平成 15 年度　資源環境学特論Ⅱ

講義のテーマ
「地理情報システム GRASS を用いて愛知県の植生分布の特徴を把握する」

目的

地理情報システム GRASS (http://grass.itc.it/index.html)は、GIS 解析に関するコマンドに加え、リモートセンシング・データなどの画像解析に関する基本的なコマンドも有するハイブリッドな GIS ソフトです。今回は、この GRASS を用い、公開されている植生情報と衛星画像から愛知県の詳細な植生分類図を作成し、他の空間情報（今回は標高データ）との比較・検討を行うプロセスにより、

- GRASS のデータ構造の把握
- ベクトルデータの編集
- 空間統計処理
- リモートセンシングデータの取り扱い
- 教師付き分類

等についての習得を目的とします。

使用データ

今回は以下に示す公開空間情報と衛星画像を用いる。

１）公開空間情報

- 国土数値情報（公共施設，自然地形メッシュ，行政界・海岸線(面)）
- 自然環境保全基礎調査データファイル

２）リモートセンシング・データ

- MODIS/Terra リモートセンシング画像データ（空間分解能約 250m，2001 年 7 月および 11 月）

使用テキスト

「Ruby プログラミング入門」原　信一郎著　まつもと　ゆきひろ監修　オーム社（￥2,800）

１．GRASS の起動とデータベースの初期設定

GRASS は、ターミナルウィンドウのコマンドライン（図１－１）上で

grass53[Enter]

とタイプして起動します。その後ターミナルウィンドウに現れるのが、図１－２の起動画面です。ここでは、以下の３つの項目を入力しなければなりません。

- LOCATION
- MAPSET
- DATABASE

各項目は図１－２（下）に示した GRASS のデータ構造に対応しており、この設定により

DATABASE/LOCATION/MAPSET

というディレクトリ（Windows や Mac ではフォルダと呼ばれる）が作成されます。

図1－1．ターミナルウィンドウ

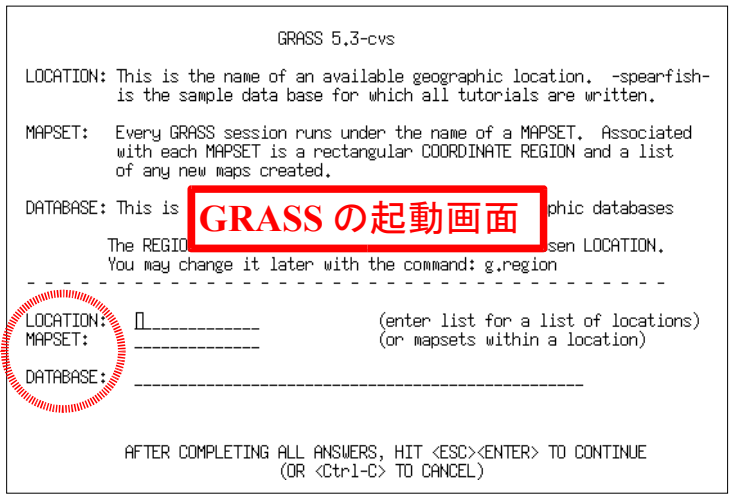


図1－2．GRASS 起動画面（左）と GRASS のデータ構造（下）

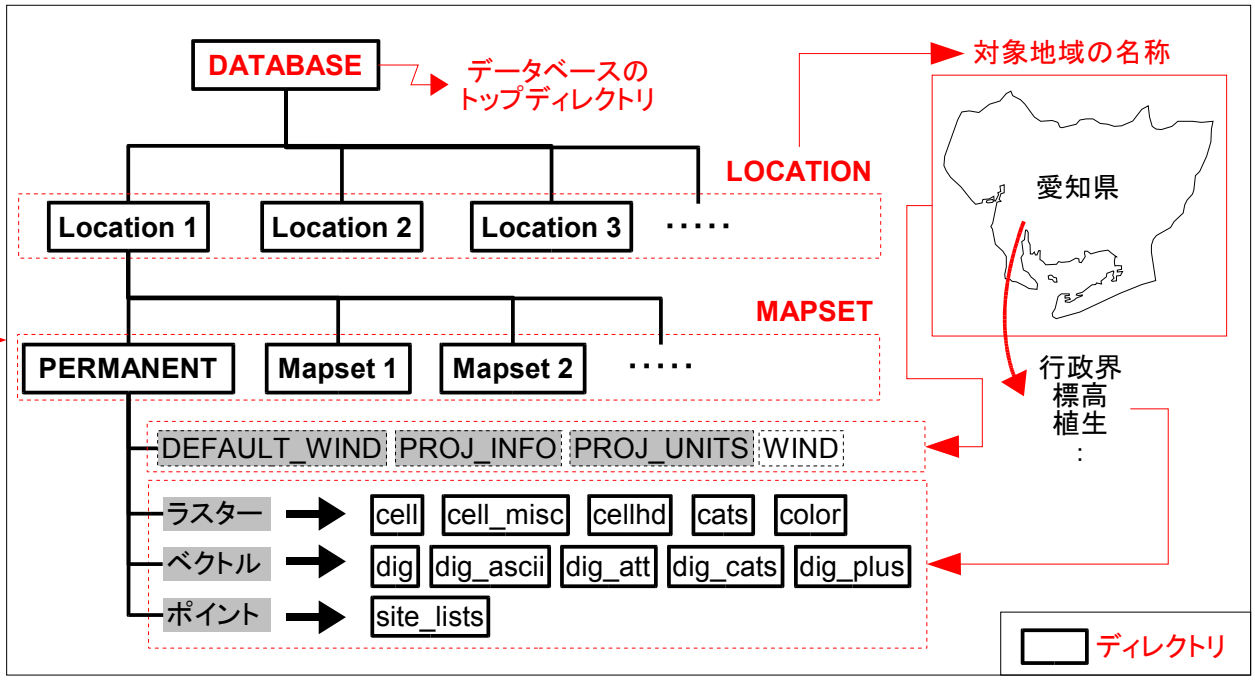


図１－２（下）に示した GRASS のデータ構造にあるように、各 LOCATION の下には必ず PERMANENT という MAPSET が作成され、この中に、

- DEFAULT_WIND　：LOCATION の東西南北端の座標等
- PROJ_INFO　　：投影情報（UTM、longitude-latitude、・・・）等
- PROJ_UNITS　　：地図の単位（度、m、・・・）等

という３つのファイルが作成されます。また、特に指定しない限り、これらの情報が同一 LOCATION の他の MAPSET にも適用されます。これらの情報は、新たな LOCATION を作成する場合、図１－２（上）の起動画面で、LOCATION・MAPSET・DATABASE に適切な設定を入力し、[ESC]→[Enter]とタイプすると、上記３つのファイルを作成するためのメッセージが順次現れますので、それに答えていけば３つのファイルが自動的に作成されます。

これら３つのファイルは、PERMANENT という MAPSET にのみ作成され、他の MAPSET には作成されません。また、GRASS 上でこれらの内容を変更することはできません。

一方、LOCATION 内に設定された解析対象地域を通常 Region と呼び、GRASS 上で LOCATION 内の任意の四角領域を設定することができ、その東西南北端の座標は WIND ファイルに書き込まれます。また、LOCATION 作成時は、DEFAULT_WIND ファイルの内容が WIND ファイルにコピーされ、Region は LOCATION と等しいということになります。これは、その LOCATION の下に新たな別の MAPSET を作成した場合も同様です。

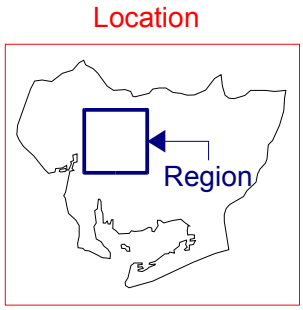


図１－３．Location と Region

１－１．愛知県を対象とする緯度経度座標系の LOCATION 作成

愛知県の東西南北端点の経度緯度は表１－１のとおりです。表２－３にある標準地域メッシュを単位として、この区画が収まる領域の東西南北端の座標は以下のとおりです。

- 経度方向：西端 136° 39′ 45″　東端 137° 51′ 00″
- 緯度方向：南端 34° 34′ 00″　北端 35° 25′ 30″

表１－１．愛知県の東西南北端点の経度緯度（国土地理院）

区　分	世界測地系(WGS84)		日本測地系(Bessel)	
	経度	緯度	経度	緯度
東　端	137° 50′ 17″	35° 12′ 45″	137° 50′ 28″	35° 12′ 33″
西　端	136° 40′ 15″	35° 08′ 31″	136° 40′ 26″	35° 08′ 19″
南　端	137° 02′ 18″	34° 34′ 38″	137° 02′ 29″	34° 34′ 26″
北　端	136° 59′ 24″	35° 25′ 30″	136° 59′ 35″	35° 25′ 18″

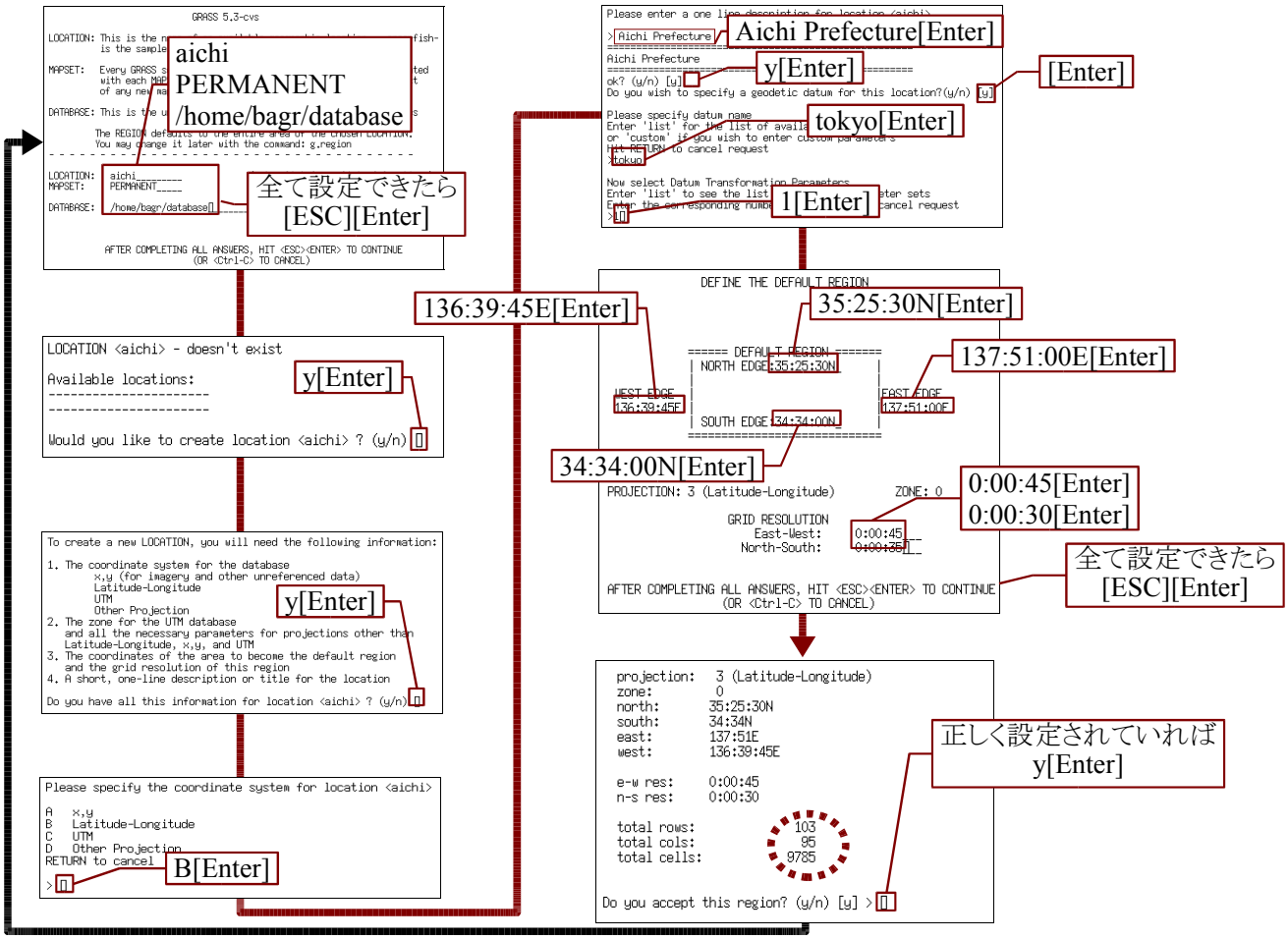


図1－4. 緯度経度座標系による愛知県の LOCATION 作成手順

愛知県を対象とする緯度経度座標系の LOCATION 作成手順は図１－４のとおりです。ただし、LOCATION 作成前にデータベースのトップディレクトリ(/home/bagr/database)を作成しておく必要があります。ディレクトリの作成は以下のコマンドにより行います（※カレントディレクトリが/home/bagr/である場合）。

mkdir database

図１－４の操作を完了すると最初の画面に戻りますので、[Esc][Enter]で図１－５のようにGRASSが起動します。正しくGRASSが起動すると、コマンドラインの左側の文字が

GRASS:~>

と変化しているはずですので、確認して下さい。

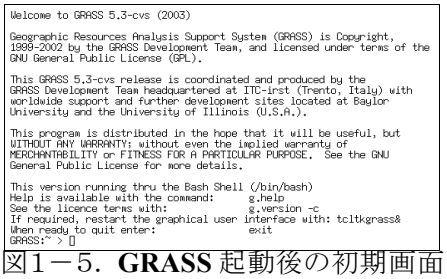


図1－5．**GRASS** 起動後の初期画面

１－２．愛知県を対象とする UTM 座標系の LOCATION 作成

今回利用するリモートセンシング・データ（MODIS/Terra）の空間解像度は約 250m です。今度は、緯度経度座標系ではなく、投影座標である UTM 座標系により、メッシュサイズ 250m の LOCATION を作成します。そのためには、まず UTM 座標系における愛知県の東西南北端の座標を調べる必要があります。今回は、表１－１の情報をもとに GRASS で緯度経度から UTM 座標を計算します。

まず、図1－5のコマンドラインで、以下のコマンドによりテキストエディタを起動します。

gedit[Enter]

テキストエディタ（図１－６)が起動したら、以下のように表１－１の座標情報を入力し（▽は半角スペース）、

137:50:28E▽35:12:33N▽E
136:40:26E▽35:08:19N▽W
137:02:29E▽34:34:26N▽S
136:59:35E▽35:25:18N▽N

aichi.in というファイル名で保存します。

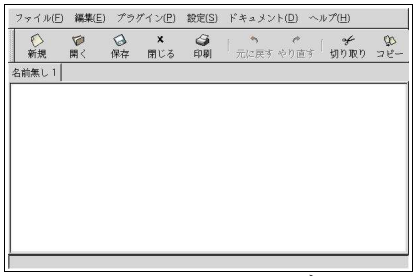


図1－6．テキストエディタ

次に、以下のコマンドラインにコマンドを入力し、実行します。

m.ll2u▽spheroid=wgs84▽zone=53▽input=aichi.in▽output=aichi.out[Enter]

結果は aichi.out というファイルに保存されていますので、以下のコマンドで内容を確認します。

cat aichi.out[Enter]

表示された内容から、250m 単位での愛知県が収まる東西南北端の座標は以下のようになります。

- 経度方向：西端 652250　東端 758750
- 緯度方向：南端 3827500　北端 3921750

上記の情報をもとに、UTM 座標系の LOCATION 作成を行いますが、新たな LOCATION 作成のためには一旦 GRASS を終了する必要があります。GRASS を終了するためには、以下のコマンドを実行します。

exit[Enter]

再度 GRASS を起動し、図１－７の手順で UTM 座標系の LOCATION を作成します。ここでも、図１－５の画面になったら、再度 GRASS を終了します。

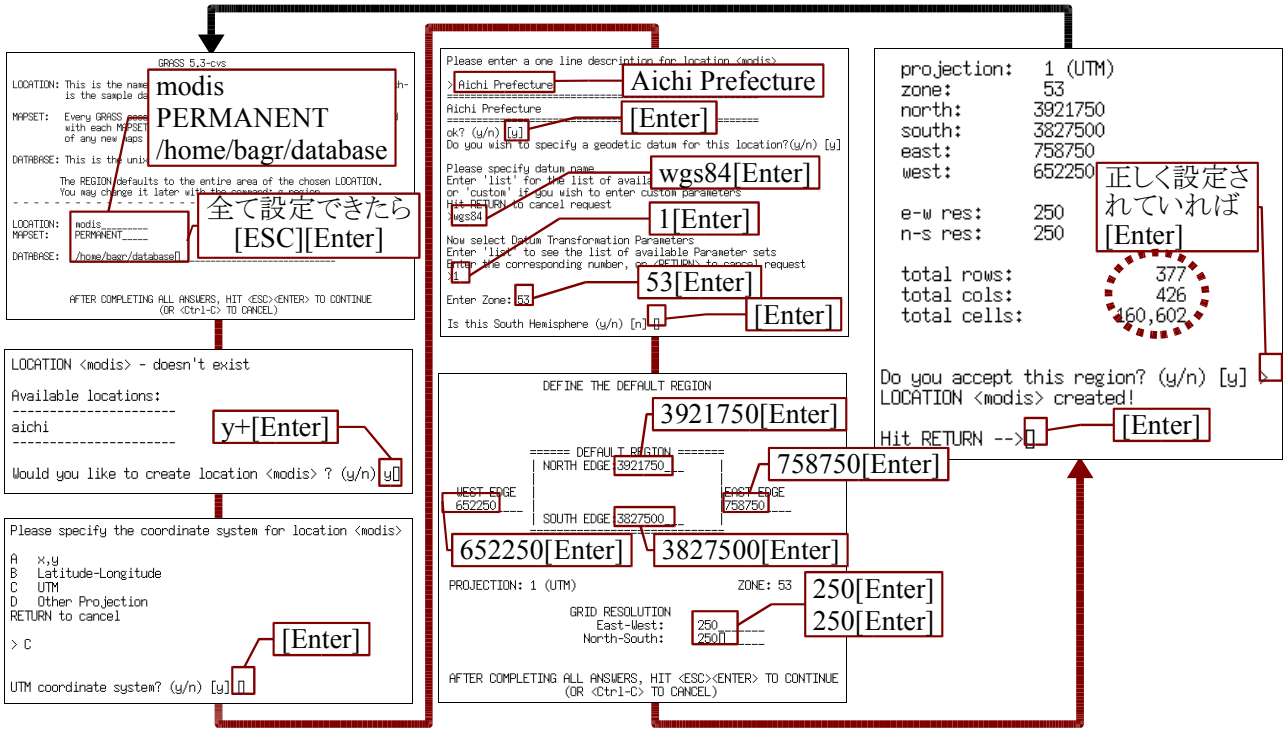


図1－7．**UTM** 座標系による愛知県の LOCATION 作成手順

２．国土数値情報

国土数値情報は、国土情報整備事業によって作成されたデジタルデータであり、国土庁が発足した昭和49年度より作成が開始され、現在でも作成・更新が続けられています。国土数値情報は、本来全国総合開発計画、国土利用計画などの国土計画の策定や実施の支援のために作られたものですが、現在ではインターネット（ http://nlftp.mlit.go.jp/ksj/ ）による無償提供が開始されています。今回は、表２－１に示した国土数値情報のデータ項目のうち３つ（赤い文字で表記）を利用します。

表２－１．国土数値情報のデータ項目

指定地域	指定地域メッシュ	三大都市圏計画区域（面）	森林・国公有地メッシュ
	都市計画区域（面）	自然公園（面）	自然環境保全区域（面）
	農業地域（面）	森林地域（面）	鳥獣保護区域（面）
	リゾート法指定地域（面）		
沿岸域	潮汐・海洋施設（点）	港湾（点）	沿岸海域メッシュ
	波向・海霧・自然漁場２次メッシュ	海岸施設・感潮限界（点）	高潮・津波テーブル（表）
	漁港（点）	増養殖施設（線）	漁礁（線）
	海底敷設線・架空線（線）	架橋（線）	環境基準類型あてはめ水域（線）
	生活環境項目（表）	漁港区域（線）	港湾区域（線）
	港域（線）	航路（線）	漁業権設定区域（線）
	鉱区（線）	海岸線（線）	海岸線台帳（表）
	海水浴台帳（表）	埋立・干拓区域（線）	埋立・干拓区域台帳（表）
	空港区域（線）	空港台帳（点）	砂利採取場（線）
	自然公園区域（線）	国土保全関連情報（線）	保護水面台帳（線）
	低地地形分類（線）	地盤沈下地域（線）	感潮限界（点）
	地下水採取規制地域台帳（線）	瀬戸内海環境保全特別措置法第五条第一項の地域（線）	環境基準類型指定水域（河川）（線）
	環境基準類型指定水域（河川）台帳（表）	環境基準類型指定水域（湖沼）（線）	環境基準類型指定水域（湖沼）台帳（表）
	大気汚染・水質汚濁総量規制地域（線）	大気汚染・水質汚濁総量規制地域台帳（表）	海岸利用施設（点）
	験潮場（点）	河川区域台帳（表）	保安林区域台帳（表）
自然	気候値メッシュ	標高・傾斜度メッシュ	土地分類メッシュ
	山岳メッシュ	谷密度メッシュ	
土地関連	地価公示（点）	都道府県地価調査（点）	土地利用メッシュ
国土骨格	道路（線）	鉄道	行政界・海岸線（面）
	道路密度・道路延長メッシュ		
施設	文化財（点）	公共施設（点）	発電所（点）
産業統計	商業統計３次メッシュ	商業統計４次メッシュ	工業統計メッシュ
水文	ダム（点）	河川・水系域テーブル（表）	湖沼メッシュ
	水系域流路延長（点）	流路延長メッシュ	流域・非集水域メッシュ
	湖沼（面）	湖沼台帳（表）	湖岸線（面）
	流域界・非集水域（面）	河川台帳（表）	単位流域台帳（表）
	流路（線）		

２－１．ポイントデータのGISデータ化

１）固定長データとは

国土数値情報は基本的に固定長データとして提供されています。まずは、これを適切に切り分ける必要があります。以下では国土数値情報（公共施設）の愛知県データ（P02_02p_23.txt)を使います。

固定長データというのは、

P 513760 4932956 124495823623 10010 1 20 地方卸売市場

P 513760 4933477 124474223623 10220 0 00 その他

といった感じのデータで、その行の何桁目であるかに意味があります。仕様書(P02-02P.htm)によると、表２－２のようになります。

表２－２．国土数値情報（公共施設）データのファイルフォーマット

項目	記述形式	開始桁	終了桁	内容
レイヤコード	A3	1	3	P
メッシュコード	I6	4	9	513760
X座標	I8	10	17	4932956
Y座標	I8	18	25	1244958
:		:	:	

＊記述形式：A→ローマ字、I→整数、N→日本語。A・I・Nの後ろの数値は桁数

＊国土数値情報のＸＹ座標系：

国土数値情報では、緯度・経度を座標系として、東経をX、北緯をYとして0.1秒単位で表現しています。なお、原点は、東経０°、北緯０°です。

例えば、東経130度30分30秒、北緯48度30分30秒の場合の国土数値情報データファイルでのＸＹ座標値の値は

Ｘ座標 = 4698300(=(130度×60分×60秒＋30分×60秒＋30秒)×10)

Ｙ座標 = 1746300(=(48度×60分×60秒+30分×60秒+30秒)×10)

となります。

２）固定長データの取り扱いに必要な **Ruby** の文法

国土数値情報（公共施設）の場合では1行に18のデータが含まれており、他のデータになるともっと多くなります。Excel上で、【区切り位置の指定】の機能を使うという手もありますが、数が多くなると手作業で行うには大変な労力がかかりますし、65536行を超えると読み込むこともできません。

この公共施設のデータをRubyで分解するためには、まず以下のRubyの文法を理解する必要があります。

①配列

配列は、要素(オブジェクト)を複数まとめて入れる"入れもの"です。例えば以下のスクリプト

```
1：a = [ ]          #配列の初期化
2：a[0]=1           #配列の最初の入れ物に「１」という数値を代入
3：a[1]="Test"      #配列の２番目の入れ物に「Test」という文字を代入
4：p a
```

を実行してみると、

```
[1, "test"]
```

という結果が得られます。この[ ]で囲まれたものが配列で、その中で「,」で区切られた数値や文字などのオブジェクトが要素となります。各要素は「変数名[要素の番号]」という指定で各要素を個別に調べたり、変更したり、追加したりすることができます。

このように、配列は番号０から始まる一続きのオブジェクトを意味します。配列変数を使用する場合は、最初にその変数が配列であることを定義するために、変数に空の配列[ ]を代入します。また、

```
1：a=[1,2,3]
```

のように、空の配列[ ]の代わりにa[0], a[1], a[2]にそれぞれ初期値を代入してもかまいません。なお、値の代入されていない番号の要素を表示させると「nil」という値がかえってきます。例えば、上のスクリプトの３行目を以下のように変更して実行してみてください。

```
３：a[10]="Test"   #配列の10番目の入れ物に「Test」という文字を代入
```

また、各要素はオブジェクトですから、その中にさらに配列を代入することも可能です。例えば、

```
1：a = [ ]          #配列の初期化
2：a[0]=1           #配列の最初の要素に「１」という数値を代入
3：a[1]=[0,1,2]     #配列の次の要素に「0,1,2」という配列を代入
4：p a
```

を実行してみてください。

さらに、配列の中の配列の各要素を個別に調べたり、変更したり、追加したりすることも可能です。例えば、上のスクリプトを以下のように変更して実行してみてください。

```
1：a = [ ]          #配列の初期化
2：a[0]=1           #配列の最初の要素に「１」という数値を代入
3：a[1]=[0,1,2]     #配列の次の要素に「0,1,2」という配列を代入
4：p a[1][1]
5：a[1][1]=10
6：a[1][3]=1
7：p a[1]
```

② 正規表現

Ruby では正規表現というものが多用されます。正規表現とは、文字列のパターンを表現するものです。あるルールにしたがった文字列を探すときのルールを表現するものと考えてもよいでしょう。例えば、「"FOO"で始まって"R"で終る文字列」というようなルールです。ちなみにこの「ルール」を表現する正規表現は以下のようになります。

```
/^FOO.*R$/
```

Ruby では // に囲まれた部分が正規表現です。^は「先頭」、$は末尾を意味し、.*は任意の文字の 0 個以上の並びを意味します。

では、正規表現のルールの書き方を説明しましょう。正規表現には上で説明したような特別な意味を持つ文字がいくつかあります。まずこれをあげておきます。

```
[ ]    : 文字範囲指定. [a-z]は a から z までのいずれか
¥w     : 英数字. [0-9A-Za-z_]と同じ
¥W     : 非英数字
¥s     : 空白文字. [ ¥t¥n¥r¥f]と同じ
¥S     : 非空白文字
¥d     : 数字. [0-9] と同じ
¥D     : 非数字
¥b     : 語境界文字(範囲指定外)
¥B     : 非語境界文字
¥b     : 後退(0x08)(範囲指定内)
.      : 直前の表現の 0 回以上の繰り返し
+      : 直前の表現の 1 回以上の繰り返し
{m,n}  : 直前の表現の m 回から n 回の繰り返し
?      : 直前の表現の 0 または 1 回の繰り返し
|      : 選択
( )    : 表現をまとめる
それ以外       : その文字そのもの
```

例えば、「^f[a-z]+」は「f からはじまる a から z までの文字の繰り返し」であり、"foobar"や"fool"などと一致します。こういう役のある一致を正規表現(regular expression)と呼びます。正規表現は文字列の検索の時に役に立つので、広く使われています。

If 文の条件式において文字列の正規表現マッチを行う場合は、演算子「＝～」を使います。例えば、

```
1 : a = [″Test″,″Program″,″program″ ]     #配列の初期化
2 : a.each  do |str|
3 :    if /^P/ =~ str   #正規表現マッチ（str ｵﾌﾞｼﾞｪｸﾄが文字列で、その最初の文字が「P」であるか）
4 :    puts str
5 :    end
6 : end
```

を実行してみると、

```
Program
```

という結果が得られます。このように、正規表現マッチでは文字が示す意味を比較しているのではなく、文字が示す文字コードにより比較を行っているので、異なる文字コードを持つ「P」と「p」は異なる文字と判断されます。また、上のスクリプトにある

```
配列.each  do |オブジェクト名|
      文
end
```

という表現は、配列の最初の番号（０）の要素から、最後の番号の要素まで、順にオブジェクト名で指定された変数に代入して、end との間に記述された内容を実行する場合に利用される、配列を扱う場合の表現です。

③ 文字列の **unpack**

「配列オブジェクト＝文字列オブジェクト.unpack(template)」により、文字列オブジェクトに記憶されている内容を template 文字列で指定した形式で配列に分解します。例えば、

```
1 : str = ″P▽▽513760493295612449 58″  注）▽は半角スペース
2 : fmt = ″A3A6A8A8″
3 : a=str.unpack(fmt)
4 : a.each  do |b|
5 :  puts  b
6 : end
```

を実行してみると、

```
P
513760
4932956
1244958
```

という結果が得られます。３行目の「"A3A6A8A8"」において、「A」は「ASCII 文字列（スペースを詰め、後続する null 文字やスペースを削除）」を、その後ろの数値はその長さを表します。したがって、３行目は、str オブジェクトの内容を fmt の内容に従って、４つ（「A3」「A6」「A8」「A8」）の文字列要素からなる配列に分解することを意味しています。

④ 文字列変換
　文字列オブジェクトには、その内容を数値に変換する method があります。例えば
```
1：str = ″12″
2：a=str+10
3：puts a
```
では、文字と数値の計算はできないのでエラーとなりますが、
```
1：str = ″12″
2：a=str.to_i+10
3：puts  a
```
を実行してみると、結果は 22 と出力されます。
　一方、
```
1：str = ″12″
2：a=str.to_f+10
3：puts  a
```
を実行してみると、結果は 22.0 と出力されます。
　このように、「 文字列オブジェクト.to_i 」は文字列を整数へ、「 文字列オブジェクト.to_f 」は文字列を実数へと変換する method です。

⑤ 文字列オブジェクト中の特定文字の削除
　文字列オブジェクト中の内容で、特定の文字を削除したい場合があります。例えば、余分なスペース文字などを削除して、ファイルのサイズを抑えたい場合などが、これに当たります。これを行うための method が文字列オブジェクトには用意されています。例えば、
```
1：a=″test″
2：print a.delete(″t″)
```
とすると、「 es 」という結果が得られるはずです。このように、文字列オブジェクトの delete method は、引数で与えられた文字列中に含まれる文字を全て文字列オブジェクト中から削除する method です。つまり、delete で与える文字列中では、文字の並びは関係なく、どうゆう文字が含まれるかが問題となります。例えば、上のスクリプトで「 "t" 」を「 "ts" 」に代えて実行して見て下さい。

⑥ ファイルへの出力
　file をオープンする open 関数については既に学んでいますが、この関数、実は以下に示す３つの引数により様々な動作を行うことができます。
```
open(file[, mode[, perm]])
```
上記の３つの引数（「file」「mode」「parm」）のうち、第１引数の file は省略することはできませんが、後の２つは省略することができます。
第２引数の mode は、以下の文字列を指定し、省略時は "r" が指定されたものとみなします。
　″r″：ファイルを読み込みモードでオープンします。
　″w″：ファイルを書き込みモードでオープンします。オープン時にファイルがすでに存在していればその内容を空にします。
　″a″：ファイルを書き込みモードでオープンします。出力は常にファイルの末尾に追加されます。
また、"+" があれば、ファイルは読み書き両用モードでオープンされますが、各モードで以下のような違いがあります。
　″r+″：ファイルの読み書き位置は先頭にセットされます。
　″w+″：″r+″と同じですが、オープン時にファイルがすでに存在していればその内容を空にします。
　″a+″：″r+″と同じですが、オープン時にファイルがすでに 存在していれば読み書き位置がファイルの末尾にセットされ ます。
　最後の第３引数 perm は、Linux などでファイルのアクセス権を指定する整数ですが、今回は説明を省略します。例えば、
```
1：f = open(″c:¥¥ruby¥¥output.dat″,″w″)
2：a = 10+1
3：f.print a, ″¥n″
4：f.close
```
を実行してみると、「ｃドライブの ruby フォルダに output.dat というファイルが作成され、ファイルに２行目の結果である「１１改行」が出力されます。３行目の「 f.print 」は、ファイル変数 f の print という method で、出力先が画面ではなくて１行目の open 関数で指定されたファイルに対して行われますが、それ以外は関数としての「print」と同様の働きをします。なお、１行目の open 関数のファイル指定で、「″c:¥¥ruby¥¥output.dat″」と「 ¥¥ 」がドライブやフォルダの区切りとして用いられていますが、これは Windows 系 OS に固有の表現です。Windows 系 OS では、ドライブやフォルダの区切りは「¥」が用いられますが、文字列中ではこの「 ¥ 」は特別な意味を持ちます。例えば「 ¥n 」が改行を表すように「 ¥ 」と特定の文字を組み合わせて特殊なコードを表現しています。そのため、″″で囲まれた文字列中で文字としての「 ¥ 」を表現したい場合は、「 ¥¥ 」と「 ¥ 」を２つ重ねることになっています。一方、Linux では、フォルダの区切りは「 / 」が用いられるので「″/home/ruby/output.dat″」のように「 / 」を重ねる必要はありません。

⑦書式付出力
　特定の桁数や小数点以下の桁数等を固定して出力したい場合がよくありますが、print 関数ではそれを指定することはできません。この場合は、print 関数にかわり、以下の printf 関数（フ

ァイル変数では printf method）が利用できます。
printf(format[, arg[, …]])
この printf 関数の format は書式を表す文字列で、arg は書式中で指定された変数を表します。
例えば、
1：a = 2.0 / 3
2：eq = ″2 / 3″
3：printf(″%s = %5.2f¥n″, eq, a)
を実行すると、
2 / 3 = 0.67 改行
と出力されます。スクリプトの２行目と出力された結果を比べてみると、「 %s 」が「 2 / 3 」へ、「 %5.2f 」が「 0.67 」へと代わっていることがわかります。printf 関数の format 文字列内において、
% [幅][.精度]指示子（[ ]で囲まれた部分は省略可能）
（正しくは、%[引数指定$][フラグ][幅][.精度]指示子ですが、[引数指定$][フラグ]に関する説明は省略します。なお、両者は共に省略可能です）
幅　　：生成される文字列の幅
. 精度　：整数を表す指示子に対しては数値列部分の長さを意味し、浮動小数の指示子に対しては小数部の桁数を意味します。
指示子：
- 文字列を表す指示子　：c, s
- 整数を表す指示子　　：d, i, u, b, o, x, X,
- 浮動小数を表す指示子: f, g, e, E, G

で表された部分は、arg で指定された変数を指定の書式により生成された文字列に変更して出力されます。また、
1：a = 2.0
2：b = 3
3：printf(″%f / %d = %5.2f¥n″, a, b, a/b)
のように、複数の変数や演算式を変数として指定することも可能です。なお、このスクリプトを実行すると
2.000000 / 3 = 0.67
という結果が得られます。このように、「 %f 」の出力結果が示すとおり浮動小数で「[幅][.精度]」を省略した場合、小数点以下の桁数は６となります。また、小数点以下の桁数のみを指定したい場合は
3：printf(″%.2f / %d = %5.2f¥n″, a, b, a/b)
のように「[.精度]」のみを指定することも可能です。

３）固定長データの分解
まず、P02_02p_23.txt のファイル・フォーマットを記載した P02-02P.htm を開き、その分解に必要な template 文字列を考えると、以下のようになります。
fmt = "A3A6A8A8A5A5A2A2A2A1A30A60A80A5A30A4A4A5"
次に、通常 GIS では、緯度・経度は度・分・秒ではなくて、度単位で指定します。例えば、45 度 30 分 30 秒であれば、45+30/60+30/3600=45.508333･･･度と表現されます。
P02_02p_23.txt にある X 座標・Y 座標は、上述したように 0.1 秒単位ですので、これを下記のように度単位に変換しなくてはなりません。
経度 = X 座標 / 36000.0
緯度 = Y座標 / 36000.0
となります。
したがって、P02_02p_23.txt から各固定長データの緯度・経度を計算して出力するスクリプトは

```
1：fmt = ″A3A6A8A8A5A5A2A2A2A1A30A60A80A5A30A4A4A5″
2：while file_name=ARGV.shift
3：    File.foreach(file_name) do |line|
4：       if /^P/ =~ line
5：          line.chomp!(″¥n″)
6：          dat = line.unpack(fmt)
7：          x = dat[2].to_f / 36000.0
8：          y = dat[3].to_f / 36000.0
9：          printf(″経度 = %f, 緯度 = %f¥n″, x, y)
10：       end
11：    end
12：end
```

※日本語入力モードへの移行は【Shift】キーと【スペース】キーを同時にタイプ
となります。

４）ポイントデータの GIS データ化
GRASS には、s.in.ascii という表形式のテキストデータを GRASS の site（ポイント）データとしてインポートするコマンドが用意されています。
① **GRASS** にインポートできるテキストデータ形式
GRASS にインポートできるテキストデータ形式は、下記のように特定の区切り文字（デフォルトはスペース）で区切られた表形式のデータです。
- ＸＹ（２Ｄ）データ形式

Ｘ【区切り文字】Ｙ【区切り文字】文字列又は数値【区切り文字】文字列又は数値･･･
例：　135.1▽35.2▽5▽名古屋市（▽は半角スペース）
　　135.2▽35.6▽6▽豊田市
※ただし、以下のように同一列に文字列と数値が混在することはできません。
例：　135.1▽35.2▽5▽名古屋市　（この場合、 5は数値）
　　135.2▽35.6▽6A▽豊田市　（この場合、 6Aは文字列）

- ＸＹＺ（３Ｄ）データ形式

Ｘ【区切り文字】Ｙ【区切り文字】Ｚ【区切り文字】文字列又は数値【区切り文字】･･･
例：　135.1▽35.2▽15.0▽名古屋市
　　135.2▽35.6▽16.6▽豊田市

② 固定長データ（ポイント）の **GRASS** データ化

GRASS は Linux 等の Unix 系 OS での使用が前提となります（エミュレータ上では Windows でも動きます）。国土数値情報等は S-JIS のコード体系で記録されていますので、日本語の名称等は EUC コード体系に変換して出力する必要があります。したがって、GRASS のポイント（サイト）・データへの変換は以下のようになります（ただし、GRASS は日本語には対応していないので、地図上に施設名称を表示したい場合はローマ字に直す必要があります）。

```
1：require▽"kconv"
2：require▽"jcode"
3：$KCODE="s"
4：fmt = "A3A6A8A8A5A5A2A2A2A1A30A60A80A5A30A4A4A5"
5：asc = open("koukyou.lst", "w")
6：while file_name=ARGV.shift
7：　　File.foreach(file_name) do |line|
8：　　　if /^P/ =~ line
9：　　　　line.chomp!("¥n")
10：　　　　dat = line.unpack(fmt)
11：　　　　x = dat[2].to_f / 36000.0
12：　　　　y = dat[3].to_f / 36000.0
13：　　　　name=dat[11].delete('□'.tosjis)
14：　　　　address=dat[12].delete('□'.tosjis)
15：　　　　asc.printf("%f▽%f▽%d▽%s▽%s¥n",xy,dat[4].to_i, name.toeuc, address.toeuc)
16：　　　end
17：　　end
18：end
19：asc.close
```

※□は全角スペース（日本語入力モードで【@】キーを２回タイプ）、▽は半角スペース

③ 国土数値情報（公共施設）データのインポート

②で GRASS データ化した国土数値情報（公共施設）データ（koukyou.lst）を GRASS にインポートする前に、GRASS を起動なければなりません。 koukyou.lst ファイルは緯度経度を座標系として利用していますので、緯度経度座標系の LOCATION(aichi)で GRASS を起動します。その後、インポートコマンドにより koukyou.lst ファイルをインポートします。

```
grass53
s.in.ascii▽sites=koukyou▽input=koukyou.lst
```

さらに、インポートされた国土数値情報（公共施設）データを表示するため、表示用のグラフィックウィンドウを開き、

```
d.mon▽start=x0▽select=x0
```

以下のコマンドにより、ポイントデータを表示します（図２－１）。

```
d.sites▽sitefile= koukyou▽color=green▽size=1
```

表示された各ポイントの内容は以下のコマンドを実行し、

```
d.what.sites▽map= koukyou
```

確認したいポイントの近くでマウスの左ボタンをクリックすれば、そのポイントの情報が表示されます。なお、確認が終了したら、必ず表示用のグラフィックウィンドウ上でマウスの右ボタンをクリックして、このコマンドを終了することを忘れないようにして下さい。

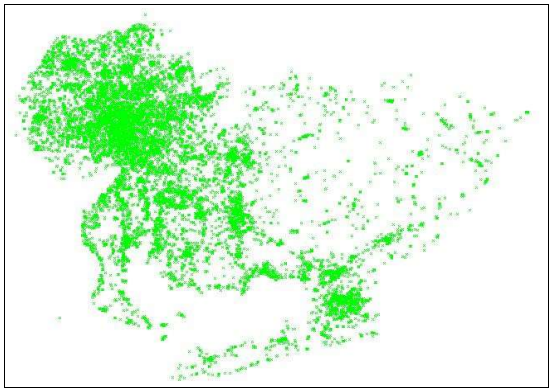


図2－1. インポートされた国土数値情報（公共施設）データ

２－２．メッシュデータのGISデータ化

１）メッシュデータとは

国土数値情報のメッシュ（GISでは、通常ラスターと表現されます）データは、総務省(旧総務庁)が定めた「統計に用いる標準地域メッシュおよび標準地域メッシュレコード」（表２－３）に従って、それぞれの区域に関する統計データが編集されています。

このメッシュコードは、メッシュデータの各区域に対し割り振られたコードで、以下の規約に従っています。

- 第１次地域区画：４桁のコードで識別され、上２桁はメッシュの南西端の緯度を1.5倍した数字、下２桁は同じ点の経度の下２桁の数を表します。
- 第２次メッシュ区画：２桁のコードで識別され、それの属する１次メッシュ区画を行列に見立てると、上１桁は西から東に向けて０から７まで振られた列番号を、下１桁は南から北に向けて０から７まで振られた行番号を表します。それの属する１次メッシュコードに続けて示されています。
- 第３次メッシュ区画：２桁のコードで識別され、それの属する２次メッシュ区画を行列に見立てると、上１桁は西から東に向けて０から９まで振られた列番号を、下１桁は南から北に向けて０から９まで振られた行番号を表します。それの属する２次メッシュコードに続けて示されています。

例えば54382323という３次メッシュコード（基準地域メッシュコード）は5438という１次地域区画中の南から２番目西から３番目にある２次地域区画中のさらに南から２番目西から３番目の３次地域区画を示していることになります。

国土数値情報の幾つかは、このメッシュデータにより提供されています。このメッシュデータのデータフォーマットは以下のとおりです。

表２－３．標準地域メッシュにおける主なメッシュ区画

区画の種類	区分方法	緯度の間隔	経度の間隔	一辺の長さ	地図との関係
１次メッシュ区画	東経100度、北緯０度を基準とし、各度の経線と、偶数緯度及びその間隔を３等分した緯線とで縦横に分割した区域	40分	１度	約80km	20万分の１地勢図の１図葉
２次メッシュ区画	１次メッシュ区画を緯線方向及び経線方向に８等分してできる区域	５分	７分30秒	約10km	２万５千分の１地形図の１図葉
３次メッシュ区画	標準(基準)地域メッシュとも呼ばれ、２次メッシュ区画を緯線方向及び経線方向に10等分してできる区域	30秒	45秒	約1km	
1/n細分メッシュ区画	３次メッシュ区画を緯線方向及び経線方向にn等分してできる区域	30/n秒	45/n秒	約1/n km	

表２－４．国土数値情報（標高・傾斜度メッシュ）のファイルフォーマット

ヘッダデータ（ヘッダの１行目）

項目	仕様	終了	内容
レイヤコード	Ａ３	3	「H」
作成機関	Ａ１０	13	国土庁「NLA」、国土地理院「GSI」、海上保安庁「MSA」
データコード	Ａ１０	23	「Ａ○○－○○M」
データの種類	Ｉ２	25	メッシュ：４
作成年度（西暦）	Ｉ４	29	複数年にわたる場合には、作成開始年度
１行の桁数	Ｉ４	33	１行の桁数を記載
データ全体の行数	Ｉ８	41	メッシュデータ全体の行数を記載

メッシュデータ（２行目以降）

項目	仕様	終了	内容
レイヤコード	Ａ３	3	「M」
メッシュの大きさ	Ｉ２	5	
１次メッシュコード	Ｉ４	9	
２次メッシュコード	Ｉ２	11	
３次メッシュコード	Ｉ２	13	
属性１			
：			

例

1行目：H GSI G01-56M 41975 299 4756

M 351376092 502 1200 30191 1 16 2 4035910 60 388880 209999 999 1009191 1 40 3 1209169 5 16 2 159999 999 309999 999 509999 999 500999 999 30999 999 30999 999 88882 88882 88882 88887 88880

2）**GRASS** におけるメッシュデータ

GIS では、システムにより縦横異なるサイズのメッシュ（長方形メッシュ）を扱えるシステムと、扱えないシステムがあります。メッシュ区画の表にあるように、地域メッシュでは緯度・経度方向で辺の長さが異なる長方形メッシュが採用されています。今回利用する GRASS では長方形メッシュも扱うことが可能です。したがって、GRASS ではこの地域メッシュにより作成されたデータをそのままインポート可能な形式にフォーマットの変換をすることで利用することが可能です。そこで、まず GRASS 読み込み可能な GRASS 2D Raster (ASCII)形式を確認すると、表２－５のとおりです。

表２－５．**GRASS** 2D Raster(ASCII) 形式

ヘッダー部

north: 3980319.16466812	北端の座標
south: 3978824.85093895	南端の座標
east: 443960	東端の座標
west: 442296	西端の座標
rows: 747	メッシュの列数
cols: 832	メッシュの行数

データ部

10.30957▽10.318124.....	北西端メッシュ～北東端メッシュ（▽は半角スペース）
:	:
15.34957▽16.318924.....	南西端メッシュ～南東端メッシュ

３）３次メッシュコードから緯度・経度座標系への変換

国土数値情報のメッシュデータとして、今回は、国土数値情報（標高・傾斜度メッシュ）の愛知県データ（G04-56M_23.txt）を使用します。このファイルに保存されているデータの種別及びフォーマットは G01-56M.htm に記載されています。自然地形メッシュ・データをフォーマットに従い分解する前に、3 次メッシュコードから緯度・経度座標系への変換を考えなくてはなりません。

例えば、"54382323"という３次メッシュコードを緯度・経度に変換するスクリプトは以下のようになります。

```
1：a=″54382323″
2：fmt=″A2A2A1A1A1A1″
3：dat=a.unpack(fmt)
4：x0=100.0+dat[1].to_f + (dat[3].to_f+dat[5].to_f / 10.0)*(7.5/60.0)
5：y0=dat[0].to_f /1.5 + (dat[2].to_f + dat[4].to_f /10.0)*(5.0/60.0)
6：x1=x0+45.0/3600.0
7：y1=y0+30.0/3600.0
8：p  x0, y0, x1, y1
```

4）メッシュデータの変換

G04-56M.htm から、標高・傾斜度メッシュを GRASS 2D Raster 形式へ変換するスクリプトを考えます。ただし、国土数値情報のメッシュ・データは各県に含まれるメッシュのみがファイルに保存されており、そのまま GRASS の GRASS 2D Raster 形式へ変換するという訳にはいきません。なぜなら、GIS のラスター・データは東西南北端の座標で指定される長方形領域のデータですが、各県が長方形の形をしているわけではありません。したがって、県外であろうとも対象とする県が収まる東西南北端の座標で指定される長方形領域全てについて、（当然、県外はデータが存在しないことを示す値を出力することになりますが）GRASS 2D Raster 形式に従ってデータを出力する必要がありますが、そこには問題点が２つあります。

① ヘッダー部の東西南北端の座標が事前に分からない。

② メッシュ・データの並びが GRASS 2D Raster 形式に従って記録されているわけではない。

まず、①の問題を解決するためには、どのようにすればよいのでしょう？データから事前に調べてその値を利用するのは困難ですし、日本全域をカバーする座標では大きすぎます。したがって、変換するデータからこれらの座標を求めなければなりません。難しく考える必要はありません。ファイル内の各３次メッシュ・コードの西端座標（経度）の最小値をヘッダー部の西端、東端座標（経度）の最大値をヘッダー部の東端、同様に南端座標（緯度）の最小値をヘッダー部の南端、北端（緯度）の最大値をヘッダー部の北端とすれば、G04-56M_23.txt 内にある全てのメッシュをカバーする領域を指定することが可能です。では、これらをどのように求めればよいのでしょう？例えば、７つの数値を要素に持つ配列中の最大・最小値求めるスクリプトは以下のようになります。

```
1：a=[0,2,4,5,1,2,3]
2：min = nil
3：max = nil
4：a.each do |v|
5：    if (min = = nil) || (min>v.to_i)
6：        min=v.to_i
7：    end
8：    if (max = = nil) || (max<v.to_i)
9：        max=v.to_i
10：   end
11：end
12：p min, max
```

※"||"は、"または"を表す演算子です。つまり、５行目では「min が nil と等しい」または「min が v.to_i

より大きい」場合、条件が満たされ６行目の文が実行されます。

上記のスクリプトを応用すれば、容易にヘッダー部の東西南北端の座標を求めることができるはずです。

では、次に②の問題はどのように解決すればよいのでしょう？②の問題を解決するためには、データを読み込んで直ぐに出力という訳にはいきません。どの範囲を出力するかは、上記の方法（①の解法）で全てのデータを読み込み終わって初めて分かることですし、読み込むメッシュの順番が出力するメッシュの順番（GRASS 2D Raster 形式）という訳でもありません。そこで、全てのデータを読み込むと同時に記憶して（①の計算も同時に行って）、その後県外や属性データのないメッシュ（当然、データがないことを示す値を設定しておく）も含め GRASS 2D Raster 形式に従って出力することになります。この場合問題になるのが、各メッシュの値をどのような形で記憶しておくかということです。

ここで、54382323 という３次メッシュコードを考えてみてください。この中で緯度方向に関連する数値は"54"（１次メッシュ）、"2"（２次メッシュ）、"2"（３次メッシュ）です。これを繋いだ数値"5422"という数値でこのメッシュの緯度を表すことができます。同じように、"3833"という数値でこのメッシュの経度を表すことができます。例えば、南西端の 3 次メッシュコードが 54382323、北東端の 3 次メッシュコードが 55391212 の区画では、図２－２のように緯度方向は 5422～5511、経度方向は 3833～3922 の各緯経度のインデックスで表されるメッシュ（例えば 3 次メッシュコードが 54382323 では、(5422, 3833)）について、順にその属性数値を出力すれば良いことになります。

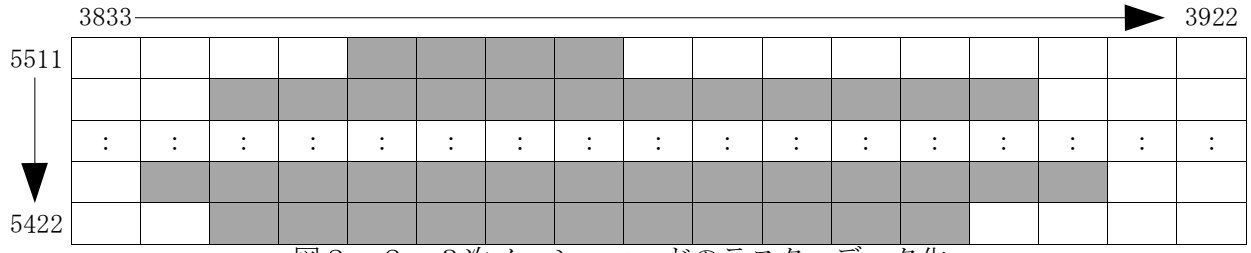


図２－２．３次メッシュコードのラスターデータ化

ただ、通常の配列では無駄な部分（配列は 0 から自動的に確保されるので、先ほどの場合、緯度方向では 0～5422、経度方向では 0～3833 の範囲は明らかに無駄な部分です）まで確保されてしまいます（当然県外部も）。これを避けるためには、Ruby のハッシュテーブルを理解する必要があります。

① ハッシュテーブル**(Hash** クラス**)**

Ruby には、配列に加え、データのまとまりを扱うためのもうひとつ重要なデータ構造として、ハッシュテーブルというものがあります。ハッシュテーブルは任意の値をキーとする配列で、連想配列とも呼ばれます。配列の生成は[ ]でしたが、ハッシュテーブルは{}で生成されます。ただし、ハッシュテーブルでも各要素へのアクセスは{}ではなく、配列と同様[ ] により行います。例えば、以下の３つのスクリプトを実行して、動作の違いを確認して下さい。

スクリプト①

```
1：ary={}
2：a="2"
3：ary[a]=a.to_i
4：p ary
```

スクリプト②

```
1：ary=[ ]
2：a="2"
3：ary[a.to_i]=a.to_i
4：p ary
```

スクリプト③

```
1：ary=[ ]
2：a="2"
3：ary[a]=a.to_i
4：p ary
```

② メッシュデータの変換

説明のとおり自然地形メッシュ・データ（G01-56M-23.txt）を GRASS 2D Raster 形式に変換するスクリプトは以下のようになります。今回は、属性値として３次メッシュ・コード自体を利用しますが、51 行目を変更すれば容易に他の属性を GRASS 2D Raster 形式に変換することができます。

```
 1：fmt = "A3A2A8"
 2：fmt << "A5A5A5A1"              ## 標高
 3：fmt << "A3A2A3A2"              ## 傾斜度
 4：fmt << "A4A1A3A2A3A2"          ## １／４細分区画１
 5：fmt << "A4A1A3A2A3A2"          ## １／４細分区画２
 6：fmt << "A4A1A3A2A3A2"          ## １／４細分区画３
 7：fmt << "A4A1A3A2A3A2"          ## １／４細分区画４
 8：fmt << "A4A1A3A2A3A2"          ## １／４細分区画５
 9：fmt << "A4A1A3A2A3A2"          ## １／４細分区画６
10：fmt << "A4A1A3A2A3A2"          ## １／４細分区画７
11：fmt << "A4A1A3A2A3A2"          ## １／４細分区画８
```

```
12 : fmt << "A4A1A3A2A3A2"          ## １／４細分区画９
13 : fmt << "A4A1A3A2A3A2"          ## １／４細分区画１０
14 : fmt << "A4A1A3A2A3A2"          ## １／４細分区画１１
15 : fmt << "A4A1A3A2A3A2"          ## １／４細分区画１２
16 : fmt << "A4A1A3A2A3A2"          ## １／４細分区画１３
17 : fmt << "A4A1A3A2A3A2"          ## １／４細分区画１４
18 : fmt << "A4A1A3A2A3A2"          ## １／４細分区画１５
19 : fmt << "A4A1A3A2A3A2"          ## １／４細分区画１６
20 : fmt1="A2A2A1A1A1A1"
21 : fmt2="A2A1A1"
22 : ascfile="topo.asc"
23 : c_min=nil;c_max=nil;r_min=nil;r_max=nil
24 : mdat={}
25 : while file_name=ARGV.shift
26 :     File.foreach(file_name) do |line|
27 :         if    /^M/ =~ line
28 :             ary= line.unpack(fmt)
29 :             meshcode = ary[2]
30 :             dat=meshcode.unpack(fmt1)
31 :             row=(dat[0]+dat[2]+dat[4]).to_i
32 :             col=(dat[1]+dat[3]+dat[5]).to_i
33 :             if(mdat[row]==nil) then
34 :                 mdat[row]={}
35 :             end
36 :             mdat[row][col]=ary[3]
37 :             if (r_min==nil)||(r_min.to_i>row)
38 :                 r_min=row
39 :             end
40 :             if (c_min==nil)||(c_min.to_i>col)
41 :                 c_min=col
42 :             end
43 :             if (r_max==nil)||(r_max.to_i<row)
44 :                 r_max=row
45 :             end
46 :             if (c_max==nil)||(c_max.to_i<col)
47 :                 c_max=col
48 :             end
49 :         end
50 :     end
51 : end
52 : rows=0
53 : i=r_max
54 : while i>=r_min
55 :     n1=((i % 100)/10).to_i
56 :     if n1<8 then
57 :         rows=rows+1
58 :     end
59 :     i=i-1
60 : end
61 : cols=0
62 : j=c_min
63 : while j<=c_max
64 :     e1=((j % 100)/10).to_i
65 :     if e1<8 then
66 :         cols=cols+1
67 :     end
68 :     j=j+1
69 : end
70 : asc=open(ascfile,"w")
71 : dat=c_min.to_s.unpack(fmt2)
72 : west=100.0+dat[0].to_f+(dat[1].to_f+dat[2].to_f/10.0)*(7.5/60.0)
73 : dat=c_max.to_s.unpack(fmt2)
74 : east=100.0+dat[0].to_f+(dat[1].to_f+dat[2].to_f/10.0)*(7.5/60.0)+45.0/3600.0
75 : dat=r_min.to_s.unpack(fmt2)
76 : south=dat[0].to_f/1.5+(dat[1].to_f+dat[2].to_f/10.0)*(5.0/60.0)
77 : dat=r_max.to_s.unpack(fmt2)
78 : north=dat[0].to_f/1.5+(dat[1].to_f+dat[2].to_f/10.0)*(5.0/60.0)+30.0/3600.0
79 : asc.print "north: ",north,"¥n"
80 : asc.print "south: ",south,"¥n"
81 : asc.print "east:  ",east,"¥n"
82 : asc.print "west:  ",west,"¥n"
```

```
 83 : asc.print ″rows: ″,rows,″¥n″
 84 : asc.print ″cols: ″,cols,″¥n″
 85 : i=r_max.to_i
 86 : while i>=r_min.to_i
 87 :     n1=((i % 100)/10).to_i
 88 :     if n1<8 then
 89 :         j=c_min.to_i
 90 :         while j<=c_max.to_i
 91 :             e1=((j % 100)/10).to_i
 92 :             if e1<8 then
 93 :                 if mdat[i][j] == nil then
 94 :                     asc.print ″▽*″
 95 :                 else
 96 :                     asc.print ″▽″,mdat[i][j].to_f/10.0
 97 :                 end
 98 :             end
 99 :             j=j+1
100 :         end
101 :         asc.print ″¥n″
102 :     end
103 :     i=i-1
104 : end
105 : asc.close
```

※ %演算子　　：剰余（余り）の計算をする演算子
※ ▽は半角スペース
※２次メッシュは0～7の値をとるので、8～9の部分は出力しない（92&93及び96&97行）
※GRASS 2D Raster形式において、「*」はNULL（値が存在しない）ことを表す。

③ **GRASS 2D Raster** データのインポート

②で作成したGRASS 2D Raster形式のデータも緯度経度座標ですから、ポイントデータ同様緯度経度座標系のLOCATION(aichi)で、以下のコマンドによりGRASSにインポートします。

```
r.in.ascii▽input= topo.asc▽output=topo
```

さらに、グラフィックウィンドウにインポートされたデータを表示するコマンドは、以下のとおりです（図２－３）。

```
d.rast▽topo
```

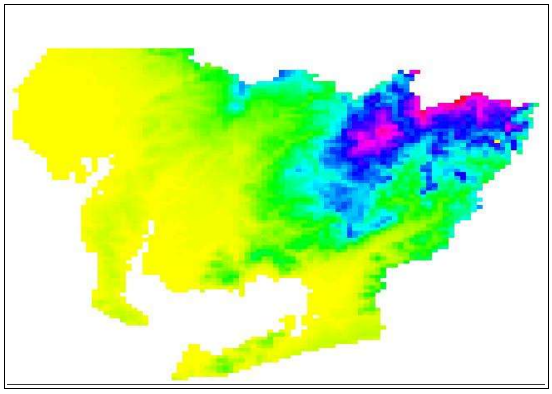


図2－3. インポートされた標高・傾斜度メッシュデータ

２－３．ポリゴン・データの GIS データ化

１）　ポリゴン・データとは

ポリゴン・データ（国土数値情報では面データと呼ばれる）は、ベクトルデータで面情報の領域を表す多角形です。ポリゴンを構成する線分をつなぎ合わせて生成した閉領域を１つのエリアとして表現しています。

国土数値情報の面データは、ヘッダ情報、ノード情報、リンク情報、エリア情報、エリア台帳データで構成されます。このうちノード情報、リンク情報、エリア情報（図２－４参照）が以下に示すように地図データに関する部分でエリア台帳データが地図データの属性に関する情報です。

- ノード情報：各リンクと呼ばれる線分の始終点、交点などを表す情報。ノード情報には、各ノードが位置する座標値などが記載されている。
- リンク情報：エリアを細かな線分に分解したもの。リンク情報には、各リンクの始終点を示すノード情報とリンクを構成する中間点の座標値などで記載されている。
- エリア情報：エリアを構成するリンク情報を記載した情報。エリア情報に記述されているリンクをつなぐことで１つのエリアが生成される。

今回は、行政界・海岸線(面)の面データ（N03-11A-23.txt）をサンプルに使います。リンク情報にはノードの点情報も含まれますから、これらのデータの中で GIS データ化に必要なのは、リンク情報、エリア情報、エリア台帳データとなります。各データは表２－６のようなフォーマットで記録されています。

表２－６のように、各情報の種類（ノード情報・リンク情報・エリア情報）は、行頭のレイヤーコードで決まり、ポリゴン境界やラインを構成する座標の情報は、リンク情報の２行目以降に記録されています。

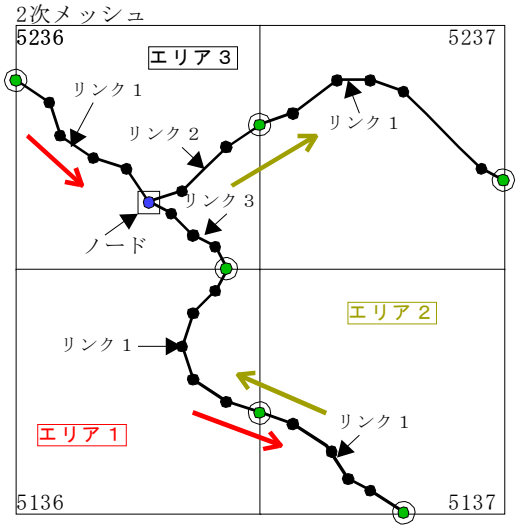


図2－4．国土数値情報の面データ

表２－６．国土数値情報（行政界・海岸線(面)）のファイルフォーマット

リンク情報（線分データ）

１行目

項目		記述形式	累積桁数	内容
レイヤコード		A3	3	「L」
起点ノード	メッシュコード	I6	9	２次メッシュコード
	番号	I6	15	
終点ノード	メッシュコード	I6	21	２次メッシュコード
	番号	I6	27	
リンク一連番号		I6	33	起点ノードのあるメッシュ内連番
リンク台帳の有無		I2	35	「０」：リンク台帳が無い
リンク属性番号		I10	45	
リンク構成中間点数		I6	51	両端のノードも含む

２行目以降（リンク構成中間点データ。１行に５点ずつ設定し、５点に満たない場合、未設定「空白」）

項目		記述形式	累積桁数	内容
中間点１（起点）	Ｘ座表	I8	8	緯度・経度を座標系として、東経をＸ、北緯をＹとして 0.1 秒単位で表現。なお、原点は、東経０°、北緯０°。
	Ｙ座標	I8	16	
中間点２	Ｘ座表	I8	24	
	Ｙ座標	I8	32	
中間点３	Ｘ座表	I8	40	
	Ｙ座標	I8	48	
中間点４	Ｘ座表	I8	56	
	Ｙ座標	I8	64	
中間点５	Ｘ座表	I8	72	
	Ｙ座標	I8	80	

エリア情報（ポリゴン・データ）

１行目

項目		記述形式	累積桁数	内容
レイヤコード		A3	3	「A」
代表点	メッシュコード	I6	9	２次メッシュコード
	Ｘ座標	I8	17	
	Ｙ座標	I8	25	
エリア一連番号		I8	33	
エリア台帳の有無		I2	35	「０」：エリア台帳無し 「１」：エリア台帳有り
エリア属性番号		I10	45	
エリア構成中間点数		I6	51	

表２－６．国土数値情報（行政界・海岸線(面)）のファイルフォーマット（続き）

２行目以降（エリア構成リンクデータ。１行に５リンクずつ設定し、５リンク未満の場合は未設定「空白」。データは右回りとし、リンクが逆向きのときは、番号に負の値「－」を設定する。）

項目		記述形式	累積桁数	内容
リンク１	メッシュコード	I6	6	２次メッシュコード
	番号	I6	12	
	フラグ	I2	14	表示する線：０　表示しない線：１
リンク２	メッシュコード	I6	20	２次メッシュコード
	番号	I6	26	
	フラグ	I2	28	表示する線：０　表示しない線：１
リンク３	メッシュコード	I6	34	２次メッシュコード
	番号	I6	40	
	フラグ	I2	42	表示する線：０　表示しない線：１
リンク４	メッシュコード	I6	48	２次メッシュコード
	番号	I6	54	
	フラグ	I2	56	表示する線：０　表示しない線：１
リンク５	メッシュコード	I6	62	２次メッシュコード
	番号	I6	68	
	フラグ	I2	70	表示する線：０　表示しない線：１

エリア台帳データ

項目	記述形式	累積桁数	内容
レイヤコード	A3	3	「ＤＡ」：エリア台帳
属性番号	A10	13	行政コード（５桁）
行数	I3	16	「１」
都道府県名	A16	32	SHIFT-JIS 漢字（８文字）
支庁名	A16	48	SHIFT-JIS 漢字（８文字）
群・政令市名	A16	64	SHIFT-JIS 漢字（８文字）
市区町村名	A16	80	SHIFT-JIS 漢字（８文字）

２）ポリゴン・データへの変換

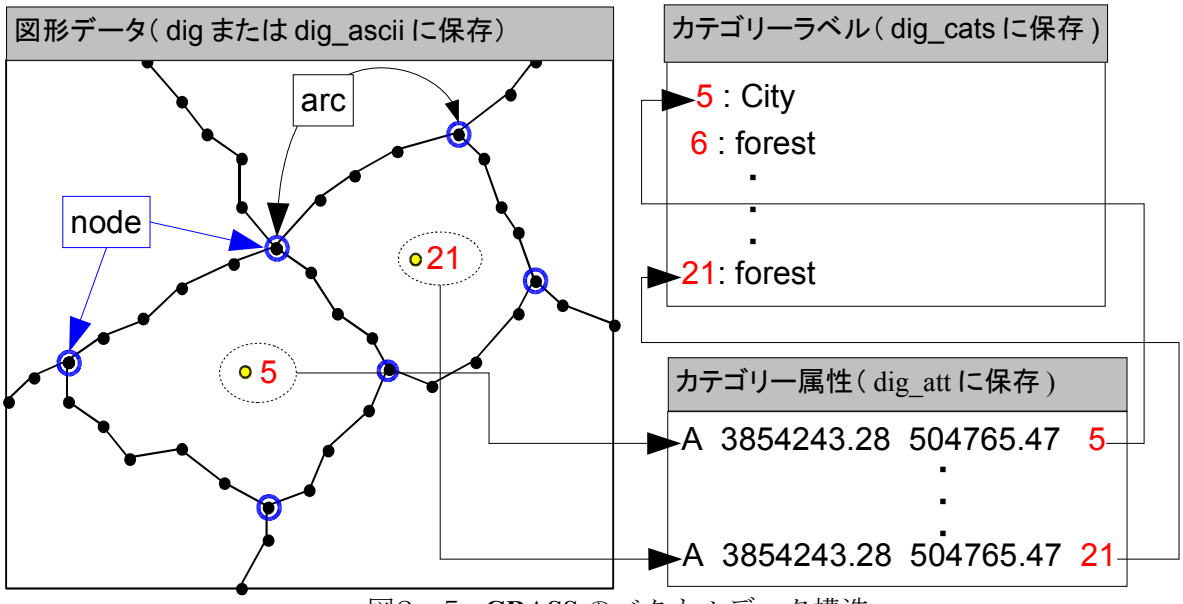


図2－5．**GRASS** のベクトルデータ構造

GRASS のベクトル型データでは、それぞれの線や領域には、カテゴリー番号と呼ぶ整数の属性値が割り当てられます。ベクトル型データの情報は、下記のように多くのデータファイルに分けられて保存されます（図２－５参照）。

① 図形データファイル（**dig**：バイナリ, **dig_ascii**：テキスト）

GRASS のベクトル型データは、arcs（アーク；弧）と呼ばれる非交差曲線からなる arc-node 表現法で保存されています。arc は xy 座標のペアの連続として保存され、１つの arc の２つの終端点は，node（ノード；節）と呼ばれています。２つの連続した xy のペアが arc の各セグメントを示します。この arc は、１つ、または結合した他との組み合わせで、より高度な地図要素である line（ライン；線（例えば、道路あるいは河川））、あるいは area（エリア；領域（例えば農地））を形成します。線的な形（linear feature）を形成する arc は、line（線）と呼ばれ、そして area（領域）の輪郭となる arc は area edge（領域境界線）、あるいは area line（領域線）と呼ばれています。

GRASS にインポートできる図形データ形式（dig_ascii 形式）はヘッダー部分（表２－７）とデータ部分（表２－８）の２つから構成されます。ヘッダー部分にはベクトルデータの諸情

報（作成者・作成日時・スケール・領域等）を示しています。
arc の情報は、このヘッダーに続いています。それぞれの arc は形の記載とそれに続く一連の xy 座標のペアより表現されます（表２－８）。形の記載部分は
arc のタイプ（A：領域境界,あるいは，L：線）＋点の数
の２つで示します（例えば、3 点からなる線はＬ3、5 点から構成される領域はＡ5 など）。そして，実際の xy 座標点が次に続きます。

② カテゴリー属性ファイル（**dig_att**）
ベクトル地図レイヤーにおけるそれぞれの線や領域に、それぞれカテゴリー番号を割り当てます。dig_att は ASCII（テキスト）形式のファイルで、各線や領域に関して、それぞれ座標マーカーとカテゴリー番号を指定した表２－９ようなものです。
座標マーカーとは、dig ファイルにおける線や領域を特定するために用いられます。そのためには、線や領域をユニークに識別できるような位置を示していなければなりません。特に、領域（A）の座標マーカーは領域の内部になければなりません。また，線（L）の座標マーカーは、線を確実に示す座標でなければならず（他の線との交点ではだめ），かつ node（節；端点）を指定してはいけません。
この dig_att ファイルに記載されていない線や領域は、unlabeled（ラベルなし）として取り扱われます。注意しなければならないのは、ベクトル→ラスター変換を行う際に、ラベルなしの領域はカテゴリーゼロとして変換され、また、ラベルなしの線は無視されます。

③ カテゴリーラベルファイル（**dig_cats**）
各カテゴリー値（番号）に対応するカテゴリー名は、表２－１０のようにカテゴリーラベルファイルに保存します。

表２－７．GRASS の dig_ascii 形式（ヘッダー部）

ヘッダーの例	意味
ORGANIZATION: O.C.U	データ作成者の組織名など
DIGIT DATE: 06/25/00	デジタル化した日付
DIGIT NAME: grass	デジタル化した者の氏名
MAP NAME: Sanda	地図名･タイトル等
MAP DATE: 08/87	地図の作成日
OTHER INFO: Test	コメント等他の情報
MAP SCALE: 50000	地図の縮尺(例は 5 万分の 1)
ZONE: 53	UTM などのゾーン
WEST EDGE: 504257.55	地図の西端の座標値
EAST EDGE: 511412.46	地図の東端の座標値
SOUTH EDGE: 3853803.35	地図の南端の座標値
NORTH EDGE: 3858192.54	地図の北端の座標値
MAP THRESH: 0.00	デジタル化の分解能(不明な場合は 0.00)
VERTI:	ヘッダー部の終わりを表す

表２－８．**GRASS** の **dig_ascii** 形式（データ部）

データ部分	意味
L 3	L（線を表す）, 点数(例;3 点の線)
▽504833.21 3854248.35	１文字目は半角スペース（▽）、座標値(**y, x** の順←注意)
▽504885.54 3854653.04	〃
▽504856.33 3854255.76	〃
A 5	A（領域を表す）, 点数(例;5 点の面)
▽504934.45 3854544.20	１文字目は半角スペース（▽）、座標値(**y, x** の順←注意)
▽504972.76 3854662.18	〃
▽504945.92 3854324.75	〃
: :	〃

表２－９．**GRASS** のベクトルデータ・カテゴリー属性ファイル

カテゴリー・属性データ		
L	3854248.33 504893.58	5
A	3854243.28 504765.47	21
:	: :	:
L=線 A=領域	座標マーカー（領域のデータでは、領域　カテゴリー番号 内の点）：座標値は **x,y** の順	

表２－１０．**GRASS** のベクトルデータ・カテゴリーラベルファイル

カテゴリー・ラベルデータ	
#N categories	N：カテゴリー数（カテゴリー番号の最大値）
Land Cover / land Use	タイトル
	１行空ける
0.00 0.00 0.00 0.00	通常このまま
1:forest	カテゴリー番号：ラベル
2:water	
：	
2.5:3.5: Case1	実数データの場合は範囲でも指定可能 最小値：最大値：ラベル

④ 図形データの変換

GRASS では、図形情報と属性情報はそれぞれ独立しています。また、GRASS では細切れのリンク（線分）を繋ぎ(snap)、座標マーカから図形中のポリゴンを認識する処理は、コマンド(v.support)により自動的に行われます。したがって、国土数値情報の面データのようにエリアごとに分断された図形データであっても、node 間の連結に問題が無ければ、少なくとも図形データだけは GRASS の dig_ascii 形式に変換することができます。この場合、エリア情報から出力するベクトルデータの東西南北端の各境界を算出し、これをもとにヘッダー情報を出力し、後に各リンク情報（中間点）を arcs（アーク；弧）の形式で順に出力すれば GRASS の dig_ascii データとすることができます。

では、行政界・海岸線(面)の面データ（N03-11A-23.txt）を GRASS で読み込むことができるポリゴン形式に変換するスクリプトは以下のようになります。

```
 1：lnks=[ ]
 2：c_min=nil;c_max=nil;r_min=nil;r_max=nil
 3：while file_name=ARGV.shift
 4：    open(file_name) do |f|
 5：        while line=f.gets
 6：            line.chomp!("¥n")
 7：            if  /^L/ =~ line then
 8：                ary= line.unpack("A3A6A6A6A6A6A2A10A6")
 9：                i=0
10：                dat=[]
11：                while i<ary[8].to_i
12：                    line=f.gets
13：                    line.chomp!("¥n")
14：                    tmp=line.unpack("A16A16A16A16A16")
15：                    tmp.each do |pt|
16：                        if i<ary[8].to_i then
17：                            dat << pt.unpack("A8A8")
18：                        end
19：                        i=i+1
20：                    end
21：                end
22：                lnks[lnks.size]=dat
23：            elsif /^A/ =~ line then
24：                ary= line.unpack("A3A6A8A8A8A2A10A6")
25：                i=0
26：                while i<ary[7].to_i
27：                    line=f.gets
28：                    line.chomp!("¥n")
29：                    tmp=line.unpack("A14A14A14A14A14")
30：                    tmp.each do |pt|
31：                        if i<ary[7].to_i then
32：                            dat0=pt.unpack("A6A6A2")
33：                            m0=dat0[0].unpack("A2A2A1A1")
34：                            if (r_min==nil)||(r_min.to_i>(m0[0]+m0[2]).to_i)
35：                                r_min=m0[0]+m0[2]
36：                            end
37：                            if (c_min==nil)||(c_min.to_i>(m0[1]+m0[3]).to_i)
38：                                c_min=m0[1]+m0[3]
39：                            end
40：                            if (r_max==nil)||(r_max.to_i<(m0[0]+m0[2]).to_i)
41：                                r_max=m0[0]+m0[2]
42：                            end
43：                            if (c_max==nil)||(c_max.to_i<(m0[1]+m0[3]).to_i)
44：                                c_max=m0[1]+m0[3]
45：                            end
46：                        end
47：                        i=i+1
```

```
48 :                         end
49 :                     end
50 :                 end
51 :             end
52 :         end
53 : end
54 : ascfile="aichi.asc"
55 : asc=open(ascfile,"w")
56 : dat=c_min.unpack("A2A1")
57 : west=100.0+dat[0].to_f+(dat[1].to_f)*(7.5/60.0)
58 : dat=c_max.unpack("A2A1")
59 : east=100.0+dat[0].to_f+(dat[1].to_f+1.0)*(7.5/60.0)
60 : dat=r_min.unpack("A2A1")
61 : south=dat[0].to_f/1.5+(dat[1].to_f)*(5.0/60.0)
62 : dat=r_max.unpack("A2A1")
63 : north=dat[0].to_f/1.5+(dat[1].to_f+1.0)*(5.0/60.0)
64 : asc.print "ORGANIZATION:¥n"
65 : asc.print "DIGIT DATE:¥n"
66 : asc.print "DIGIT NAME:¥n"
67 : asc.print "MAP NAME: Digital Map¥n"
68 : asc.print "MAP DATE:¥n"
69 : asc.print "MAP SCALE:          25000¥n"
70 : asc.print "OTHER INFO:¥n"
71 : asc.print "ZONE:                          0¥n"
72 : asc.printf("WEST EDGE: %f¥n",west)
73 : asc.printf("EAST EDGE: %f¥n",east)
74 : asc.printf("SOUTH EDGE: %f¥n",south)
75 : asc.printf("NORTH EDGE: %f¥n",north)
76 : asc.print "MAP THRESH:  0.0¥n"
77 : asc.print "VERTI:¥n"
78 : lnks.each do |line|
79 :     if line!=nil then
80 :         asc.printf("A %d¥n",line.size)
81 :         line.each do |pnt|
82 :             asc.printf("▽%16.13f▽%16.14f¥n",pnt[1].to_f/36000.0,pnt[0].to_f/36000.0)
83 :         end
84 :     end
85 : end
86 : asc.close
```

※▽は半角スペース

⑤ 属性・カテゴリーラベルファイルの変換

ここで問題となるのは、ポリゴンの図形情報と属性（カテゴリー）情報の結合です。GRASSでは、上述したように座標マーカーに属性（カテゴリー）情報が結合されます。この座標マーカーは、ポリゴンの内部になければなりませんが、行政界・海岸線(面)のデータ（N03-11A-23.txt）には座標マーカーに使えるようなポリゴン内部の点は情報として含まれていません。

しかし、この行政界・海岸線(面)のデータに限っては、先に使った公共施設のデータを利用することができます。この公共施設のデータには、各市区町村の市・区役所または役場の座標データが含まれています。政令指定都市では市役所と区役所が重複する可能性がありますが、その他の市町村では重複しませんし、ほぼ各市町村のエリア内に存在することが期待できます。

したがって、政令指定都市以外では市役所または役場の座標データを用いることにより、また政令指定都市では区役所の座標データを用いることにより、飛び地は無理としても、それ以外の部分については、ほぼこれらの座標に座標マーカーを設定することが可能と考えられます。

また、公共施設のデータの管理者（コード）には、行政コードという行政ごとの独自の番号が割り振られていますので、これをカテゴリー番号として用いれば、重複を防げますし、行政コードを利用した他の情報との連携がとり易くなります。

では、公共施設 のデータ(P02_02p_23.txt)を属性・カテゴリーラベルファイルそれぞれに変換するスクリプトは以下のようになります。

```
 1 : require▽"kconv"
 2 : require▽"jcode"
 3 : $KCODE="s"
 4 : fmt="A3A6A8A8A5A5A2A2A2A1A30A60A80A5A30A4A4A5"
 5 : attfile="aichi.att"
 6 : catfile="aichi.cats"
 7 : att=open(attfile,"w")
 8 : cats=[]
 9 : cmax=-1
10 : while file_name=ARGV.shift
11 :   File.foreach(file_name) do |line|
12 :       if /^P/ =~line
13 :           line.chomp!("n")
```

```
14 :            dat = line.unpack(fmt)
15 :            if (dat[6].to_i==2)&&(dat[7].to_i==1)&&(dat[8].to_i>1)
16 :                x = dat[2].to_f / 36000.0
17 :                y = dat[3].to_f / 36000.0
18 :                str=dat[14].delete("□".tosjis)
19 :                att.printf("A▽%f▽%f▽%d¥n", x, y, dat[13].to_i )
20 :                if cats.include?([dat[13].to_i,str])==false then
21 :                    cats[cats.size]=[dat[13].to_i,str]
22 :                    if cmax<dat[13].to_i then
23 :                        cmax=dat[13].to_i
24 :                    end
25 :                end
26 :            end
27 :        end
28 :    end
29 : end
30 : att.close
31 : cat=open(catfile,"w")
32 : cat.printf("#%d▽categories¥n",cmax)
33 : cat.printf("City▽Name¥n¥n0.00▽0.00▽0.00▽0.00¥n")
34 : cats.each do |line|
35 :    cat.printf("%d:%s¥n",line[0],line[1].toeuc)
36 : end
37 : cat.close
```

※□は全角スペース、▽は半角スペース

⑥ 図形データファイルのインポート

dig_ascii 形式の図形データ（aichi.asc ファイル）も緯度経度座標ですから、緯度経度座標系の LOCATION(aichi)で GRASS にインポートします。ただし、dig_ascii 形式の図形データは、以下のディレクトリになければなりません。

DATABASE/LOCATION/MAPSET/dig_ascii

緯度経度座標系の LOCATION(aichi)では、

DATABASE　：/home/bagr/database
LOCATION ：aichi
MAPSET　：PERMANENT

です。したがって、GRASS にインポートするためには、以下のコマンドでこのディレクトリを作成なければなりません。

mkdir▽/home/bagr/database/aichi/PERMANENT/dig_ascii

次に、 aichi.asc ファイルをこのディレクトリにコピーし、GRASS にインポートします。

cp▽aichi.asc▽/home/bagr/database/aichi/PERMANENT/dig_ascii
v.in.ascii▽input=aichi.asc▽output=aichi

上記のコマンドにより、 aichi.asc ファイルは GRASS のベクトルファイルに変換され、以下のディレクトリに aichi というファイル名で保存されます。

DATABASE/LOCATION/MAPSET/dig

⑦ 属性・カテゴリーラベルファイルのインポートとトポロジー構築

属性・カテゴリーラベルファイル（aichi.att ファイル及び aichi.cats ファイル）は、それぞれ以下のディレクトリになければなりません。

属性ファイル　：DATABASE/LOCATION/MAPSET/dig_att
カテゴリーラベルファイル　：DATABASE/LOCATION/MAPSET/dig_cats

また、そのファイル名は対応するベクトル・ファイル（dig ファイル）と同じでなければなりません。しかがって、上記２つのディレクトリを作成します。

mkdir▽/home/bagr/database/aichi/PERMANENT/dig_att
mkdir▽/home/bagr/database/aichi/PERMANENT/dig_cats
cp▽aichi.att▽/home/bagr/database/aichi/PERMANENT/dig_att/aichi
cp▽aichi.cats▽/home/bagr/database/aichi/PERMANENT/dig_cats/aichi

最後に、以下のコマンドによりトポロジー構築を行います。

v.support▽-s▽map=aichi

このコマンドにより、属性ファイルで指定された座標マーカーから、その座標マーカーが指定するポリゴンやラインを形成するための arc の抽出及び相互（連結）関係が解析され、以下のディレクトリ

DATABASE/LOCATION/MAPSET/dig_plus

にベクトル・ファイル（dig ファイル）と同名のファイルとして保存されます。「-s」は同時にスナップを実行するためのオプションで、隣接する arc を接続することができます。

インポートされたベクトルファイルは、以下のコマンドで確認できます。

d.vect.area▽map=aichi▽fillcolor=blue▽linecolor= black

⑧ ベクトルデータの編集

表示された図（図２-６）をよく見てみると、一部市町村（ポリゴン）で内部の色が背景色と同じものがあります。これは、属性ファイルで指定された座標マーカーがポリゴン内部に存在しなかったためです。GRASS には、ベクトルデータの編集のためのコマンドも用意されてい

ますので、今回はそれを用いて座標マーカーが正しく設定されなかったポリゴンの編集を行います。

その前に、この部分がどの市町村に属し、その行政コードが何なのかを確認しておく必要があります。既に国土数値情報（公共施設）データはインポートしているので、そのデータを用い、この区域内の公共施設から行政コードを確認します（２－１．ポイントデータの GIS データ化参照）。

ベクトルデータの編集を行う場合、以下のコマンドを利用します。

v.digit

コマンド実行後、図２－７に示した一連の操作により設定を行います。

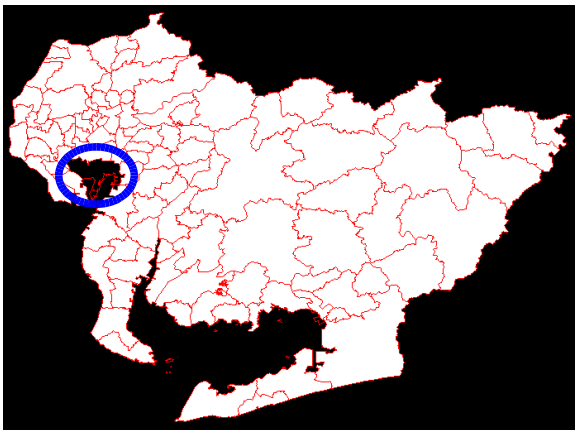


図2－6. ベクトルファイルの確認

設定が終了したら図２－５の【MAIN MENU】の画面で Shift と Q のキーを同時にタイプし、ベクトルデータの編集を終了します。さらに、ベクトルデータの編集を行った場合は、必ずトポロジー構築を再実行する必要がありますので、以下のコマンドを実行します。

v.support ▽map=aichi

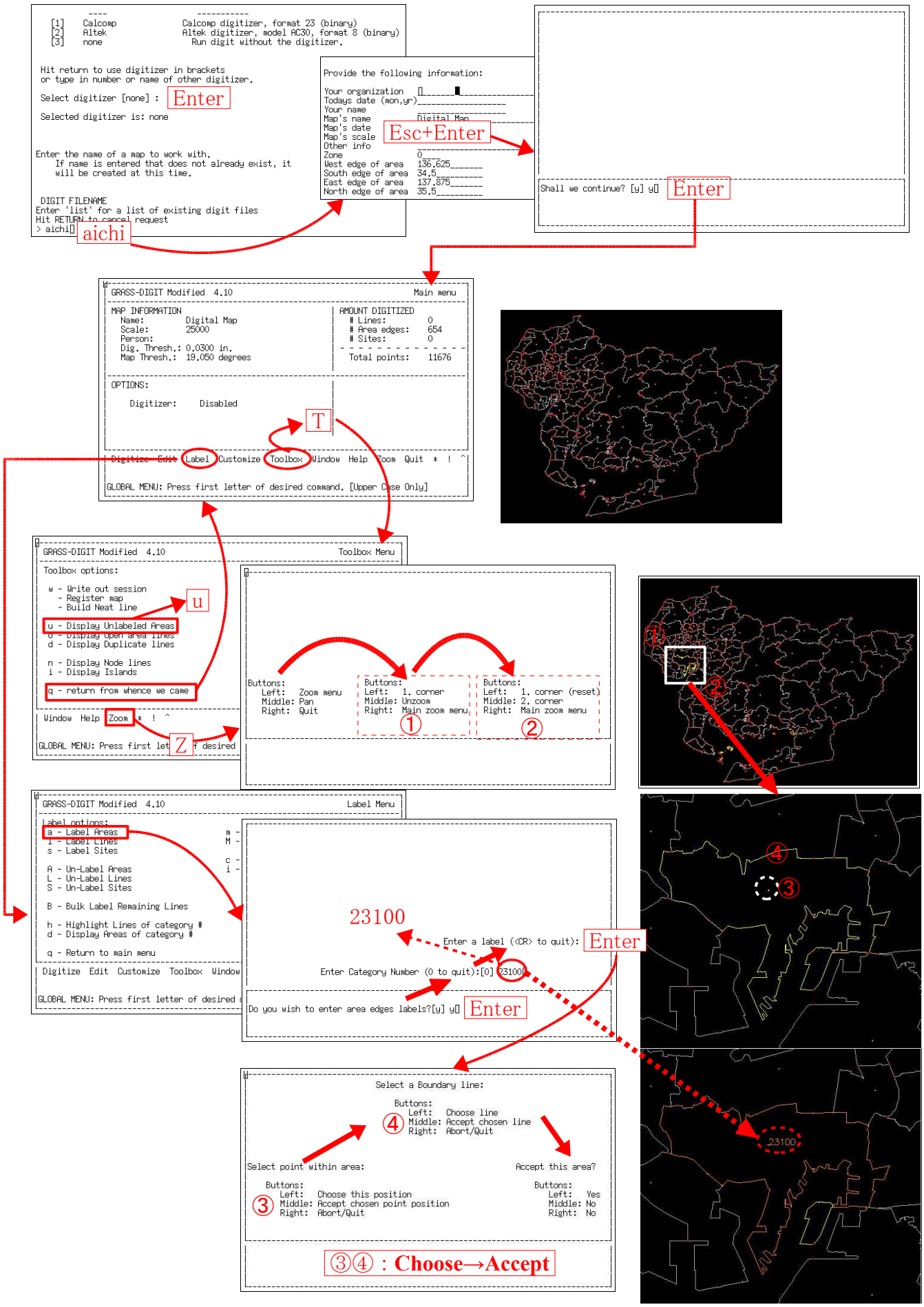


図2－7. ベクトルファイルの編集

３．自然環境保全基礎調査データファイル

自然環境保全基礎調査は、全国的な観点から我が国における自然環境の現況および改変状況を把握し、自然環境保全の施策を推進するための基礎資料を整備するために、環境省が昭和４８年度より自然環境保全法第４条の規定に基づきおおむね５年ごとに実施している調査です。

■自然環境保全法第４条

「国は、おおむね５年ごとに地形、地質，植生及び野生動物に関する調査その他自然環境保全のために講ずべき施策の策定に必要な基礎調査を行うよう努めるものとする。」

一般に、「緑の国勢調査」と呼ばれ、陸域、陸水域、海域の各々の領域について調査項目を分類し国土全体の状況を調査している。また、調査結果は報告書及び地図等にとりまとめられたうえ公表されており（植生調査及び湿地調査の調査結果は http://www.biodic.go.jp/kiso/fnd_f.html にて一部公開されている）、これらの報告書等は、自然環境の基礎資料として、自然公園等の指定・計画をはじめとする自然保護行政の他、環境アセスメント等の各方面において活用されています。

３－１．公開されているデータ

http://www.biodic.go.jp/dload/mesh_vg.html で公開されている自然環境保全基礎調査データファイルには、以下のものがあります。

- 植生調査共通コード等（veg_c02.lzh ：veg_gunraku.csv 等）
- 第４回基礎調査　３次メッシュデータ(1988-1992)（lveg04m01.lzh ：veg04mesh.csv 等）
- 第５回基礎調査　３次メッシュデータ(1992-1996)（veg05m01.lzh ：veg05mesh.csv 等）

これら lzh 形式で圧縮されたファイルには、上記に示したＣＳＶ形式のデータファイル等が格納されています。ＣＳＶ形式のデータファイルの内容は、「植生調査共通コード等」では以下ような形式（veg_gunraku.csv）：

″植生区分コード″,″群落コード″,″群落名″,″集約群落コード″,″自然度コード″

で、一方３次メッシュデータでは以下ような形式：

″メッシュコード″,″群落コード″

で記録されており、各ファイルの内容は図３－１のような関係にあります。

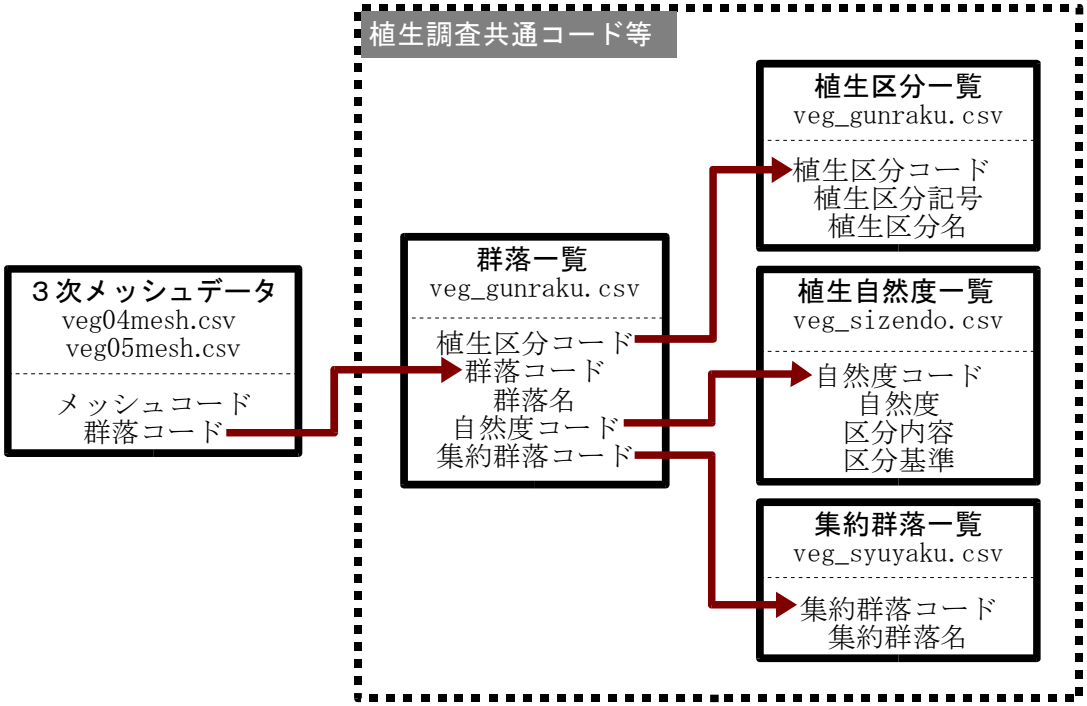


図3－1．自然環境保全基礎調査データファイルの構造

また、メッシュコードを除く各コードの内容は以下のように定義されています。

- 植生区分
  全国の植生を、植生帯、自然植生・代償植生、立地の別などにより区分したもの。市街地等の人為的なものを含め、10 区分。
- 群落名
  植物社会学的群落名を採用し、原則として各都道府県で使用された凡例名をそのまま使用。
- 集約群落名
  類似の群落名を統合したもの（集計に使用）。
- 自然度
  植物社会学的な観点から土地の自然性がどの程度残されているかを示す一つの指標として導入したもの。10 ランクに区分。

※ これらのうち、植生区分コードと自然度コードは数字のみで構成されていますが、群落名と集約群落名は数字とローマ字で構成されています。

一方、この３次メッシュデータは、現存植生図上で地域基準メッシュの中央に直径 5mm の測定円（約 5ha）を設定し、円内で最も広い面積を占める群落をそのメッシュの代表とする（小円選択法）ことにより作成されています。この手法は、昭和 50 年度に検討、植生調査の解析に採用されたもので小面積の

読みとりの欠如を小さくできるとともに、偶然性を是正できることが特徴とされます。また、読みとりにおける条件は以下のとおりです。

- メッシュ内の測定円に植生が一部でも含まれたとき、含まれた範囲について原則を適用する。（海岸線等）
- 測定円に陸地が含まれても、植生図中に群落が表示されておらず、陸地面積が微少な時は除外する。（極めて小さい島嶼等）
- 読みとり範囲で、図の読みとりが不可能なときは「不明区分」を用いて表示する。
- 陸域で囲まれるような湖沼、河川については、現存植生図中に該当凡例がなくても開放水域とする。
- 測定円に２都道府県以上がまたがる場合は、最大面積を占める都道府県のうち、最も広い面積の群落をそのメッシュの代表とする。

３－２．自然環境保全基礎調査データファイルのＧＩＳデータ化

１）デリミタによる文字列の分解・結合

固定長データでは、その行の何桁目であるかに意味がありました。しかし、ＣＳＶファイルでは、文字列をデリミタ（区切り文字のこと。ＣＳＶファイルでは通常「, 」）で区切っているため、1行内でその行内の文字列をどのように区切るかはデリミタの位置によって決められます。したがって、行により文字列をどのように区切るかが異なる場合があり、unpack メソッドは利用できません。

このような場合に利用できる文字列オブジェクトのメソッドとして、split メソッドが ruby には用意されています。例えば、

```
1：a = ″1,2,3″
2：b = a.split(″,″)
3：p  b
```

を実行すると、

```
[″1″, ″2″, ″3″]
```

という結果が得られるはずです。

一方、複数の要素で構成される配列の各要素を、デリミタで区切って結合する場合は、join メソッドが ruby には用意されています。例えば、

```
1：a = [″1″,″2″,″3″]
2：b = a.join(″,″)
3：p  b
```

を実行すると、

```
″1,2,3″
```

という結果が得られるはずです。また、join メソッドでは、配列の要素に数値が含まれていた場合、その数値は文字列に変換されて結合されることになります。例えば、

```
1：a = [1,2,3]
2：b = a.join(″,″)
3：p  b
```

を実行すると、

```
″1,2,3″
```

という結果が得られるはずです。

２）Ruby における「 ″」と「 ’ 」の違い

ruby で文字列を指定する場合、その文字列を「 ″」でくくるか「 ’ 」でくくるかにより、その内部の文字列がどのように記憶されるかが異なります。例えば、次の２つの例を実行してみてください。

例 1

```
puts ″1¥n1″
```

例 2

```
puts ’1¥n1’
```

このように、「 ’ 」で文字列をくくった場合は、その内部の文字をそのまま記憶しますが、「 ″」でくくった場合は、「¥n」（改行）で示されるような文字はそれが示す特殊な文字コードに解釈されて記憶されます。また、「 ″」でくくられた文字列内で、特殊な文字（例えば「 ″」や「 ¥」等）をそのままの文字として記憶させたい場合は、その前に「 ¥」をつけることによりそれが可能になります。例えば、次の例を実行してみてください。

例

```
puts ″¥¥n ¥″″
```

３）３次メッシュデータのＧＩＳデータ化

自然環境保全基礎調査データファイルは、３次メッシュ（約１×１ ㎞）の中心に設定された約 5ha（直径約 250m）の測定円内の代表的な植生タイプです。ラスター・データとして取り扱うよりは、むしろポイント・データとして取り扱うのが妥当と考えられます。

① ３次メッシュ中央座標の計算

３次メッシュコードは、メッシュ内の南西端の座標を代表しています。測定円はその中央に設定されますから、その中央の座標を計算しなければなりません。例えば、３次メッシュコードから、その中央の座標を計算するスクリプトは以下のようになります。

```
1：a=″54382323″
2：fmt=″A2A2A1A1A1A1″
3：dat=a.unpack(fmt)
4：x=100.0+dat[1].to_f+(dat[3].to_f+dat[5].to_f/10.0)*(7.5/60.0)+(45.0/2.0)/3600.0
5：y=dat[0].to_f/1.5+(dat[2].to_f+dat[4].to_f/10.0)*(5.0/60.0)+(30.0/2.0)/3600.0
```

```
6：p  x, y
```

② 植生調査共通コードの問題点

３次メッシュデータに用いられている群落コード及び集約群落コードは、上述したように数字とローマ字で構成されています。全てが数字とローマ字で構成されていれば良いのですが、数字だけのものと数字とローマ字の組み合わせで構成されているものがあるため、そのままではGRASSのsite(ポイント）データとしてインポートするには問題があります。

また、群落コードは、数字とローマ字で構成されているだけでなく、含まれるローマ字を削除すると重複するコードが多数存在してしまいます。これは、都道府県で使用された凡例をそのまま使用して群落名が構成されているため、類似の群落名が「同一の数値＋異なるローマ字」という方針でコード化されたためと考えられます。したがって、GRASSのsite(ポイント）データとしてインポートする場合は、群落コードを用いず連番などで代替しなければならない。これに対して、集計のために類似の群落名を統合した集約群落コードでは、数字とローマ字で構成されていますが、こちらはローマ字を削除しても重複するコードはありませんので、ローマ字を削除すればそのままGRASSのsite(ポイント）データとしてインポートすることができます。

③ 変換テーブル

３次メッシュデータには メッシュコードと群落コードのみが保存されているため、GRASSにインポートするためには先に述べたような連番等に入れ替えなければなりません。また、群落名や植生区分名等の名称もGRASSにインポートするためには、群落コードと各名称の関係が必要になります。それは下記のようなスクリプトとなります。

```
 1：require▽″kconv″
 2：require▽″jcode″
 3：$KCODE=″s″
 4：gunfile=″veg_gunraku.csv″
 5：kubfile=″veg_kubun.csv″
 6：sizfile=″veg_sizendo.csv″
 7：syufile=″veg_syuyaku.csv″
 8：kub2name={}
 9：open(kubfile,″r″)▽do▽|f|
10：    line=f.gets
11：    while▽line=f.gets
12：        ary=line.chop.delete('″').split(″,″)
13：        kub2name[ary[0].to_i]=ary[2]
14：    end
15：end
16：siz2name={}
17：open(sizfile,″r″)▽do▽|f|
18：    line=f.gets
19：    while▽line=f.gets
20：        ary=line.chop.delete('″').split(″,″)
21：        siz2name[ary[0].to_i]=ary[2]
22：    end
23：end
24：syu2name={}
25：open(syufile,″r″)▽do▽|f|
26：    line=f.gets
27：    while▽line=f.gets
28：        ary=line.chop.delete('″').split(″,″)
29：        syu2name[ary[0].delete('A').to_i]=ary[1]
30：    end
31：end
32：gun2syu={}
33：gun2name={}
34：gun2siz={}
35：gun2kub={}
36：gun2id={}
37：id=1
38：open(gunfile,″r″)▽do▽|f|
39：    line=f.gets
40：    while▽line=f.gets
41：        ary=line.chop.delete('″').split(″,″)
42：        gun2id[ary[1]]=id
43：        gun2kub[ary[1]]=ary[0].to_i
44：        gun2name[ary[1]]=ary[2]
45：        gun2syu[ary[1]]=ary[3].delete('A').to_i
46：        gun2siz[ary[1]]=ary[4].to_i
47：        printf(″%s,%d,%s,%s,%s,%s¥n″,ary[1],gun2id[ary[1]],gun2name[ary[1]],syu2name
    [gun2syu[ary[1]]],kub2name[gun2kub[ary[1]]],siz2name[gun2siz[ary[1]]])
48：        id=id+1
49：    end
```

```
50：end
```

④ ３次メッシュデータの **GIS** データ化

では、群落の連番・群落名・自然度コード・自然度を GIS データ化するスクリプトは、上記のスクリプトの後に下記の内容を加えることになります。自然環境保全基礎調査データファイルは全国を一つのファイルにまとめていますが、今回は愛知県のみが対象ですから、図１－６で設定した範囲内の情報だけが必要になります。不要なデータは処理を遅くしますから、下記のスクリプトでは、図１－６で設定した範囲（53～56 行）内のみに限って出力（65～67 行）すようにしています。

```
51：fmt="A2A2A1A1A1A1"
52：meshfile="veg05mesh.csv"
53：txtfile="veg_gunraku.txt"
54：txt▽=▽open(txtfile,▽"w")
55：x0=136+39/60.0+45/3600.0
56：x1=137+51/60.0
57：y0=34+34/60.0
58：y1=35+25/60.0+30/60.0
59：open(meshfile,"r")▽do▽|f|
60：    line=f.gets
61：    while▽line=f.gets
62：        ary=line.chop.delete('"').split(",")
63：        meshcode▽=▽ary[0]
64：        dat=meshcode.unpack(fmt)
65：        x=100.0+dat[1].to_f+(dat[3].to_f+dat[5].to_f/10.0)*(7.5/60.0)+(45.0/2.0)/3600.0
66：        y=dat[0].to_f/1.5+(dat[2].to_f+dat[4].to_f/10.0)*(5.0/60.0)+(30.0/2.0)/3600.0
67：        if▽(x>=x0)&&(x<=x1)&&(y>=y0)&&(y<=y1)▽then
68：            txt.printf("%f▽%f▽%d▽%s▽%d▽%s¥n",▽x,▽y,▽gun2id[ary[1]],▽gun2name
    [ary[1]].toeuc,▽gun2siz[ary[1]],▽siz2name[gun2siz[ary[1]]].toeuc)
69：        end
70：    end
71：end
72：txt.close
```

※▽は半角スペース

⑤ 自然環境保全基礎調査データファイルのインポート

veg_gunraku.txt ファイルも緯度経度座標ですから、緯度経度座標系の LOCATION(aichi)で GRASS にインポートします。国土数値情報（公共施設）データで使用したインポートコマンドにより veg_gunraku.txt ファイルを以下のようにインポートします。

```
s.in.ascii▽sites=gunraku▽input= veg_gunraku.txt
```

以下のコマンドにより、ポイントデータを表示します（図３－２）。

```
d.sites▽sitefile=gunraku▽color=green▽size=1
```

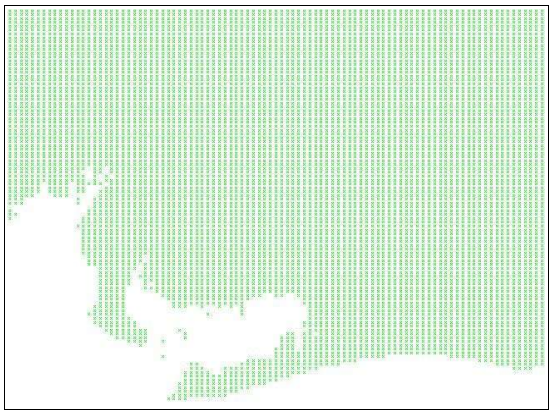


図3－2. インポートされた自然環境保全基礎調査データ

４．MODIS/Terra リモートセンシング画像データ

MODIS センサーは、TERRA 衛星及び AQUA 衛星に搭載されている受動型センサーで、それぞれの衛星は陸域の観測及び海域の観測に利用されています。MODIS センサーによる観測画像はブロードキャスト・データとして準リアルタイムに公開されるとともに、レジストレーション・投影変換・幾何補正等の基本的な処理や LAI/FPAR・NDVI 推定などの様々な高度な処理が施されたデータが蓄積・公開されており、グローバルなリモートセンシング解析において重要なデータとなっています。

※参考

受動型センサー：太陽から放射された電磁波が物体に当たって反射される電磁波を記録するタイプ

能動型センサー：自らエネルギー源を持ち、様々な物体との相互作用による反射応答を記録するタイプ

この高度な処理済データは（表４－１）、the Land Processes Distributed Archive Center (LP DAAC)(http://edcdaac.usgs.gov/tutorial/datapool.html)からダウンロードすることができます。なお、LP DAAC で配布されている MODIS データは、下図の各グリッド領域ごとに CEOS HDF 形式のファイルで配布されており、その地理座標系としては SIN(Sinosoid)または ISIN(Integerd Sinosoid)座標系が用いられています。

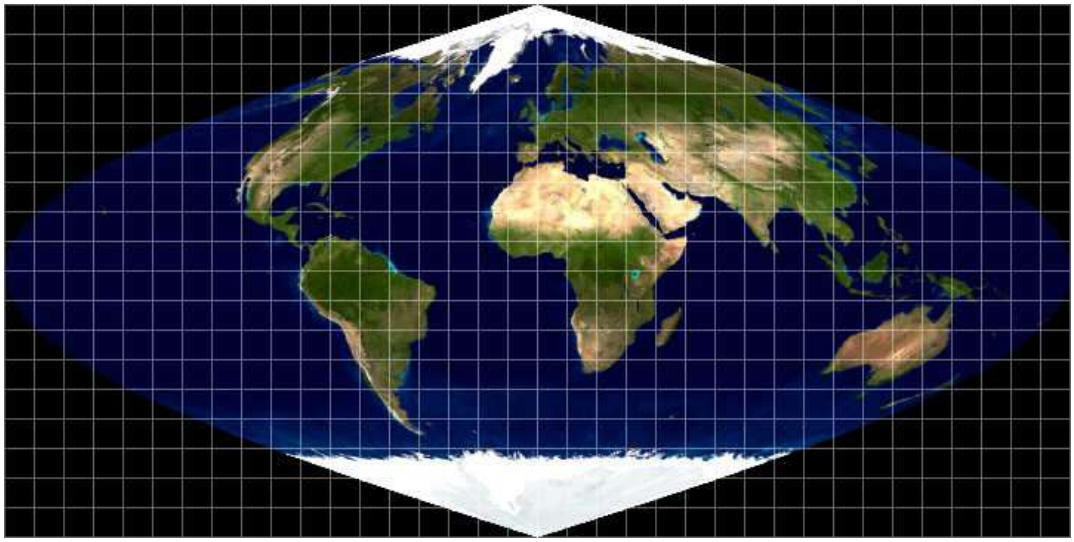


図4－1. LP DAAC で配布されている MODIS データの配布単位

表４－１．LP DAAC で配布されている MODIS の高度な処理済データ

Data Set	Description
MOD09A1.4	MODIS/Terra Surface Reflectance 8-Day L3 Global 500m SIN Grid
MOD09GHK.4	MODIS/Terra Surface Reflectance Daily L2G Global 500m SIN Grid
MOD09GQK.4	MODIS/Terra Surface Reflectance Daily L2G Global 250m SIN Grid
MOD09GST.4	MODIS/Terra Surface Reflectance Quality Daily L2G Global 1km SIN Grid
MOD09Q1.4	MODIS/Terra Surface Reflectance 8-Day L3 Global 250m SIN Grid
MOD11A1.4	MODIS/Terra Land Surface Temperature/Emissivity Daily L3 Global 1km SIN Grid
MOD11A2.4	MODIS/Terra Land Surface Temperature/Emissivity 8-Day L3 Global 1km SIN Grid
MOD11B1.4	MODIS/Terra Land Surface Temperature/Emissivity Daily L3 Global 5km SIN Grid
MOD11C1.4	MODIS/Terra Land Surface Temperature/Emissivity Daily L3 Global 0.05Deg CMG
MOD11C2.4	MODIS/Terra Land Surface Temperature/Emissivity 8-Day L3 Global 0.05Deg CMG
MOD11C3.4	MODIS/Terra Land Surface Temperature/Emissivity Monthly L3 Global 0.05Deg CMG
MOD11_L2.4	MODIS/Terra Land Surface Temperature/Emissivity 5-Min L2 Swath 1km
MOD13A1.4	MODIS/Terra Vegetation Indices 16-Day L3 Global 500m SIN Grid
MOD13A2.3	MODIS/Terra Vegetation Indices 16-Day L3 Global 1km ISIN Grid
MOD13A2.4	MODIS/Terra Vegetation Indices 16-Day L3 Global 1km SIN Grid
MOD13Q1.4	MODIS/Terra Vegetation Indices 16-Day L3 Global 250m SIN Grid
MOD14.4	MODIS/Terra Thermal Anomalies/Fire 5-Min L2 Swath 1km
MOD14A2.4	MODIS/Terra Thermal Anomalies/Fire 8-Day L3 Global 1km SIN Grid
MOD15A2.3	MODIS/Terra Leaf Area Index/FPAR 8-Day L4 Global 1km ISIN Grid
MOD15A2.4	MODIS/Terra Leaf Area Index/FPAR 8-Day L4 Global 1km SIN Grid

Data Set	Description
MOD17A2.3	MODIS/Terra Net Photosynthesis 8-Day L4 Global 1km ISIN Grid
MOD17A2.4	MODIS/Terra Net Photosynthesis 8-Day L4 Global 1km SIN Grid
MOD17A3.3	MODIS/Terra Net Primary Production Yearly L4 Global 1km ISIN Grid
MOD43B1.4	MODIS/Terra BRDF/Albedo Model-1 16-Day L3 Global 1km SIN Grid
MOD43B3.4	MODIS/Terra Albedo 16-Day L3 Global 1km SIN Grid
MOD43B4.4	MODIS/Terra Nadir BRDF-Adjusted Reflectance 16-Day L3 Global 1km SIN Grid
MOD43C1.4	MODIS/Terra Albedo 16-Day L3 Global 0.05Deg CMG
MOD43C2.4	MODIS/Terra BRDF/Albedo Parameters 16-Day L3 Global 0.05Deg CMG
MOD43C3.4	MODIS/Terra Nadir BRDF-Adjusted Reflectance 16-Day L3 Global 0.05Deg CMG

４-１．MODIS/TERRA Vegetation Indices 16-Day L3 Global 250m SIN Grid データ

今回用いるのは、空間解像度約 250m の MODIS/TERRA Vegetation Indices 16-Day L3 Global 250m SIN Grid データです（表４－２）。LP DAAC で公開されている高度な処理済データは、通常一定期間のコンポジットデータとなっています。これは、衛星の軌道が雲の上にあるので、天候により陸域の観測ができなかったり、観測状態が良くないデータが混在するため、一定期間内の最も状態の良いデータを用いて画像を再構築するためです。

MODIS/TERRA Vegetation Indices 16-Day L3 Global 250m SIN Grid データは、植生指標(VI: Vegetation Index)である NDVI（Normalized Vegetation Index）及び EVI（Enhanced Vegetation Index)と、それらの計算に必要な観測画像が、そのデータの状態(Quality)とともに、1 つの CEOS HDF 形式のファイルにまとめれています。なお、このデータの各グリッド領域のファイルサイズは無圧縮で約 500MB（参考：CD1 枚の容量は 640MB 又は 700MB)あります。

表４－２．MODIS/TERRA Vegetation Indices 16-Day L3 Global 250m SIN Grid データの概要

SDS	UNITS	DATA TYPE-bit	FILL VALUE	VALID RANGE	Divide by SCALE FACTOR
250m 16 days NDVI	NDVI	16-bit signed integer	-3000	-2000- 10000	10000
250m 16 days EVI	EVI	16-bit signed integer	-3000	-2000- 10000	10000
*250m 16 days NDVI Quality	bit field	16-bit unsigned integer	65535	0 - 65536	na
*250m 16 days EVI Quality	bit field	16-bit unsigned integer	65535	0 - 65536	na
250m 16 days red reflectance MODIS Band # 1, 620 - 670 nm	reflectance	16-bit signed integer	-1000	0 - 10000	10000
250m 16 days NIR reflectance MODIS Band # 2, 841- 876 nm	reflectance	16-bit signed integer	-1000	0 - 10000	10000
250m 16 days blue reflectance ^MODIS Band # 3, 459 - 479 nm	reflectance	16-bit signed integer	-1000	0 - 10000	10000
250m 16 days MIR reflectance ^MODIS Band # 7, 2105- 2155 nm	reflectance	16-bit signed integer	-1000	0 - 10000	10000
250m 16 days average view zenith angle View zenith of the chosen pixel	degree	16-bit signed integer	-10000	-9000 - 9000	100
250m 16 days average sun zenith angle Sun zenith of the chosen pixel	degree	16-bit signed integer	-10000	-9000 - 9000	100
250m 16 days average relative azimuth angle Relative Azimuth of the chosen pixel	degree	16-bit signed integer	-4000	-3600 - 3600	10

※赤字は今回使用する SDS。

植生指標(VI)とは

緑色の葉に含まれるクロロフィルの反射率は、0.5～0.7μｍで 20%未満であるのに対して、0.7～1.3μｍの近赤外域では 60%を越えます。春から夏の植生の生育期には緑葉は可視光、特に赤の光をより強く吸収し、近赤外の光はより強く反射するようになります。また、植被率が増加すると全体として近赤外の反射率が増加するので、赤(Red)と近赤外(NIR)の波長帯の反射率等の差あるいは比は、植生の活性、植被率、葉面積指数（LAI）といった植生パラメーターと相関を持つようになります。

これらの特徴を利用して、赤と近赤外の波長帯の反射率等から計算される植生指標が幾つも提案されています。その代表が正規化植生指標(NDVI: Normalized Difference Vegetation Index)で、以下の式により算出されます（-1～1 の間の値となりますが、通常これに適当な数値をかけた値が用いられます）。

$$NDVI = (NIR - Red) / (NIR + Red)$$

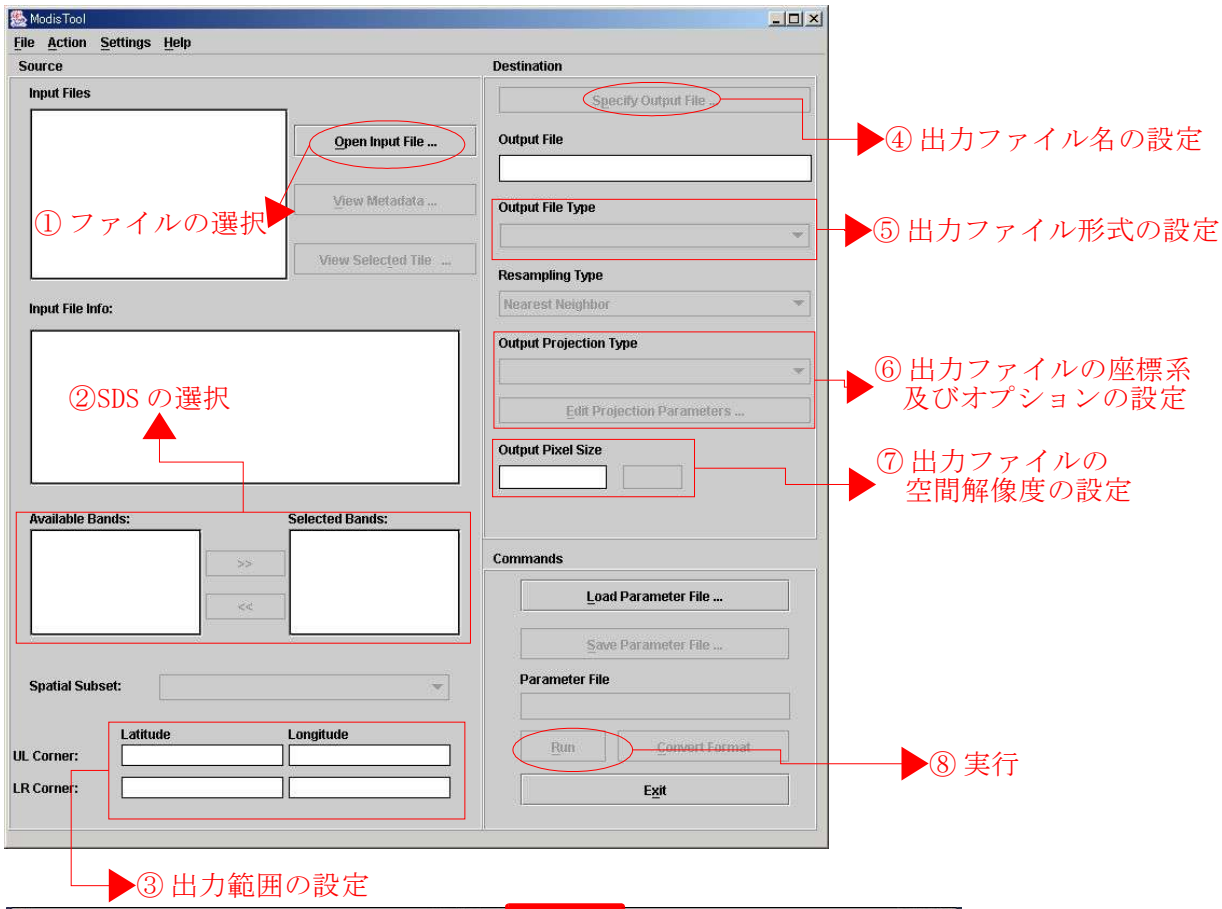
４－２．MODIS Reprojection Tool

GRASS では、CEOS HDF 形式のデータを直接インポートする機能はありませんが、MODIS Reprojection Tool (http://lpdaac2.usgs.gov/landdaac/tools/modis/index.asp)を利用すれば、

- 複数のグリッドデータの結合
- グリッド内の領域の切り出し

- データ種目(SDS)の選択
- 緯経度・投影変換

等の処理だけでなく、GRASS にインポート可能なデータ形式（GEOTIFF 形式または RAW BINARY 形式）への変換を行うことができます。



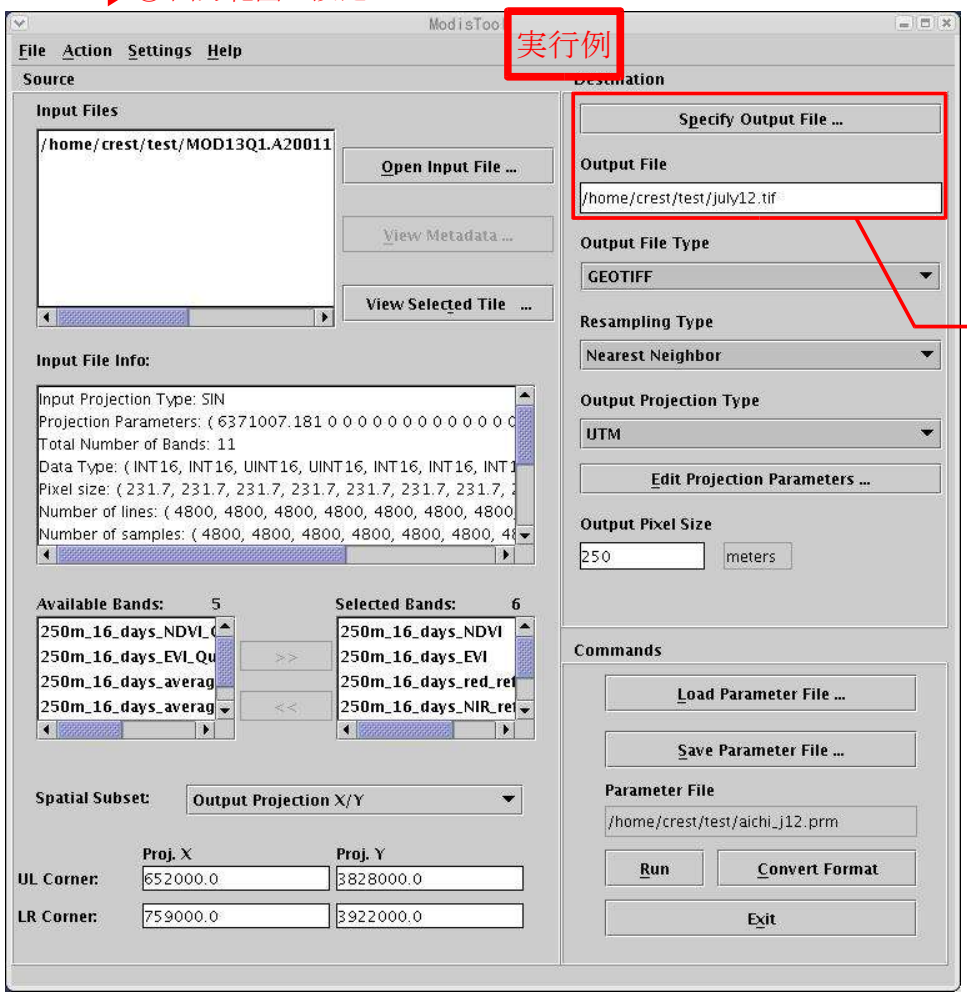


図4－2．**MODIS Reprojection Tool (MRT)** の利用手順(上)と設定例(下)

今回は、表４－２に示したMODIS/TERRA Vegetation Indices 16-Day L3 Global 250m SIN Gridデータの６つのSDSについて、MRTにより下記に示した設定で既に処理を行ったデータを利用します。

- 形式　：RAW BINARY形式
- 解像度：250m
- 座標系：UTM座標（Zone: 53）
- 範囲　：(652250, 3827500)-(758750, 3921750)

MRTでは、出力形式がRAW BINARYの場合、【④出力ファイル名の設定】で指定されたファイル名から、【②SDSの選択】で選択されたSDS（Selected Bandsにある SDS)それぞれについて、１つの画像を出力します。例えば、④で

/home/bagr/july12.hdr

と指定した場合、250m 16 days NDVIのSDSが選択されていれば、このSDSのRAW BINARYデータを

/home/bagr/july12.250m_16_days_NDVI.dat

というファイルに出力します。また、/home/bagr/july12.hdrというファイルも同時に作成され、その中には出力データに関する様々な情報（範囲、座標系、解像度・・・）が記録されます（図４－３）。

```
PROJECTION_TYPE = UTM
UTM_ZONE = 53
PROJECTION_PARAMETERS = (
        0.000000000     0.000000000     0.000000000
        0.000000000     0.000000000     0.000000000
        0.000000000     0.000000000     0.000000000
        0.000000000     0.000000000     0.000000000
        0.000000000     0.000000000     0.000000000 )

# COORDINATE_ORIGIN = UL
UL_CORNER_LATLON = ( 35.427547075 136.677154504 )
UR_CORNER_LATLON = ( 35.405569685 137.849307966 )
LL_CORNER_LATLON = ( 34.578015064 136.659902680 )
LR_CORNER_LATLON = ( 34.556716246 137.820030036 )
# UL_CORNER_XY = ( 652250.000000000 3921750.000000000 )
# UR_CORNER_XY = ( 758750.000000000 3921750.000000000 )
# LL_CORNER_XY = ( 652250.000000000 3827500.000000000 )
# LR_CORNER_XY = ( 758750.000000000 3827500.000000000 )
NBANDS = 6
BANDNAMES = ( 250m_16_days_NDVI 250m_16_days_EVI 250m_16_days_red_reflectance 250m_16_days_NIR_reflectance
250m_16_days_blue_reflectance 250m_16_days_MIR_reflectance )
DATA_TYPE = ( INT16 INT16 INT16 INT16 INT16 INT16 )
NLINES = ( 377 377 377 377 377 377 )
NSAMPLES = ( 426 426 426 426 426 426 )
PIXEL_SIZE = ( 250.000000 250.000000 250.000000 250.000000 250.000000 250.000000 )
MIN_VALUE = ( -2000 -2000 0 0 0 0 )
MAX_VALUE = ( 10000 10000 10000 10000 10000 10000 )
BACKGROUND_FILL = ( -3000 -3000 -1000 -1000 -1000 -1000 )
DATUM = WGS84
# SCALE_FACTOR = ( 10000.000000 10000.000000 10000.000000 10000.000000 10000.000000 10000.000000 )
# OFFSET = ( 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 )
```

図４－３．RAW BINARY形式のヘッダー情報

４－３．　MODISデータのインポート

MODISデータはUTM座標系ですから、緯度経度座標系のLOCATION(aichi)を終了し、UTM座標系のLOCATION(modis)で再度GRASSを起動します。

MODISのRAW BINARY形式の画像をインポートするためのコマンドは以下のとおりです。

```
r.in.bin▽-b▽-s▽input=july12.250m_16_days_EVI.dat▽output=july12.evi▽bytes=2▽
        north=3921750▽south=3827500▽west=652250▽east=758750▽rows=377▽cols=426▽
                                                        anull=-3000
```

下線部のオプションは、図４－３に記載されている情報を用いています。さらに、インポートした画像の表示（図４－４）は、メッシュ（ラスター）データ同様以下のコマンドで行います。

```
d.rast▽july12.evi
```

MODISデータのような画像データが、一般的なラスターデータと異なるのは、通常画像データは表４－２のように複数の画像をセットにして使用します。今回用いるのは、衛星による観測画像だけでなく、それにより計算された画像も含みますが、表４－２のUNITSの欄がreflectanceとなっているものは、衛星に観測された値で、それぞれ特定の波長帯において物質（ここでは地球の表面）が太陽光を反射する強度を表します。

通常我々が見ているデジタルカラー画像は、可視域の３つの波長帯（赤・緑・青）（通常、波長帯のことをバンドと呼んでいます）の反射強度（パソコンでは、通常０～２５５の値を持つ）を組み合わせることにより表現されています。リモートセンシング画像の表示では、観測された各波長帯のうち、いずれか３つの反射強度を赤・緑・青いずれかに割り当てることにより、様々な画像表現を行います。通常、赤・緑・青に同じ赤・緑・青の波長帯の反射強度を割り当

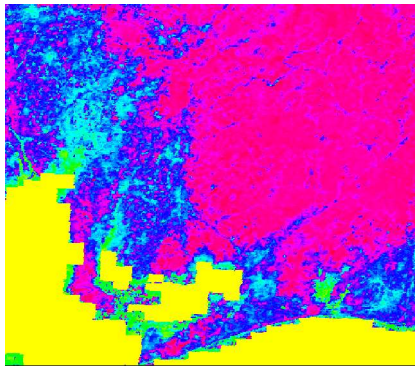


図4－4. インポートされた**MODIS**画像

てて表示した画像をフルカラーまたはナチュラルカラー画像と、赤・緑・青に別の長帯の反射強度を割り当てて表示した画像をフォルスカラー画像と呼んでいます。

　これらの画像表現をコンピュータ上で行うためには、まず各波長帯の反射強度で構成されている各メッシュの値を０～２５５の値に正規化する必要があります。GRASS では、以下の操作によりこれを行います。

r.colors▽map=july12.evi▽color=grey

この操作では、各メッシュの値が変化するのではなく、各メッシュの値が０～２５５のどの値に対応するのかを表す変換テーブルのようなものを作成します。このテーブルは表示時のみに利用され、各メッシュの値は何も変更されません。例えば、上記の操作後に同じ画像を

d.rast▽july12.evi

により再表示すると図４－５のようになります。

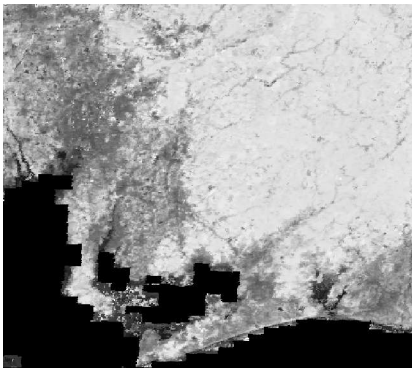


図4－5. 再表示された **MODIS** 画像

　今回は、表４－２に示したように４つのバンドを含む６つの SDS を利用します。また、今回は july12（7 月 12 日から 16 日間のコンポジットデータ）だけでなく、july28, november01, november17 の４時期（２ヶ月×２）のデータを利用します。したがって、表４－２のとおり各時期で６つのデータがありますから、合計 24 枚の画像を利用します。各画像を上記のようなコマンドで一つ一つインポートするのは大変ですから、今回は下記に示した Linux のシェルスクリプトを利用して、一括して処理を行います。

```
1：lst=("july12" "july28" "november01" "november17")
2：opt="bytes=2 north=3921750 south=3827500 west=652250 east=758750 rows=377 cols=426"
3：anull1=" anull=-3000"
4：anull2=" anull=-1000"
5：for days in ${lst[@]};do
6：    r.in.bin -b -s input=${days}.250m_16_days_EVI.dat output=${days}.evi $opt $anull1
7：    r.in.bin -b -s input=${days}.250m_16_days_NDVI.dat output=${days}.ndvi $opt  $anull1
8：    r.in.bin -b -s input=${days}.250m_16_days_MIR_reflectance.dat output=${days}.mir $opt $anull2
9：    r.in.bin -b -s input=${days}.250m_16_days_NIR_reflectance.dat output=${days}.nir $opt $anull2
10：   r.in.bin -b -s input=${days}.250m_16_days_blue_reflectance.dat output=${days}.blue $opt $anull2
11：   r.in.bin -b -s input=${days}.250m_16_days_red_reflectance.dat output=${days}.red $opt $anull2
12：   r.colors map=${days}.evi color=grey
13：   r.colors map=${days}.mir color=grey
14：   r.colors map=${days}.ndvi color=grey
15：   r.colors map=${days}.nir color=grey
16：   r.colors map=${days}.blue color=grey
17：   r.colors map=${days}.red color=grey
18：done
```

５．植生分類

５－１．教師付き分類と教師無し分類

植生分類を行う手段としては、教師付き分類と教師無し分類に大別することができます。教師付き分類とはグランドトゥルース（地上での実測値）を使用する分類法で、教師無し分類はグランドトゥルースを使用しない分類法です。

教師付き分類は、画像からあらかじめ定められたカテゴリー（植生区分）を最もよく代表していると思われる領域を抽出する手法です。このとき指定される領域又は領域群をトレーニングエリア又はトレーニングデータと呼びます。トレーニングエリアは、例えば、オペレータがあらかじめ地図データから現地の情報を把握している場合、その領域と画像データとの対応付けを行い、画像データの分類を行う手法です。教師付き分類は、標本の無作為性および独立性に問題が生じる可能性があるという欠点があります。

一方、教師無し分類では、画像データの画素をランダムに選択し、統計的手法を使用して、いくつかのクラスタに分類し、カテゴリーに対応づける手法です。教師無し分類では、生成されたクラスタとカテゴリーとの対応づけが困難であるという欠点があります。

５－２．投影変換

これまでに、異なる座標系を持つ、２つのLOCATION(aichi, modis) を利用してきました。GRASS では、同じLOCATION 内であれば、MAPSET が異なってもデータを利用することができますが、異なるLOCATION 間ではそのままデータを利用し合うことはできません。特に、今回のように両者の利用する座標系が異なる場合は、データをそのままコピーしても、本来同じ場所のデータであっても両者は全く異なる場所のデータとして扱われてしまうでしょう。

GIS では様々な座標系の地図を利用しますから、ある座標系から別の座標系へデータを変換する手法が用意されています。そのような場合に利用するのが、投影変換という手法です。今回は、LOCATION(aichi)にインポートした緯度経度座標系のデータを、LOCATION(modis)のUTM 座標系に投影変換します。GRASS では、ポイント・ラスター・ベクトルの各データ種目ごとに異なるコマンドが用意されていますが、その利用の仕方はほとんど同じです。ただし、ラスターでは、投影変換によりメッシュの形状及びサイズ自体が変わってしまうので、通常は合わせて何らかの補間を行います。

では、LOCATION(aichi)のポイント・ラスター・ベクトルの各データの投影変換はを下記のコマンドで行います。

```
s.proj▽input=gunraku▽output=gunraku▽mapset=PERMANENT▽location=aichi
v.proj▽-s▽input=aichi▽output=aichi▽mapset=PERMANENT▽location=aichi
r.proj▽input=topo▽output=topo▽mapset=PERMANENT▽location=aichi▽method=cubic▽
                                                          resolution=250
```

投影変換が終了したら、以下のコマンドで各データがLOCATION(aichi)からLOCATION(modis)に正しく変換されているか確認します。

```
d.vect.area▽map=aichi▽fillcolor=blue▽linecolor= black
d.rast▽topo
d.sites▽sitefile=gunraku
```

５－３．ポイント→ラスター変換

今回は、自然環境保全基礎調査データの自然度をトレーニングデータとして利用して教師付き分類を行います。GRASS ではポイントデータをトレーニングデータとして利用することはできないので、ラスターデータに変換して利用します。GRASS では、ポイントデータからラスターデータへの変換は、下記のコマンドにより行います。

```
s.to.rast▽input=gunraku▽output=gunraku▽size=0▽findex=2▽string=2
```

変換が終了したら、以下のコマンドでどのように変換されたか確認します。

```
d.rast▽gunraku
```

５－４．教師付き分類

１）イメージグループの作成

GRASS では、個々の画像を単位として扱うだけでなく、複数の画像を１つのグループとして扱うことにより、画像の合成や分類などより高度な画像解析が可能になります。GRASS では、このグループとして２つの単位概念があります。

その１つがイメージグループで、もう一つがそのイメージグループ中に作られるサブグループです。イメージグループは解析対象となる画像全ての画像群で、サブグループはその中で分類に利用される画像群です。サブグループにイメージグループ内の全ての画像を指定することもできますし、同じ画像を異なるサブグループに重複して指定することもできます（図５－１）。画像分類を使用しない場合はサブグループを設定する必要はありませんが、画像分類を利用する場合は必ず設定する必要があります。

今回は、植生分類にGRASS の教師付き分類の機能を利用しますので、イメージグループとサブグループの設定が必要です。ラスターデータ（今回はMODIS 画像）からのイメージグループの作成は以下のコマンドで行います。

```
i.group
```

このコマンドでのイメージグループとサブグループの設定手順は図５－２のとおりです。

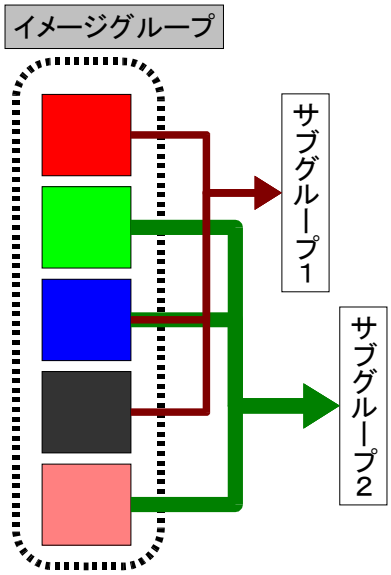


図５－１．イメージグループとサブグループ

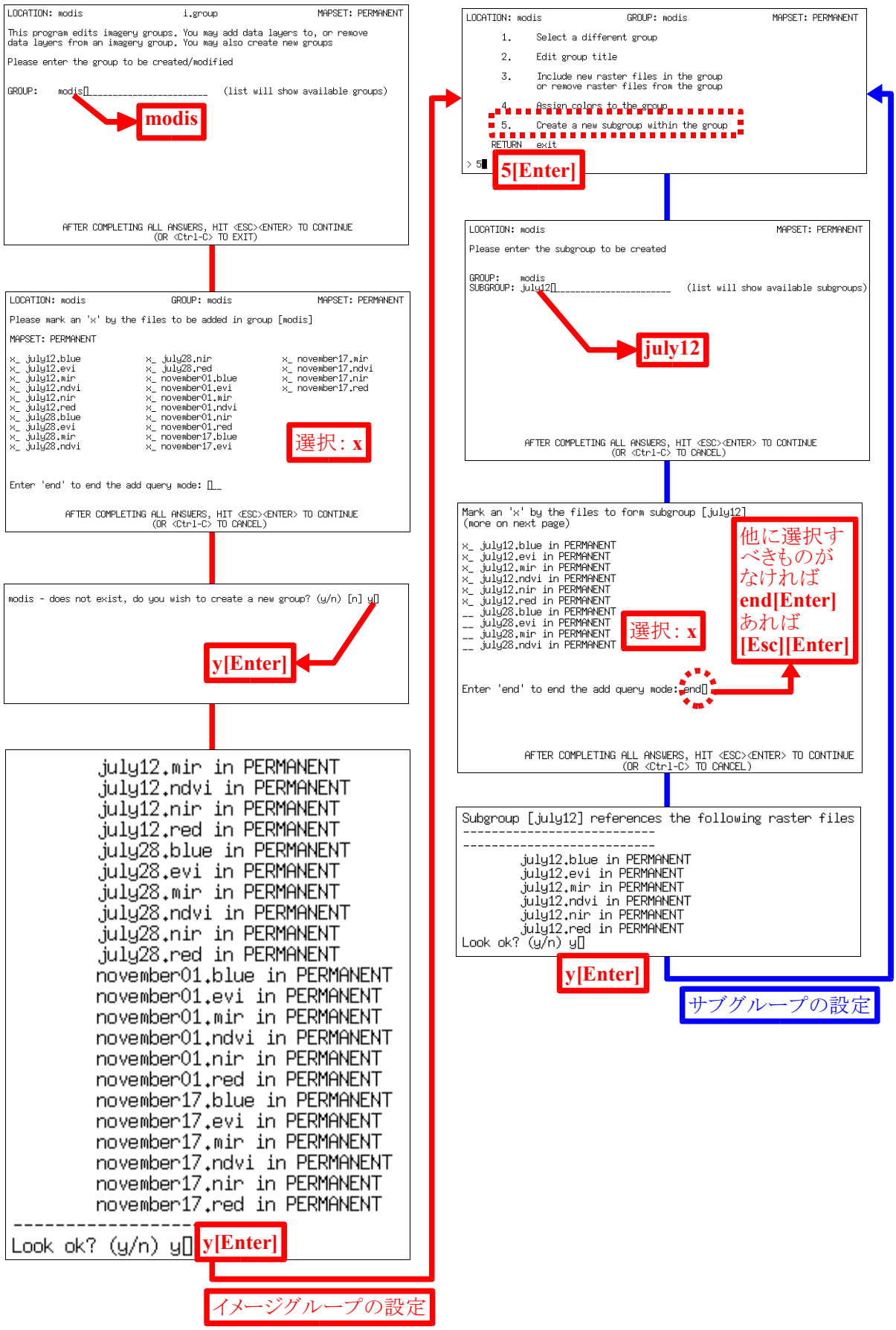


図5－2. イメージグループとサブグループの設定

２）マスク設定

MODIS 画像には、愛知県以外の部分や海域も含まれています。この画像から愛知県以外の部分を削除して表示するには、どうすればよいでしょうか？その方法としては２つあります。

GRASS には、マスク設定を行うことにより、ラスターデータの処理（表示・各種演算）を特定の部分のみに制限する機能があります。Region では方形のエリアしか設定できませんが、マスクでは制限する部分の形状に制限はありません。MASK という名前の０と１のカテゴリー値のみで構成されたラスターデータを作れば、以後ラスターデータの処理は、MASK というラスターデータが１の値を持つメッシュだけに適用され、その他のメッシュは処理から除外されます。例えば、投影変換した愛知県のベクトルデータから以下のコマンドでラスター変換します。

v.to.rast∇input=aichi∇output=aichi

変換されたラスターデータでは、市町村ごとに行政コードがメッシュの値として変換されますので、以下のコマンドで表示すると、市町村ごとに色分け表示されるのがわかります。

d.rast∇aichi

ただし、マスク設定を行うためには、愛知県内のメッシュは全て１の値を持つ必要があるので、以下のラスター演算コマンドを用いてマスク設定を行います。

r.mapcalc∇″MASK=if(aichi>0,1,0)″

では、その効果を以下のコマンドで見てみましょう。

d.rast∇july12.evi

※マスク設定では、分類画像自体は何も変わりません。表示等の処理を行う場合に、MASK というラスターデータが１を持つ部分以外が除外されるだけです。マスクを削除して、再び分類画像を表示するとそれがわかります。

g.remove∇rast=MASK
d.rast∇july12.evi

３）トレーニングマップによるシグネチャー（**signature**）ファイルの作成

今回は、５－３でラスター変換した自然環境保全基礎調査データ(gunraku)をトレーニングマップとして用います。教師付き分類を行うためには、まずサブグループに設定された画像から、トレーニングマップ内に存在する各分類項目（今回は自然度の各区分）と同一の位置のピクセル（メッシュ）のデータを抽出し、分類の基礎となる統計値を計算する必要があります。それを行うコマンドは、以下のとおりです。

i.gensig∇group=modis∇subgroup=july12∇signaturefile=july12.sig∇trainingmap=gunraku
i.gensig∇group=modis∇subgroup=all∇signaturefile=all.sig∇trainingmap=gunraku

上記は、２つのサブグループを対象に、それぞれ統計値の計算を行い、シグネチャーファイル(july12.sig, all.sig)に出力しています。

４）教師付き分類の実行

３）で計算した シグネチャーファイルを用いた対象地域（画像全体）の分類は、以下のコマンドにより行います。

i.maxlik∇group=modis∇subgroup=july12∇sigfile=july12.sig∇class=july12.class
i.maxlik∇group=modis∇subgroup=all∇sigfile=all.sig∇class=all.class

これにより、トレーニングマップで設定した各分類項目（今回は自然度）に、シグネチャーファイルに記録されている統計値を用いて最尤法により画像分類が行われます（図５－３）。ただし、トレーニングマップで設定した分類項目全てが分類された画像に現れるとは限りません。統計的に有意でない（無意味である）と判断された分類項目は、分類時に自動的に削除されます。

では、分類画像と凡例（今回は自然度コード）を以下のコマンドで表示します

d.rast∇july12.class
d.mon∇start=x1∇select=x1
d.legend∇map=july12.class
d.mon∇select=x0
d.rast∇all.class
d.mon∇select=x1
d.erase
d.legend∇map=all.class

本来、d.legend コマンドは凡例の数値（カテゴリー値：自然度コード）だけでなく、カテゴリーラベル（自然度）もその説明として表示できるのですが、今回のカテゴリーラベルは日本語のため画面には表示されません（半角英数字の文字であれば表示可能です）。

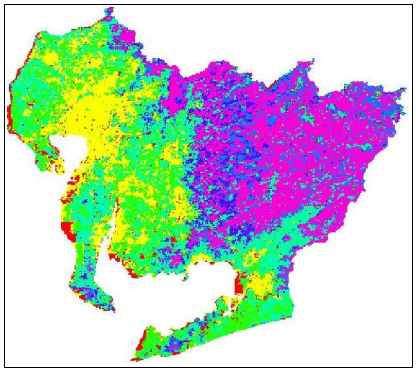


図５－３．分類画像 **(all.class)**

５－５．市町村別の自然度構成（面積比）の集計

市町村ごとに各分類項目（今回は自然度）がどれくらいの割合を占めているのか、集計してみましょう。ラスターデータの集計（面積）は以下のコマンドで行います。

r.report∇map=aichi, july12.class∇units=p
r.report∇map=aichi, all.class∇units=p