PNMapcalc 1.0

09 May 2018

Paweł Netzel

Table of Contents

Introduction

PNMapcalc is a standalone raster map calculator. It works under Windows (7, 8.1, 10) and Linux (FC25). The application depends on GDAL (2.1.2). So, it is necessary to install GDAL before PNMapcalc installation. Windows version of PNMapcalc is provided with an installation program/setup. Windows PNMapcalc setup will install all needed dependencies.

PNMapcalc's user defines macros and scripts to perform calculations on raster layers. These macros should be written using C language syntax. It is possible to define simple operations such as adding two raster layers or NDVI calculation. User can also define complex scripts to perform more advanced raster analysis (for example: calculation of Koeppen-Geiger climate classification).

PNMapcalc is a fast and efficient software. It compiles macro to raw binary code and calls it for every raster cell. Thanks to that, the application is 4 times faster than GRASS r.mapcalc module.

PNMapcalc features:

- up to 255 input layers;
- up to 255 output layers;
- C language syntax;
- local auxiliary variables;
- global memory variables;
- command line macros and files with scripts;
- mathematical functions.

PNMapcalc limitations:

- input layers are treated as floating point layers;
- pixel-to-pixel analysis only, no neighborhood operators;
- forcing the same projection, resolution and extent for all input layers;
- output files are GeoTIFFs only;
- GDAL dependency.

PNMapcalc is a free and open source software. It is released under GNU GPL v.3 license.

A user can find binaries, sources, and manuals on the web page:

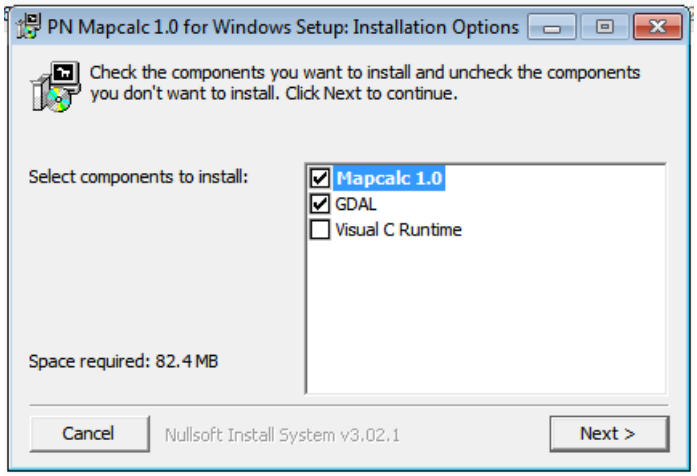
http://pawel.netzel.pl/index.php?id=software#a_mapcalc

Installation

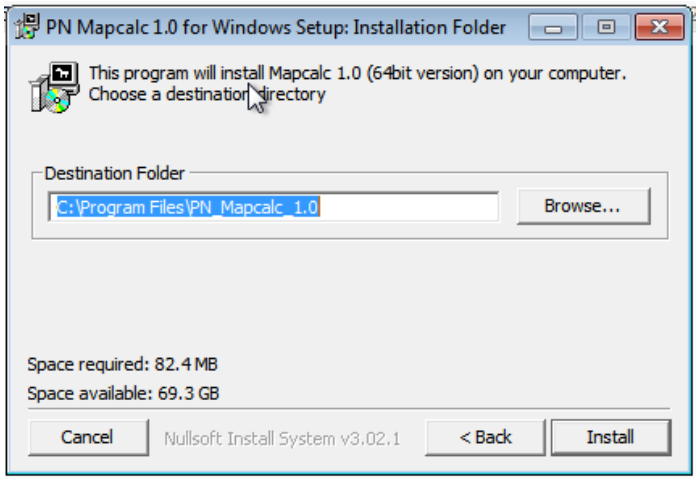
To install PNMapcalc, a user has to run setup program or install ready to use package. These binaries can be downloaded form program web page listed in the section „Introduction”.

Windows

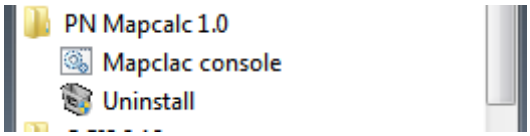
PNMapcalc was tested under Windows 7 and Windows 10. The application is packaged in one setup exe program. The setup will install PNMapcalc. Additionally, a user can select two extra packages: GDAL (usually this package is necessary) and Visual C runtime libraries.



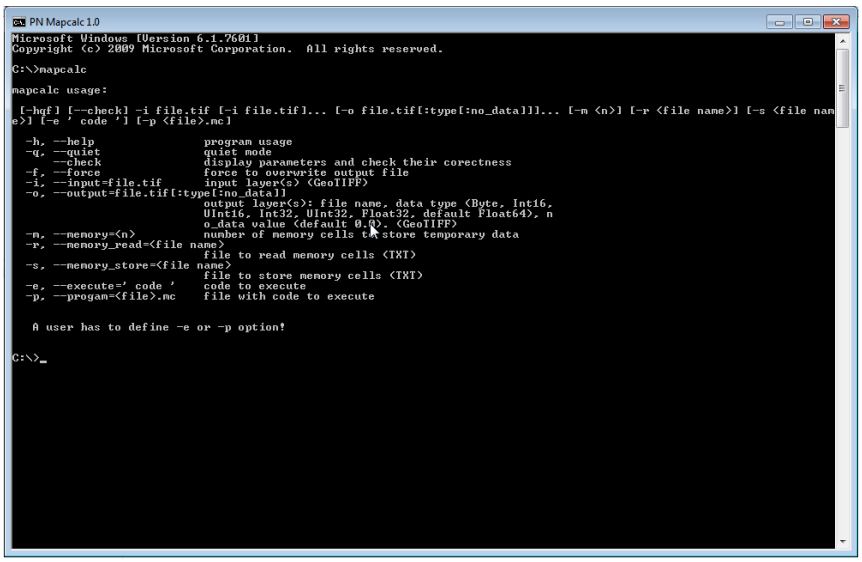
The user can also select a destination folder where the program will be installed.



After successful installation, the new position will be created in Windows start menu.



The main program to run PNMapcalc is *Mapcalc console*. This shortcut will open new windows console with mapcalc environment ready to work.



In this CMD session the user can also call all GDAL utilities.

Linux

PNMapcalc is available in a binary form as RPM package. The package is for Fedora Core 25.

It contains mapcalc binary, necessary libraries, and headers. The package can be installed using *dnf* package manager. *dnf* will install all dependences, in particular – GDAL.

The command to install PNMapcalc package:

```
dnf install mapcalc-1.0-FC25.x86_64.rpm
```

Alternatively, it is possible to build PNMapcalc from sources. If system contains GDAL and TCC, the user can compile the application by invoking

```
make
```

and install it by

```
make install
```

PNMapcalc will be installed in /usr/bin directory. It will look for tcc library, its components, and headers in /usr/lib and /usr/lib/tcc.

How to use

PNMapcalc runs macros to perform calculations on spatial data. It is a command line tool. So, a user has to know both command line syntax to run mapcalc command and macro syntax to program calculation procedure or expression.

Command line syntax

To run PNMapcalc, a user has to call *mapcalc* program. The program can describe itself. One can rut it without any option or with *-h* or *--help* option. The result is as follows:

```
$ ./mapcalc

usage:

./mapcalc [-hqf] [--check] -i file.tif [-i file.tif]... [-o
file.tif[:type[:no_data]]]... [-m <n>] [-r <file name>] [-s <file name>]
[-e ' code '] [-p <file>.mc]

  -h, --help                 program usage
  -q, --quiet                quiet mode
      --check                display parameters and check their correctness
  -f, --force                force to overwrite output file
  -i, --input=file.tif       input layer(s) (GeoTIFF)
  -o, --output=file.tif[:type[:no_data]]
                             output layer(s): file name, data type (Byte,
                             Int16, UInt16, Int32, UInt32, Float32,
                             default Float64), no_data value (default 0.0).
                             (GeoTIFF)
  -m, --memory=<n>           number of memory cells to store temporary data
  -r, --memory_read=<file name>
                             file to read memory cells (TXT)
  -s, --memory_store=<file name>
                             file to store memory cells (TXT)
  -e, --execute=' code '     code to execute
  -p, --program=<file>.mc    file with code to execute

   A user has to define -e or -p option!
```

The minimum number of arguments is one input raster layer and macro to run. Macro can be provided in-line or as script file with a macro definition.

There are four groups of arguments:

- input/output layer definition: -i, -o;
- memory buffer definition: -m, -r, -s;
- macro definition: -e, -p, --check;
- others: -q, -f, -h.

input/output layer options

A program user has to enter file name of input file using *-i* option. It could be an absolute or relative path to the file. The option *-i* can be used multiple times. That enables a user to define many input raster layers.

The option *-o* is to define output layer. This option can be used multiple times as well. A user can add two parameters after a file name/path. The first one defines data type of output layer. The second – no-data value.

memory buffer options

One can define storage memory buffer in PNMapcalc. This storage can be used for different tasks. For example: to calculate global statistics of raster layer, to build frequency tables or histograms, to reclassify input data etc.

A program user can define how many memory cells should be available. The option *-m* followed by integer number it is for that.

PNMapcalc can read initial data to fill memory buffer from text file. A user has to point path to text file with data to read with option *-r*. Each line in this file should contain memory cell index, blank space, memory cell value. An example of such file is below.

```
0 41623.279724212028668262
1 35513.439075932932610158
2 28021.639102664925303543
3 28041.327999936092965072
4 20369.025455332906858530
```

The input file can contain lines for all memory cell but this is not a critical condition. One can define only selected cells in the text file. Rest of the memory cells will be initialized with zeroes. PNMapcalc will read the data from text file BEFORE start any calculations.

Memory buffer can be stored AFTER all calculations will be finished. If user add *-w* option, PNMapcalc will store ALL memory cells into a text file. Internal structure of the file will be identical to the structure of the input file.

macro options

An equation or macro or program (very complex macro) can be entered in two ways: as a command line parameter or as a file with macro. To specify macro in a command line, the option *-e* has to be used. The macro should be enclosed in quotation marks. In Linux, it can be single quotation characters. In Windows, it should be double quotation characters.

If the macro is large and complex, it can be stored in a text file. Formatting and text aligning in such file is free and should satisfy rules of C language.

The option *-p* specifies the file name (with or without full/relative path). The macro will be read into memory and processed in the same way as entered from command line.

PNMapcalc has additional option --*check* to run „dry” calculation. If this option is present in mapcalc call, the program will only check syntax and try to compile the macro without any calculations. The user can use this option during the macro build.

other options

Among others options, PNMapcalc has a small set auxiliary option. The option *-q* forces the program to run without any messages. The option *-f* tells the program to overwrite output raster layers. By default, the program prevents raster layers overwriting. The option *-h* displays a short usage info.

Macro and macro’s predefined variables

A user should know C syntax to create macros for mapcalc application. But there is no need to learn C language as a language or to learn how to compile and build executables in C.

The only knowledge necessary to work with PNMapcalc is a few C expressions and C-style punctuation. Such knowledge is enough to define most of the spatial raster operations. The macro language of PNMapcalc contains all mathematical functions available in C. If needs arise, a user can create more complex macros and even programs implementing advanced algorithms.

When any doubt occurs, refer to ANSI C language documentation and tutorials.

PNMapcalc works in the following way: reads macro, compiles macro, walks from cell to cell across region defined by input files and runs the macro for each cell. Thanks to the binary format of the macro, calculations are very fast.

Input data, output results, and memory buffer are available in macro through set of variables. These variables are named using capitalics.

Location

Macro should know “where are we”. Answer to this question is couple of variables: COL, ROW. These variables contain location (in terms: which column and which row) of current calculations. Both variables are type of integer.

Input and output

There are four variables responsible for communication with raster layers: IN[], OUT[], INPNUM, OUTNUM. IN[] are an array. That means a user can call an array element with index. All arrays in C are indexed starting from zero. So, a user has to type-in IN[3] to obtain the value of the 4th element of the array. IN array has INPNUM elements. The index can have a value from 0 to INPNUM-1. The value of INPNUM equals to the number of *-i* options in the program command line call.

OUT[] array and OUTNUM variable work in the same way. The only difference is set of layers. OUT[] and OUTNUM are defined base on number output layers.

INPNUM and OUTNUM are integers. IN[] and OUT[] arrays are arrays of double type numbers. Even if input layer contains integer numbers these numbers are converted to double.

Order of layers represented in IN[] and OUT[] arrays depends on order of layers occurrence in command line program call.

Memory buffer

Memory buffer is represented by MEM[] array. The number of elements of this array is defined by *-m* option. Macro knows the number of elements in MEM[] array thanks to MEMNUM integer variable. MEM[] array is an array of double numbers.

Examples

In following examples, you can find hints and different approaches to using PNMapcalc. Examples are ordered from simplest to the most complex. All examples are dedicated to run in Linux. Windows system needs small modifications: different path element separator (in options *-i*,*-o*,*-r*,*-s*) and double quotation marks (in option *-e*).

Example 1: adding layers

Let's assume that you have two layers: *layer1.tif* and *layer2.tif*. Both layers have the same resolution and extent. The layers are in the same projection or both are defined in geographical coordinates. The data stored in the layers are in floating point format. In the case of integer data, PNMapcalc will convert integers to doubles and do calculations using floating point format.

Basic call of PNMapcalc should be:

```
mapcalc -i layer1.tif -i layer2.tif -o sum.tif -e 'OUT[0]=IN[0]+IN[1];'
```

A version of the call with long parameter's format:

```
mapcalc --input=layer1.tif
        --input=layer2.tif
        --output=sum.tif
        --execute='OUT[0]=IN[0]+IN[1];'
```

The call shown above contains two inputs: *layer1.tif* and *layer2.tif* and one output: *sum.tif*. The output layer will be created with default options. That means: Float64 data format without no-data value definition. The first input layer will be represented as IN[0] in the macro, the second as IN[1]. The array IN[] is an array of floating point numbers with double precision. Similarly, the output layer sum.tif is represented as a first element of floating point array OUT[] (array elements are indexed starting from zero);

If the user wants to get an output layer that contains integers and has no-data value, the mapcalc syntax should looks as follow:

```
mapcalc -i layer1.tif -i layer2.tif
        -o sum.tif:Int32:-9999 -e 'OUT[0]=IN[0]+IN[1];'
```

Int32 means that output values will be converted to 4-byte integer. The conversion removes fractional part of the value. The value -9999 will be used as no-data marker.

Sometimes, the default rounding function is not desired. The user can explicitly round output values in the macro. For example:

```
mapcalc -i layer1.tif -i layer2.tif
        -o sum.tif:Int32:-9999 -e 'OUT[0]=floor(IN[0]+IN[1]);'
```

The resultant layer will have the same projection, resolution, and extent as input layers.

Example 2: calculation of NDVI

The normalized difference vegetation index (NDVI) is a simple graphical indicator that can be used to analyze remote sensing measurements, typically, but not necessarily, from a space platform, and assess whether the target being observed contains live green vegetation or not.

In the case of Landsat 5 TM data, the NDVI is calculated as follows:

```
NDVI = (band 4 - band 3) / (band 4 + band 3)
```

This equation can be undefined in the case of zero in denominator. Because mapcalc does calculation on double precision floating point values, dividing by zero will return *+Inf* or *-Inf* value. Sometimes, this behavior is acceptable. If not, the user should handle such case.

A macro to calculate NDVI can looks like following:

```
mapcalc  -i band4.tif
         -i band3.tif
         -o ndvi.tif:Float32:-9999
         -e ‘double x = IN[1] + IN[0];
            if(x==0.0) OUT[0]=-9999;
            else OUT[0]=(IN[0]-IN[1])/x;’
```

The macro contains local variable *x*. This variable is local in the context of spatial cell calculation. The variable *x* is type of double and it is initialized with sum of both input layers: band3 and band4. If variable x contains zero, ndvi will have no-data value (*-9999*).

Example 3: a *mask* file creating

Masking spatial data is used very often during spatial analysis. PNMapcalc is a convenient tool to create a mask using complex conditions. This example is presenting how to create a mask for selecting raster cell belonging to a given country.

The user has raster layer covering all Europe. Each country has its own index. The user is going to create a mask for country with index 38.

Command line call of mapcalc should be as follows

```
mapcal -i europe.tif -i my_country_mask.tif:Byte
       -e ‘OUT[0]=(IN[0]==38)?1:0;’
```

The expression: (IN[0]==38)?1:0 means: if cell of europe.tif contains value 38 then set value 1 else set value 0. The resultant map will contain ones over country area and zeroes elsewhere.

Example 4: statistics calculation

Let’s use results of example 2 and example 3 to obtain average NDVI value of the selected country.

The user should use a memory buffer to calculate statistics over given area.

Command line call of mapcalc should be as follows

```
mapcalc -i ndvi.tif -i my_country_mask.tif
```

```
-m 3 -e 'if(IN[1]>0) {MEM[0]+=1;
         MEM[1]+=IN[0]; MEM[2]=MEM[1]/MEM[0];}'
-s result.txt
```

PNMapcalc will create result.txt file. The file will contain three lines. First line, corresponding to MEM[0], will contain number of cells in the raster map. Second line, corresponding to MEM[1], will contain total sum of all cell values. Third line, corresponding to MEM[2], will contain average NDVI value. Of course, these statistics will be calculated for cells where mask value (file *my_country_mask.tif*) is greater than zero. Note, that there is no output raster layer.

Example 5: reclassification of input layer

Consider following scenario: the user has landcover raster map (*lc_classes.tif*) and he is going to reclassify the map to 4, more general classes: water, forest, agriculture, urban areas. PNMapcalc supports such operation in a very easy way.

First of all, the user has to prepare a text file containing contents of the memory buffer. Because the input layer contains 12 different, distinct classes:

- 1 – water1,
- 2 – water2,
- 3 – water3,
- 11 – agriculture1,
- 12 – agriculture2,
- 13 – agriculture3,
- 14 – agriculture4,
- 15 – forest1,
- 16 – forest2,
- 21 – urban_area1,
- 22 – urban_area2,
- 23 – urban_area3.

memory file has to contain definitions for all 12 classes. Output classes will be numbered in the following way

- 0 – no data,
- 1 – water,
- 2 – urban area,
- 3 – agriculture,
- 4 – forest.

The contents of memory file (*mem_map.txt*) is following

```
1 1
2 1
3 1
11 3
12 3
13 3
14 3
15 4
16 4
21 2
```

```
22 2
23 2
```

Memory cells other than listed in *mem_map.txt* file will be initialized with 0 value by definition.

Command line call of mapcalc should be as follows

```
mapcalc  -i lc_classes.tif
         -m 24 -r mem_map.txt
         -o lc_reclass.tif:Byte:0 -e ‘OUT[0]=MEM[(int)IN[0]];’
```

The expression (int)IN[0] means: take floating point number of the first array element and convert it to integer. It is necessary to use this conversion because arrays can be indexed only with integers. The expression results with integer index of MEM[] array.

Example 6: implementation of Koeppen-Geiger climate classification

Koeppen-Geiger climate classification needs information about temperature and precipitation. This information should be collected for each month of the year. Such data can be downloaded from WoldClim project (http://www.worldclim.org).

Let’s assume that the user has downloaded all necessary layers and stored them as:

- temperature layers: t01.tif, t02.tif, …., t12.tif
- precipitation layers: p01.tif, p02.tif, …., p12.tif

Each temperature layer contains information about multiyear average of monthly temperature. Each precipitation layer contains information about multiyear average of monthly sum of precipitation.

Spinoni provided convenient algorithm to do KG classification. In the example, classification to 13 climate classes is presented. Rules to classify climate data are too complex to insert as a command line parameter. For such a case, PNMapcalc has an ability to read macro from a file.

Contents of the macro file (*KG_classification_13.mc*) is shown below:

```
// KG classes
  #define EF  1
  #define ET  2
  #define BW  3
  #define BS  4
  #define Am  5
  #define Aw  6
  #define Af  7
  #define CS  8
  #define CW  9
  #define CF 10
  #define DS 11
  #define DW 12
  #define DF 13

// Temperature 10*C deg.

  double Tcold  =  10000.0;
  double Thot   = -10000.0;
```

```
double Tmon10 =  0.0;
double Ta;
double MAT  = 0.0;
double MATw = 0.0;
double MATs = 0.0;
int i;

for(i=1; i<=12; i++) {
  Ta = IN[i-1];
  if(Ta<Tcold) Tcold=Ta;
  if(Ta>Thot) Thot=Ta;
  if(Ta>100.0) Tmon10+=1.0;
  MAT+=Ta;
  if((i>3) && (i<10)) {
    // summer
    MATs+=Ta;
  } else {
    // winter
    MATw+=Ta;
  }
}
Tcold  *= 0.1;
Thot   *= 0.1;
Tmon10 *= 0.1;
MAT    /= 120.0;

// Precipitation mm/month

double MAP  = 0.0;
double MAPw = 0.0;
double MAPs = 0.0;
double Pr;
double Pdry  = 10000.0;
double Pwdry = 10000.0;
double Psdry = 10000.0;
double Pwwet = 0.0;
double Pswet = 0.0;

for(i=1; i<=12; i++) {
  Pr=IN[i-1+12];
  if(Pdry>Pr) Pdry = Pr;
  MAP += Pr;
  if((i>3) && (i<10)) {
    // summer
    if(Psdry>Pr) Psdry = Pr;
    if(Pswet<Pr) Pswet = Pr;
    MAPs+=Pr;
  } else {
    // winter
    if(Pwdry>Pr) Pwdry=Pr;
    if(Pwwet<Pr) Pwwet=Pr;
    MAPw+=Pr;
  }
}

double p;
double Pthre;

// Summer/winter
if(MATw>MATs) {
  p=MAPw; MAPw=MAPs; MAPs=p;
  p=Pwdry; Pwdry=Psdry; Psdry=p;
  p=Pwwet; Pwwet=Pswet; Pswet=p;
}
```

```
// P threshold
if(MAPw > 0.7*MAP)
  Pthre= 20.0*MAT;
else if(MAPs>0.7*MAP)
  Pthre = 20.0*MAT+280.0;
else
  Pthre = 20.0*MAT+140.0;

int alpha, beta, delta;

alpha=((Psdry<30.0) && (Psdry<Pwwet/3.0) && (MAPs<MAPw));
beta=((Pwdry<Pswet/10.0) && (MAPw<MAPs));
delta=!(alpha && beta);

int KG=0;

if(Thot<10.0) {
  if(Thot>0.0)
    KG=ET;
  else
    KG=EF;
} else {
  if(Pthre>=MAP) {
    if(MAP<Pthre/2.0)
      KG=BW;
    else
      KG=BS;
  } else {
    if(Tcold>=18.0) {
      if(Pdry<60.0) {
        if(Pdry>=100.0-MAP/25.0)
          KG=Am;
        else
          KG=Aw;
      } else
        KG=Af;
    } else if(Tcold<-3.0) {
      if(alpha)
        KG=DS;
      else if(beta)
        KG=DW;
      else if(delta)
        KG=DF;
    } else {
      if(alpha)
        KG=CS;
      else if(beta)
        KG=CW;
      else if(delta)
        KG=CF;
    }
  }
}

OUT[0] = (double)KG;
```

The macro contains many different C syntax elements such as

- constants definitions,
- local variables definition and initialization,

- logical, integer, and double variables,
- conditional expressions,
- type casting and conversion,
- comments.

Command line call of mapcalc should be following:

```
mapcalc  -i t01.tif -i t02.tif -i t03.tif -i t04.tif -i t05.tif -i t06.tif
         -i t07.tif -i t08.tif -i t09.tif -i t10.tif -i t11.tif -i t12.tif
        -i p01.tif -i p02.tif -i p03.tif -i p04.tif -i p05.tif -i p06.tif
         -i p07.tif -i p08.tif -i p09.tif -i p10.tif -i p11.tif -i p12.tif
         -o kg13.tif:Byte:0
         -p macros/KG_classification_13.mc
```

PNMapcalc will read data from 24 input files and create one output file. The resultant file will be type of Byte where no-data cells will contain value zero. The value zero will be defined as no-data value. PMMapcalc will read macro from file *KG_classification_13.mc* placed in subdirectory *macros*. The file suffix *mc* is optional. The order of input files is significant.