1 Program 3dcalc

1.1 Purpose

Do arithmetic on 3D datasets, voxel-by-voxel [no inter-voxel computation].

1.2 Usage

3dcalc [options]

1.3 Options

-verbose = Makes the program print out various information as it progresses.

-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 dataset]

-fscale = Force scaling of the output to the maximum integer range. This only has effect if the output datum is byte or short (either forced or defaulted). This option is often necessary to eliminate unpleasant truncation artifacts. [The default is to scale only if the computed values seem to need it – are all less than 1 or there is at least one value beyond the integer upper limit.]

- In earlier versions of 3dcalc, scaling (if used) was applied to all sub-bricks equally – a common scale factor was used. This would cause trouble if the values in different sub-bricks were in vastly different scales. In this version, each sub-brick gets its own scale factor. To override this behavior, use the '-gscale option'.

-gscale = Same as '-fscale', but also forces each output sub-brick to get the same scaling factor. This may be desirable for 3D+time datasets, for example.

-a dname = Read dataset 'dname' and call the voxel values 'a' in the expression that is input below. 'a' may be any single letter from 'a' to 'z'.

- If some letter name is used in the expression, but not present in one of the dataset options here, then that variable is set to 0.
- If the letter is followed by a number, then that number is used to select the sub-brick of the dataset which will be used in the calculations. For example, '-b3 dname' specifies that the variable 'b' refers to sub-brick 3 of the dataset (indexes start at 0). N.B.: Another way to achieve the effect of '-b3' is described below in the 'INPUT DATASET SPECIFICATION' section.

-expr "expression" Apply the expression within quotes to the input datasets, one voxel at a time, to produce the output dataset. ("sqrt(a*b)" to compute the geometric mean, for example).

-prefix pname = Use 'pname' for the output dataset prefix name. [default='calc']

-session dir = Use 'dir' for the output dataset session directory. [default='./'=current working directory]

1.4 3D+TIME DATASETS

This version of 3dcalc can operate on 3D+time datasets. Each input dataset will be in one of these conditions:
(A) Is a regular 3D (no time) dataset; or
(B) Is a 3D+time dataset with a sub-brick index specified ('-b3'); or
(C) Is a 3D+time dataset with no sub-brick index specified ('-b').
If there is at least one case (C) dataset, then the output dataset will also be 3D+time; otherwise it will be a 3D dataset with one sub-brick. When producing a 3D+time dataset, datasets in case (A) or (B) will be treated as if the particular brick being used has the same value at each point in time.

Multi-brick 'bucket' datasets may also be used. Note that if multi-brick (bucket or 3D+time) datasets are used, the lowest letter dataset will serve as the template for the output; that is, '-b fred+tlrc' takes precedence over '-c wilma+tlrc'. (The program 3drefit can be used to alter the .HEAD parameters of the output dataset, if desired.)

1.5 INPUT DATASET SPECIFICATION

An input dataset is specified using one of these forms:

'prefix+view', 'prefix+view.HEAD', or 'prefix+view.BRIK'.

You can also add a sub-brick selection list after the end of the dataset name. This allows only a subset of the sub-bricks to be read in (by default, all of a dataset's sub-bricks are input). A sub-brick selection list looks like one of the following forms:

fred+orig[5]	==>	use only sub-brick #5
fred+orig[5,9,12]	==>	use #5, #9, and #12
fred+orig[5..8] or [5-8]	==>	use #5, #6, #7, and #8
fred+orig[5..13(2)] or [5-13(2)]	==>	use #5, #7, #9, #11, and #13

Sub-brick indexes start at 0. You can use the character '$' to indicate the last sub-brick in a dataset; for example, you can select every third sub-brick by using the selection list
fred+orig[0..$(3)]

- The sub-bricks are read in the order specified, which may not be the order in the original dataset. For example, using fred+orig[0..$(2),1..$(2)] will cause the sub-bricks in fred+orig to be input into memory in an interleaved fashion. Using fred+orig[$..0] will reverse the order of the sub-bricks.

- The '$', '(', ')', '[', and ']' characters are special to the shell, so you will have to escape them. This is most easily done by putting the entire dataset plus selection list inside forward single quotes, as in 'fred+orig[5..7,9]'.

1.6 1D TIME SERIES

You can also input a '*.1D' time series file in place of a dataset. In this case, the value at each spatial voxel at time index n will be the same, and will be the n-th value from the time series file. At least one true dataset must be input. If all the input datasets are 3D (single sub-brick) or are single sub-bricks from multi-brick datasets, then the output will be a 'manufactured' 3D+time dataset For example, suppose that 'a3D+orig' is a 3D dataset:

3dcalc -a a3D+orig -b b.1D -expr "a*b"

The output dataset will 3D+time with the value at (x,y,z,t) being computed by a3D(x,y,z)*b(t). The TR for this dataset will be set to 1 second – this could be altered later with program 3drefit. Another method to set up the correct timing would be to input an unused 3D+time dataset – 3dcalc will then copy that dataset's time information, but simply do not use that dataset's letter in -expr.

1.7 COORDINATES

If you don't use '-x', '-y', or '-z' for a dataset, then the voxel spatial coordinates will be loaded into those variables. For example, the expression

'a*step(x*x+y*y+z*z-100)'

will zero out all the voxels inside a 10 mm radius of the origin.

1.8 PROBLEMS

- Complex-valued datasets cannot be processed.
- This program is not very efficient (but is faster than before).

1.9 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 ,	
sind ,	cosd ,	tand ,	median,				

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,
sind, cosd, tand take arguments in degrees (vs. radians),
median(a,b,c,...) computes the median of its arguments [median takes a variable number of arguments].

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.

The following 27 new [Mar 1999] functions are used for statistical conversions, as in the program 'cdf':

fico_t2p(t,a,b,c),	fico_p2t(p,a,b,c),	fico_t2z(t,a,b,c),
fitt_t2p(t,a) ,	fitt_p2t(p,a) ,	fitt_t2z(t,a) ,
fift_t2p(t,a,b) ,	fift_p2t(p,a,b) ,	fift_t2z(t,a,b) ,
fizt_t2p(t) ,	fizt_p2t(p) ,	fizt_t2z(t) ,
fict_t2p(t,a) ,	fict_p2t(p,a) ,	fict_t2z(t,a) ,
fibt_t2p(t,a,b) ,	fibt_p2t(p,a,b) ,	fibt_t2z(t,a,b) ,
fibn_t2p(t,a,b) ,	fibn_p2t(p,a,b) ,	fibn_t2z(t,a,b) ,
figt_t2p(t,a,b) ,	figt_p2t(p,a,b) ,	figt_t2z(t,a,b) ,
fipt_t2p(t,a) ,	fipt_p2t(p,a) ,	fipt_t2z(t,a) .

See the output of 'cdf -help' for documentation on the meanings of and arguments to these functions. (After using one of these, you may wish to use program '3drefit' to modify the dataset statistical auxiliary parameters.)

1.10 Notes

- 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. The arithmetic parser and evaluator is written in Fortran-77 and is derived from a program written long ago by RW Cox to facilitate compiling on an array processor hooked up to a VAX. It's a mess, but it works.