Chapter 6

Map & Table Calculation

6.1 Introduction

Calculating and modeling of spatial and attribute data is performed in ILWIS by using Map calculation, Table calculation, the Pocket Line Calculator or by running a script. Map calculations usually result in enhanced images or newly created maps combining tabular and spatial data. Table calculation provides the user the opportunity to calculate new parameters which are stored in a table. The Pocket Line Calculator enables quick calculations and retrieval of information from maps and tables. By running a script the user may perform a whole series of calculations after one other which is comparable with running a batch-file. Operators and functions used in MapCalc, TabCalc and the pocket line calculator are mainly the same.

6.1.1 Overview of operators and functions

Operators on domain Value and Image

Arithmetic	+	-	*	/	MOD	DIV
Relational	=	<	<=	>	>=	<>
	eq	lt	le	gt	ge	ne

Operators on domain Bool

Logical	AND	OR	XOR	NOT

Functions on domain Value and Image

Undefined values	ISUNDEF(*a*)	IFUNDEF(*a,b*)	IFUNDEF(*a,b,c*)	
	IFNOTUNDEF(*a,b*)		IFNOTUNDEF(*a,b,c*)	
Conditional	IFF(*a,b,c*)			
Relational	INRANGE(*a,b,c*)			
Exponential	SQ(*a*)	SQ(*a,b*)	SQRT(*a*)	
	HYP(*a,b*)	POW(*a,b*)	EXP(*a*)	
Logarithmic	LOG(*a*)	LN(*a*)		
Random	RND(*n*)	RND(0)	RND()	
Sign	ABS(*a*)	-(*a*)	NEG(*a*)	SGN(*a*)
Rounding	CEIL(*a*)	FLOOR(*a*)	ROUND(*a*)	
MinMax	MIN(*a,b*)	MIN(*a,b,c*)	MAX(*a,b*)	MAX(*a,b,c*)
NDVI	NDVI(*a,b*)			
Trigonometric	SIN(*a*)	COS(*a*)	TAN(*a*)	
	ASIN(*a*)	ACOS(*a*)	ATAN(*a*)	ATAN2(*y,x*)
Angular	DEGRAD(*a*)	RADDEG(*a*)		
Hyperbolic	SINH(*a*)	COSH(*a*)	TANH(*a*)	
Classify/Group	CLFY(*a*, *DomainGroup*)			
Conversion	STRING (*a*)			

(*a*, *b* and *c* refer to an expression resulting in a Value; *n* refers to a long integer; *a* in an IFF is a Bool)

Predefined values

Predefined values	PI	PI2	PIDIV2	PIDIV4	EXP(1)

Operators and functions on non-value domains

Relational Class/ID	=	<>	eq	ne
Conditional Class/ID	IFF(*a,b,c*)			
Undefined Class/ID	ISUNDEF(*a*)		IFUNDEF(*a,b*)	IFUNDEF(*a,b,c*)
	IFNOTUNDEF(*a,b*)		IFNOTUNDEF(*a,b,c*)	
Special Class/ID	+	IN(*s1,s2*)	INMASK (*s, mask*)	STRPOS (*s1,s2*)
	LENGTH(*s*)	LEFT(*s,i*)	RIGHT(s,i)	SUB*(s*)
	STRLT (*s1,s2*)	STRLE (*s1,s2*)	STRGT(*s1,s2*)	STRGE(*s1,s2*)
Conversion	VALUE (*s*)			

(s refers to an expression resulting in a Class, Group ID or a String; *i* refers to an integer; *a* in an IFF is a Bool)

Special Map Calculations

Predefined variables	%L	%C	%X	%Y
Create attribute map	Map.Column		Map.Table.Column	
Two-dim tables	TwodimTableName[Map1,Map2]			
Neighbourhood operations	NBMIN	NBMINP	NBMAX	NBMAXP
	NBSUM	NBCNT	NBPRD	NBPRDP
	NBPRDP	NBFLT	NBDIS	NBPOS
Iterations	ITER	ITERP		

Special Table Calculations

Predefined variables	%R	%K
Statistics	AVG(*col*)	STDEV(*col*)
Aggregations	AGGAVG (*col,g,w*)	AGGCNT (*col,g,w*)
	AGGMIN (*col,g,w*)	AGGMED (*col,g,w*)
	AGGMAX (*col,g,w*)	AGGPRD (*col,g,w*)
	AGGSTD (*col,g,w*)	AGGSUM (*col,g,w*)
Join columns	ColumnJoin (Table,Column)	

(*col* refers to column, *g* refers to the key to group the values, *w* refers to weight)

6.1.2 Special Topics

Calculations on coordinates

Retrieval of coordinates	MAPCRD	PNTCRD	CRDX
	CRDY	COORD	MINCRDX
	MAXCRDX	MINCRDY	MAXCRDY
Other functions	TRANSFORM	DIST	DIST2
	MAPVALUE	MAPROW	MAPCOL
	RASVALUE		

Calculations on point data

Retrieval of coordinates	PNTNR	PNTVAL	PNTCRD

Calculations on data properties

Retrieval of data properties	MAPMIN	MAPMAX	PIXSIZE
	PIXAREA	MAPROWS	MAPCOLS

Calculations on colors

Color separation	CLRRED	CLRGREEN	CLRBLUE
	CLRYELLOW	CLRMAGENTA	CLRCYAN
	CLRGREY	CLRHUE	CLRSAT
	CLRINTENS		
Retrieval of colors	MAPCOLOR	RPRCOLOR	
Assigning colors	COLOR	COLORHSI	

6.2 Map calculation

6.2.1 Introduction

Map calculation is used for analyses and transformation of spatial data and modeling operations. It integrates spatial and attribute data. Map calculation enables the user to perform image enhancements and to produce and display new maps.

One may type a map calculation formula directly on the command line of the Main window or double-click the MapCalc item in the Operation-list.

The following operations are available:

- manipulation of one or more raster maps by using different types of operators and functions
- creation of attribute maps from raster maps using an attribute table
- classification of a domain Value map according to a domain Group
- integration of 2 raster maps according to 2-dimensional tables
- neighbourhood operations
- calculations using map rows and columns
- calculations using X- and Y-coordinates
- calculations using colors
- application of user-defined functions

6.2.2 General syntax of MapCalc formulae

A MapCalc formula or statement consists of an output raster map name that will contain the result of the calculation, a definition symbol (=) or an assignment symbol (:=), and a expression.

```
OutputMap = Expression
```

or

```
OutputMap:= Expression
```

Example

A simple MapCalc formula reads:

```
Map3 = Map1 + Map2
```

Map1

1	0	2	2
3	2	1	4
0	3	1	2
4	2	0	3

Map2

0	4	1	2
4	3	3	1
1	2	4	0
3	1	2	0

Map3

1	4	3	4
7	5	4	5
1	5	5	2
7	3	2	3

The values of the pixels in Map3 represent the sums of the values in the corresponding pixels of Map1 and Map2.

Definition symbol = and assignment symbol :=

By using the definition symbol = on the command line a dependent map is created: the definition of how this map was created is stored. In ILWIS 1.4, dependent maps did not exist.

By using the assignment symbol := an editable map or table is created. This means that the dependency link is broken and the output values are directly assigned to the output pixels.

For more information, refer to Basic concepts: dependent data objects

Output map name

When typing a MapCalc formula on the command line, you generally start with an output map name. Otherwise you fill in the dialog box after double-clicking the MapCalc item in the Operation- list. If necessary, the output object's domain or its domain and value range may be specified in a pair of curly brackets after the output object name. Then the specified domain should be an existing one.

To create for instance a map OUT with domain value name 'MyVal', this would look like:

```
OUT {dom=MyVal} = expression
```

To create for instance an output map OUT2 with domain MYVAL2 and to store values in map OUT2 which range between minimum value 0 and maximum value 10000 with a precision of 1, this would look like:

```
OUT2{dom=MyVal2;vr=0:10000:1} = expression
```

Expression

The expression usually contains operators and/or functions to specify what calculation has to be performed. Operators used in an expression are for instance: + - / * etc. An overview of MapCalc operators and functions is presented in the next topic.

All data objects used on the command line, regardless of whether they are input or output objects, should start with a character between A and Z. Further, the name of a data object cannot exceed 8 characters (except for column names). When using class or group names or IDs in an expression, "double quotes" are needed around the class names, group names or IDs.

- When you type $ in front of an output map name, the output map is directly shown in a map window.
- Advanced users who are sure about the correctness of an expression may end their expression with a semi-colon(;). In this way, dialog boxes are skipped by accepting all the default values.

Example

```
Coffee = Landuse = ”coffee”
```

This means in words: The output map, named `coffee`, shows where the landuse class name is ”`coffee`” and where it is not. This is one of the most simple types of expression. It returns an output map with a Bool domain. The expression Landuse = ”coffee” is tested whether it is true of false.

Landuse

B	B	S	B
S	S	C	C
S	C	C	S
B	C	C	C

Coffee

F	F	F	F
F	F	T	T
F	T	T	F
F	T	T	T

Read in the Landuse map for B: Bare Soil, for C: Coffee, and for S: Shrubland. True (T) is returned in the pixel information when the land use was coffee. False (F) indicates that the land use was something else.

- It does not matter whether you type spaces around the definition or assignment symbol, or in the expression or not.
- It does not matter whether you type in capitals or in small characters; a MapCalc formula is not case-sensitive. In the Reference Guide and in the ILWIS Help, often capitals are used for operators and functions. This is only for editing purposes. It is not necessary in MapCalc, TabCalc or the Pocket line calculator.
- The output map name may be a new or an existing map name. When you specify a new output map name, the answers are written into a newly created map. When you specify an existing output map name, a message appears, asking the user to confirm to overwrite the existing map.
- A MapCalc formula does not really have a limitation on length; when the command line seems full, you can just continue typing.
- With the left and right arrow keys and the Home and End key of the keyboard, you can move back and forth in your formula. You can also use the somewhat larger MapCalc dialog box where you can type a complicated MapCalc formula on multiple lines.

- ☞ The command line has a history: press the up arrow key to retrieve previously used expressions; the down arrow key is used to 'scroll' forward again.
- ☞ Advanced users who are certain that the typed formula is a correct one may skip dialog boxes of operations by ending the expression with a semi-colon (;).

6.2.3 Description of operators and functions

6.2.3.1 Operators on domain Value and Image

There are many similarities but also some differences in the way you can manipulate images or maps with a domain Value and how you can manipulate maps with a domain Class, Group or ID. Therefore they are listed separately. Below, the operators and functions are explained that can be used in Map calculation on images and maps with a domain Value.

Arithmetic operators

+	*a* + *b*	add operator;
-	*a* - *b*	subtract operator;
*	*a* * *b*	multiply operator;
/	*a* / *b*	divide operator;
MOD	*a* MOD *b*	modulus operator; returns the remainder of *a* divided by *b*, e.g. 20 mod 3, returns 2
DIV	*a* DIV *b*	integer division operator; dividing integer *a* by *b* returns the quotient, e.g. 20 div 3, returns 6

☞ The MOD and DIV operators require typed spaces.

Examples

```
Map3 = Map1 + Map2
```

Map1

1	0	2	2
3	2	1	4
0	3	1	2
4	2	0	3

Map2

0	4	1	2
4	3	3	1
1	2	4	0
3	1	2	0

Map3

1	4	3	4
7	5	4	5
1	5	5	2
7	3	2	3

A pixel value in `Map3` is the sum of the corresponding pixel values of `Map1` and `Map2`.

The following example shows the creation of a new map in which the pixel values of the input map are divided by a factor 20.

```
Map2 = Map1 DIV 20
Map3 = Map1 MOD 20
```

Map1

50	78	63	59
29	43	85	17
12	6	18	26
32	90	64	8

Map2

2	3	3	2
1	2	4	0
0	0	0	1
1	4	3	0

Map3

10	18	3	19
9	3	5	17
12	6	18	6
12	10	4	8

A pixel value in `Map2` is the corresponding value of `Map1` divided by 20. Using operator DIV returns an integer in the output map not paying attention to rest values.

`Map3` displays the modulus (rest value) of the division of `Map1` by 20.

Relational operators

=	eq	*a* = *b*	*a* eq *b*	equal to
<	lt	*a* < *b*	*a* lt *b*	less than
<=	le	*a* <= *b*	*a* le *b*	less than or equal to
>	gt	*a* > *b*	*a* gt *b*	greater than
>=	ge	*a* >= *b*	*a* ge *b*	greater than or equal to
<>	ne	*a* <> *b*	*a* ne *b*	not equal to

When a relational operator is used, ILWIS tests whether the outcome of the statement containing this operator is true or false (Bool domain).
When using the symbols it does not matter whether you type spaces around the operators or not. When using the characters spaces are required on both sides of the operator.

Example

```
Result = DEM > 1400
```

DEM

800	900	1000	1100
900	1000	1200	1400
1200	1300	1500	1600
1300	1400	1600	1800

Result

F	F	F	F
F	F	F	F
F	F	T	T
F	F	T	T

The output map has a Bool domain. For pixels in map `DEM` (Digital Elevation Model) that have a value greater than 1400, the expression is true. Otherwise `Result` returns false.
Relational operators are often used in combined with a conditional IFF function. To test whether values lie in a certain range, e.g. greater than 1000 but less than 1600, the INRANGE function may be used as well.

6.2.3.2 Operators on domain Bool

Logical operators

AND	*a* and *b*	returns true if both expressions *a* and *b* are true
OR	*a* or *b*	returns true if one or both of the expressions *a* and *b* is true
XOR	*a* xor *b*	returns true if only one of the expressions *a* and *b* is true
NOT	not *a*	returns true if expression *a* is false

Examples of the logical AND operator

The result of *a* and *b* is true if both expressions *a* and *b* are true; in other cases false is returned.

```
Result1 = (DEM > 1000) AND (DEM < 1600)
```

DEM

800	900	1000	1100
900	1000	1200	1400
1200	1300	1500	1600
1300	1400	1600	1800

Result1

F	F	F	T
F	F	T	T
T	T	T	F
T	T	F	F

If a pixel in map `DEM` (Digital Elevation Model) has a value larger than 1800 and at the same time the value of this pixel is less than 2400, then both expressions are true. For pixels where not both expressions are true, False is assigned.

```
Result2 = (Landuse = "Coffee") AND (DEM > 1400)
```

Landuse

B	B	S	B
S	S	C	C
S	C	C	S
B	C	C	C

DEM

800	900	1000	1100
900	1000	1200	1400
1200	1300	1500	1600
1300	1400	1600	1800

Result2

F	F	F	F
F	F	F	F
F	F	T	F
F	F	T	T

If pixels in map `Landuse` have class name `Coffee`, and the corresponding pixels in map `DEM` have a value larger than 1400, both expressions are true which is shown in the output map `Result2`. False is assigned when one or both of the expressions is not true (in mathematics the logic AND is called an intersection).

Example of the logical OR operator

The result of *a* or *b* is true if either expression *a*, or expression *b* or if both expressions *a* and *b* are true; in other cases false is returned.

```
Result = (Landuse = "Coffee") OR (DEM > 1400)
```

Landuse

B	B	S	B
S	S	C	C
S	C	C	S
B	C	C	C

DEM

800	900	1000	1100
900	1000	1200	1400
1200	1300	1500	1600
1300	1400	1600	1800

Result

F	F	F	F
F	F	T	T
F	T	T	T
F	T	T	T

For map Landuse read: C= coffee, S= shrubs and B= bare soil

True is assigned in map `Result`, if a pixel in map `Landuse` has class name "`Coffee`", or if the value of that pixel in map `DEM` is larger than 1400. By using this expression, the output map will contain all areas where the landuse type is `Coffee`, and all areas where the height is more than 1400 m. In mathematics the logical OR is called a union.

6.2.3.3 Functions on domain Value and Image

Conditional IFF function

IFF(*a*,*b*,*c*)	If condition *a* is true, then return the outcome of expression *b*, else (when condition *a* is not true) return the outcome of expression *c*. Mind the double ff in IFF (standing for IF Function). The conditional IFF may be used for all types of input data: values, IDs, groups and classes.

Examples of IFF functions on values

```
Result1 = IFF (DEM > 1400, 10, 0)
```

This means: if a pixel in map `DEM` (digital elevation model) has a value greater than 1400, then assign the value 10 to this pixel in output map `Result1`, else assign a 0.

DEM

800	900	1000	1100
900	1000	1200	1400
1200	1300	1500	1600
1300	1400	1600	1800

Result1

0	0	0	0
0	0	0	0
0	0	10	10
0	0	10	10

The IFF function can be combined with other operators and functions.

```
Result2 =  IFF((Landuse="coffee") AND (DEM>=1400), 10, 0)
```

Landuse

B	B	S	B
S	S	C	C
S	C	C	S
B	C	C	C

DEM

800	900	1000	1100
900	1000	1200	1400
1200	1300	1500	1600
1300	1400	1600	1800

Result2

0	0	0	0
0	0	0	10
0	0	10	0
0	10	10	10

If a pixel belongs to class coffee in map Landuse and the corresponding pixel in map DEM has a value greater than or equal to 1400, then assign 10 to this pixel in output map Result2, else assign a 0. If you wish to assign undefineds to the remaining pixels, you can replace the 0 in the formula by ?.

It is also possible to use more than one IFF function in a formula.

```
Result3 = IFF(DEM < 900, 1, IFF(DEM < 1100, 2, IFF(DEM <
          1600, 3, 4)))
```

DEM

800	900	1000	1100
900	1000	1200	1400
1200	1300	1500	1600
1300	1400	1600	1800

Result3

1	2	2	3
2	2	3	3
3	3	3	4
3	3	4	4

In words this means: if the DEM has a value of less than 900 then the pixel value will be 1. Then: if the value of the DEM is less than 1100 (in fact between 900 and 1100) a pixel will get value 2. From the remaining pixels the ones which have a value lower than 1600 will get value 3 and the rest value 4.
This formula is nothing else then a kind of classification. You could also come to the same result by applying a classification function (CLFY).

The INRANGE function

INRANGE(*a*,*b*,*c*)	tests whether values of expression or map a are contained by a range or closed interval with endpoints b and c. Mathematic notation: $b \leq a \leq c$ or $a \in [b;c]$

Example of the INRANGE function
Below, the INRANGE function is combined with a conditional IFF.

```
Result1 = IFF(INRANGE (DEM, 1000, 1600), 50, 1)
```

DEM

800	900	1000	1100
900	1000	1200	1400
1200	1300	1500	1600
1300	1400	1600	1800

Result1

1	1	50	50
1	50	50	50
50	50	50	50
50	50	50	1

If a pixel in map DEM (Digital Elevation Model) has a value that is larger than or equal to 1000 and less than or equal to 1600, then 50 is assigned in output map Result1. For other pixels, a 1 is assigned.

```
Result2 = IFF (INRANGE (DEM, 1000, 1600), DEM, ? )
```

DEM

800	900	1000	1100
900	1000	1200	1400
1200	1300	1500	1600
1300	1400	1600	1800

Result2

?	?	1000	1100
?	1000	1200	1400
1200	1300	1500	1600
1300	1400	1600	?

If a pixel in map DEM (Digital Elevation Model) has a value that is larger than or equal to 1000 and less than or equal to 1600, then the original height value of map DEM is assigned in output map Result2. For other pixels, the undefined ? is assigned.

Exponential functions

SQ(*a*)	a^2	square function: *a*a*; *a* square
SQ(*a*,*b*)	$a^2 + b^2$	square function: *a*a + b*b*; *a* square plus *b* square
SQRT(*a*)	$\sqrt{a}$	square root function: calculates the positive square root of *a*
HYP(*a*,*b*)	$\sqrt{(a^2 + b^2)}$	hypotenuse: calculates the positive square root of the sum of *a* square and *b* square
POW(*a*,*b*)	a^b	exponential function: *a* raised to the power *b*. The n-th root of *a* is found by using this function in the form of: POW(*a*, 1/n)
EXP(*a*)	e^a	exponential function: value e (i.e. 2.718) raised to the power *a*

Memory refreshment: Laws of exponents

Negative exponents: $a^{-n} = 1/a^n$

Fractional exponents: $a^{m/n} = \sqrt[n]{(a^m)}$

Multiplication: $a^n a^m = a^{n+m}$

Division: $a^n/a^m = a^{n-m}$

Raising to a power: $(a^n)^m = a^{n*m}$

Example of an exponential function

The exponential HYP function may be used to calculate slope values in percentages, from 2 input maps which contain height differences in X-direction (map DX) and in Y-direction (map DY). Map DX and DY were created by using the filters dfdx and dfdy on a digital elevation model (DEM).

```
Slopepct = (HYP(dx,dy)/pixelsize) * 100
```

For the *pixelsize* fill in the pixel size of the maps (if pixels are 20 by 20m, use 20). The results may be higher than 100% as a slope of 45° is equal to 100%. Slopes of 60° are equal to 173%.

Logarithmic functions

LOG(*a*)	^{10}log(*a*)	logarithm: calculates the 10-based logarithm of *a*
LN(*a*)	^{e}log(*a*)	natural logarithm: calculates the e based (2.718) logarithm of *a*

Memory refreshment: Laws of logarithms

Multiplication: log(n*m) = log(n) + log(m)

Division: log(n/m) = log(n) - log(m)

Raising to a power: log(n^m) = m * log(n)
ln(EXP(n)) = n

Memory refreshment: Changing the base of a logarithm

If you would like to calculate ^{b}log(*a*), the *b*-based logarithm of *a*, instead of ^{10}log(*a*), you can divide the old logarithm by the logarithm of the new base:

$^{b}\log(a) = {}^{10}\log(a)/{}^{10}\log(b)$

Example of a logarithmic function

```
Result2 = LOG(map1)/LOG(2)
```

Map1

38	46	12	92
81	3	55	69
21	60	88	37
74	19	5	46

Result2

5.3	5.5	3.6	6.5
6.3	1.6	5.8	6.1
4.4	5.9	6.5	5.2
6.2	4.3	2.3	5.5

Map `Result2` contains the values of ^{2}log(`map1`).

Random functions

RND(*n*)	returns integer values in the range 1 to *n*. To simulate a die, use this function in the form of: RND(6)
RND(0)	returns a 0 or 1 at random
RND()	returns random real values in the range [0;1> , i.e. between 0 and 1, including 0 but excluding 1

☞ Use always **:=** otherwise new random values will be assigned in every new calculation.

☞ Integer *n* has a maximum value of 2 billion ($2*10^9$)

Example of a random function

For statistical purposes you might need a map with random values. That can be done using the following formula:

```
Mapran := RND(0)
```

Then a dialog box appears to define the output raster map. Accept the default domain Value. You can choose a georeference as well. The pixels in the output map will get randomly the value 0 or 1.

Sign operator functions

-*a*	returns *a* multiplied by -1
NEG(*a*)	returns *a* multiplied by -1
ABS(*a*)	returns the absolute (=positive) value of *a*
SGN(*a*)	returns -1 for negative values of a, 0 when a is 0, and 1 for positive values of *a*

Example of a sign function

The input map has both positive and negative values which are, when negative, converted to positive values.

```
Map2 = ABS(Map1)

Map3 = NEG(Map1)

Map3 = -Map1

Map4 = SGN(Map1)
```

Map1

4	-2	-8	3
5	1	-1	7
-9	-6	5	-4
3	2	-3	8

Map2

4	2	8	3
5	1	1	7
9	6	5	4
3	2	3	8

Map3

-4	2	8	-3
-5	-1	1	-7
9	6	-5	4
-3	-2	3	-8

Map4

-1	1	1	-1
-1	-1	1	-1
1	1	-1	1
-1	-1	1	-1

Rounding functions

ROUND(*a*)	rounds a to a integer value, e.g. ROUND(3.5) returns 4, ROUND(-3.5) returns -3
FLOOR(*a*)	rounds down; returns the largest integer value smaller than input value (truncation),e.g. FLOOR(3.6) returns 3, FLOOR(-3.6) returns -4
CEIL(*a*)	rounds up; returns the smallest integer value larger than input value, e.g. CEIL(3.6) returns 4, CEIL(-3.6) returns -3

MinMax functions

MIN(*a*,*b*)	calculates the minimum of two expressions *a* and *b*
MIN(*a*,*b*,*c*)	calculates the minimum of three expressions *a*, *b* and *c*
MAX(*a*,*b*)	calculates the maximum of two expressions *a* and *b*
MAX(*a*,*b*,*c*)	calculates the maximum of three expressions *a*, *b* and *c*

Using these functions, you can for instance calculate for each pixel the minimum or maximum value of 2 or 3 input maps; substitute *a*,*b*,*c* with the map names.

Example of a MinMax function

```
Resmax = MAX(map1,Map2)
```

Map1

0	4	1	2
4	3	3	1
1	2	4	0
3	1	2	0

Map2

1	0	2	2
3	2	1	4
0	3	1	2
4	2	0	3

Resmax

1	4	2	2
4	3	3	1
1	3	4	2
4	2	2	3

NDVI function for images

NDVI(*a*,*b*)	(*b* - *a*) / (*b* + *a*) used to calculate the NDVI (Normalized Difference Vegetation Index) of 2 images. Use for a the band with visible or red reflectance and for b the band with near infrared reflectance values.

Example of a NDVI function

```
VegInd = NDVI(TmBand3,TmBand4)
```

Output map VegInd contains NDVI values calculated from TmBand3 and TmBand4. TmBand3 represents the visible reflectance and TmBand4 represents the near-infrared reflectance.

- Vegetated areas will generally yield high values because of their relatively high near-infrared reflectance and low visible reflectance.
- In contrast, water, clouds, and snow have larger visible reflectance than near-infrared reflectance. Thus, these features yield negative index values.
- Rock and bare soil areas have similar reflectances in the two bands and result in vegetation indices near zero.

Trigonometric functions

SIN(*a*)	sine; returns real values in the range -1 to 1
COS(*a*)	cosine; returns real values in the range -1 to 1
TAN(*a*)	tangent
ASIN(*a*)	arcsin; returns real values in radians in the range -π/2 to π/2
ACOS(*a*)	arccos; returns real values in radians in the range 0 to π
ATAN(*a*)	arctan; returns real values in radians in the range -π/2 to π/2
ATAN2(*y*,*x*)	returns the angle in radians of two input values (*y* is vertical, *x* is horizontal) in the range -π to π

Example of a trigonometric function

```
Cosine = COS(Map1)
```

Map1

0	4	1	2
4	3	3	1
1	2	4	0
3	1	2	0

Cosine

1.00	-0.65	0.54	-0.42
-0.65	-0.99	-0.99	0.54
0.54	-0.42	-0.65	1.00
-0.99	0.54	-0.42	1.00

Map1 has input values in radians. Map2 returns the cosine of the values in Map1.

- ATAN(y/x) = ATAN2(x,y) if x and y are both positive.
- The function ATAN and especially ATAN2 are often used in calculation of slope maps and aspect maps. Then use RADDEG(ATAN2(DX,DY)+PI).
- For the functions SIN, COS and TAN, the input angles have to be specified in radians. To convert degrees to radians, use the angular function DEGRAD. For example, the formula SIN(DEGRAD(60)) calculates the sine of 60°.
- For the functions ASIN, ACOS, ATAN and ATAN2, the output values are in radians. To convert radians to degrees, use the angular function RADDEG. For the functions ASIN and ACOS, the input values must be in the range -1 to 1.

Angular conversion functions

DEGRAD(*a*)	degrees to radians function: *a**2π/360
RADDEG(*a*)	radians to degrees function: *a**360/2π MOD 360

The DEGRAD function converts degree values in map *a* to radians: *a* is multiplied with 2π/360. The DEGRAD function is often used in combination with trigonometric functions.

The RADDEG function converts radian values in map *a* to degrees: *a* is multiplied with 360/2π MOD 360. ILWIS uses a default range of 0 - 360 with a precision of 0.01. The RADDEG function is often used in combination with trigonometric functions.

Examples of an angular conversion function

In the following example, the input map has pixel values in radians which are converted to degrees in the output map and rounded to the nearest integer.

```
Degrees = ROUND(RADDEG(Map1))
```

Map1

0.30	0.44	0.70	0.56
0.10	1.45	0.49	0.42
0.52	0.79	0.28	0.05
1.03	0.58	0.37	0.80

Degrees

17	25	40	32
6	83	28	24
30	45	16	3
59	33	21	46

Examples of conversion functions

To calculate the tangent (in degrees) of `map1`, use the following formula:

```
map2 = TAN(DEGRAD(map1))
```

To convert a map with slope values in percentages (`slopepct`) to a map with slope values degrees (`slopedeg`), use the following expression:

```
slopedeg = RADDEG(ATAN(slopepct/100))
```

Hyperbolic functions

SINH(*a*)	hyperbolic sine: $(e^{a} - e^{-a})/2$
COSH(*a*)	hyperbolic cosine: $(e^{a} + e^{-a})/2$
TANH(*a*)	hyperbolic tangent: tanh(*a*) = sinh(*a*)/cosh(*a*)

Hyperbolic functions are related to a hyperbole ($x^2 - y^2 = r^2$), in the same way as trigonometric functions are related to a circle ($x^2 + y^2 = r^2$). In the formulas above, *a* represents the X of the hyperbole.

Predefined values

PI	value π : 3.141593...
PI2	value 2 * π : 6.283185...
PIDIV2	Value ½ π : 1.570796...
PIDIV4	Value 1/4 π: 0.785398...
EXP(1)	value e : 2.718282...

The predefined values available in ILWIS are often applied in combination with trigonometric functions. Most calculations using predefined values are performed in table calculations. For a more detailed description refer to the section Table calculation.

6.2.3.4 Operators and functions on non-value maps

Below, the operators and functions are explained that can be used in Map calculation on maps with a domain Class, Group or ID. For an overview of all operators and functions available, please, refer to the schematic overview at the start of this chapter.

When using class or group names or IDs within an expression, these names and IDs should be put between double quotes, e.g. ”coffee”. In domains of the Class or Group type, you can enter codes and class names/group names. These codes can be an abbreviation of your class or group names. In expressions, also the codes can be used.

Relational operators

=	eq	*a* = *b*	*a* eq *b*	tests whether the outcome of expression *a* is equal to the outcome of expression *b*
<>	ne	*a* <> *b*	*a* ne *b*	tests whether the outcome of expression *a* is not equal to the outcome of expression *b*

Relational operators may be written as symbols (=, <>) or as letters (eq, ne). When using the symbols it does not matter whether you type spaces around the operators or not. When using the letters, e.g. in scripts, spaces are required on both sides of the operator. Pixels are assigned True when the expression is true and else False.

Example of the relational = operator

```
Result1 = Landuse = ”coffee”
```

B	B	S	B
S	S	C	C
S	C	C	S
B	C	C	C

F	F	F	F
F	F	T	T
F	T	T	F
F	T	T	T

Read in the `Landuse` map for B: Bare soil, for C: Coffee, and for S: Shrubland. Relational operators are often used in combination with a conditional IFF function.

Example of the logical AND operator
The result of *a* and *b* is true if both expressions *a* and *b* are true; in other cases false is returned.

In the following example a `Landuse` map is use used indicating the `Landuse` of 16 square parcels and a map `Commval` giving the commercial value of those parcels.

```
ResAND1 = (Landuse = ”residential”) AND (CommVal > = 20)
```

Landuse

R	C	I	R
I	C	C	R
R	R	I	R
C	C	R	I

Commval

10	35	20	25
35	15	40	15
10	40	30	5
20	5	20	10

ResAND1

F	F	F	T
F	F	F	F
F	T	F	F
F	F	T	F

Read in the map `Landuse` for R: Residential, for C: Commercial and for I: Industrial.

Using this expression, map `ResAND1` is created from 2 maps: `Landuse` and `CommVal` (commercial value). When a pixel has the class "`residential`" in map `Landuse` and the corresponding pixel in the map `Commval` has a value >=20 the expression is true. Therefore True is assigned to that pixel in the output map `ResAND1`. For all the other pixels the expression is False.

In the following example, a new map is created from a map showing the location and numbers of parcels and its attribute table.

```
ResAND2 =  (Parcel.Landuse = "residential") AND
           (Parcel.Commval > = 20)
```

`Parcel`

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Parcel

Parcelnr	Landuse	Commval
1	Residential	10
2	Commercial	35
3	Industrial	20
4	Residential	25
5	Industrial	35
6	Commercial	15
7	Commercial	40
8	Residential	15

Parcelnr	Landuse	Commval
9	Residential	10
10	Residential	40
11	Industrial	30
12	Residential	5
13	Commercial	20
14	Commercial	5
15	Residential	20
16	Industrial	10

The attribute table Parcel contains columns with information on `Landuse` and the Commercial Value for every parcel. Applying the ResAND2 formula gives the following result:

`ResAND2`

F	F	F	T
F	F	F	F
F	T	F	F
F	F	T	F

Using the second expression, map ResAND2 is created from a map with parcels and its attribute table Parcel. When a record in the table has "residential" in column Landuse, and has a commercial value larger than 20 in column CommVal the expression is True which is assigned to the corresponding pixels in map ResAND2.

Example of the logical OR operator

The result of A or B is true if either expression A, or expression B or if both expressions A and B are true; in other cases false is returned.

```
ResOR = (Landuse = "bare") OR (Landuse = "shrub")
```

Landuse

B	B	S	B
S	S	C	C
S	C	C	S
B	C	C	C

ResOR

T	T	T	T
T	T	F	F
T	F	F	T
T	F	F	F

For pixels from the map Landuse belonging to the class "bare" or to class "shrub", the expression is true. For class names, false is returned.

Of course, the OR operator can also be used to compare to different input maps, for instance when you are interested in the total area where the landuse is coffee (map Landuse), or the elevation is more than 1400 (map DEM). This can be done as follows:

```
ResOR = (Landuse = "Coffee") or (DEM > 1400)
```

Landuse

B	B	S	B
S	S	C	C
S	C	C	S
B	C	C	C

DEM

800	900	1000	1100
900	1000	1200	1400
1200	1300	1500	1600
1300	1400	1600	1800

ResOR

F	F	F	F
F	F	T	T
F	T	T	T
F	T	T	T

The map ResOR shows the result of testing for each pixel if one of the expressions or both expressions are true.

Conditional IFF function

IFF(*a*,*b*,*c*)	If condition *a* is true, then return the outcome of expression *b*, else (when condition *a* is not true) return the outcome of expression *c*. *a*,*b*,*c* may be class names, IDs, or group names, columns of domain Class, ID, or Group or expressions.

☞ The conditional IFF may be used for all types of input data: values, Ids, groups and classes.

Examples of conditional IFF functions:

```
Result1 = IFF (Landuse = "coffee", 20, 1)
```

This is the most simple type of expression using an IFF function. In this case it means: If in map `Landuse` a pixel belongs to class coffee, then assign value 20 to this pixel in output map `Result1`, else assign 1.

Landuse

B	B	S	B
S	S	C	C
S	C	C	S
B	C	C	C

Result1

1	1	1	1
1	1	20	20
1	20	20	1
1	20	20	20

Read in the `Landuse` map for B: Bare soil, for C: Coffee, and for S: Shrubland. The IFF function can be combined with different other operators and functions.

```
Result2 = IFF ((Landuse="coffee") AND (DEM>=1400), 1, 0)
```

Landuse

B	B	S	B
S	S	C	C
S	C	C	S
B	C	C	C

DEM

800	900	1000	1100
900	1000	1200	1400
1200	1300	1500	1600
1300	1400	1600	1800

Result2

0	0	0	0
0	0	0	1
0	0	1	0
0	1	1	1

If a pixel belongs to class "`coffee`" in map `Landuse` and the corresponding pixel in the `DEM` has a value greater than or equal to 1400, then assign 1 to this pixel in output map `Result2`, else assign a 0.

```
Result3 = IFF((Landuse = "coffee") AND INRANGE(DEM,
          1000,1400), 10, 0)
```

Landuse

B	B	S	B
S	S	C	C
S	C	C	S
B	C	C	C

DEM

800	900	1000	1100
900	1000	1200	1400
1200	1300	1500	1600
1300	1400	1600	1800

Result3

0	0	0	0
0	0	10	10
0	10	0	0
0	10	0	0

If a pixel belongs to class "coffee" in map `Landuse` and the corresponding pixel in the `DEM` has a value in between 1000 and 1400, then assign a 1 to this pixel in output map `Result3`, else assign a 0.

When you wish to keep the original height values in map `Result3`, you may replace the 1 in the formula with `DEM`; when you wish to assign undefineds to the remaining pixels, you may replace the 0 in the formula by ?.

6.2.4 Special Map Calculations

6.2.4.1 Calculating with undefined values

In every data set (map or table) you might encounter missing values or illegal outcomes of operations (e.g. a division by zero or the square root of a negative value). These are called undefined and are represented by a question mark (?). The outcome of calculations using an undefined value will always result in another undefined value with only one exception. This and the testing for, and assigning of undefined values is described below.

Testing for undefined values

ISUNDEF(*a*)	tests whether *a* is undefined; returns a Bool
IFUNDEF(*a*,*b*)	returns *b* when *a* is undefined, else *a* is returned
IFUNDEF(*a*,*b*,*c*)	returns *b* when *a* is undefined, else c is returned
IFNOTUNDEF(*a*,*b*)	returns *b* when *a* is not undefined, else the undefined remains
IFNOTUNDEF(*a*,*b*,*c*)	returns *b* when *a* is not undefined, else the undefined is replaced by *c*

☞ Do not use for example the expression: Map1 = ? because then an undefined is defined for every pixel and further calculations are of no use.

Example of testing for undefined values
The ISUNDEF function tests whether *a* is undefined or not known. Argument *a* can be the outcome of an expression or simply a map name or a pixel in a map. Function ISUNDEF may be used on maps with any domain type.

Below, the ISUNDEF function is combined with a conditional IFF function. Map `Result1` can be obtained by three statements:

```
Result1 = IFF(isundef(Map1), 10, Map1)
Result1 = IFUNDEF(Map1, 10, MAP1)
Result1 = IFUNDEF(Map1, 10)
```

Map1

9	3	?	4
8	?	1	?
1	5	2	6
?	?	3	8

Result1

9	3	10	4
8	10	1	10
1	5	2	6
10	10	3	8

If a pixel of `Map1` is undefined, then value 10 is assigned to this pixel in map `Result1`; else the original value is assigned.

The function IFUNDEF(*a*, *b*, *c*) gives the same result as the expression IFF(ISUNDEF(*a*), *b*, *c*).

```
Result2 = IFF(ISUNDEF(Landuse),"unknown", landuse)
Result2 = IFUNDEF(Landuse,"unknown", Landuse)
Result2 = IFUNDEF(Landuse,"unknown")

Result3 = IFNOTUNDEF (Landuse, 10)
```

Landuse

?	B	S	B
S	S	C	C
?	C	?	S
B	C	C	?

Result2

U	B	S	B
S	S	C	C
U	C	U	S
B	C	C	U

Result3

?	10	10	10
10	10	10	10
?	10	?	10
10	10	10	?

In `Result2` "`unknown`" is assigned for every undefined pixel in map `Landuse`. All other pixels remain of the same value, class, id etc. as they were in the input map.

In `Result3`, every undefined pixel in map `Landuse` remains undefined. Every defined pixel is assigned a value of 10.

Assigning undefined values

To assign undefined values, use a question mark ? when the output domain is a domain Value, and use "?" when the output domain is a domain Class, Group or ID. Below, an example is given combined with a conditional IFF.

Example of assigning undefined values

```
Result4 = IFF (Map1 = 10, ?, Map1)
```

Map1

9	3	10	4
8	10	1	10
1	5	2	6
10	10	3	8

Result4

9	3	?	4
8	?	1	?
1	5	2	6
?	?	3	8

If a pixel of `Map1` has value 10, then undefined is assigned to this pixel in map `Result4`; else the pixel will get the original value.

```
Result5 = IFF (landuse = "unknown", "?", landuse)
```

Landuse

U	B	S	B
S	S	C	C
U	C	U	S
B	C	C	U

Result5

?	B	S	B
S	S	C	C
?	C	?	S
B	C	C	?

If a pixel in map `Landuse` has class name "`unknown`", then undefined is assigned to this pixel in map `Result5`; else the original class name is assigned.

6.2.4.2 Predefined variables

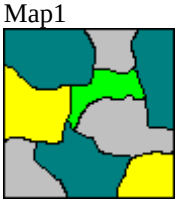
%C	used to specify a column number in a raster map
%L	used to specify a line number in a raster map
%X	used to specify the X-coordinate in a raster map
%Y	used to specify the Y-coordinate in a raster map

The predefined variables %C and %L are useful to perform calculations on specific column or line numbers in raster maps.

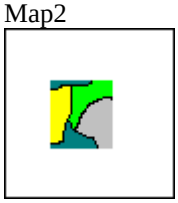
Example of predefined variables %C and %L

```
Map2 = IFF(INRANGE(%C, 100,400) AND (INRANGE (%L, 200,
       500)), Map1, "?")
```

Map1



Map2

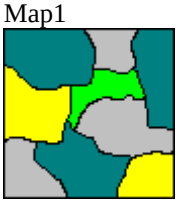


Map 2 has the same content as map1 if the column number is in the range of 100 to 400 and the line number in the range of 200 to 500. Outside this area Map2 is undefined. For more examples refer to Special Topics.

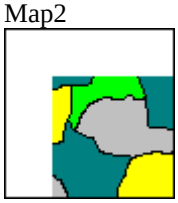
Example of predefined variables %X and %Y

```
Map2 =IFF((%X > 80000) AND (%Y < 8080000), Map1, "?")
```

Map1



Map2



Map 2 has the same content as map1 if the X-coordinate is higher than 800000 and the Y-coordinate is lower than 8080000. Outside this area Map2 is undefined. For more examples refer to Special Topics.

6.2.4.3 Create an attribute map

Raster maps of the domain type Class, ID or Group can have an attribute table with additional information on the elements in the map. An attribute table must share the same domain Class, ID or Group as the map(s) to which it refers. The domain provides the relational link between the table and the map. Therefore, the attributes in a column from an attribute table, can be put in a map, a so-called attribute map. Attribute maps may be created through operation Attribute Map. You can also directly create an attribute map by typing one of the following statements on the command line of the Main window:

```
Outmap = MapName.ColumnName
```

```
Outmap = MapName.TableName.ColumnName
```

```
Outmap = MapAttribute(MapName,ColumnName)
```

where:

Outmap	is the name of your output map
MapName	is the name of your input map (domain Class , Group or ID)
TableName	is the name of a table (using same domain as input map)
ColumnName	is the name of a column

Example of creating an attribute map

In the following example a `landuse` map is used showing the geographic position of five landuse classes. In the attribute table there are columns with the description of the landuse class and a commercial value for a certain landuse. A new map is created showing the commercial values for landuse.

```
Landval = Landuse.CommValue
```

This formula will do in this example because the attribute table has the same name as the input map. Of course, you may also use the expressions:

```
Landval = Landuse.Landuse.CommValue
```

```
Landval = MapAttribute(Landuse, Commval)
```

`Landuse(.mpr)`

1	1	2	2
1	2	2	2
3	3	4	2
5	3	4	4

Landuse(.tbt)

LUclass	Description	Comm Value
1	Bare	0
2	Shrub	50
3	Coffee	100
4	Grassland	75
5	unknown	0

Landval

0	0	50	50
0	50	50	50
100	100	75	50
0	100	75	75

The input values of the landuse map (which are the landuse classes) are replaced by the commercial value of that specific landuse class.

The first statement only works when an attribute table is linked to the input map (see the properties dialog box of the input map. For the second statement, the table does not have to be linked to the map. However the second type of statement only works when the domain used by the table is the same domain as used by the map. In such a way you may also use histogram tables and representations (.his .hsa .rpr).

6.2.4.4 Classifying a map

CLFY(a , DomainGroup)	classifies the values of a according to a domain Group.

Classified maps are less detailed then the original ones. But they often more easy to read because of the limited number of groups or classes.

- To temporary classify a map, you can simply create a representation value or gradual; a domain Group is not needed. For more information, see How to classify a map.
- To permanently classify a map, you have to predefine a Domain Group in which you indicate the upper limit for each class.

Example of classifying a value map
In this example an input map1 has values between 0 and 100 and you defined your Domain Group with the name ”class” as follows :
Upper boundary 25: Low
Upper boundary 50: Intermediate
Upper boundary 100: High

You may classify your map according to this domain using the formula:

```
Classmap=(Map1, class)
```

Map1

38	46	12	92
81	3	55	69
21	60	88	37
74	19	5	46

Classmap

I	I	L	H
H	L	H	H
L	H	H	I
H	L	L	I

All pixels of map1 with a value between 0 and 25 are assigned class Low (L) in the classified map. If they are between 25 and 50 they get class Intermediate (I), else class High (H) is assigned.

A domain Value map can be classified or sliced as well through the operation Slicing. Ranges of values of the input map are grouped together into one or more output classes. A domain Group should be created beforehand; it lists the upper boundaries of the groups and the group names. Then, double click the Slicing operation in the Operation-list and a dialog box appears on the screen. Fill in your raster map that needs to be classified, an output map name and choose the domain according to which your map needs to be classified or sliced. To force immediate calculation tick the show box.

☞ For ILWIS 1.41 users: the domain Group acts as a classify table.

To perform a multi-spectral image classification, use Sample and then the Classify operation.

6.2.4.5 Two-dimensional tables

Two-dimensional tables are used to combine or reclassify two raster maps with a Class, Group or ID domain. It defines a new class or a value for each possible combination of input classes, groups or IDs. A two-dimensional table view consists of rows representing one domain, and columns representing another domain. In the two-dimensional table, you have to assign a value, class name or ID to each possible combination of your input domains

Application of a two-dimensional table on the command line of the Main window, requires two input raster maps which have the domains as used by the two-dimensional table. The output raster map then contains the values, classes or IDs that you entered in the fields of the two-dimensional table.

When you create a new two-dimensional table, it is directly displayed in a table window where you can also edit it directly. To open and edit an existing two-dimensional table, double-click a two-dimensional table in the Catalog, double-click the Open or Edit item in the Operation-list, or select the Edit Object command on the Edit menu of the Main window and select a two-dimensional table.

A two-dimensional table can be applied by typing the following expression on the command line of the Main window:

```
OUTMAP = Twodimtablename[map1,map2]
```

where:

OUTMAP	is the name of your output map
Twodimtablename	is the name of your two-dimensional table (extension .TA2)
Map1 and Map2	are your input maps from which the domains are used in the two-dimensional table.

Example of a two-dimensional table

Two small raster maps are combined to one output map according to a two-dimensional table. The domain of Map1 contains the IDs: A, B, C, D and E and Map2 has the classes: P, Q, R, and S.

The two-dimensional table may for instance look like:

Twodim.ta2

	P	Q	R	S
A	Very Low	Very Low	Very Low	Very Low
B	Very Low	Low	Intermediately Low	Intermediate
C	Very Low	Intermediately Low	Intermediate	Intermediately High
D	Very Low	Intermediate	Intermediately High	High
E	Very Low	Intermediately High	High	Very High

Application of this table to Map1 and Map2 results in Map3 by using one of the following formulas:

```
Map3 = Twodim.ta2[map1,map2]
Map3 = Twodim[map1,map2]
```

Map1

A	A	D	B
D	A	B	B
B	C	C	B
A	D	B	A

Map2

Q	P	S	P
P	Q	Q	R
R	R	T	P
P	R	Q	P

Map3

VL	VL	H	VL
VL	VL	L	IL
IL	I	H	VL
VL	IH	L	VL

Map3 has pixel values according the classification given in the two-dimensional table twodim. Each combination of corresponding pixel values is assigned its new value.

6.2.4.6 Neighbourhood operations

Neighbourhood operations are a special spatial analysis in ILWIS. They are calculations on pixels in which the outcome depends on the neighbouring pixels. Neighbourhood operations may be performed on user-selected pixels as well as on whole maps. Just as in filtering procedures, neighbourhood operations make use of a filter. This window of 3 by 3 cells is moved over the raster map. Each cell of the output map is calculated according to the specified neighbourhood expression. The cell numbers in the moving window are coded as follows:

1	2	3
4	5	6
7	8	9

This means that the left neighbour of the central pixel is coded number 4 and the lower right pixel number 9. By definition the central pixel itself is included and has a value of 5. The result of the calculation is stored in the central pixel.
If a neighbourhood operation is performed on a pixel on the top or bottom line or on the very first or last column of a raster map, new neighbours are created by duplicating this boundary line or column. In the case of the neighbour position variable: NBPOS, the outer lines or rows added will have value 0.

Syntax of neighbourhood operations

```
Map2 = Map1#[expression]
```

where:

`Map1`	is the input map.
`#`	is the neighbourhood operator indicating all neighbours are to be used.
`[ ]`	is used to indicate the selection of a specific neighbour .
`expression`	is any expression with output values in the range 1 to 9 to select a specific neighbour.

Examples
In this example the altitude difference will be calculated in the X and Y direction. The DX and DY maps are the output maps using a digital elevation model as input map.

```
DX = dem#[4] - dem#[6]
DY = dem#[2] - dem#[8]
```

This means in words:
Subtract the right neighbour of the central pixel from the left neighbour of the central pixel and store the result in the central pixel. Subtract the lower neighbour of the central pixel from the upper neighbour of the central pixel and store the result in the central pixel.

Now the height difference is known over a horizontal distance of two pixels in two directions. There are several neighbourhood functions available for performing operations on neighbours of a pixel. These functions are listed below:

Neighbourhood functions

NBMIN (nbexpr)	returns for each pixel the lowest neighbouring value
NBMAX (nbexpr)	returns for each pixel the highest neighbouring value
NBSUM (nbexpr)	returns for each pixel sum of the neighbouring values
NBPRD (nbexpr)	returns for each pixel the predominant of the neighbouring values
NBCNDP (nbexpr)	tests for which of the neighbouring pixels a certain condition is true; when this is the case for more pixels, then the precedence is: central pixel number 5, and then 1,2,3,4,6,7,8,9; when none of the neighbours satisfies the specified condition the outcome value is 0
NBCNT (nbexpr)	tests the number of neighbours which satisfy the specified condition
NBFLT (fltname)	applies a pre-defined linear filter of 3 by 3 cells to all neighbouring pixels
NBDIS (nbexpr)	returns the distance between the central pixel and its neighbouring ones
NBPOS (nbexpr)	returns the position of a neighbour in the neighbour expression
NBMINP (nbexpr)	returns the position of the neighbour with the lowest value; when this is the case for more pixels, then the precedence is: central pixel number 5, and then 1,2,3,4,6,7,8,9
NBMAXP (nbexpr)	returns the position of the neighbour with the highest value; when this is the case for more pixels, then the precedence is: central pixel number 5, and then 1,2,3,4,6,7,8,9
NBPRDP (nbexpr)	returns the position of the neighbour with the predominant value; when this is the case for more pixels, then the precedence is: central pixel number 5, and then 1,2,3,4,6,7,8,9

NBDIS is a predefined variable which is only useful when multiplied by the pixel size (see example).

Examples of neighbourhood functions

Calculating a classified aspect map:

In this example a classified aspect map will be calculated which contains 9 classes. The input map is a DEM (digital elevation model) By using the neighbourhood function NBMINP, a pixel is assigned the position number of the lowest pixel in the imaginative window. This indicates the direction of the slope as follows:

Pixel number	Direction
1	NW
2	N
3	NE
4	W
5	Flat
6	E
7	SW
8	S
9	SE

```
aspect = NBMINP (DEM#)
```

DEM

3696	3693	3694	3700	3711
3691	3680	3680	3693	3698
3689	3680	3680	3680	3692
3691	3688	3680	3673	3680

aspect

9	8	7	7	7
6	5	5	4	7
3	5	9	8	7
3	2	6	9	8

Iterations

Iterations are a special type of calculations. They are a successive repetition of a mathematical operation, using the result of one calculation as input for the next. The calculation stops when the difference of the output compared to the input is negligible or if the number of steps is reached which was defined before. Iterations are often used in combination with neighbourhood operations. Such an application might be for instance the selection of an item or area which fits a certain condition, starting from one pixel (see: example calculation of a flooded area)

In ILWIS, 4 iteration functions are available listed below:

MAPITER (startmap, iterexpr)	performs iterations on the startmap according the iteration expression until no pixel changes anymore
MAPITER (startmap, iterexpr, times)	performs a specified number of iterations on the startmap according the iteration expression
MAPITERPROP (startmap, iterexpr)	performs iterations with propagation on a startmap until no pixel changes anymore; the newly calculated value for a pixel is used in calculating the next line instead of in the next iteration
MAPITERPROP (startmap, iterexpr,times)	performs a specified number of iterations with propagation on a startmap; the newly calculated value for a pixel is used in calculating the next line instead of in the next iteration

After each iteration ILWIS shows the number of changed pixels. This number is the total of changes after performing one iteration in all directions (up, down, right and left).

Distance calculation

You may calculate the distance from pixels in a map to one or more specified pixels. To do so you need to create a startmap in which you find these specified pixels. It is recommended to give those pixels a value 0, because the output map contains pixel values which are the sum of the distance between the two pixels and the value of the pixel in the start map. The distance calculation in neighbourhood operations uses a predefined filter NBDIS. This is a 3 by 3 filter as shown below:

√2	1	√2
1	0	1
√2	1	√2

Use the pixel editor to create the start map:

- Firstly, display a map showing the area in which you want to perform the distance calculation and choose File, Create, Create Raster Map from the menu in the map window. Then you fill in the dialog box. In this example `Start` is filled in as the raster map name. Accept the default georeference which is the same as in the map displayed in the map window and choose domain Value. Then, choose one or more pixels which act as the starting point(s) for the distance calculation. Give them value 0. When leaving the pixel editor the map `Start` has one or several defined pixel with value 0, the rest is undefined. Display the `Start` map to force calculation.
- Now the actual distance calculation can be performed. Iteration with propagation is used until there are no changes anymore in any of the pixel values. Type:

  ```
  NBMIN(Start# + NBDIS * PIXSIZE)
  ```

  Iterations sometimes take quite a lot of time, especially with large maps. Therefore the calculation time may be reduced to split the calculation in two steps. First, the distance filter has to be applied in the iteration and this temporary result will be stored. Subsequently, this temporary result will be multiplied with the pixel size.
- To start the calculation, double-click the Iteration item in the Operation-list of the Main window. The Iteration dialog box appears on the screen. In this example the initial map is raster map `Start`. In the expression box, type

  ```
  NBMIN(Start# + NBDIS)
  ```

  This means: Calculate for every neighbouring pixel the temporary distance to the nearest specified pixel in the start map and store the result in map `Temp`. The default is accepted for the Stop criterion: Until No changes. Keep the option propagation selected because then the new pixel value is immediately used in the calculation of the next pixel. `Temp` is filled in as the output raster map; select the Show check box. The defaults are accepted for the domain and then click OK.
  You may also type directly in the command line of the Main window:

  ```
  Temp = MAPITERPROP (Start, NBMIN( Start# + NBDIS))
  ```

- After accepting the display options, map `Temp` is displayed in which the pixel values represent the nearest distance to the earlier specified pixel(s). The pixel size is not taken into account yet. The final result is obtained by multiplying the temporary result with the pixel size. In this example the pixel size is 10m. Type the following formula on the command line of the Main window:

  ```
  Mapdis = (Temp * PIXSIZE(Start))
  ```

  Startmap

?	?	?	?	?
?	?	?	?	?
?	?	?	0	?
?	?	?	?	?

  Temp

3.8	2.8	2.4	2.0	2.4
3.4	2.4	1.4	1.0	1.4
3.0	2.0	1.0	0	1.0
3.4	2.4	1.4	1.0	1.4

  Mapdis

38	28	24	20	24
34	24	14	10	14
30	20	10	0	10
34	24	14	10	14

  The map `Mapdis` is the result of a distance calculation. The starting pixel has value 0. After the iterations with propagation, each pixel in map `Mapdis` represents the distance to the starting pixel.

Determining flat areas and pits in a DEM

An area is considered to be flat when in the moving window of 3 by 3 pixels all pixels have the same value. By using one of the following expression you can check which pixels have 8 neighbours with the same altitude as the central pixel.

```
Flat = NBCNT8(DEM#=DEM)=8
```

```
Flat = NBCNT(DEM#=DEM)=9
```

A so-called pit is a pixel which has a value lower than all of its surrounding pixels. A DEM will be checked for the presence of pits. This is done using the following expression:

```
Pit = NBMINP(DEM#)=5
```

This results in a map with domain Bool. If from all neighbours, the central pixel has the lowest value, that central pixel will be assigned True, else False.

Calculation of the flooded area, given dam site and dam altitude

When a dam is constructed in a valley, an area upstream will be flooded up to a certain water level. To determine the exact area to be flooded you can use neighbourhood operations in ILWIS. Then, you can also calculate the volume of the water body.

- The first step is determination of the dam site, dam altitude, freeboard and the designed water level *h*.
- If not available create a DEM (digital elevation model) of the area.
- Then, make a raster map showing the exact location of the dam. To do so, start with the display of the DEM and use the pixel editor to create a new raster map. Choose in the menu of the map window File, Create, Create Raster Map. The Create Raster Map dialog box appears on the screen. Fill in the new map name (e.g. `Dam`). Then, select the georeference which is used by the DEM (accept the default) and as domain select `Value` and press OK. Zoom in to the

area where you want to build the dam. Now, you can select pixels which will form the dam in map Dam. The pixels indicating the dam should be connected in a proper way (see figures below). Otherwise pixels at the opposite site of the dam are also used in the calculation later.

wrong

correct

The left mouse button is used to select the pixels while holding down the Ctrl key of the keyboard. When all pixels are selected double-click a pixel (still holding down the Ctrl key) and give the dam the code of the altitude of the top of the dam

- To combine the DEM and the raster map Dam.mpr type on the command line of the Main window:

  ```
  Demnew = IFUNDEF(Dam, DEM, Dam)
  ```

  This means that if map Dam is undefined the value of the DEM is assigned, else the value given in map Dam. Thus the altitude of the dam (in this example 3360m) is included in the Digital Elevation Model.
- To determine the area to be flooded you need to create a map indicating one pixel in this area. Using neighbourhood operations this pixel acts as the starting point for the calculation. Use the pixel editor to create the new raster map. Firstly, display the map Damnew.mpr and choose File, Create, Create Raster Map from the menu in the map window. Then fill in the dialog box. In this example fill in Start as the raster map name. Accept the default georeference which is the same as in Demnew and choose domain Bool. Choose a pixel in the area to be flooded, double-click the pixel and make it ”True”. When leaving the pixel editor the map Start has only one defined pixel, the rest is undefined. Display the map to force calculation.
- Now the actual calculation of the area can be performed. Iteration with propagation is used until there are no changes anymore in any of the pixel values. To start the calculation, double-click the Iteration item in the Operation-list of the Main window. The Iteration dialog bog appears on the screen. In this example the initial map is raster map Start. Type in the expression box:

  ```
  IFF (Demnew > 3340, Start, NBMAX(Start1#))
  ```

  This means: if the altitude in the new Digital Elevation Model is higher than 3340 m, than return the pixel values of raster map Start. Else, assign the maximum value of the neighbouring pixels found in raster map Start (which is ”True”). In the first iteration there is only one starting pixel which is ”True”. Every iteration, the neighbouring pixels which satisfy the conditions (altitude < 3340 m) will get the same value as that starting pixel. This will continue until next neighbouring pixels have an altitude of more than 3340 m. (The upper side

of the dam was at an altitude of 3360 m; the freeboard of the dam is 20 m; the actual water level is at 3340 m altitude)

- The default is accepted for the Stop criterion: Until No changes. Keep the option propagation selected because then the new pixel value is immediately used in the calculation of the next pixel. `Flooded` is filled in as the output raster map and select the show box. The defaults are accepted for the domain and then click OK.
- After accepting the display options, the area which will be flooded appears on the screen.
- To find out the size of the flooded area calculate a histogram by double clicking the Histogram operation in the Operation-list. Open the list box and choose map Flooded. The size of the area which is flooded (True) can be read from this histogram table.
- The next step is the calculation of the volume of the water body. Therefore cross the newly calculated map `Flooded` with the DEM `Damnew`. The crossing of two maps is a relatively time consuming operation. You can limit the calculation time and the size of the cross table by making a new DEM showing only the flooded area. Do that by typing in the command line of the Main window:

  ```
  Demsmall = IFF(Flooded, Damnew, ?)
  ```

  This means that outside the flooded area all pixels are undefined and not taken into account in the crossing operations.
- The crossing is started by double-clicking the Cross operation in the Operation-list of the Main window. Then, fill in the cross dialog box. The first map you choose from the context-sensitive menu is the map `Flooded` and the second map is the new reduced DEM `Demsmall.` Fill in `Flooddam` as the output table name and select the show box. You do not need a cross map because the calculation will be performed using the cross table. Therefore do not select the option output map. Finally click on O.K. and the cross table is calculated and appears on the screen. It shows the combinations of input values of the map `Flooded` and the map `Demsmall`, the number of pixels that occur for each combination and the area for each combination.
- To calculate the depth you only have to subtract the altitude given in the DEM `Demsmall` from the planned water level (3340 m). Type in the command line of the table window:

  ```
  Depth = 3340 - Demsmall
  ```

- Now you can calculate the volume per area having a certain depth as follows:

  ```
  Vol = Depth * Area
  ```

- Finally sum all volumes of all different water depths together. This can be done using the aggregation function AGGSUM.

  ```
  Sum = AGGSUM (Vol)
  ```

The column Sum shows the total volume of water in cubic meters for the entire area which will be flooded after creating the dam.