Subject: Re: "ALOG2" ?
Posted by steinhh on Sat, 03 Apr 1999 08:00:00 GMT
View Forum Message <> Reply to Message

> Does anyone have an IDL function to compute the logarithm
> base 2 of the elements of an array?
>
> I would like it to work like the the C function frexp() and
> the IEEE floating point function logb().

Nope, I don't have an IDL function, but it's not that
difficult to write a DLM that interfaces to the C functions.

Below is a dlm file and corresponding C file that makes
logb() and frexp() available as builtin IDL functions.
Extension to other, similar functions should be fairly
straightforward. I haven't checked the routines extensively,
but I think they work as expected.

Regards,

Stein Vidar

log2.dlm----------
```
# $Id: log2.dlm,v 1.1 1999/04/03 09:29:38 steinhh Exp steinhh $
MODULE LOG2
DESCRIPTION Base 2 log functions.
VERSION $Revision: 1.1 $
BUILD_DATE $Date: 1999/04/03 09:29:38 $
SOURCE S.V.H.HAUGAN
FUNCTION LOGB 1 1
FUNCTION FREXP 2 2
```
---

log2.c----------
```
#include <unistd.h>
#include <stdio.h>
#include <math.h>
#include "export.h"


#define NULL_VPTR ((IDL_VPTR) NULL)

/* Help function to create temporary variable of given type, with same
   dimensionality as the source */

void *result_var(IDL_VPTR src, int type, IDL_VPTR *res)
{
```

```
  /* Is it an array? */

  if (src->flags&IDL_V_ARR) {
    return (void*) IDL_MakeTempArray(type, src->value.arr->n_dim,
         src->value.arr->dim,
         IDL_ARR_INI_NOP,res);
  }

  /* Nope, scalar: */

  if ( (*res=IDL_Gettmp()) == NULL_VPTR ) {
    IDL_Message(IDL_M_NAMED_GENERIC,IDL_MSG_LONGJMP,
  "Couldn't create temporary variable");
  }

  (*res)->type = type;
  return & (*res)->value.c;
}


IDL_VPTR LOGB(int argc, IDL_VPTR argv[])
{
  IDL_VPTR src,dst;

  float *f_src,*f_dst;
  double *d_src,*d_dst;

  IDL_MEMINT nn;

  src = argv[0];

  IDL_EXCLUDE_UNDEF(src);
  IDL_ENSURE_SIMPLE(src);
  IDL_EXCLUDE_COMPLEX(src);
  IDL_EXCLUDE_STRING(src);

  if (src->type != IDL_TYP_FLOAT && src->type != IDL_TYP_DOUBLE) {
    src = IDL_CvtDbl(1,argv); /* MIGHT make a temporary variable */
  }

  /* We can do operation "in place" if src is a temp. (but not const!) */
  /* Otherwise we need to make storage space for result                 */

  if (src->flags & IDL_V_TEMP && !(src->flags & IDL_V_CONST)) dst = src;
  else result_var(src,src->type,&dst);

  /* Get pointers to data spaces, and number of elements into nn */
```

```
  if (src->flags&IDL_V_ARR) {
    nn = dst->value.arr->n_elts;
    f_dst = (float*) dst->value.arr->data;
    f_src = (float*) src->value.arr->data;
    d_dst = (double*) dst->value.arr->data;
    d_src = (double*) src->value.arr->data;
  } else {
    nn=1;
    f_dst = &dst->value.f;
    f_src = &src->value.f;
    d_dst = &dst->value.d;
    d_src = &src->value.d;
  }

  if (src->type==IDL_TYP_FLOAT) while (nn--) *f_dst++ = logbf(*f_src++);
  else                          while (nn--) *d_dst++ = logb(*d_src++);

  return dst;
}

#define GETVARDATA(v,n) ((v->flags&IDL_V_ARR) \
    ? (*(n) = v->value.arr->n_elts, v->value.arr->data) \
    : (*(n) = 1, & v->value.c ) )

IDL_VPTR FREXP(int argc, IDL_VPTR argv[])
{
  IDL_VPTR src,dst,ds2,tmp;

  float *f_src,*f_dst;
  double *d_src,*d_dst;
  IDL_LONG *i_ds2;

  IDL_MEMINT nn;

  src = argv[0];

  IDL_EXCLUDE_UNDEF(src);
  IDL_ENSURE_SIMPLE(src);
  IDL_EXCLUDE_COMPLEX(src);
  IDL_EXCLUDE_STRING(src);

  if (src->type != IDL_TYP_FLOAT && src->type != IDL_TYP_DOUBLE) {
    src = IDL_CvtDbl(1,argv); /* MIGHT make a temporary variable */
  }

  /* We can do operation "in place" if src is a temp. (but not const!) */
  /* Otherwise we need to make storage space for result                 */
```

```
  if (src->flags & IDL_V_TEMP && !(src->flags & IDL_V_CONST)) dst = src;
  else result_var(src,src->type,&dst);


  /* Now, take care of second argument (output) */

  ds2 = argv[1];

  /* Avoid expressions - we check this before doing calculations, although */
  /* the IDL_VarCopy does it again later                                   */

  IDL_EXCLUDE_EXPR(ds2);

  /* Generate temporary variable with enough space & correct type */
  i_ds2 = (IDL_LONG*) result_var(src,IDL_TYP_LONG,&tmp);

  switch (src->type) {
  case IDL_TYP_FLOAT:
    f_src = (float*) GETVARDATA(src,&nn);
    f_dst = (float*) GETVARDATA(dst,&nn);
    while (nn--) *f_dst++ = frexpf(*f_src++,i_ds2++);
    break;
  case IDL_TYP_DOUBLE:
    d_src = (double*) GETVARDATA(src,&nn);
    d_dst = (double*) GETVARDATA(dst,&nn);
    while (nn--) *d_dst++ = frexp(*d_src++,i_ds2++);
    break;
  }

  /* Make sure we copy (i.e. move) the temporary variable to the output arg */
  /* This automatically frees any space occupied previously by output arg */
  IDL_VarCopy(tmp,ds2);

  return dst;
}


IDL_SYSFUN_DEF main_def[] =
  {{(IDL_FUN_RET) LOGB,"LOGB",  1, 1},
   {(IDL_FUN_RET) FREXP,"FREXP",  2, 2}
  };


int IDL_Load(void)
{
  return IDL_AddSystemRoutine(main_def,IDL_TRUE,2); /* Just add'em */
}
 ----------------------------------------
```