Title
C Wrappers in XLISP-STAT

Author
de Leeuw, Jan

Publication Date
1999-06-24

USING C WRAPPERS IN XLISP-STAT

JAN DE LEEUW

ABSTRACT. This paper discusses incorporating C (and FORTRAN) functions into the XLISP-STAT statistical computing environment, by using shared libraries loaded at runtime. We provide a number of examples that can be used as templates. They can be used (as in S-plus) to speed up XLISP-STAT programs, but also (as in SWIG) to produce graphical user interfaces for existing C and FORTRAN programs. The appendices discuss a number of completed projects following these lines which considerably extend XLISP-STAT.

Contents

Date: June 24, 1999.
We thank Luke Tierney for help along the way.

1. Introduction

`XLISP-STAT` [Tierney, 1990] can be extended in two different ways. By far the most common one is to write additional code in Lisp, which is read into the interpreter at run-time. Much less common is to extend the system by writing additional code in `C`.

There are basically two reasons to use `C`. In the first place, we may have legacy `C` code, for instance from books such as Press et al. [1988], and we may not have the time or the resources to translate this `C` to Lisp. Secondly, we may have a project, or a part of a project, which needs to run very fast, faster than is possible in the interpreted Lisp environment of `XLISP-STAT`. In the interpreted environment we can do fast prototyping, and we can write elaborate and elegant graphical interfaces. Fast floating point computation, however, is only possible if the necessary functions are already linked as object code into the `XLISP-STAT` executable.

There are two ways to incorporate `C` in `XLISP-STAT`. It can be done at *compile-time*, when the `XLISP-STAT` executable is build. One simply writes additional functions in `C`, and one uses the `XLISP-STAT` application programmer interface to make them accessible to the interpreter. This is not hard to do, but it has a major disadvantage. Anybody who does this will have created a personal copy of `XLISP-STAT`, different from all other copies in the world. It will be difficult to maintain and to upgrade, unless the extensions are incorporated (by Luke Tierney) in the canonical `XLISP-STAT` source tree. Extending at compile-time also means you need the tools to build a complete `XLISP-STAT` distribution. On the Mac, for instance, this implies you need to have the Metrowerks CodeWarrior Pro tools, because that is the only supported development environment.

Alternatively, it is possible to load precompiled `C` code at *run-time*. This is what we discuss in this note. It requires you to have utilities for building shared libraries, but this can be done with many different sets of tools. On the Mac, for instance, the free `MPW` and `GNU` environments, with the `MrC` and `gcc` compilers, can be used.

The Lisp tools and `XLISP-STAT` "wrapper" extensions to build these shared libraries have been developed by Luke Tierney. The technical aspects and the implementation are discussed in detail by Tierney [1998c]. Background information on dynamic loading, native pointers, and shared libraries is in Tierney [1998b,d,f]. Applications that create a regular expression library and a socket interface for `XLISP-STAT` are in Tierney [1998e,g].

In this paper we strip away as much of the technical detail as possible, and concentrate on simple computational examples. For a real understanding of the implementation on the various operating systems, we refer to Tierney's papers. They can all be found at the URL

```
www.stat.umn.edu/~luke/xls/projects/
```

2. Lisp Code , Byte Code, Object Code

We can be a little bit more specific here. Let us take the inner product function as an example. The inner product is defined in the file `linalg.lsp` in the `XLISP-STAT` distribution. That file gets byte-compiled during installation, and normally it sits as a byte-compiled function in the `XLISP-STAT` workspace. We can show this by looking in the function slot of the `inner-product` function.

```
> (symbol-function 'inner-product)
#<Byte-Code-Closure-INNER-PRODUCT: #54ae758>
```

Byte-compilation transforms Lisp code into instructions for the `XLISP-STAT` virtual machine, which runs on the various platforms that `XLISP-STAT` has been ported to. Usually, byte-compiled code is considerably faster than interpreted Lisp code. But much less fast than the native object code for the specific processor that regular `C` or `FORTRAN` compilers produce.

Fortunately, the `inner-product` function is just a high-end interface to the `blas-ddot` function. It does some error testing in Lisp, and then calls `blas-ddot` to do the work. And if we look in the corresponding function slot, we see

```
> (symbol-function 'blas-ddot)
#<Subr-BLAS-DDOT: #548aa18>
```

Thus `blas-ddot` is a *subr*, which means a compiled function living in object code in the `XLISP-STAT` executable. It is taken from the C version of the `BLAS` Anderson et al. [1992], which is a library of highly efficient building blocks for numerical linear algebra. In the `XLISP-STAT` distribution it is in the file `blas.c`. Thus the critical parts of `inner-product`, which is where the computation happens, are efficient.

Let us illustrate with an example. The autocovariance (of lag ℓ) of a sequence $x_1, \ldots, x_T$ is given by

$$\gamma_\ell(x) = \frac{1}{T}\sum_{t=1}^{T-\ell}(x_t - \overline{x})(x_{t+\ell} - \overline{x})$$

Thus we take `x`, put it in deviations from the mean, chop off the first ℓ elements to get, say, `x-tail`, chop off the last ℓ elements to get `x-head`, take the inner product of `x-tail` and `x-head`, and divide by T. Here it is in Lisp.

```
(defun conv (lag x)
 (inner-product (butlast x lag) (butfirst x lag))
)

(defun butfirst (x &optional (n 1))
  (select x (which (<= n (iseq (length x)))))
)
```

If we want to compute the first `maxlag` autocovariances, we use mapping and say

```
(defun autocovar (x maxlag)
  (let ((n (length x))
        (z (- x (mean x))))
  (/ (mapcar #'(lambda (k) (conv k z)) (iseq maxlag)) n)
))
```

It seems that this should be quite efficient, because as we have seen most of the computation is done in `blas-ddot`, which is the engine of `inner-product`. So let's give it a try. The machine is a 300 MHz G3 with 128 MB of RAM, with MacOS 8.6, adn with `XLISP-STAT` 3.52.9 (beta). We apply it to the Zürich sunspot data, a series of length 2820, first by using raw Lisp.

```
> (symbol-function 'autocovar)
#<Closure-AUTOCOVAR: #57eb358>
>(time (autocovar a 100))
The evaluation took 4.47 seconds; 2.88 seconds in gc.
```

The large amount of garbage collecting indicates there is not enough memory available in the workspace. We say

```
>(expand 100)
100
>(time (autocovar a 100))
The evaluation took 2.20 seconds; 0.42 seconds in gc.
```

In this case, byte compiling does not make a difference.

```
> (compile 'autocovar)
AUTOCOVAR
> (symbol-function 'autocovar)
#<Byte-Code-Closure-AUTOCOVAR: #57dbc58>
> (time (autocovar data 100))
The evaluation took 2.58 seconds; 0.68 seconds in gc.
```

In all cases, the computations seem to take about $1.6 - 1.8$ seconds. This includes the mapping and the selection from the list.

It is well known, for instance Newton [1988, pag 24], that the autocovariance can be computed by applying two Fourier transforms to the sequence. This is implemented in the function below.

```
(defun autocovar (x maxlag)
  (let* ((n (length x))
 (m (+ n maxlag))
 (y (append (- x (mean x)) (repeat 0 maxlag)))
 (r (/ (squared-abs (fft y)) m))
 (a (* (/ m n) (/ (realpart (fft r)) m))))
  (select a (1+ (iseq maxlag)))
))

(defun squared-abs (x)
(let ((r (realpart x))
      (c (imagpart x)))
(+ (* r r) (* c c))
))
```

We now find

```
> (time (autocovar data 100))
The evaluation took 0.12 seconds; 0.02 seconds in gc.
```

This is a dramatic difference. It is due to the fact that now almost all of the computing is done on the `C` level, with virtually no manipulation of lists. Besides that, the subr `fft` implements the fast Fourier transform, which seems to be living up to its name.

3. Shared Libraries: Example 1

Let us discuss a simpel first example of using shared libraries and the wrapper system. We share write a `C` version of the function `conv` we earlier did in Lisp. The file `cconv.c` looks like this.

```
double cconv (int l, int n, double * x)
{
double s = 0.0, * y = x + (n - l), * z = x + l, * u = x;
while (u < y)
     s +=  *u++ * *z++;
return (s);
}
```

Writing the additional glue code that links the `C` function to the `XLISP-STAT` application is the next step. This is OS dependent and not very simple. Tierney has written `XLISP-STAT` functions that writes these "wrappers" for you.

In our application the wrapper is in the file `conv.wrp` below.

```
(wrap:c-lines "double cconv (int, int, double *);")
(wrap:c-function base-conv "cconv"
    (:integer :integer (:cptr "double")) :flonum)

(defun conv (lag vec)
  (let ((n (length vec))
        (vec (coerce vec '(vector c-double))))
  (base-conv lag n
        (wrapptrs::cast-c-double (array-data-address vec)))
))
```

The first two lines are the critical ones, the second part is an `XLISP-STAT` function that calls the `C` function in the library. If we run `conv.wrp` through the make-wrappers function, using

```
(wrap:make-wrappers "conv.wrp")
```

it generates a file `conv.lsp` and a file `conv.c`. The `conv.lsp` file has a byte-compiled version of the Lisp code in the wrapper file, but instructions to load the shared library. The `conv.c` file contains the glue to link the `C` function to the `XLISP-STAT` system. The `wrap:c-lines` and `wrap:c-function` are two macros in the `wrap` package that determine the structure of the `conv.c` file. Observe that `wrap:c-function` gives the Lisp symbol name given to the `C` function, and it explains the type of the arguments and the result. Also observe the handling of pointers in the `conv` Lisp function, using macros from the `wrapptrs` package. In order for everything to work you need to load the file `wrap.lsp`, which creates the `wrap` package, and to load the file `wrapptrs.lsp`, which loads the shared library `wrapptrs.dll` and creates the package `wrapptrs`. The `wrapptrs` package itself is already and example of applying `wrap:make-wrappers` to the file `wrapptr.wrp`.

Now link `conv.c` and `cconv.c` into a shared library `conv.dll`. In the link you should also include the `XLISP-STAT` interpreter, and whatever libraries from your development system needed. Loading `conv.lsp` into the `XLISP-STAT` interpreter makes the shared library (and thus the `C` function) available. And here is the result.

```
> (time (autocovar data 100))
The evaluation took 0.43 seconds; 0.20 seconds in gc.
```

We see an improvement compared to the raw Lisp with a factor of 5. Not as good as using the fft, but surprisingly good anyway. It seems that the Lisp `inner-product` function, which is a Lisp wrapper for `blas-ddot`, does introduce quite a bit of overhead.

4. Shared Libraries: Example 2

We can expect more gain in situations where the Lisp function we are replacing have many loops and list manipulations. In this example we take a function, written by Rick Schoenberg, to make contours on a scatterplot. `XLISP-STAT` has a `contour-function`, which draws the contours of a function of two variables. So in order to draw contours in a scatterplot, using n data-points (x_i, y_i, z_i), we need a smoother that interpolates the function and the use `contour-function` on this smoother. Schoenberg uses a low-pass Gaussian two-dimensional filter to do the interpolations. Here is the Lisp code.

```
(defun rsmooth-2d (i j x y z b1 b2)
  (let* ((w (/ (* (exp (/ (- (^ (- x i) 2)) (* 2 b1 b1)))
                  (exp (/ (- (^ (- y j) 2)) (* 2 b2 b2))))
             (* 2 pi b1 b2))))
  (/ (sum (* z w)) (sum w))
))

(defun rcontour (x y z
    &key levels (xnum 5) (ynum 5)
            (x1 (min x)) (x2 (max x)) (y1 (min y)) (y2 (max y))
    (b1 (/ (- x2 x1) (log (length z))))
    (b2 (/ (- y2 y1) (log (length z))))
    (smoother #'rsmooth-2d))
(contour-function #'(lambda (i j)
                    (funcall smoother i j x y z b1 b2))
          x1 x2 y1 y2 :num-points xnum :levels levels)
)
```

Clearly it is essential to make the smoother efficient. Here is a `C` version, in the file `cgauss.c`.

```
#include <math.h>
#define pi 3.141592653589793
#define SQUARE(x) ((x) * (x))

double gaussian_2_smooth (int n, double x0, double y0,
   double b1, double b2, double * x, double * y, double * z){
int i; double sum1 = 0.0, sum2 = 0.0,
  term1, term2, term3, weight;
for (i = 0 ; i < n ; i++) {
term1 = exp (- SQUARE(x0 - *(x + i)) / (2.0 * SQUARE(b1)));
term2 = exp (- SQUARE(y0 - *(y + i)) / (2.0 * SQUARE(b2)));
term3 = 2.0 * pi * b1 * b2;
weight = (term1 * term2) / term3;
sum1 += weight * *(z + i);
sum2 += weight;}
return (sum1 / sum2);}
```

The wrapper file `gauss.wrp` is

```
(wrap:c-lines "double gaussian_2_smooth
    (int, double, double, double, double, double *, double *, double *);")
(wrap:c-function rsmooth-2d-base "gaussian_2_smooth"
    (:integer :flonum :flonum :flonum :flonum
    (:cptr "double") (:cptr "double") (:cptr "double")) :flonum)

(defun rsmooth-2d (i j x y z b1 b2)
  (let ((n (length x))
        (x (coerce x '(vector c-double)))
        (y (coerce y '(vector c-double)))
        (z (coerce z '(vector c-double))))
  (rsmooth-2d-base n i j b1 b2
(wrapptrs:cast-c-double (array-data-address x))
(wrapptrs:cast-c-double (array-data-address y))
(wrapptrs:cast-c-double (array-data-address z)))
))

(defun rcontour (x y z
    &key levels (xnum 5) (ynum 5)
    (x1 (min x)) (x2 (max x)) (y1 (min y)) (y2 (max y))
    (b1 (/ (- x2 x1) (log (length z))))
    (b2 (/ (- y2 y1) (log (length z))))
    (smoother #'rsmooth-2d))
(contour-function #'(lambda (i j)
    (funcall smoother i j x y z b1 b2))
  x1 x2 y1 y2 :num-points xnum :levels levels)
)
```

If we link `gauss.c` and `cgauss.c`, we must make sure that code for the `exp` function is also linked in, by using some sort of `C` math library. After loading `gauss.lsp` we have

```
> (symbol-function 'rsmooth-2d-base)
#<Subr: #40b9888>
> (symbol-function 'rsmooth-2d)
#<Closure-RSMOOTH-2D: #40bbb18>
```

Now for the time comparison. The example

```
(def x (* (uniform-rand 1000) 10))
(def y (* (uniform-rand 1000) 10))
(def z (+ (* (- x 3) (- x 3)) (* (- y 5) (- y 5))))
(rcontour x y z :levels (iseq 20))
```

takes 1.68 seconds on the G3 in the Lisp version, and 0.18 seconds in the `C` version. Almost ten times faster, and in many applications, possibly much larger than this example, that can make a huge difference.

5. Shared Libraries: Example 3

In this example, we use `FORTRAN` to replace `C`. Given the enormous amount of legacy numerical code in `FORTRAN`, this is an important extension. Obviously using `FORTRAN` presupposes we have a development system that compiles and links both languages. We shall use the Absoft compilers, running in the MPW environment.

In `FORTRAN`, we can write external functions and subroutines. External functions return a value, so they seem especially appropriate, but we shall look at using subroutines as well. One important property of `FORTRAN` is that arguments to subroutines or functions are passed by reference and not by value. This means that if we call `FORTRAN` rotuines from `C`, we pass pointers to the parameters, but in the `FORTRAN` routine itself we calculate as if values were passed. This makes life just a tiny bit more complicated.

Again, we will use the same convolution example. The `FORTRAN` source, in the file `fconv.f`, is

```
        real*8 function cconv (l, n, x)
        integer*4 l, n
        real*8 sum, x(n)
        sum = 0.0
        do 10, i=1,n - l
10      sum=sum + x(i) * x(i + l)
        cconv = sum
        return
        end
```

As a wrapper file we use

```
(wrap:c-lines "double cconv (int *, int *, double *);")
(wrap:c-function base-conv "cconv"
   ((:cptr "int") (:cptr "int") (:cptr "double")) :flonum)

(defun conv (lag vec)
      (let ((n (coerce (list (length vec)) '(vector c-int)))
    (lag (coerce (list lag) '(vector c-int)))
    (vec (coerce vec '(vector c-double))))
      (base-conv (wrapptrs:cast-c-int (array-data-address lag))
 (wrapptrs:cast-c-int (array-data-address n))
 (wrapptrs:cast-c-double (array-data-address vec)))
    ))
```

This works exactly the same as the `C` version. But observe that it needs the hack, where scalars are converted to one-element vectors (which we need to do in order to use `array-data-address`).

The same result can be attained by using a `FORTRAN` subroutine. The code is

```
    subroutine cconv (l, n, x, sum)
    integer*4 l, n
    real*8 sum, x(n)
    sum = 0.0
    do 10, i=1,n - l
10  sum = sum + x(i) * x(i + l)
    return
    end
```

and the wrapper code is

```
(wrap:c-lines "void cconv (int *, int *, double *, double *);")
(wrap:c-function base-conv "cconv"
    ((:cptr "int") (:cptr "int")
     (:cptr "double") (:cptr "double")) :void)

(defun conv (lag vec)
  (let* ((n (coerce (list (length vec)) '(vector c-int)))
 (lag (coerce (list lag) '(vector c-int)))
 (vec (coerce vec '(vector c-double)))
 (sum (coerce (list 0.0) '(vector c-double))))
  (base-conv (wrapptrs:cast-c-int (array-data-address lag))
     (wrapptrs:cast-c-int (array-data-address n))
     (wrapptrs:cast-c-double (array-data-address vec))
     (wrapptrs:cast-c-double (array-data-address sum)))
(aref (pointer-protected (array-data-address sum)) 0)
))
```

Appendix A. Introduction

In the Appendices we give documentation and references necessary to use the various ports to XLISP-STAT we have made so far. Generally, each of these libraries defines a package, and we give documentation for the external symbols.

It is best to set up the packages using the new autoload system Tierney [1998a].

The documentation below is incomplete, because documentation strings have not been added to all external functions yet. The appendices will be updated regularly if more documentation (and more modules) are added. We plan to add modules for generalized eigenvalue and singular value computation for nonsymmetric matrices, for writing pdf, for producing pdf and ps plots. We also plan to add functions to the optimization, solving, and smoothing modules. But this may take a long time.

If you are interested in obtaining the wrappers and libraries for these modules, just drop me an email. Of course you can only use the compiled versions if you have a PowerMac of some sort. But the wrapper files, the `_autoidx.lsp` index files, and the code for the libraries will make it possible to compile your own versions.

Appendix B. cephes

The cephes library is written, in C, by Stephen Mosier. The material is discussed extensively in the book Mosier [1989]. The source code is on netlib, at

www.netlib.org/cephes/index.html

We have not used all of cephes, only the special function part. The documentation below is copied in many cases from the source code.

```
DRAND:
Args: (&optional (n 1))
Returns a typed vector with n uniform random numbers between
1.0 and 2.0 using the Wichman-Hill generator.

ZETA:
Args: (x)
                  inf.
                   -    -x
     zetac(x)  =   >   k   ,   x > 1,
                   -
                  k=2

   is related to the Riemann zeta function by

        Riemann zeta(x) = zetac(x) + 1.


PSI:
Args: (x)
                d      -
     psi(x)  =  -- ln | (x)
                dx

   is the logarithmic derivative of the gamma function.


DAWSON:
Args: x
                               x
                               -
                        2     | |        2
    dawsn(x)  =  exp( -x  )   |    exp( t  ) dt
                            | |
                             -
                             0
```

```


INVERSE-INCOMPLETE-BETA:
Args: (a b y)
   Given y, the function finds x such that
                    x
       -            -
      | (a+b)      | |  a-1     b-1
    -----------    |   t   (1-t)   dt = y.
     -     -     | |
    | (a) | (b)   -
                   0


INCOMPLETE-GAMMA:
Args: (a x &key (complement nil))
If complement is nil
                             x
                              -
                     1       | |  -t  a-1
    igam(a,x)  =   -----     |   e   t   dt.
                    -      | |
                   | (a)    -
                             0
else
    igamc(a,x)   =   1 - igam(a,x)

                              inf.
                                -
                       1       | |  -t  a-1
                 =   -----     |   e   t   dt.
                      -      | |
                     | (a)    -
                               x


FRESNEL:
Args (x)
             x
             -
            | |
   C(x) =   |   cos(pi/2 t**2) dt,
          | |
           -
            0
```

```

             x
             -
            | |
   S(x) =   |   sin(pi/2 t**2) dt.
          | |
           -
            0
Returns a typed vector with (s c).

COMPLEMENTARY-ERROR-FUNCTION:
NIL

RECIPROCAL-GAMMA:
Args: (x)
Returns one divided by the gamma function of the argument.

MODIFIED-BESSEL-THIRD-KIND:
Args: (x &key (exp nil) (order 0))
Modified Bessel functions of the third kind,
of order 0 or 1 or of integer order.
For the functions of order 0 or 1, one can choose to
use exponential scaling.

BESSEL:
Args: (x &key (order 0))
Bessel functions of order 0 or 1 or of integer or
non-integer order.

SPENCE:
Args: (x)
                      x
                      -
                     | | log t
   spence(x)  =  -   |   ----- dt
                   | |   t - 1
                    -
                    1


BESSEL-SECOND-KIND:
Args: (x &key (order 0))
Bessel functions of the second kind of order 0 or 1 or
of integer and non-integer order.

```

```
FAC:
Args (x)
Returns the factorial of (the integer) x.

GAUSS-HYPERGEOMETRIC-2F1:
Args: (a b c x)
    hyp2f1( a, b, c, x )  =   F ( a, b; c; x )
                             2 1

             inf.
              -   a(a+1)...(a+k) b(b+1)...(b+k)   k+1
     =  1 +   >   -----------------------------  x   .
              -         c(c+1)...(c+k) (k+1)!
            k = 0


MODIFIED-BESSEL:
Args: (x &key (exp nil) (order 0))
Modified Bessel functions of order 0 or 1 or of non-integer order.
For the functions of order 0 or 1, one can choose to
use exponential scaling.

INCOMPLETE-BETA:
Args: (a b x)
                    x
       -            -
      | (a+b)      | |  a-1     b-1
    -----------    |   t   (1-t)   dt.
     -     -     | |
    | (a) | (b)   -
                   0


COMPLETE-ELLIPTIC-FIRST-KIND:
Args: (m)
              pi/2
               -
              | |
              |           dt
   K(m)  =    |    ------------------
              |                   2
            | |    sqrt( 1 - m sin t )
             -
              0

```

```

ELLIPTIC-FIRST-KIND:
Args: (phi m)
                  phi
                   -
                  | |
                  |           dt
   F(phi_m)  =    |    ------------------
                  |                   2
                | |    sqrt( 1 - m sin t )
                 -
                 0


STRUVE:
Args (v x)
Computes the Struve function Hv(x) of order v, argument x.

JACOBIAN-ELLIPTIC:
Args (u m)
Evaluates the Jacobian elliptic functions sn(u|m), cn(u|m),
and dn(u|m) of parameter m between 0 and 1, and real
argument u. These functions are periodic, with quarter-period on the
real axis equal to the complete elliptic integral
ellpk(1.0-m).  Relation to incomplete elliptic integral:
If u = ellik(phi,m), then sn(u|m) = sin(phi),
and cn(u|m) = cos(phi).  Phi is called the amplitude of u.
Returns a typed vector with (sn cn dn phi).

COMPLETE-ELLIPTIC-SECOND-KIND:
Args: (m1)
              pi/2
               -
              | |               2
   E(m)  =    |    sqrt( 1 - m sin t ) dt
            | |
             -
              0


SINE-COSINE-INTEGRALS:
Args: (x &key (hyperbolic nil))
   Approximates the integrals

                            x
```

```
                            -
                           |  cos t - 1
     Ci(x) = eul + ln x +  |  --------- dt,
                           |      t
                          -
                          0
               x
               -
              |  sin t
     Si(x) =  |  ----- dt
              |    t
             -
              0

   where eul = 0.57721566490153286061 is Euler's constant. If
   HYPERBOLIC is t, then hyperbolic sines and cosines are used.
   Returns results in a typed vector (c s).


ERROR-FUNCTION:
Args: (x)
                             x
                              -
                   2         | |          2
     erf(x)  =  --------      |    exp( - t  ) dt.
                sqrt(pi)   | |
                            -
                            0


CONFLUENT-HYPERGEOMETRIC-1F1:
Args: (a b x)
                            1            2
                         a x    a(a+1) x
     F ( a,b;x )  =  1 + ---- + --------- + ...
    1 1                  b 1!   b(b+1) 2!


INVERSE-COMPLEMENTED-INCOMPLETE-GAMMA:
Args: (a p)
   Given p, the function finds x such that

                              inf.
                                -
                       1       | |  -t  a-1
```

```
             p   =   -----     |   e   t   dt.
                      -      | |
                     | (a)    -
                               x


GAMMA:
Args: (x)
Returns the gamma function of x.

EXPONENTIAL-INTEGRAL:
Args: (n x)
                   inf.
                     -
                    | |   -xt
                    |    e
        E (x)  =    |    ----  dt.
         n          |      n
                  | |     t
                   -
                    1


AIRY:
Args: x
Solves the differential equation y''(x)=xy. The two independent
solutions a and b, and their derivatives a' and b', at x are returned
in the typed vector (a a' b b').
```

Appendix C. GD

```
GDIMAGESETBRUSH:
Args: (gd gd_brush)
GD and GD_BRUSH are gdImagePtrs. The image in GD_BRUSH is
used as a brush in the image in GD. Returns NIL.

GDIMAGEGETINTERLACED:
Args: (gd)
GD is a gdImagePtr, the function returns T if the
image is interlaced and NIL if it is not.

GDIMAGECOLORTRANSPARENT:
Args: (gd color)
Sets the index of the transparent color in the image
pointed to by GD to COLOR. If there are no transparent
colors, call this function with COLOR = -1. Returns NIL.

GDIMAGEPOLYGON:
Args: (gd x y color)
GD is a gdImagePtr, and COLOR is an integer corrsponding to
one of the colors of the image. X and Y are lists with the
coordinates of the vertices of the polygon. A polygon is drawn
in the image. Returns NIL.

GDIMAGEINTERLACE:
Args: (gd interlace)
If INTERLACE is T, the image pointed to by GD will be interlaced, if
INTERLACE is NIL it will not. Returns NIL.

GDIMAGECOLORDEALLOCATE:
Args: (gd color)
Deallocates the color indexed by COLOR in the image
pointed to by GD. Returns NIL.

GDIMAGEDASHEDLINE:
Args: (gd start_x start_y end_x end_y color)
Draws a dashed line from (START_X,START_Y) to (END_X,END_Y) in the image pointed
to by GD. Deprecated. Use gdImageSetStyle instead. Returns NIL.

GDIMAGECHAR:
Args: (gd gf x y char color)
GD is a gdImagePtr, and COLOR is an integer corrsponding to
one of the colors of the image. X and Y are the starting
coordinates of the character CHAR which is drawn horizontally
```

```
from left to right in size GF. Returns NIL.

GDIMAGECOLOREXACT:
Args: (gd r g b)
Returns the index of the allocated color in the image
pointed to by GD that has RGB-values R, G, and B. Returns
-1 if there is no such color.

GDIMAGECREATEFROMGIF:
Args: (filename)
Reads GIF file from FILENAME, and returns
a gdImagePtr to an image. Returns NIL.

GDIMAGECOPYRESIZED:
Args: (gd_dst gd_src dst_x dst_y src_upper_left_x src_upper_left_y dst_width dst_h
Copies and possibly resizes a rectangular region from the image pointed to by GD_S
by GD_DST. The region copied has the upper left corner (SRC_UPPER_LEFT_X,SRC_UPPER
and width SRC_WIDTH and height SRC_HEIGHT. The region is copied to the point (DST_
width DST_WIDTH and height DST_HEIGHT. Returns NIL.

GDIMAGECOPY:
Args: (gd_dst gd_src dst_x dst_y src_upper_left_x src_upper_left_y width height)
Copies a rectangular region from the image pointed to by GD_SRC to the image point
by GD_DST. The region copied has the upper left corner (SRC_UPPER_LEFT_X,SRC_UPPER
and width WIDTH and height HEIGHT. The region is copied to the point (DST_X,DST_Y)

GDIMAGESETSTYLE:
Args: (gd style)
GD is a gdImagePtr and STYLE is a list of allocated colors of the
image. Defines a style color for dashed lines. Returns NIL.

GDIMAGEFILL:
Args: (gd start_x start_y color)
Floods a portion of the image pointed to by GD with COLOR. The
portion flooded is the surrounding region of the point (START_X,START_Y)
with the same color as the starting point. Returns NIL.

GDIMAGEDESTROY:
Args: (gd)
Destroys the image pointed to by GD. Returns NIL.

GDIMAGECOLORSTOTAL:
Args: (gd)
GD is a gdImagePtr, the function returns the number of
currently allocated colors in the image.
```

```

GDIMAGESTRING:
Args: (gd gf x y string color)
GD is a gdImagePtr, and COLOR is an integer corrsponding to
one of the colors of the image. X and Y are the starting
coordinates of the STRING which is drawn horizontally
from left to right in characters of size GF. Returns NIL.

GDIMAGEGETPIXEL:
Args: (gd row col)
Returns the color value of the pixel in ROW and COL
of the image pointed to by GD.

GDIMAGECOLORALLOCATE:
Args: (gd r g b)
Allocates a color in the image pointed to by GD, with
RGB-values R, G, and B. Returns the color index.

GDIMAGEBLUE:
Args: (gd color)
GD is a gdImagePtr, and COLOR is one of its allocated colors.
The function returns the blue component of the color.

GDIMAGEGREEN:
Args: (gd color)
GD is a gdImagePtr, and COLOR is one of its allocated colors.
The function returns the green component of the color.

GDIMAGESETTILE:
Args: (gd gd_tile)
GD and GD_TILE are gdImagePtrs. The image in GD_TILE is
used as a tile in the image in GD. Returns NIL.

GDIMAGERECTANGLE:
Args: (gd upper_left_x upper_left_y lower_right_x lower_right_y color)
Draws a rectangle in color COLOR with upper left corner at (UPPER_LEFT_X,UPPER_LEF
and lower right corner at (LOWER_RIGHT_X,LOWER_RIGHT_Y) in the image pointed to by

GDIMAGERED:
Args: (gd color)
GD is a gdImagePtr, and COLOR is one of its allocated colors.
The function returns the red component of the color.

GDIMAGEFILLTOBORDER:
Args: (gd start_x start_y color)
```

```
Floods a portion of the image pointed to by GD with FLLOD_COLOR. The
portion flooded begins at the point (START_X,START_Y)and stops at border
with color BORDER_COLOR. Returns NIL.

GDIMAGECOLORCLOSEST:
Args: (gd r g b)
Returns the index of the allocated color in the image
pointed to by GD that is closest to the color with
RGB-values R, G, and B.

GDIMAGECREATE:
Args: (nrow ncol)
Returns a gdImagePtr to an (empty) image with a height of NROW pixels
and a width of NCOL pixels.

GDIMAGEGD:
Args: (gd filename)
GD is a gdImagePtr. The corresponding image is written
in GD format to the file FILENAME. Returns NIL.

GDIMAGEARC:
Args: (gd start_x start_y end_x end_y color)
Draws a segment of an ellips in color COLOR centered at (CENTER_X,CENTER_Y),
of width WIDTH and height HEIGHT, starting at BEGIN_DEGREE and ending at
END_DEGREE in the image pointed to by GD. Returns NIL.

GDIMAGELINE:
Args: (gd start_x start_y end_x end_y color)
Draws a line from (START_X,START_Y) to (END_X,END_Y)
in the image pointed to by GD. Returns NIL.

GDIMAGESX:
Args: (gd)
GD is a gdImagePtr, the function returns the width of the
image in pixels.

GDIMAGESY:
Args: (gd)
GD is a gdImagePtr, the function returns the height of the
image in pixels.

GDIMAGESTRINGUP:
Args: (gd gf x y string color)
GD is a gdImagePtr, and COLOR is an integer corrsponding to
one of the colors of the image. X and Y are the starting
```

```
coordinates of the STRING which is drawn vertically
from bottom to top in characters of size GF. Returns NIL.

GDIMAGEGETTRANSPARENT:
Args: (gd)
GD is a gdImagePtr, the function returns the current
transparent color of the image.

GDIMAGESETPIXEL:
Args: (gd row col)
Sets the color value of the pixel in ROW and COL
of the image pointed to by GD. Returns NIL.

GDIMAGECREATEFROMGD:
Args: (filename)
Reads GD file from FILENAME, and returns
a gdImagePtr to an image.

GDIMAGECHARUP:
Args: (gd gf x y char color)
GD is a gdImagePtr, and COLOR is an integer corrsponding to
one of the colors of the image. X and Y are the starting
coordinates of the character CHAR which is drawn vertically
from bottom to top in size GF. Returns NIL.

GDIMAGEGIF:
Args: (gd filename)
GD is a gdImagePtr. The corresponding image is written
in GIF format to the file FILENAME. Returns NIL.

GDIMAGEFILLEDRECTANGLE:
Args: (gd upper_left_x upper_left_y lower_right_x lower_right_y color)
Draws a filled rectangle in color COLOR with upper left corner at (UPPER_LEFT_X,UP
and lower right corner at (LOWER_RIGHT_X,LOWER_RIGHT_Y) in the image pointed to by

GDIMAGECREATEFROMXBM:
Args: (filename)
Reads XBM file from FILENAME, and returns
a gdImagePtr to an image.

GDIMAGEFILLEDPOLYGON:
Args: (gd x y color)
GD is a gdImagePtr, and COLOR is an integer corrsponding to
one of the colors of the image. X and Y are lists with the
coordinates of the vertices of the polygon. A filled polygon
```

```
is drawn in the image. Returns NIL.
```

Appendix D. pppack

```
REINSCH-SMOOTHING-SPLINE:
Args: (x y &key (dy (repeat 1.0 (length x))) (s (float (length x))))

PIECEWISE-POLYNOMIAL-VALUE:
NIL

PIECEWISE-POLYNOMIAL-FROM-B-SPLINE:
NIL

ALL-B-SPLINE-VALUES:
Args: (knot jhigh x left)
Computes the values of all non-zero B-splines with knots KNOT of order JHIGH at X.
Here LEFT is the index such that KNOT[LEFT] < X < KNOT[LEFT+1], and KNOT has
length LEFT + JHIGH. We must have JHIGH <= 20.

CUBIC-SPLINE-INTERPOLANT-VALUE:
NIL

B-SPLINE-VALUE:
NIL

CUBIC-SPLINE-INTERPOLANT:
Args: (x y jderiv &key (plot t) (min (min x)) (max (max x)) (numpoints 100))
Plots the JDERIV-th derivative of the cubic interpolating spline through the scatt
sequences x and y.

B-SPLINE:
Args: (knot bcoef jderiv &key (min (min knot)) (max (max knot)) (numpoints 100))
Plots the JDERIV-th derivative of the B-spline with (n + k) knots KNOT and n coeff
Note: k = length (KNOT) - length (BCOEF) is the order of the spline. KNOT is suppo
be nondecreasing.
```

Appendix E. specfun

The specfun library is another collection of special functions, which has a great deal of overlap with cephes. The code was written, in `FORTRAN`, by W.J. Cody. The source code is on netlib, at

`www.netlib.org/specfun/index.html`

There is also a version in

`www.netlib.org/toms/715`

which corresponds with Cody [1993].

Of special interest, perhaps, is the function machar, which dynamically computes machine parameters Cody [1988]. If called from `XLISP-STAT`, we obtain

```
ibeta       2
it          53
irnd        5
ngrd        0
machep      -52
negeps      -53
iexp        11
minexp      -1022
maxexp      1024
eps         2.220446049250313E-16
epsneg      1.1102230246251565E-16
xmin        2.2250738585072014E-308
xmax        1.7976931348623157E+308
```

```
BESY0:
NIL

BESY1:
NIL

RIBESL:
NIL

EXPEI:
NIL

BESJ0:
NIL

DLGAMA:
NIL

BESJ1:
```

```
NIL

BESK0:
NIL

BESK1:
NIL

BESI0:
NIL

PSI:
NIL

BESI1:
NIL

DERFC:
NIL

DAW:
NIL

DGAMMA:
NIL

MACHAR:
NIL

BESEK0:
NIL

BESEK1:
NIL

BESEI0:
NIL

BESEI1:
NIL

RYBESL:
NIL

RJBESL:
```

```
NIL

RKBESL:
NIL

EONE:
NIL

ANORM:
NIL

DERF:
NIL

DERFCX:
NIL

EI:
NIL
```

Appendix F. probability

```
NONCENTRAL-CHISQ-CDF
Args: (x dfr pnonc)
Returns the value of the  Noncentral ChiSquare (DFR,PNONC) distribution
function at X.


NONCENTRAL-CHISQ-QUANT
Args (p dfr pnonc)
Returns the P-th quantile of the Noncentral ChiSquare (DFR,PNONC) distribution.


NONCENTRAL-F-CDF
Args: (x dfr1 dfr2 pnonc)
Returns the value of the  Noncentral F (DFR1, DFR2, PNONC) distribution
function at X.


NONCENTRAL-F-QUANT
Args (p dfr1 dfr2 pnonc)
Returns the P-th quantile of the Noncentral F (DFR1, DFR2, PNONC) distribution.


NONCENTRAL-T-CDF
Args: (x dfr pnonc)
Returns the value of the Noncentral t (DFR,PNONC) distribution
function at X.


NONCENTRAL-T-QUANT
Args (p dfr pnonc)
Returns the P-th quantile of the Noncentral t (DFR,PNONC) distribution.
```

Appendix G. smoothing

```
CUBIC-SPLINE-DATA-SMOOTHER:
Args: (x y &key (d (repeat 1.0 (length x))) (var -1.0) (job 0) (plot t))
Interface to Hutchinson's cubgcv cubic spline smoother. X has N abscissae,
Y has N ordinates. D are the relative standard deviations. If unknown, set
D = 1.0. If known, then set VAR = 1. If VAR < 0 then generalized cross-validation
is used to estimate the smoothing parameter, and VAR returns the error variance.
If VAR > 0 then the smoothing parameter is estimated by estimating the MSE and
VAR is unchanged. If VAR = 0 an interpolating cubic spline is calculated.
If JOB = 0 standard errors are not computed, if JOB =1 they are computed.
If PLOT is non-zero, the resulting smoother is plotted.

LOCAL-POLYNOMIAL-RIDGE-REGRESSION:
NIL

KERNEL-REGRESSION-LOCAL-BANDWIDTH:
NIL

KERNEL-REGRESSION-GLOBAL-BANDWIDTH:
NIL
```

Appendix H. solving

```
CPOLY
Args: (coefs)
Computes the roots of a polynomial with complex coefficients COEFS.
```

Appendix I. Optimization

```
QUADRATIC-PROGRAM
Args: (dmat dvec amat bvec meq &key (ierr 0))
This routine uses the Goldfarb/Idnani algorithm to solve the
following minimization problem:

      minimize  -d^T x + 1/2 *  x^T D x
        where   A1^T x  = b1
                A2^T x >= b2

the matrix D is in DMAT, assumed to be symmetric and positive definite
and of order n. Vector d is in DVEC. Matrix A, containing both A1 and
A2 is in the n x q array AMAT, b is in the q-vector BVEC. MEQ indicates
how many of the q constraints are equality constraints. The program
returns a list with the optimum value, the number of iterations, the
optimum solution, and the vector indicating which constraints are active.
FORTRAN by Berwin A. Turlach <bturlach@stats.adelaide.edu.au>.
```

Appendix J. transform

```
FCT2D:
NIL

IFCT2D:
NIL

FCT:
NIL

IFCT:
NIL
```

Appendix K. density

```
LSCV-KD-CRITERION:
Args: (data bw)
DATA is some 1-d sequence, BW is the bandwidth.
Computes the least squares cross validation criterion
to be minimized in bandwidth selection for kernel density estimation

KERNEL-DENS-PLUGIN:
Args: (x &key (z (rseq (min x) (max x) 50)) (plot t) (kerndens t))


SHEATHER-JONES-SEQ-BANDWIDTH:
Args: (data)
Return sheather-jones solve-the-equation bandwidth for kernel density estimation
(Journal of the Royal Statistical Society, Series B, 1991, Vol. 53, pp 683-690).
To be used with gaussian kernel. Data size must be less than 1600.

WARPED-HISTOGRAM:
Args: (x h m &key (k 3) (plot t))
X is a sequence of data. H is the bandwidth, and M is the number of
small bins in a large bin. K is the kernel (1: uniform, 2: triangle,
3: epanechnikov, 4: quartic, 5:triweight, and PLOT indicates if a plot
should be made. KERNDENS adds a dashed kernel-density plot.
```

References

E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorenson. *LAPACK User's Guide*. SIAM, 1992.

W. J. Cody. Algorithm 665: MACHAR: A subroutine to dynamically determine machine parameters. *ACM Transactions on Mathematical Software*, 14(4):303–311, December 1988.

W. J. Cody. Algorithm 715: SPECFUN: A portable FORTRAN package of special function routines and test drivers. *ACM Transactions on Mathematical Software*, 19(1):22–32, March 1993. See remark Price [1996].

S. Mosier. *Methods and Programs for Mathematical Functions*. Prentice-Hall, 1989.

H. Joseph Newton. *TIMESLAB: A Time Series Analysis Laboratory.* Wadsworth & Brooks/Cole, 1988.

W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.R. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1988.

David T. Price. Remark on Algorithm 715. *ACM Transactions on Mathematical Software*, 22(2):258–258, June 1996. See Cody [1993].

L. Tierney. *LISP-STAT. An Object-Oriented Environment for Statiatical Computing and Dynamic Graphics*. Wiley, 1990.

L. Tierney. A New Autoload System for XLISP-STAT. Technical report, Department of Statistics, University of Minnesota, 1998a.

L. Tierney. Dynamic loading basic. Technical report, Department of Statistics, University of Minnesota, 1998b.

L. Tierney. Interfacing to C code. Technical report, Department of Statistics, University of Minnesota, 1998c.

L. Tierney. Native pointers in XLISP-STAT. Technical report, Department of Statistics, University of Minnesota, 1998d.

L. Tierney. Regular Expressions in XLISP-STAT. Technical report, Department of Statistics, University of Minnesota, 1998e.

L. Tierney. Shared Libraries for XLISP-STAT. Technical report, Department of Statistics, University of Minnesota, 1998f.

L. Tierney. Sockets. Technical report, Department of Statistics, University of Minnesota, 1998g.

UCLA Statistics Department
E-mail address: deleeuw@stat.ucla.edu