:
447:     xmin=xmax=AX(0); ymin=ymax=AY(0);
448:     for (a=1; a<AN; a++) {
449:         if (AX(a)<xmin) xmin=AX(a); else if (AX(a)>xmax) xmax=AX(a);
450:         if (AY(a)<ymin) ymin=AY(a); else if (AY(a)>ymax) ymax=AY(a);
451:     }
452:
453:     initscr(); clrscr();
454:     window(D(xmin-1),D(ymin-1),D(xmax+1),D(ymax+1),0);
455:
456:     #ifdef PERIMETER
457:     displayedge(&network,1);
458:     #endif
459:
460:     cn=(argc==3)?atoi(argv[2])-1:2;
461:
462:     for (e=0; e<EN; e++)
463:         if (EL(e)>0 && (n=node[l=Link(node,e)])>cn)
464:             TrianglePaint(ER(e),ET(e),EL(e),n+((l==i || l==o)?8:0));
465:
466:     for (e=0; e<EN; e++)
467:         if (ER(e)>0)
468:             if (node[Link(node,(EL(e)>0)?e:-EL(e)-2)]>0 &&
469:                 node[Link(node,ER(e))]>0) {
470:                 p=voronoi(&network,ER(e));
471:                 movep(p.x,p.y);
472:                 p=voronoi(&network,(EL(e)>0)?e:-EL(e)-2);
473:                 drawl(p.x,p.y,7,-1);
474:             }
475:
476:     d.x=0.005*D(xmax-xmin); d.y=0.005*D(ymax-ymin);
477:
478:     for (e=0; e<EN; e++)
479:         if ((n=node[e])>2) {
480:             p=voronoi(&network,(EL(e)>0)?e:-EL(e)-2);
481:             if (e==i || e==o)
482:                 plotp(p.x,p.y,n+8);
483:             else {
484:                 movep(p.x-d.x,p.y); drawl(p.x+d.x,p.y,n+8,-1);

```



```

485:                movep(p.x,p.y-d.y); drawl(p.x,p.y+d.y,n+8,-1);
486:            }
487:        }
488:    }

ecc.c :

1:
2: /*
3:     conductance of electrical current through 2 dimensional channel
4: */
5:
6: double conductance(width,length)
7: double width,length;
8: {
9:     return(width/length);
10: }

vfc.c :

1:
2: /*
3:     conductance of viscus flow through 2 dimensional channel
4: */
5:
6: double conductance(width,length)
7: double width,length;
8: {
9:     return(width*width*width/length);
10: }

pcnet.c :

1:
2: /*
3:     pcnet
4:     calculate conductance of all channel of the network.
5:
6:     compile
7:     cc -o {pcnec | pcnvf} pcnet.c {ecc.c | vfc.c}
8:     usage
9:     {pcnec | pcnvf} PoreChannelFile
10: */
11:
12: #include <stdio.h>
13: #include <stdlib.h>
14: #include <alloc.h>
15:
16: extern double conductance();
17:
18: typedef struct {
19:     double area,x,y;
20: } Pore;
21:
22: typedef struct {
23:     int pore1,pore2;
24:     double area,length,conductance;
25: } Chan;
26:
27: typedef struct {
28:     double xl,yd,xr,yu;
29:     int pn,cn;
30:     Pore *pore;
31:     Chan *chan;
32: } PC;
33:
34: #define XL (pc->xl)
35: #define YD (pc->yd)
36: #define XR (pc->xr)
37: #define YU (pc->yu)
38:
39: #define PN (pc->pn)
40: #define P (pc->pore)
41: #define PA(p) (P[p].area)
42: #define PX(p) (P[p].x)
43: #define PY(p) (P[p].y)
44:
45: #define CN (pc->cn)
46: #define C (pc->chan)
47: #define C1(c) (C[c].pore1)
48: #define C2(c) (C[c].pore2)
49: #define CA(c) (C[c].area)

```

```

50: #define CL(c) (C[c].length)
51:
52: int loadpc(pc,name)
53: PC *pc;
54: char *name;
55: {
56:     FILE *file;
57:     int an,en,tn,node,coor,cunt,freq,id,p,c;
58:     double area;
59:
60:     if ((file=fopen(name,"r"))==NULL) return(0);
61:
62:     fscanf(file,"%d",&an);
63:     fscanf(file,"%d",&en);
64:     fscanf(file,"%d %lf",&tn,&area);
65:
66:     fscanf(file,"%lf %lf %lf %lf",&XL,&YD,&XR,&YU);
67:
68:     fscanf(file,"%d",&node);
69:     fscanf(file,"%d",&node);
70:     fscanf(file,"%d",&node);
71:
72:     for (freq=0; freq<node; freq+=cunt)
73:         fscanf(file,"%d %d %lf",&coor,&cunt,&area);
74:
75:     fscanf(file,"%d %d",&PN,&CN);
76:
77:     if ((P=(Pore *)calloc(PN,sizeof(*P)))==NULL ||
78:         (C=(Chan *)calloc(CN,sizeof(*C)))==NULL) return(-1);
79:
80:     for (p=0; p<PN; p++)
81:         fscanf(file,"%d %d %lf %lf %lf",
82:             &id,&tn,&PA(p),&PX(p),&PY(p));
83:
84:     for (c=0; c<CN; c++)
85:         fscanf(file,"%d %d %d %d %lf %lf",
86:             &id,&tn,&C1(c),&C2(c),&CA(c),&CL(c));
87:
88:     fclose(file); return(1);
89: }
90:
91: void printpc(pc)
92: PC *pc;
93: {
94:     int p,c;
95:
96:     printf("%lf\t%lf\t%lf\t%lf\n",XL,YD,XR,YU);
97:
98:     printf("%d\n",PN);
99:     for (p=0; p<PN; p++) printf("%lf\t%lf\t%lf\n",PA(p),PX(p),PY(p));
100:
101:     printf("%d\n",CN);
102:     for (c=0; c<CN; c++)
103:         printf("%d\t%d\t%lf\n",C1(c),C2(c),C[c].conductance);
104: }
105:
106: void Error(msg)
107: char *msg;
108: {
109:     fputs(msg,stderr); fputc('\n',stderr); exit(1);
110: }
111:
112: void main(argc,argv)
113: int argc;
114: char **argv;
115: {
116:     char usage[256];
117:     PC pc;
118:     int c;
119:
120:     if (argc!=2) {
121:         sprintf(usage,"usage : %s PoreChannelFile",argv[0]);
122:         Error(usage);
123:     }
124:
125:     switch (loadpc(&pc,argv[1])) {
126:         case 0 : Error("file not found."); break;
127:         case -1 : Error("memory allocation error."); break;
128:     }
129:
130:     for (c=0; c<pc.cn; c++)
131:         pc.chan[c].conductance=

```

```

132:         conductance(pc.chan[c].area/pc.chan[c].length,
133:                     pc.chan[c].length);
134:
135:     printpc(&pc);
136: }

netsolve.c :

1:
2: /*
3:     netsolve
4:     calculate potentials at the nodes on the network.
5:
6:     compile
7:     cc -o netsolve netsolve.c
8:     usage
9:     netsolve ConductanceFile
10: */
11:
12: #include <stdio.h>
13: #include <stdlib.h>
14: #include <alloc.h>
15:
16: static int      Allocate(n,a,b)
17: int             n;
18: double         ***a,**b;
19: {
20:     int         i,j;
21:
22:     if ((*b=(double *)calloc(n,sizeof(**b)))==NULL ||
23:         (*a=(double **)calloc(n,sizeof(**a)))==NULL) return(0);
24:     for (i=0; i<n; i++) {
25:         if (((*a)[i]=(double *)calloc(n,sizeof(**a)))==NULL) return(0);
26:         (*b)[i]=0.0; for (j=0; j<n; j++) (*a)[i][j]=0.0;
27:     }
28:     return(1);
29: }
30:
31: static void      SweepOut(n,a,b)
32: int             n;
33: double         ***a,*b;
34: {
35:     int         i,j,k,i1,n1;
36:     double      p,q;
37:
38:     n1=n-1;
39:     for (i=0; i<n1; i++) {
40:         i1=i+1;
41:         j=i; p=a[i][i]; if (p<0.0) p=(-p);
42:         for (k=i1; k<n; k++) {
43:             q=a[k][i]; if (q<0.0) q=(-q);
44:             if (p<q) {
45:                 j=k; p=q;
46:             }
47:         }
48:         if (j!=i) {
49:             for (k=i; k<n; k++) {
50:                 q=a[i][k]; a[i][k]=a[j][k]; a[j][k]=q;
51:             }
52:             q=b[i]; b[i]=b[j]; b[j]=q;
53:         }
54:         for (k=i1; k<n; k++) a[i][k]/=p;
55:         b[i]/=p;
56:         for (k=i1; k<n; k++)
57:             if ((p=a[k][i])!=0.0) {
58:                 for (j=i1; j<n; j++) a[k][j]-=(p*a[i][j]);
59:                 b[k]-=(p*b[i]);
60:             }
61:     }
62:     b[n1]/=a[n1][n1];
63:     for (i=n1-1; i>=0; i--) {
64:         p=0.0;
65:         for (j=i+1; j<n; j++) p+=(a[i][j]*b[j]);
66:         b[i]-=p;
67:     }
68: }
69:
70: static void      Error(msg)
71: char             *msg;
72: {
73:     fputs(msg,stderr); fputc('\n',stderr); exit(1);
74: }

```

```

75:
76: void    main(argc,argv)
77: int      argc;
78: char     **argv;
79: {
80:     int    pn,p,pn2,cn,c,*pore1,*pore2;
81:     double xl,yd,xr,yu,area,*x,*y,**a,*b,*cond,p1,p2;
82:     FILE    *file;
83:
84:     if (argc!=2) Error("usage : netsolve ConductanceFile");
85:
86:     if ((file=fopen(argv[1],"r"))==NULL) Error("file not found.");
87:
88:     fscanf(file,"%lf %lf %lf %lf",&xl,&yd,&xr,&yu);
89:
90:     fscanf(file,"%d",&pn);
91:     if ((x=(double *)calloc(pn,sizeof(*x)))==NULL ||
92:         (y=(double *)calloc(pn,sizeof(*y)))==NULL)
93:         Error("can not allocate pore position area.");
94:
95:     for (p=0; p<pn; p++)
96:         fscanf(file,"%lf %lf %lf",&area,&(x[p]),&(y[p]));
97:
98:     pn2=pn-2;
99:     if (!Allocate(pn2,&a,&b))
100:         Error("can not allocate calculation area.");
101:
102:     fscanf(file,"%d",&cn);
103:     if ((pore1=(int *)calloc(cn,sizeof(*pore1)))==NULL ||
104:         (pore2=(int *)calloc(cn,sizeof(*pore2)))==NULL ||
105:         (cond=(double *)calloc(cn,sizeof(*cond)))==NULL)
106:         Error("can not allocate channel data area.");
107:
108:     for (c=0; c<cn; c++) {
109:         fscanf(file,"%d %d %lf",&(pore1[c]),&(pore2[c]),&(cond[c]));
110:         if (pore1[c]!=pore2[c] && (pore1[c]>2 || pore2[c]>2)) {
111:             if (pore1[c]>2) {
112:                 p=pore1[c]-3;
113:                 a[p][p]+=cond[c];
114:                 if (pore2[c]>2)
115:                     a[p][pore2[c]-3]-=cond[c];
116:                 else
117:                     if (pore2[c]==1) b[p]+=cond[c];
118:             }
119:             if (pore2[c]>2) {
120:                 p=pore2[c]-3;
121:                 a[p][p]+=cond[c];
122:                 if (pore1[c]>2)
123:                     a[p][pore1[c]-3]-=cond[c];
124:                 else
125:                     if (pore1[c]==1) b[p]+=cond[c];
126:             }
127:         }
128:     }
129:     fclose(file);
130:
131:     SweepOut(pn2,a,b);
132:
133:     printf("%d\n",pn);
134:     printf("%lf\t%lf\t1.0\n",x[0],y[0]);
135:     printf("%lf\t%lf\t0.0\n",x[1],y[1]);
136:     for (p=2; p<pn; p++) printf("%lf\t%lf\t%lf\n",x[p],y[p],b[p-2]);
137:
138:     printf("%d\n",cn);
139:     for (c=0; c<cn; c++) {
140:         switch(pore1[c]) {
141:             case 1 :
142:                 p1=1.0; break;
143:             case 2 :
144:                 p1=0.0; break;
145:             default :
146:                 p1=b[pore1[c]-3];
147:         }
148:         switch(pore2[c]) {
149:             case 1 :
150:                 p2=1.0; break;
151:             case 2 :
152:                 p2=0.0; break;
153:             default :
154:                 p2=b[pore2[c]-3];
155:         }
156:         printf("%d\t%d\t%lf\n",pore1[c],pore2[c],cond[c]*(p1-p2));

```

```

157:     }
158: }

netdiv.c :

1:
2: /*
3:     netdiv
4:     calculate divergence of the flow at the node of the network.
5:
6:     compile
7:     cc -o netdiv netdiv.c
8:     usage
9:     netdiv PotentialFile NodeName
10: */
11:
12: #include <stdio.h>
13: #include <stdlib.h>
14:
15: static void    Error(msg)
16: char          *msg;
17: {
18:     fputs(msg,stderr); fputc('\n',stderr); exit(1);
19: }
20:
21: void    main(argc,argv)
22: int     argc;
23: char    **argv;
24: {
25:     FILE    *file;
26:     int     pore,pn,p,found,cn,c,num,pore1,pore2;
27:     double  x,y,z,xp,yp,zp,div,flow;
28:
29:     if (argc!=3) Error("usage : netdiv PotentialFile pore");
30:
31:     if ((file=fopen(argv[1],"r"))==NULL) Error("file not found.");
32:
33:     pore=atoi(argv[2]);
34:
35:     fscanf(file,"%d",&pn);
36:     for (found=0, p=0; p<pn; p++) {
37:         fscanf(file,"%lf %lf %lf",&x,&y,&z);
38:         if (pore==p+1) {
39:             found=1; xp=x; yp=y; zp=z;
40:         }
41:     }
42:
43:     if (!found) Error("pore is not found.");
44:
45:     fscanf(file,"%d",&cn);
46:     for (div=0.0, num=0, c=0; c<cn; c++) {
47:         fscanf(file,"%d %d %lf",&pore1,&pore2,&flow);
48:         if (pore==pore1) {
49:             ++num; div+=flow;
50:         }
51:         if (pore==pore2) {
52:             ++num; div-=flow;
53:         }
54:     }
55:     fclose(file);
56:
57:     printf("%d\t%lf\t%lf\t%lf\t%lf\t%lf\n",pore,xp,yp,zp,num,div);
58: }

```