Procesamiento de series de tiempo en GRASS GIS

Aplicaciones en Ecología y Ambiente

Dra. Verónica Andreo
CONICET - INMeT

Río Cuarto, 2018

Raster data processing in GRASS GIS

Overview

Overview

- Basics about raster maps in GRASS GIS

Overview

- Basics about raster maps in GRASS GIS
- NULL values

Overview

- Basics about raster maps in GRASS GIS
- NULL values
- MASK

Overview

- Basics about raster maps in GRASS GIS
- NULL values
- MASK
- Computational region

Overview

- Basics about raster maps in GRASS GIS
- NULL values
- MASK
- Computational region
- Resampling and interpolation methods

Overview

- Basics about raster maps in GRASS GIS
- NULL values
- MASK
- Computational region
- Resampling and interpolation methods
- Reports and Statistics

Overview

- Basics about raster maps in GRASS GIS
- NULL values
- MASK
- Computational region
- Resampling and interpolation methods
- Reports and Statistics
- Regresion analysis

Overview

- Basics about raster maps in GRASS GIS
- NULL values
- MASK
- Computational region
- Resampling and interpolation methods
- Reports and Statistics
- Regresion analysis
- Map algebra

Basic raster concepts in GRASS GIS

A "raster map" is a gridded array of cells. It has rows and columns, with a data point (or null value indicator) in each cell. They may exist as 2D grids or 3D cubes.

- Boundaries are described by the north, south, east, and west fields.
- Extent is described by the outer bounds of all cells within the map.

Further info: Raster Intro manual page

Raster data precision

- **CELL DATA TYPE:** a raster map of INTEGER type (whole numbers only)
- **FCELL DATA TYPE:** a raster map of FLOAT type (4 bytes, 7-9 digits precision)
- **DCELL DATA TYPE:** a raster map of DOUBLE type (8 bytes, 15-17 digits precision)

Further info: Raster semantics wiki

General raster rules in GRASS GIS

- **Output** raster maps have their *bounds and resolution equal to those of the computational region*
- **Input** raster maps are automatically *cropped/padded and rescaled to the computational region*
- **Input** raster maps are automatically masked if a raster map named *MASK* exists.

Exception: All r.in.* programs read the data cell-by-cell, with no resampling

NULL values

- **NULL**: represents "no data" in raster maps, different from 0 (zero) data value.
- Operations on NULL cells lead to NULL cells
- NULL values are handled by r.null.

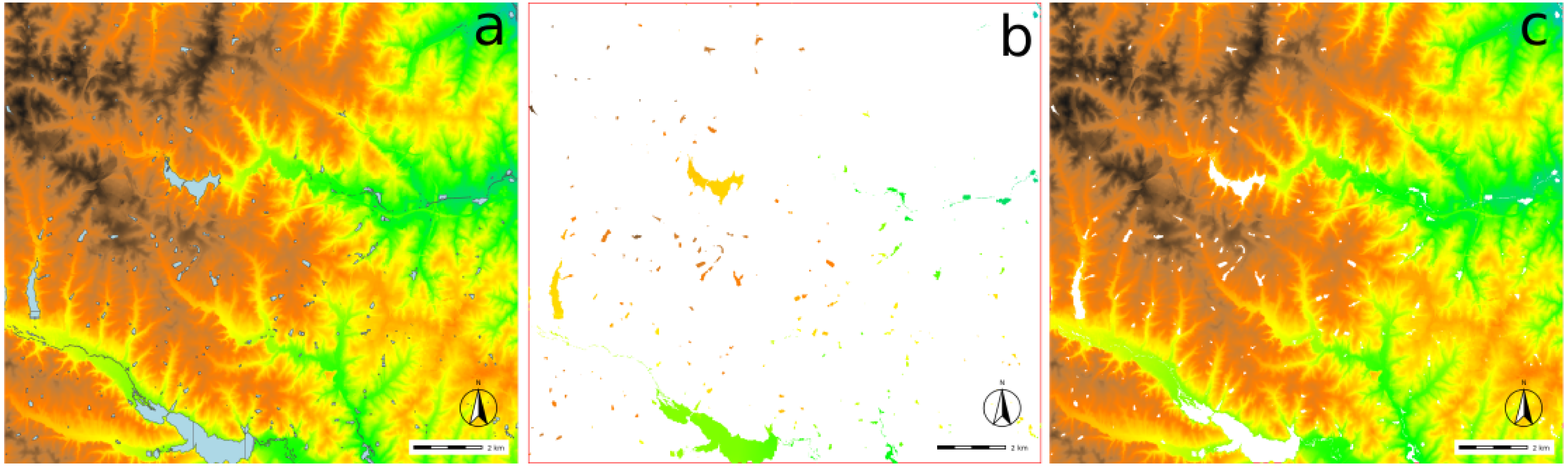
```
# set the nodata value
r.null map=mapname setnull=-9999

# replace NULL by a number
r.null map=mapname null=256
```

MASK

A raster map named MASK can be created to mask out areas: all cells that are NULL in the MASK map will be ignored (also all areas outside the computation region).

Masks are set with r.mask or creating a raster map called *MASK*.

Vector maps can be also used as masks



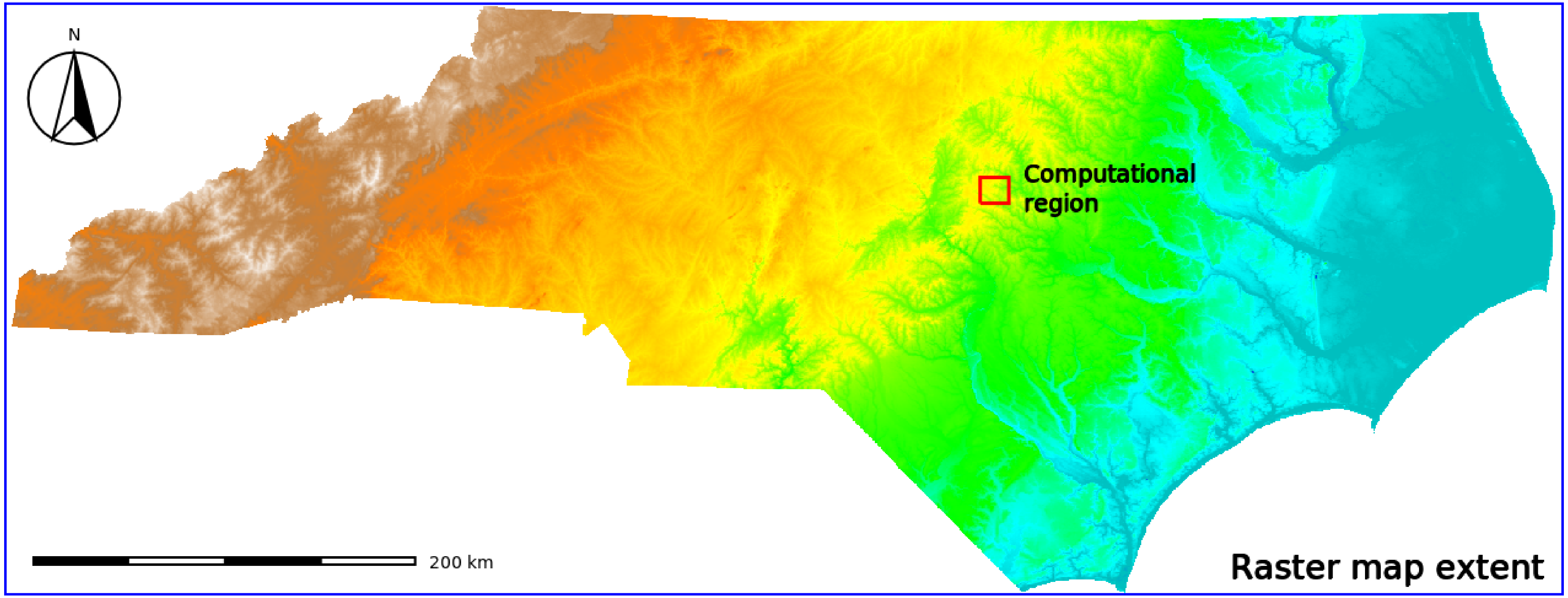
a- Elevation raster and lakes vector maps. b- Only the raster data inside the masked area are used for further analysis. c- Inverse mask.

MASK examples

```
# use vector as mask
r.mask vector=lakes

# use vector as mask, set inverse mask
r.mask -i vector=lakes

# mask categories of a raster map
r.mask raster=landclass96 maskcats="5 thru 7"

# create a raster named MASK
r.mapcalc expression="MASK = if(elevation < 100, 1, null())"

# remove mask
r.mask -r
```

Note: A mask is only actually applied when reading a GRASS raster map, i.e., when used as input in a module.

Computational region



It can be set and changed by means of g.region to the extent of a vector map, a raster map or manually to some area of interest.

- **Computational region** is defined by actual region extent and raster resolution. *It applies to raster operations.*
- **Raster map region** is defined by map extents and map resolution. Each raster map has its own values. Computational region overrides raster region.
- **Display region** is the extent of the current map display independent of the current computational region and the raster region.

User can set the current computational region from display region

Import/export, MASK and region

- r.in.* modules + r.import: The full map is always imported (unless cropping to region is set). Importantly, we can set the region to align with raster resolution (and extent).
- r.out.* modules: Raster export adheres to computational region (extent and resolution) and respects MASK if present. Nearest neighbour interpolation is applied by default.

Note: *In import and export, vector maps are always considered completely.*

Resampling and interpolation methods

See Interpolation wiki page

- Downscaling: - r.resample: nearest neighbour resampling for discrete data - r.resamp.interp: nearest neighbor, bilinear, and bicubic resampling methods for continuous data - r.resamp.rst: Regularized Spline with Tension (RST) interpolation 2D

- Upscaling: - r.resamp.stats: Resamples raster map layers to a coarser grid using aggregation - r.resamp.rst: Regularized Spline with Tension (RST) interpolation 2D

- Gap-filling 2D: - r.fillnulls: Regularized Spline with Tension (RST) interpolation 2D for gap-filling (e.g., SRTM DEM) - r.resamp.bspline: Bicubic or bilinear spline interpolation with Tykhonov regularization - r.resamp.tps: Thin Plate Spline interpolation with regularization and covariables

Note that there are also methods to interpolate sparse vector data and obtain continuous surfaces

Raster map reports and statistics

- r.report: reports area and cell numbers
- r.coin: reports coincidence of two raster map layers
- r.volume: estimates volume for clumps
- r.surf.area: estimates area of a raster map

```
r.report map=zipcodes,landclass96 units=h,p
r.coin first=zipcodes second=landclass96 units=p
```

- r.univar: calculates univariate statistics from the non-null cells of a raster map.
- r.stats: calculates the area present in each of the categories or intervals of a raster map
- r.statistics and r.stats.zonal: zonal statistics
- r.neighbors: local stats based in neighbors

```
# univar stats
r.univar map=elevation

# average elevation in zipcode areas
r.stats.zonal base=zipcodes cover=elevation method=average
  output=zipcodes_elev_avg
```

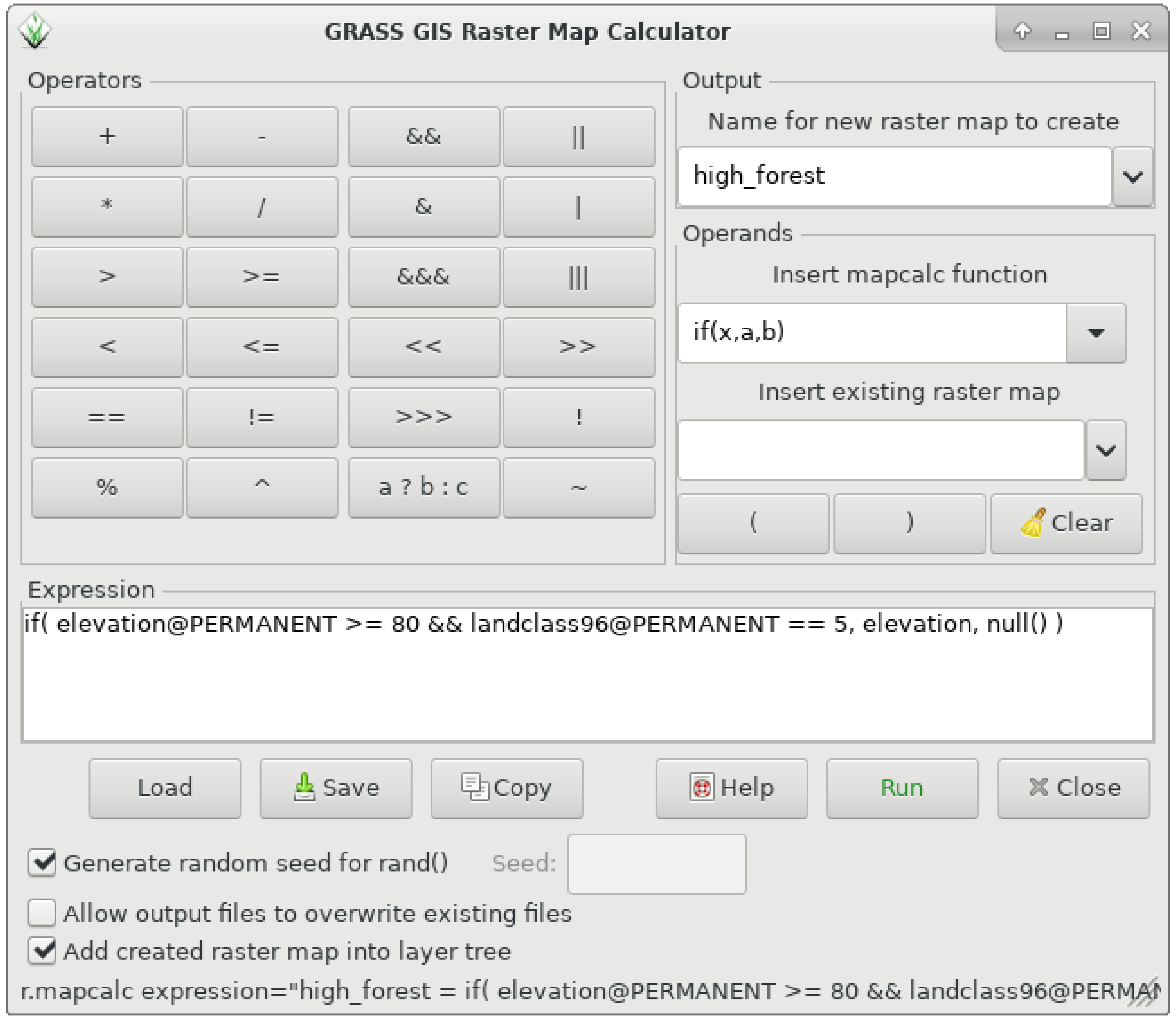
Regression analysis

Both linear (r.regression.line) and multiple regression (r.regression.multi) are supported

```
# linear regression
g.region raster=elev_srtm_30m -p
r.regression.line mapx=elev_ned_30m mapy=elev_srtm_30m

# multiple linear regression
g.region raster=soils_Kfactor -p
r.regression.multi mapx=elevation,aspect,slope mapy=soils_Kfactor \
  residuals=soils_Kfactor.resid estimates=soils_Kfactor.estim
```

Raster map algebra

r.mapcalc



Name for new raster map to create: high_forest

Insert mapcalc function: if(x,a,b)

Insert existing raster map:

Expression

```
if( elevation@PERMANENT >= 80 && landclass96@PERMANENT == 5, elevation, null() )
```

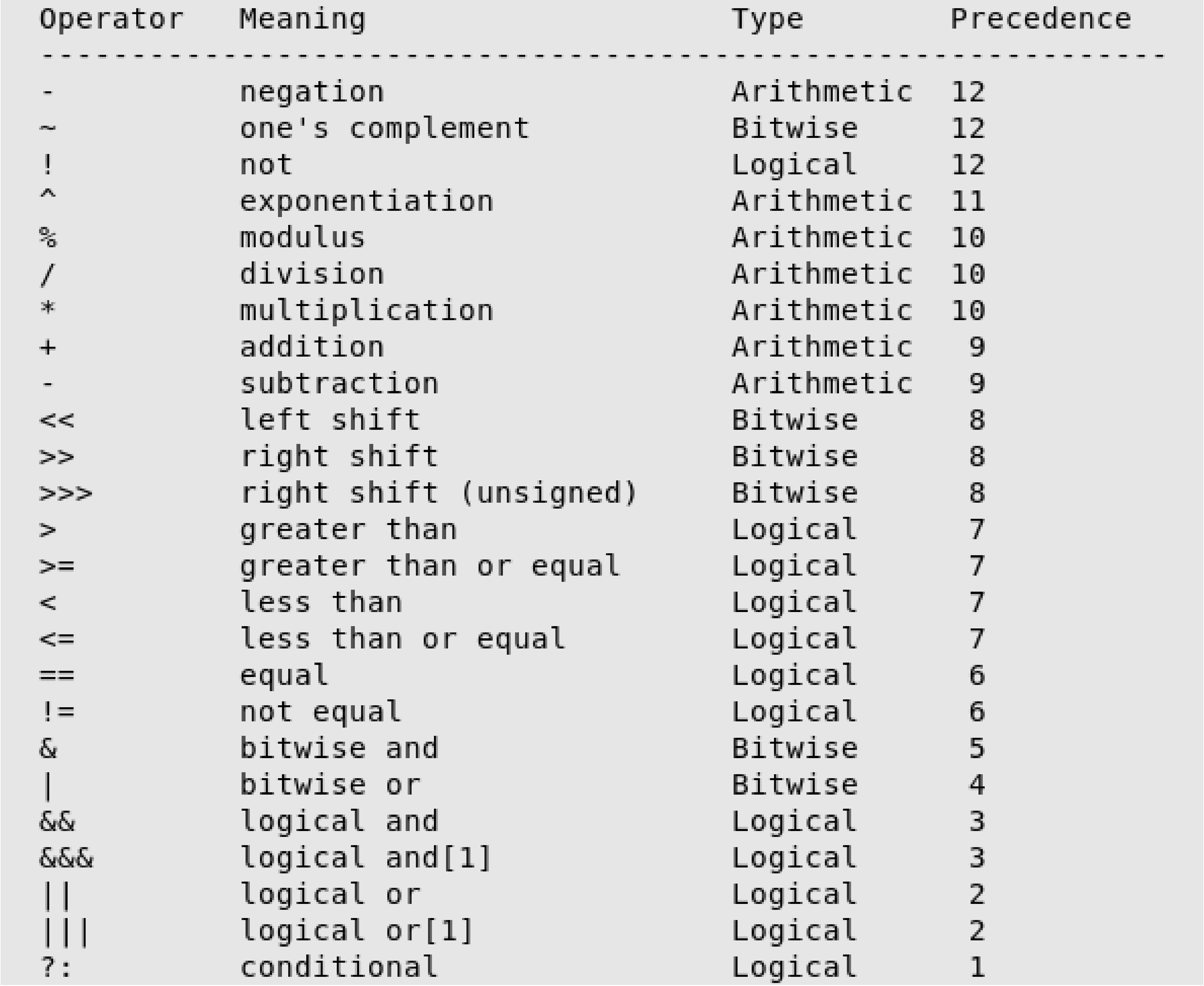
☑ Generate random seed for rand() Seed:

☐ Allow output files to overwrite existing files

☑ Add created raster map into layer tree

```
r.mapcalc expression="high_forest = if( elevation@PERMANENT >= 80 && landclass96@PERMAN
```

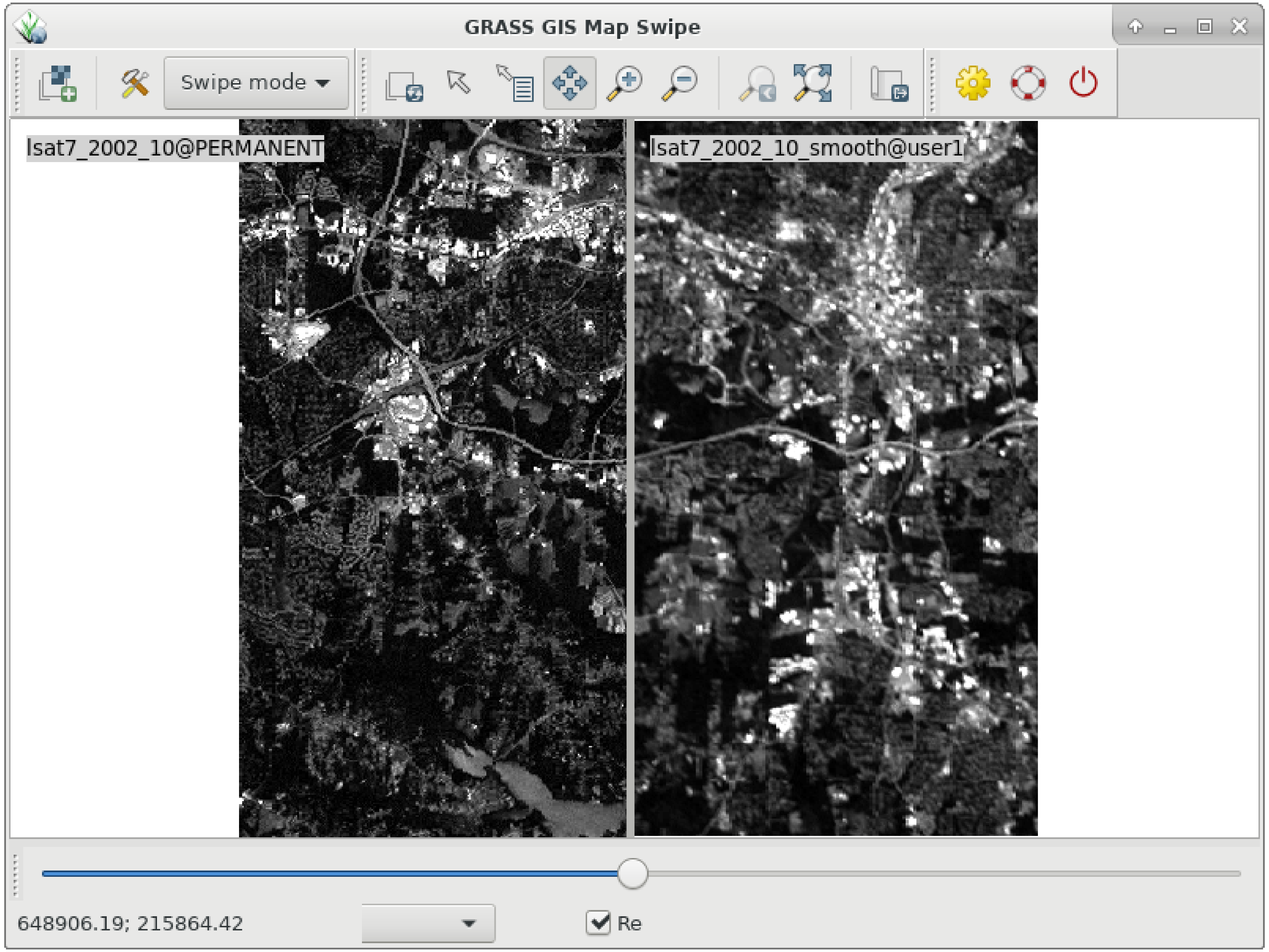
Operators



```
Operator   Meaning                    Type        Precedence
--------------------------------------------------------------
-          negation                   Arithmetic  12
~          one's complement           Bitwise     12
!          not                        Logical     12
^          exponentiation             Arithmetic  11
%          modulus                    Arithmetic  10
/          division                   Arithmetic  10
*          multiplication             Arithmetic  10
+          addition                   Arithmetic   9
-          subtraction                Arithmetic   9
<<         left shift                 Bitwise      8
>>         right shift                Bitwise      8
>>>        right shift (unsigned)     Bitwise      8
>          greater than               Logical      7
>=         greater than or equal      Logical      7
<          less than                  Logical      7
<=         less than or equal         Logical      7
==         equal                      Logical      6
!=         not equal                  Logical      6
&          bitwise and                Bitwise      5
|          bitwise or                 Bitwise      4
&&         logical and                Logical      3
&&&        logical and[1]             Logical      3
||         logical or                 Logical      2
|||        logical or[1]              Logical      2
?:         conditional                Logical      1
```

Neighborhood operator **[row,col]**

```
# example of a low pass filter
r.mapcalc \
expression="lsat7_2002_10_smooth = (lsat7_2002_10[-1,-1] +
                                    lsat7_2002_10[-1,0] +
                                    lsat7_2002_10[1,1] +
                                    lsat7_2002_10[0,-1] +
                                    lsat7_2002_10[0,0] +
                                    lsat7_2002_10[0,1] +
                                    lsat7_2002_10[1,-1] +
                                    lsat7_2002_10[1,0] +
                                    lsat7_2002_10[1,1]) / 9"
```



```
g.gui.mapswipe first=lsat7_2002_10 second=lsat7_2002_10_smooth
```

Functions

```
function                description                                     type
---------------------------------------------------------------------------
abs(x)                  return absolute value of x                      *
acos(x)                 inverse cosine of x (result is in degrees)      F
asin(x)                 inverse sine of x (result is in degrees)        F
atan(x)                 inverse tangent of x (result is in degrees)     F
atan(x,y)               inverse tangent of y/x (result is in degrees)   F
cos(x)                  cosine of x (x is in degrees)                   F
double(x)               convert x to double-precision floating point    F
eval([x,y,...,]z)       evaluate values of listed expr, pass results to z
exp(x)                  exponential function of x                       F
exp(x,y)                x to the power y                                F
float(x)                convert x to single-precision floating point    F
graph(x,x1,y1[x2,y2..]) convert the x to a y based on points in a graph F
graph2(x,x1[,x2,..],y1[,y2..])
                        alternative form of graph()                     F
if                      decision options:                               *
if(x)                   1 if x not zero, 0 otherwise
if(x,a)                 a if x not zero, 0 otherwise
if(x,a,b)               a if x not zero, b otherwise
if(x,a,b,c)             a if x > 0, b if x is zero, c if x < 0
int(x)                  convert x to integer [ truncates ]              I
isnull(x)               check if x = NULL
log(x)                  natural log of x                                F
log(x,b)                log of x base b                                 F
max(x,y[,z...])         largest value of those listed                   *
median(x,y[,z...])      median value of those listed                    *
min(x,y[,z...])         smallest value of those listed                  *
mode(x,y[,z...])        mode value of those listed                      *
nmax(x,y[,z...])        largest value of those listed, excluding NULLs  *
nmedian(x,y[,z...])     median value of those listed, excluding NULLs   *
nmin(x,y[,z...])        smallest value of those listed, excluding NULLs *
nmode(x,y[,z...])       mode value of those listed, excluding NULLs     *
not(x)                  1 if x is zero, 0 otherwise
pow(x,y)                x to the power y                                *
rand(a,b)               random value x : a <= x < b                     *
round(x)                round x to nearest integer                      I
round(x,y)              round x to nearest multiple of y
round(x,y,z)            round x to nearest y*i+z for some integer i
sin(x)                  sine of x (x is in degrees)                     F
sqrt(x)                 square root of x                                F
tan(x)                  tangent of x (x is in degrees)                  F
xor(x,y)                exclusive-or (XOR) of x and y                   I
```

if statement

Determine the forested areas located above a certain elevation

```
# set region
g.region rast=landclass96

# report of land classes
r.report map=landclass96 units=p

# univariate statistics for elevation
r.univar map=elevation

# select areas higher 120m and with forest land class:
r.mapcalc expression="forest_high = \
  if(elevation > 120 && landclass96 == 5, 1, null())"
```

Advanced raster algebra

`eval` function

When the output of the computation should be only one map but the expression is so complex that it is better to split it to several expressions:

```
r.mapcalc expression= "eval(elev_200 = elevation - 200,
                            elev_5 = 5 * elevation,
                            elev_p = pow(elev_5, 2));
                            elevation_result = (0.5 * elev_200) + 0.8 *
```

QUESTIONS?

Thanks for your attention!!

Move on to:

Exercise 3: Landscape, hydrology and terrain analysis

Presentation powered by

GitPitch