MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A. I. Memo No. 774

27 September 1984

Some Scientific Subroutines in LISP

Gerald Roylance

Abstract. Here's a LISP library of mathematical functions that calculate hyperbolic and inverse hyperbolic functions, Bessel functions, elliptic integrals, the gamma and beta functions, and the incomplete gamma and beta functions. There are probability density functions, cumulative distributions, and random number generators for the normal, Poisson, chi-square, Student's T, and Snedecor's F functions. Multiple linear regression, Fletcher-Powell unconstrained minimization, numerical integration, root finding, and convergence. Code to factor numbers and to do the Solovay-Strassen probabilistic prime test.

Acknowledgements. This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contracts N00014-80-C-0505 and N00014-80-C-0622.

© Massachusetts Institute of Technology 1984

1. Introduction

Several times in science and engineering we need to evaluate rather exotic functions -- elliptic integrals, Bessel functions, and probability density functions, for example. Here's where you can get your hands on some LISP code that will evaluate some of these functions. The code runs in both MACLISP and LISPM LISP.

The functions in the library are scattered in several different files. The descriptions below will describe the function name, how it should be used, and the file where that function can be obtained. For example, the function FACTOR is in PS:<GLR.FUNCT>MOD.LSP. Before it can be used, the code must first be loaded by doing one of:

```
(LOAD    "PS:<GLR.FUNCT>MOD.FASL")      ; For MACLISP
(LOAD "OZ:PS:<GLR.FUNCT>MOD.QFASL")     ; For MIT LISPM
(LOAD "OZ:PS:<GLR.FUNCT>MOD.QBIN")      ; For Symbolics LM2
(LOAD "OZ:PS:<GLR.FUNCT>MOD.BIN")       ; For Symbolics 3600
```

After the file has been loaded, just call the function with the right arguments:

```
(FACTOR 12345678987654321) ==> (3 3 3 3 37 37 333667 333667)
```

If you have to compile one of the library files or you want to read the source version of a library file into your LISP, then you will have to load a special macro called IMPORT-FILE that is defined in the file PS:<GLR.LISP>IMPORT.LSP. You may want to use the IMPORT-FILE macro yourself. Its format is:

```
(IMPORT-FILE "OZ:PS:<GLR.FUNCT>MOD.LSP")
```

The argument is the source filename with a LISPM style hostname (the MACLISP version of the macro knows to ignore the host). The behavior of the macro depends on whether it is being compiled, evaluated, or loaded. If the IMPORT-FILE form is evaluated or loaded (EVAL-WHEN (EVAL LOAD) ...) and the imported file has not already been loaded, it will load a binary file if it can find one or the source file if it cannot. At compile time (EVAL-WHEN (COMPILE) ...), the IMPORT-FILE form turns into

```
(PROGN 'COMPILE <first form in source file> <load code>)
```

By convention, the first form in the imported source file should be a DECLARE of all the external functions and variables that the imported file defines. Including the declarations allows some compile time error checking and eliminates spurious warnings about undefined functions. The <load code> conspires to load the binary or source file when when the file we are currently compiling gets loaded.

2. Number Theory and Combinatorial Functions

(FACTORIAL N) PS:<GLR.FUNCT>COMBIN.LSP

The number of ways to permute N objects.

(CHOOSE N M) PS:<GLR.FUNCT>COMBIN.LSP

The number of ways to place M distinguishable objects on N distinguishable plates.

(BELL-NUMBER N) PS:<GLR.FUNCT>COMBIN.LSP

The number of ways to place N distinguishable objects on N indistinguishable plates.

(CATALAN-NUMBER N) PS:<GLR.FUNCT>COMBIN.LSP

The number of ways to fully parenthesize a string of n symbols.

(FIB N) PS:<GLR.FUNCT>FIB.LSP

The Nth (FIXNUM) Fibonacci number. FIB(0)=0, FIB(1)=1, ... Uses a log(N) algorithm.

(FACTOR N) PS:<GLR.FUNCT>MOD.LSP

Find the prime factors of the integer N (FIXNUM or BIGNUM). Returns a sorted list of the factors.

(PRIME-TEST N &optional (TRIALS 50.)) PS:<GLR.FUNCT>MOD.LSP

Tests the primality of N using a probabilistic algorithm (see <Solovay>). Returns NIL, PRIME, or PROBABLY-PRIME ($P\{error\}=2^{-TRIALS}$). TRIALS must be FIXNUM.

(SMALLER-PRIME N) PS:<GLR.FUNCT>MOD.LSP

Finds the first prime that is smaller than N. Repeatedly calls PRIME-TEST.

(JACOBI-SYMBOL P Q) PS:<GLR.FUNCT>MOD.LSP

Computes the Jacobi Symbol of P and Q.

(TOTIENT N) PS:<GLR.FUNCT>MOD.LSP

Computes Euler's totient function of N. Calls FACTOR as a subroutine.

3. Discrete Fourier Transform

These functions are used to compute discrete fourier transforms <Oppenheimer>. The inputs are two FLONUM arrays that represent the real and imaginary parts of the input. The algorithms require input arrays for the transform to be bit reversed; there are functions for doing the reversal. The functions also require some initialization. The length of the DFT must be a power of 2.

$$X[k] = \Sigma_{m=0}^{N-1} \exp(-j\,2\,\pi\,k\,m\,/\,N)$$
$$x[k] = (1/N)\,\Sigma_{m=0}^{N-1} \exp(\,j\,2\,\pi\,k\,m\,/\,N)$$

(DFT-INIT LOGN) PS:<GLR.FUNCT>DFT.LSP

DFT-INIT returns a structure containing some tables that are needed by the DFT algorithm. This structure should be saved away and given to the DFT routine each time it is called. The structure need only be computed once for each size DFT. The length of the DFT is 2^{LOGN}.

(DFT-FORWARD X-REAL X-IMAG TABLES) PS:<GLR.FUNCT>DFT.LSP

This function does a discrete Fourier transform. The results are written back into X-REAL and X-IMAG (the previous contents are lost) and are in sequential order (ie -- not bit reversed). TABLES is as returned by DFT-INIT.

(DFT-REVERSE X-REAL X-IMAG TABLES) PS:<GLR.FUNCT>DFT.LSP

This is just like DFT-FORWARD, but it does the inverse transform. The input X arrays should be in sequential (not bit-reversed) order. The resulting arrays are also in sequential order.

(DFT-REVERSE-ARRAY ARRAY TABLES) PS:<GLR.FUNCT>DFT.LSP

This bit reverses the first 2^{LOGN} elements of ARRAY.

(DFT-610 X-REAL X-IMAG TABLES) PS:<GLR.FUNCT>DFT.LSP

This function does a forward transform. X-REAL and X-IMAG are bit-reversed input arrays (to reverse them, use DFT-REVERSE-ARRAY). The results are returned in X-REAL and X-IMAG in sequential order.

(DFT-618 X-REAL X-IMAG TABLES) PS:<GLR.FUNCT>DFT.LSP

The reverse transform. Input arrays are sequential order, output arrays are bit-reversed. Use DFT-REVERSE-ARRAY to put them in sequential order.

4. Trigonometric and other Functions

These approximations come from <Abramowitz>, <DEC>, and <Hastings>.

(EXP10 X) PS:<GLR.FUNCT>EXTFCN.LSP

Computes 10↑X.

(LOG10 X) PS:<GLR.FUNCT>EXTFCN.LSP

Log base 10 of X.

(TAN X)
(SEC X)
(CSC X)
(COT X)
(ASIN X)
(ACOS X)
(ASEC X)
(ACSC X)
(ACOT Y X) PS:<GLR.FUNCT>HYPER.LSP

Various trigonometric functions.

(ERROR-FUNCTION X) PS:<GLR.FUNCT>EXTFCN.LSP

Error function. eps < 1.5E-7.

$$(2/\mathrm{sqrt}(\pi)) \int_0^x \exp(-t^2)\, dt$$

(BESSEL-I N X) PS:<GLR.FUNCT>BESSEL.LSP

Modified Bessel function for integer order N. eps < 1.6E-7

(BESSEL-J N X) PS:<GLR.FUNCT>BESSEL.LSP

Bessel function for integer order N. eps < 5E-8.

(GAMMA-FUNCTION A) PS:<GLR.FUNCT>GAMMA.LSP

Gamma function.

$$\Gamma(a) = \int_0^\infty t^{a-1} e^{-t}\, dt$$

(GAMMA-FUNCTION-INCOMPLETE A X) PS:<GLR.FUNCT>GAMMA.LSP

Incomplete Gamma function. 0 <= x <= inf, but breaks if X is large because of roundoff error.

$$\Gamma(a, x) = \int_0^x t^{a-1} e^{-t}\, dt$$

(BETA-FUNCTION A B) PS:<GLR.FUNCT>GAMMA.LSP

Beta function. A and B are FLONUM.

$$\beta(a, b) = \int_0^1 t^{a-1} (1-t)^{b-1}\, dt,\ a>0, b>0$$

(BETA-FUNCTION-INCOMPLETE A B X) PS:<GLR.FUNCT>GAMMA.LSP

Incomplete Beta function. A and B are FLONUM, but one of A or B must be an integer (eg, 3.0).

$$\beta(a, b, x) = \int_0^x t^{a-1} (1-t)^{b-1}\, dt,\ 0<a, 0<b, 0<=x<=1$$

(ELLIPTIC-INTEGRAL-K M) PS:<GLR.FUNCT>ELLIP.LSP

Elliptic integral of the first kind, k(M). 0 <= M < 1. eps = 2.0E-8.

$$\int_0^{\pi/2} (1 - m \sin^2(\theta))^{-0.5} \, d\theta$$

(ELLIPTIC-INTEGRAL-KC M) PS:<GLR.FUNCT>ELLIP.LSP

Complementary elliptic integral of the first kind, k'(M). k'(M) = k(1-M).

(ELLIPTIC-INTEGRAL-E M) PS:<GLR.FUNCT>ELLIP.LSP

Elliptic integral of the second kind, e(M). eps < 2.0E-8.

$$\int_0^{\pi/2} \mathrm{sqrt}(1 - m \sin^2(\theta)) \, d\theta$$

(ELLIPTIC-INTEGRAL-EC M) PS:<GLR.FUNCT>ELLIP.LSP

Complementary elliptic integral of the second kind, e'(M). e'(M) = e(1-M).

(ELLIPTIC-SINE u M) PS:<GLR.FUNCT>ELLIP.LSP

Elliptic sine function (SN(u,M))

(ELLIPTIC-COSINE u M)) PS:<GLR.FUNCT>ELLIP.LSP

Elliptic cosine function (CN(u,M))

5. Linear Regression

Say we are trying to fit some (x, y) data to a function that looks like:

```
Y'(x) = a0* 1 + a1*g1(x) + a2*g2(x) + ... + an*gn(x)
```

where the a[i] are constants that we are trying to determine, the g[i] are linearly independent functions that we specify, and Y'(x) is the value of the fitted equation. Such a fit is called a linear regression and here are some functions that will do it.

(FIT FCTN X Y N &optional (MODE 0)) PS:<GLR.FUNCT>FIT.LSP

Let X be a one dimensional array of x values (the x values could themselves be vectors). Y is a one dimensional array of FLONUM values for the corresponding X input. N is the number of terms in the equation we are fitting (ie, n in the equation above). FCTN is a function of N+1 arguments that evaluates the equation above. That is FCTN looks like:

```
(LAMBDA (X A0 A1 A2 A3 ... AN)
  ... )
```

and calculates the above equation. FIT returns a list of the coefficients Ai as the CDR of its value.

FIT actually uses the procedure REGRESS in PS:<GLR.FUNCT>REGRES.LSP, which is modeled after <Bevington>. If you want to do something a little more complicated, then ask me about

that procedure.

6. Functional Minimization

(FMFP FUNCT N X G EST EPS LIMIT)

PS:<GLR.FUNCT>FMFP.LSP

Fletcher-Powell's functional minimization procedure (adapted from <Kuester> and <Luenberger>). This finds a minimum of a multivariate, unconstrained, nonlinear function. That is, find the vector X such that F(X) is a local minimum. The arguments to this function are:

```
N       number of independent variables
X       vector[N] of initial variable values
        contains the result vector on exit
G       vector[N] in which to store the gradient
EST     estimate of minimum value of objective function
EPS     test value representing the expected absolute error in movement
LIMIT   maximum number of iterations
FUNCT   user supplied objective function that computes F and the gradient
        (FUNCT N X G) returns F(X) and
        also fills in the vector[N] G with the gradient
```

(MARQUARDT N K X Y Z FUNC DERIV B BMIN BMAX BV)

PS:<GLR.FUNCT>MARQ.LSP

Marquardt's parameter fitting procedure (adapted from <Kuester>). Given N data points (X[i], Y[i]) and a function F with K parameters B[j], find the K parameters B[j] that best fit the data. The fitted values are returned in array Z. The arguments are:

```
N     -- number of data points (FIXNUM)
K     -- number of unknowns    (FIXNUM)
B     -- vector[K] of (FLONUM) unknowns
BMIN -- vector[K] of (FLONUM) minimum values of B[j]
BMAX -- vector[K] of (FLONUM) maximum values of B[j]
X     -- vector[N] independent variable data points
         (this vector might be a vector of vectors)
Y     -- vector[N] of (FLONUM) dependent variable
Z     -- vector[N] of (FLONUM) computed values of dependent variable
BV    -- vector[K] of (FLONUM) codes
           1.0 -> numerical derivatives
           0.0 -> do not change this unknown
FUNC -- (FUNC X K B N Z ZOFF) computes the function F (using the
         K parameters in vector B) for each of the X[0] ... X[N-1]
         and puts the respective results into Z[ZOFF+0] ... Z[ZOFF+N-1]
DERIV-- (not used)
```

7. Hyperbolic & Inverse Hyperbolic Functions

This functions use the Common Lisp <Steele> names, but are only defined for real arguments. They call EXP to compute their results.

(COSH X)
(SINH X)
(TANH X)
(COTH X)
(SECH X)
(CSCH X) PS:<GLR.FUNCT>HYPER.LSP

Hyperbolic functions. FLONUM argument and FLONUM result.

(ACOSH X)
(ASINH X)
(ATANH X)
(ACOTH X)
(ASECH X)
(ACSCH X) PS:<GLR.FUNCT>HYPER.LSP

Inverse hyperbolic functions. FLONUM argument and FLONUM result.

8. Numerical Integration

(INTEGRATE-TRAPEZOIDAL F X0 X1 N) PS:<GLR.FUNCT>INTEGR.LSP

Uses the trapezoidal rule to integrate the function F (of one FLONUM argument) from X0 to X1 with N (FIXNUM) iterations.

(INTEGRATE-SIMPSON F X0 X1 N) PS:<GLR.FUNCT>INTEGR.LSP

Uses Simpson's rule to integrate the function F (of one FLONUM argument) from X0 to X1 with N (FIXNUM) iterations.

(INTEGRATE-EULER F X0 Y0 H X1) PS:<GLR.FUNCT>RUNGE.LSP

Uses the forward Euler method to integrate F(x, y) from position (X0, Y0) to a position (X1, Y1) using a step size of H. Y1 is the value of INTEGRATE-EULER. F is a function of X and Y (FLONUM) and should return DY/DX for the given position.

(INTEGRATE-RUNGE-KUTTA F X0 Y0 H X1) PS:<GLR.FUNCT>RUNGE.LSP

This is just like EULER except it uses a fourth order RUNGE-KUTTA method instead of Euler's method.

(INTEGRATE-MULTI-STEP-H F X0 Y0 H X1
&optional (CD 1.0E-8) (CH 1.0E-5)) PS:<GLR.FUNCT>RUNGE.LSP

Uses a multi-step predictor-corrector method to integrate DERIV (see <Hamming> page 407). Integrates from X0 to X1 using an initial step size of H. When the routine estimates that the error is below CD, the step size is doubled; when the error is above CH, the step size is halved. For reasonableness, CD << CH.

9. Root Finding Functions

(BISECTION-ROOT-SEARCH F X0 X1) PS:<GLR.FUNCT>ROOT.LSP

Uses bisection search to find a zero of F(X) between X0 and X1.

(FALSE-POSITION-SEARCH F X0 X1 EPS) PS:<GLR.FUNCT>ROOT.LSP

Uses false position search to find a zero of F(X) with initial guesses X0 and X1.

(FALSE-POSITION-CONVERGE F X0 X1 EPS) PS:<GLR.FUNCT>ROOT.LSP

Uses false position convergence to find an X = F(X) with initial guesses X0 and X1.

(CONVERGE F X0 EPS) PS:<GLR.FUNCT>ROOT.LSP

Find an X = F(X) with initial guess of X0. Uses Wegstein's method.

10. Probability and Statistics

The following functions are useful in probability and statistics. There are functions for computing probability density functions p(x), computing cumulative distributions P(x), and generating random numbers from a particular distribution. Most of the approximations come from <Abramowitz>. Most of the generators use the ratio of uniform deviates method <Kinderman>.

(UNIFORM-DENSITY X)
(UNIFORM-CUMULATIVE X)
(UNIFORM-RANDOM-NUMBER) PS:<GLR.FUNCT>STATIS.LSP

Probability functions for the uniform distribution.

$$p(x) = 1 \quad 0 <= x <= 1$$

(NORMAL-DENSITY X)
(NORMAL-CUMULATIVE X) PS:<GLR.FUNCT>STATIS.LSP

Probability functions for normal (Gaussian) distribution.

$$p(x) = (1/\mathrm{sqrt}(2\pi)) \exp(-x^2/2)$$

(NORMAL-TAIL ALPHA) PS:<GLR.FUNCT>STATIS.LSP

Given a probability ALPHA, find the X such that ALPHA=1-P(X), where P(X) is the cumulative distribution. This function provides a one-sided tail.

(NORMAL-RANDOM-NUMBER) PS:<GLR.FUNCT>STATIS.LSP

Generates a random number from the unit normal distribution.

(EXPONENTIAL-DENSITY X LAMBDA)
(EXPONENTIAL-CUMULATIVE X LAMBDA)
(EXPONENTIAL-RANDOM-NUMBER LAMBDA) PS:<GLR.FUNCT>STATIS.LSP

Exponential probability functions.

$$p(x, \lambda) = \lambda \exp(-\lambda x), \quad \lambda > 0$$

(POISSON-DENSITY N TIME)
(POISSON-CUMULATIVE N TIME)
(POISSON-RANDOM-NUMBER TIME) PS:<GLR.FUNCT>STATIS.LSP

Poisson probability functions. Probability of exactly N (FIXNUM) arrivals within TIME (FLONUM) for a Poisson process with an average arrival rate (λ) of 1.

$$p(n, m) = (m^n \exp(-m))/n!, \quad m = \lambda t$$

(CHI-SQUARE-DENSITY X N)
(CHI-SQUARE-CUMULATIVE X N)
(CHI-SQUARE-RANDOM-NUMBER N) PS:<GLR.FUNCT>STATIS.LSP

The CHI-SQUARE distribution with N (FIXNUM) degrees of freedom.

(T-DENSITY X N)
(T-CUMULATIVE X N)
(T-RANDOM-NUMBER N) PS:<GLR.FUNCT>STATIS.LSP

Student's T distribution with N (FIXNUM) degrees of freedom. N must be even in T-CUMULATIVE.

(T-TWO-SIDED X N) PS:<GLR.FUNCT>STATIS.LSP

For the T distribution, the probability that T falls within -X to +X. N must be an even FIXNUM.

(F-DENSITY X M N)
(F-CUMULATIVE X M N)
(F-RANDOM-NUMBER M N) PS:<GLR.FUNCT>STATIS.LSP

Snedecor's F distribution with M and N (FIXNUM) degrees of freedom.

(GAMMA-DENSITY X A)
(GAMMA-CUMULATIVE X A)
(GAMMA-RANDOM-NUMBER A) PS:<GLR.FUNCT>STATIS.LSP

Gamma probability functions with parameter A (FLONUM). The random number generator must have A > 1.0.

$$p(x, a) = (1/\Gamma(a))\, x^{a-1}\, e^{-x}, \quad a>0$$

(BETA-DENSITY X A B)
(BETA-CUMULATIVE X A B)
(BETA-RANDOM-NUMBER A B) PS:<GLR.FUNCT>STATIS.LSP

Beta probability functions with parameters A and B (FLONUM). The random number generator is slow, so keep A and B small (say below 10).

$$p(x, a, b) = (1/\beta(a,b))\, x^{a-1}\, (1-x)^{b-1}, \quad a>0,\ b>0$$

(CAUCHY-DENSITY X)
(CAUCHY-CUMULATIVE X)
(CAUCHY-RANDOM-NUMBER) PS:<GLR.FUNCT>STATIS.LSP

Probability functions for the Cauchy distribution.

$$p(x) = 1 / (\pi\, (1 + x^2))$$

11. Bibliography

Abramowitz Abramowitz and Stegun, eds, **Handbook of Mathematical Functions,** National Bureau of Standards, 1964

Bevington Philip R. Bevington, **Data Reduction and Error Analysis for the Physical Sciences,** McGraw-Hill, New York, 1969.

DEC Digital Equipment Corporation, **PDP-11 Paper Tape Software Programming Handbook,** DEC-11-GGPA-D, Section 7.7.

Hamming R. W. Hamming, **Numerical Methods for Scientist and Engineers,** McGraw-Hill, 1973.

Hastings Cecil Hastings, Jr., **Approximations for Digital Computers,** Princeton University Press, Princeton, NJ, 1955.

Kinderman A. J. Kinderman and J. F. Monahan, "Computer Generation of Random Variables Using the Ratio of Uniform Deviates," **ACM Transactions on Mathematical Software,** Vol 3 No 3, September 1977, pp 257-260.

Kuester James Kuester and Joe Mize, **Optimization Techniques with Fortran,** McGraw-Hill, New York, 1973.

Luenberger David G. Luenberger, **Introduction to Linear and Nonlinear Programming,** Addison Wesley, Reading, MA, 1965.

Oppenheimer Alan Oppenheimer and Ronald Schafer, **Digital Signal Processing**, Prentice-Hall, 1975.

Solovay Robert Solovay and Volker Strassen, "A Fast Monte-Carlo Test for Primality," **SIAM Journal on Computing**, 1977, pp 84-85.

Steele Guy Steele, Jr, **Common Lisp Reference Manual**, Mary Poppins Edition, Carnegie-Mellon University, 29 November 1983.