1 Program imcalc

1.1 Purpose

Do arithmetic on 2D images, pixel-by-pixel.

1.2 Usage

imcalc options

1.3 Options

-datum type = Coerce the output data to be stored as the given type, which may be byte, short, or float. [default = datum of first input image]

-a dname = Read image 'dname' and call the voxel values 'a' in the expression. 'a' may be any letter from 'a' to 'z'.

Note: If some letter name is used in the expression, but not present in one of the image options here, then that variable is set to 0.

-expr "expression" Apply the expression within quotes to the input images, one voxel at a time, to produce the output image.

("sqrt(a*b)" to compute the geometric mean, for example)

-output name = Use 'name' for the output image filename.

[default='imcalc.out']

1.4 Problems

- Complex-valued images cannot be processed (byte, short, and float are OK).
- Evaluation of expressions is not very efficient.

1.5 Expressions

Arithmetic expressions are allowed, using + - * / ** and parentheses. As noted above, datasets are referred to by single letter variable names. At this time, C relational, boolean, and conditional expressions are NOT implemented. Built in functions include:

sin ,	cos ,	tan ,	asin ,	acos ,	atan ,	atan2,	
sinh ,	cosh ,	tanh ,	asinh ,	acosh ,	atanh ,	exp ,	
log ,	log10,	abs ,	int ,	sqrt ,	max ,	min ,	
J0 ,	J1 ,	Y0 ,	Y1 ,	erf ,	erfc ,	qginv,	qg ,
rect ,	step ,	astep,	bool ,	and ,	or ,	mofn ,	

where

qg(x) = reversed cdf of a standard normal distribution

qginv(x) = inverse function to qg,
min, max, atan2 each take 2 arguments,
J0, J1, Y0, Y1 are Bessel functions (see Watson),
erf, erfc are the error and complementary error functions.

The following functions are designed to help implement logical functions, such as masking of 3D volumes against some criterion:

step(x)	=	{1 if x>0 ,	0 if x<=0},
astep(x,y)	=	{1 if abs(x) > y ,	0 otherwise} = step(abs(x)-y)
rect(x)	=	{1 if abs(x)<=0.5,	0 if abs(x)>0.5},
bool(x)	=	{1 if x != 0.0 ,	0 if x == 0.0},
and(a,b,...,c)	=	{1 if all arguments are nonzero,	0 if any are zero}
or(a,b,...,c)	=	{1 if any arguments are nonzero,	0 if all are zero}
mofn(m,a,...,c)	=	{1 if at least 'm' arguments are nonzero,	0 otherwise}

These last 3 functions take a variable number of arguments.

All computations are carried out in double precision before being truncated to the final output 'datum'.

Note that the quotes around the expression are needed so the shell doesn't try to expand * characters, or interpret parentheses.

(Try the 'ccalc' program to see how the expression evaluator works.)