DEM (数値標高モデル) から DDM (流水線図) を作成する アルゴリズムの改良と C 言語プログラム

野上道男

An Algorithm and a C-program Source for Automated Drainage Network Extraction

Michio NOGAMI

Abstract : We developed a flooding-type algorithm for drainage network extraction and show a full program source in the language C. The program generates automatically Drainage Direction Matrices (DDMs) from Digital Elevation Models (DEMs) with sink holes. By using a random technique of random walk type, the drainage networks are processed adequately in flat plains or alluvial fans.

Keywords: 数値標高モデル (DEM), 流水線図 (DDM)、自動流域抽出 (automated drainage extraction)

1. 流水線追跡問題

国土地理院によって全国整備されつつある50m-DEMはその格子間隔の細密さによって、地形研究のみならず、応用的な面でも画期的なものである。DEMは等高線地形図がそうであったように、すべての数値地図情報の基図であるといえることができる (Nogami, 1989)。

またこのような精密なDEMによって流域の設定が容易に行えるようになり、地形学 (地形計測など) や水文学 (とくに流域水質管理/流域水収支・水資源/流出洪水問題など) においてDEMの重要性が増す (野上, 1995)。

水がどの方位に流れるかを示す流水線図 (Drainage Direction Matrix; DDM) がDEMから得られれば、それを用いて流域単位の各種演算処理を容易に行うことができる (吉山, 1990; 1994; Nogami, 1995)。例えば、ある注目点を通る流水線に連なる上流の格子点の数を数え、単位格子面積を乗ず

ればその点の流域面積となる。

もっと実践的にこのアルゴリズムを述べよう。流域面積を格納するためにDEMと同じ大きさの2次元配列を用意する。全ての格子点についてそこを出発点として、DDMに従って図幅の縁または海に達するまで格子点を移動する。そのとき移動するたびに、その通過格子点の面積値に1を加える (蛇足かも知れないが、例えば流域内人口を計算するのであれば、1ではなく出発点の人口を加える)。このようにして2重ループによるすべての出発格子点からの走査が終われば流域面積の配列が完成している。

以上のように、DEMによる流域面積測定はDDMによって簡単に行うことができる。問題はそれの基礎となるDDMの生成である。この問題は流水線追跡問題 (あるいは流域抽出問題) として研究されてきた (例えば, Marks *et al*, 1984; 野上・杉浦, 1986; 野上, 1991a・b; Nogami, 1995; Jenson, 1991; Chorowicz *et al*, 1992; McComack *et al*, 1993; Pike, 1993)。

水は周囲の最も低い方向に流れるから、DEMによって注目点から4方向あるいは8方向の標高を参照して流水線方向を定める。これはごく簡単な原理であるが、実際にコンピュータ処理で流水線追跡を自動化しようとするといくつかの困難点があり、こ

野上: 〒192-03 八王子市南大沢1-1
東京都立大学理学研究科
Dept. Geogr. Tokyo Metropolitan Univ.
Minami-Osawa 1-1, Hachioji, Tokyo
E-Mail: nogami@comp.metro-u.ac.jp
<http://www.sci.metro-u.ac.jp/~nogami/>

れまで多くの論文のテーマとなるほどの複雑な問題であることが分かっている。

2. 洪水型凹陷値処理

流水線追跡問題を複雑にする原因はDEM上で凹陷地となる標高値（以後凹陷値とする）や平坦地の存在である。火山地形やカルスト地形あるいは砂丘地形のように、地表の水系網が完全でない場合には凹陷値は普通に存在するが、それとは別にDEMには多数の凹陷値が本源的に存在する。このことはDEMの格子点間隔をいくら細かくしても同じことである。凹陷値があると単純な探索プログラム（例えば、野上・杉浦,1986）では無限ループに入ってしまう。このことをさけるため凹陷値ではそこが凹陷値でなくなるまで、標高値に1を加えるなどして、その標高値を少しずつ上げることを考える。このことは凹地に水が溜まって、水位が上がり出口が見つかるそこから流れ出すことに類比される。そこでこのような流水線の発見方法を洪水型アルゴリズムと呼ぶ。「洪水」と溢流という自然現象に準拠していることがこのアルゴリズムの利点である(Nogami, 1995)。

3. 平坦地における流水線の処理

凹陷値が埋まったとして、次の問題は平坦地にどのように流水線を設定するかである。本質的な平坦さは湖面で見られるが、湖の流出河川から湖に流入する河川のすべての流域を認識する必要がある場合などには、湖にも流水線を設定しなければならない。また緩勾配の沖積低地でもDEMのもとになった地図の等高線間隔が開いているので、DEM上に平坦地が広い面積で現れる。

全くの平坦に流水線を設定することはできないので、平坦値の標高を高めて勾配を与えることにする。すなわち「洪水面」に勾配を与えることにする。

勾配は格子間隔に依存するが、DEMの格子間隔を変えることはできず、また計算速度の点から標高値として整数を使いたいの、DEMの標高値に倍数をかけることにする。たとえば、50m-DEMで標高値を1000倍した場合、水平でない最小勾配は、

1/50000となる。この値は日本の大きな沖積平野の勾配よりは小さく、琵琶湖でも湖頭と湖尻の水位差が1メートル未満となるから妥当な値と考えられる。この値より大きな値を設定すると、洪水に水没する地域の流水線は任意のものとなる。

このように洪水に勾配を与えた場合にさらに問題が生ずる。それは標高差を格子間で1と固定すると、たとえば海からの距離が等しい点は等しい洪水面標高値となり、直線水路が長く続くような不自然な水系網となる。そこで、洪水面の格子点ごとにランダムな「波」を立てることにする。この「波」の波高は標高値(m)を1000倍したとき、経験的に10ではやや大きすぎ、経験的には5位が適当のようである。「波」が高いと水路が小さく屈曲する程度が高くなる。波の高さは乱数で与えているので、それをさけて流れる流路はランダムウォークの性質を持つわけである。

ここに掲載したプログラムでは、水が流れ得る方向が2つ以上ある場合は、乱数を発生させてそのうちの1つを選択している。これまでに発表された流水線追跡アルゴリズムの多くはこのようなランダムテクニックを採用していない。谷底の狭い山地内の水路網では区別が付かないが、この技法を採用していないプログラムでは、新しい火山や扇状地、あるいは沖積低地・湖水面に対して作成された流水線が極めて不自然となるという欠陥がある。

4. 生成された流水線と河川網の不一致

確率処理を導入した洪水法で生成された流水線は、「洪水」によって「水没」する地域（峡谷の上流の盆地など）はもちろんのこと、緩勾配の平野部でも現実の河川水路とはその位置が一致しないという欠点がある。たとえば国土地理院のCD-ROM版の250m-DEMでは、琵琶湖に洪水を起こすと、宇治川の峡谷部でDEM上の峡谷が埋まっているので、関ヶ原の峠(190m)を越えて溢流してしまう。古いMT版のDEM(KS110)ではそのようなことはなかった。利根川はどちらの版でも南流して東京湾に注ぐ。水は低い方に流れるというアルゴリズムと堤防のデータの入っていないDEMでは利根川を東流させることは出来ない。堤防の形状が読めるよ

うな詳細なDEMがあれば話は別である。

なお旧版のMT版KS110 (250m-DEM) では格子点の標高であったが、CD-ROM版ではその格子点を西南の角とする格子の中央点の値として計算し直している。この時の内挿法 (山田ほか, 1997) のため、峡谷部で谷埋めの傾向が強まったDEMとなっている。

5. 流水線追跡 (DDM生成) プログラムの実際

DDMを作成するアルゴリズムを記載した論文はあるが、実際のプログラムリストを掲載したものはない。技術の進歩や蓄積が研究の進展をもたらすという考えから、ここにプログラムのソースリストを掲載する。アルゴリズムの記述しかない場合、それから完全に作動するプログラムを作成するまでにはかなりの労力を要するであろう。

開発言語:

掲載したリストは Microsoft社の Visual C++ Ver.5.0用となっている。VC++にはWindowsのDOS窓で実行できる32ビットのコンパイラ (cl) が含まれている。Windows95やNTでは仮想記憶が使えるのでこのプログラムのような大きな配列を使うことができる。ワークステーションのUNIXのCでもincludeするライブラリ (のパス) を変更する程度でほとんどこのままコンパイル可能である。このプログラムは2次元配列のために多量のメモリを必要とする。2バイト整数を用いれば、やや精度が悪くなるがメモリを節約できる。

アルゴリズムおよびプログラムの要点:

もともになるDEMは2バイト符号無し整数のバイナリファイルで、海に0xFFFFの値を当てている。これと異なるDEM (例えばアスキーファイルでしかも海を-9999などに指定しているもの) の場合はフォーマットを予め変更しておく必要がある。

国土地理院のCD-ROM版DEMの場合、1m単位の表示なので富士山の標高値でも2バイト整数の範囲に収まる。しかし250mについて1m以下の勾配 (1/4000) の沖積低地は普通に存在するので、洪水型流水線追跡の精度を保つために、このプログラムでは内部的には4バイト整数を用いて精度を確保している。すなわち標高値を #define 文の MLT で指

定する分だけ倍率をかけて、4バイト整数としている。MLTの値は250m-DEMでは200、50m-DEMでは1000位が望ましい。勾配の小さな沖積低地を含まない山地流域だけを扱う場合は、この値を10として2バイト整数のDEMとして良いことはもちろんである。この方が計算時間を短縮でき、メモリも少なくて済む。

なお地図の4つの縁には予めそこに海のコードと値を埋め込んでいる。従ってその1つ内側の点の流水線方向は信頼できない。流域が地図の外に広がっている場合は流域面積などは当然計算できないわけであるが、このプログラムが生成するDDMでは地図の縁の近くに分水嶺があるかのようにになっているので注意が必要である。

DDMでは海に0xFFを当て、8方位のコードには、コンピュータのテンキイボード配列の数字 (5の位置から見たもの) を用いた。8の方向がN、6の方向がEである。この方法では流水線が衝突する場合はコードの和が10になるので、DDMを利用するプログラムでそれを検出する部分が若干簡単に記述できることが利点となっている。

プログラムの内部では、海または地図の縁に連なる流水線は確定した流水線であるとして、この8方位コードに100を加えて (従ってその値は101-109となる)、海とともにそれ以後の探索の対象から外すようにしている。全ての点のDDMの値が100以上になったとき、探索が終了する。

凹陷値を持つ点や全く平坦な点で「洪水」をおこした場合 (標高に1を加えた場合のこと)、それに隣接する点の流線方向に影響が及ぶので、DDMの値をクリヤして (0を代入して) 探索の対象に編入している。

8方位を使用しているため、流水線の交差を防ぐ必要があり、そのためのルーチンも加えられている。

探索の主要ルーチンはC言語ではあまり推奨されないgoto文でループを抜け出している。このようなルーチンをgoto文を使わずに関数で記述することもできるが、関数呼び出しのためのオーバーヘッドが生ずるなどして計算時間が長くなる。このプログラムでは計算時間の短縮を一つの目標にしている。

プログラムのソースリストには注としてもっと詳細な説明を付している。

参考文献

- 野上道男 (1991a): 地理情報とくにラスタ型数値地図利用の高度化. *Geoinformatics (情報地質)*, 2, 331-339.
- 野上道男 (1991b): 等高線図から DEM を作成する方法. *地図*, 29(3), 20-26
- 野上道男 (1995): 細密DEMによる地形計測. *地理学評論*, 68A, 465-474
- 野上道男・杉浦芳夫 (1986): 数値地理学演習. 古今書院, 275pp.
- 吉山 昭 (1990): DEM を用いた流域の地形量の計測. *地学雑誌*, 99, 136-143
- 吉山 昭 (1994): 河道の勾配と流域の地形特性量との関係――国土数値情報を用いた計測―― *地理評*, 67A, 794-802
- 山田美隆・前島正吉・村上広史 (1997): DCW DEMの精度評価. *G I S - 理論と応用*, 5-3, 53-58
- Chorowicz, J., C. Ichoku, S. Riazanoff, Y.-J. Kim and B. Cervelle (1992): A combined algorithm for automated drainage network extraction. *Water Resources Research*, 28, 1293-1302
- Jenson, S. (1991): Applications of hydrologic information automatically extracted from digital elevation models. *Hydrol. Processes* 5, 31-44
- Marks, D., J. Dozier and J. Frew (1984): Automated basin delineation from digital elevation data. *Geo-Processing* 2, 299-311
- McComack, J. E., M. N. Gahegan, S. A. Roberts, J. Hogg and B.S. Hoyle (1993): Feature-based derivation of drainage networks. *Int. J. Geogr. Information System*, 7, 263-279
- Nogami, M. (1989): Numerical Land Information in Japan. *Transact. Jour. Geomorphological Union*, 10-A, 147-156
- Nogami, M. (1995): Geomorphometric measures for digital elevation models. *Zeitschrift fur Geomorphologie*, N.F. Suppl.-Bd. 101, 53-67
- Pike, R. J. (1993): A bibliography of geomorphometry with a topical key to the literature and an introduction to the numerical characterization of topographic form. *U.S. Geol. Survey Open-file report* 93-262A

```

/*****
* calddm.c      DDM（流水線図）の作成
*
*      coded by M. Nogami, coded 29 Dec.,'94      rev. 10 Sept.,'97
*****/

#include <stdio.h>    // このプログラムはDEMからDDMを作成する.
#include <fcntl.h>    // (WE*NS*5) バイトの配列用メモリを使用している
#include <sys/types.h> // このままで、Visual C++ Ver. 5.0 でコンパイル可能
#include <sys/stat.h>
#include <io.h>
#include <process.h>
#include <stdlib.h>

// ----- 以下を書き換えてコンパイルする -----
#define fin "d:tak.dem" // 入力DEMのファイル名（2 Bバイナリファイル）
#define fout "tak.ddm" // 出力DDMのファイル名（1 Bバイナリファイル）
#define fout1 "tak.fld" // 出力flooded DEM（2 Bバイナリファイル）
#define WE 6400        // 地図の大きさ 東西方向のピクセル数
#define NS 6400        // 南北方向のピクセル数
#define MLT 1000       // 内部処理でDEM値を何倍にするか
#define WH 5           // 洪水の波の高さ（経験値）
// ----- ここまでを書き換える -----
#define SEA 0xFFFF     // 2 バイトファイルの海のコード
#define sea 0xFF        // 1 バイトファイルの海のコード
int      dem[NS][WE];  // 内部ではDEMに 4 バイト整数を用いる
unsigned char ddm[NS][WE]; // DDMコードはテンキーボード配列
unsigned short int buf[WE]; // 8: N, 9: NE, 6: E, 3: SE
unsigned long int count, repeat, lcount; // 2: S, 1: SW, 4: W, 7: NW
// 未処理点: 0  確定点: +100
void detection(int, int); // 方向を探索する関数. 引数は座標値

void main(void)
{ int fdi,fdo,fdo1;
  int m,n;
  if((fdo=open(fout, O_CREAT | O_RDWR | O_BINARY, S_IWRITE))== -1)
    { perror(fout); exit(1); }
  if((fdo1=open(fout1, O_CREAT | O_RDWR | O_BINARY, S_IWRITE))== -1)
    { perror(fout1); exit(1); }
  if((fdi=open(fin, O_RDONLY | O_BINARY))== -1)
    { perror(fin); exit(1); }
  for (n=0; n<NS; n++)
    { read(fdi,(char *)buf,WE*2);
      for (m=0; m<WE; m++) // 海のコードと MLT 倍値の設定
        if (buf[m]==SEA || buf[m]==0) // 0 m地帯は海と見なす
          { dem[n][m]=0; ddm[n][m]=sea; }
    }
}

```

```

    else { dem[n][m]=(int)buf[m]*MLT; ddm[n][m]=0; } // 未処理値の設定
}
close(fdi);
for (n=0; n<NS; n++) // 地図の縁を海として処理する
    { dem[n][0]=0; dem[n][WE-1]=0; ddm[n][0]=sea; ddm[n][WE-1]=sea; }
for (m=0; m<WE; m++)
    { dem[0][m]=0; dem[NS-1][m]=0; ddm[0][m]=sea; ddm[NS-1][m]=sea; }
repeat=0L;
do
{ count=0L;
  if (repeat%10 == 0) printf("¥ncount=%d repeat=%d¥n",count,repeat);
  repeat++;
  if (count==0L)
  { for (n=1; n<NS-1; n++) // 全点から走査探索を行う
    for (m=1; m<WE-1; m++)
      if (ddm[n][m]<=100) detection(n,m);
  }
} while(count != 0); // 全ての点が確定点になるまで繰り返す
for (n=1; n<NS-1;n++) // 確定点のコードをもとのコードに戻す
  for (m=1; m<WE-1;m++)
    if (ddm[n][m] != sea) if (ddm[n][m]>100) ddm[n][m] -= 100;
    else printf("¥nError Data found %d¥n",ddm[n][m]);
write(fdo,(char *)ddm,WE*NS); close(fdo);
for (n=0; n<NS; n++) // D E M (洪水後) の書き出し
{ for (m=0; m<WE; m++)
  if (ddm[n][m]==sea) buf[m]=SEA;
  else buf[m]=(unsigned short int)((dem[n][m]+MLT/2)/MLT);
  write(fdo1, (char *)buf, WE*2);
}
close(fdo1);
}

void detection(int n, int m)
{ int kn,km,wn,wm,k,j,num,rnum;
  unsigned char drc, ndrc, dir[9];
  int dat, mindat, wdat;
  wn=n; wm=m;
ENT: // 方向探索ループの入り口
  if (count%1000000L==0) printf("*");
  if (wn==0 || wn==(NS-1) || wm==0 || wm==(WE-1)) return;
  if (ddm[wn][wm]>100) return; // 海および確定点の場合はスキップ
  else count++; // 未処理および未確定点の数を数える
  dat=dem[wn][wm]; num=0;
  if (ddm[wn-1][wm+1]>100 && dem[wn-1][wm+1]<dat) { dir[num]=109; num++; }

```

```

if (ddm[wn][wm+1]>100 && dem[wn][wm+1]<dat) { dir[num]=106; num++; }
if (ddm[wn+1][wm+1]>100 && dem[wn+1][wm+1]<dat) { dir[num]=103; num++; }
if (ddm[wn+1][wm]>100 && dem[wn+1][wm]<dat) { dir[num]=102; num++; }
if (ddm[wn+1][wm-1]>100 && dem[wn+1][wm-1]<dat) { dir[num]=101; num++; }
if (ddm[wn][wm-1]>100 && dem[wn][wm-1]<dat) { dir[num]=104; num++; }
if (ddm[wn-1][wm-1]>100 && dem[wn-1][wm-1]<dat) { dir[num]=107; num++; }
if (ddm[wn-1][wm]>100 && dem[wn-1][wm]<dat) { dir[num]=108; num++; }
if (num!=0) // そこより低いところに確定点発見
{ rnum=rand()%num; ddm[wn][wm]=dir[rnum];
return; // 乱数で選んでそこも確定点となる
}

num=0;
if (ddm[wn-1][wm+1]>100 && dem[wn-1][wm+1]==dat) { dir[num]=109; num++; }
if (ddm[wn][wm+1]>100 && dem[wn][wm+1]==dat) { dir[num]=106; num++; }
if (ddm[wn+1][wm+1]>100 && dem[wn+1][wm+1]==dat) { dir[num]=103; num++; }
if (ddm[wn+1][wm]>100 && dem[wn+1][wm]==dat) { dir[num]=102; num++; }
if (ddm[wn+1][wm-1]>100 && dem[wn+1][wm-1]==dat) { dir[num]=101; num++; }
if (ddm[wn][wm-1]>100 && dem[wn][wm-1]==dat) { dir[num]=104; num++; }
if (ddm[wn-1][wm-1]>100 && dem[wn-1][wm-1]==dat) { dir[num]=107; num++; }
if (ddm[wn-1][wm]>100 && dem[wn-1][wm]==dat) { dir[num]=108; num++; }
if (num!=0) // 平坦なところに確定点発見
{ dem[wn][wm]++; count++; // そこに流れるように、標高を 1 あげて、
for (k=-1; k<2; k++) // 周囲の流線方向を 0 クリヤして
for (j=-1; j<2; j++)
if (ddm[wn+k][wm+j]<100) ddm[wn+k][wm+j]=0;
rnum=rand()%num; ddm[wn][wm]=dir[rnum];
return; // しかしその方向だけはセットする
}

if (ddm[wn][wm]>0) goto LOOPENT; // 未確定点では探索は不要だから、次に進む
mindat=SEA*MLT; num=0; // 未処理点では初期値設定後、8 方位探索の開始
wdat=dem[wn-1][wm+1]; // NE
if (wdat<mindat) { dir[0]=9; mindat=wdat; num=1; }
else if (wdat==mindat) { dir[num]=9; num++; }
wdat=dem[wn][wm+1]; // E
if (wdat<mindat) { dir[0]=6; mindat=wdat; num=1; }
else if (wdat==mindat) { dir[num]=6; num++; }
wdat=dem[wn+1][wm+1]; // SE
if (wdat<mindat) { dir[0]=3; mindat=wdat; num=1; }
else if (wdat==mindat) { dir[num]=3; num++; }
wdat=dem[wn+1][wm]; // S
if (wdat<mindat) { dir[0]=2; mindat=wdat; num=1; }
else if (wdat==mindat) { dir[num]=2; num++; }
wdat=dem[wn+1][wm-1]; // SW
if (wdat<mindat) { dir[0]=1; mindat=wdat; num=1; }

```

```

else if (wdat==mindat) { dir[num]=1; num++; }
wdat=dem[wn][wm-1];          // W
if (wdat<mindat) { dir[0]=4; mindat=wdat; num=1; }
else if (wdat==mindat) { dir[num]=4; num++; }
wdat=dem[wn-1][wm-1];        // NW
if (wdat<mindat) { dir[0]=7; mindat=wdat; num=1; }
else if (wdat==mindat) { dir[num]=7; num++; }
wdat=dem[wn-1][wm];          // N
if (wdat<mindat) { dir[0]=8; mindat=wdat; num=1; }
else if (wdat==mindat) { dir[num]=8; num++; }
if (mindat==0)                // 最低点が海ならば、
{ rnum=rand()%num; ddm[wn][wm]=dir[rnum]+100;
  return;                      // 乱数で方向を選んで確定点とする
}
if (dat<=mindat)               // 凹陷地あるいは平坦地ならば
{ if (dat==mindat) dem[wn][wm]+=1+rand()%WH; // 平坦地なら波のある洪水発生
  else dem[wn][wm]=mindat; // 凹陷地なら周囲と同じ高さまでの洪水発生
  for (k=-1; k<2; k++)        // これまでの周囲の流線方向を0クリアして
    for (j=-1; j<2; j++)
      if (ddm[wn+k][wm+j]<100) ddm[wn+k][wm+j]=0;
  goto ENT;                   // 繰返しへ戻る
}
rnum=rand()%num; ddm[wn][wm]=dir[rnum]; // 行き先の方向を選択した
LOOPENT:
count++; lcount++;
if (lcount%1000000L==0) printf("."); // プログラムの進行を表示する
if (wn==0 || wn==(NS-1) || wm==0 || wm==(WE-1)) return;
kn=wn; km=wm; drc=ddm[wn][wm]; // 次の点への移動開始
if (drc==9) {wn--; wm++; }
else if (drc==6) { wm++; }
  else if (drc==3) {wn++; wm++; }
    else if (drc==2) {wn++; }
      else if (drc==1) {wn++; wm--; }
        else if (drc==4) { wm--; }
          else if (drc==7) {wn--; wm--; }
            else if (drc==8) {wn--; }
              else printf("¥nERR %d¥n",drc);
ndrc=ddm[wn][wm];
if (ndrc==0) goto ENT;        // 次の点が、未処理点なので、繰返しへ戻る
if (ndrc>100) { ddm[kn][km] += 100; return; } // 確定点なので、そこも確定
goto ENT;                     // いずれでもないので探査を続ける
}

```