Behavior Research Methods, Instruments, & Computers
1992, 24 (4), 560-564

PRAXIS: Brent's algorithm for function minimization

KARL R. GEGENFURTNER
New York University, New York, New York

Implementations of Brent's (1973) PRincipal AXIS (PRAXIS) algorithm in the widely used C and PASCAL programming languages are presented. The algorithm minimizes a multivariate function without using derivatives. An example computer program that calculates a maximum likelihood estimate of the parameters of a psychometric function illustrates the use of the routine. Another algorithm, Localmin, *also due to Brent (1973), efficiently finds the minimum of a univariate function. An example program uses this algorithm to estimate the polychoric correlation coefficient from a* p×q *table of observed frequencies.*

The problem. Finding the minimum of a multivariate function f is a common problem in psychological research. For example, fitting a model to a set of data points involves minimizing the deviation of the model's predictions from the data points. A simple case with which we are all familiar consists of linear regression using least squares. To find the best estimates of the slope and the intercept, we differentiate the error function with respect to each of the parameters, set these derivatives to 0, and solve the resulting system of equations for the unknown parameters.

This standard recipe, however, fails in many cases when the model function is more complicated. Partial derivates might not exist, or the resulting set of equations might not have an explicit solution. In those cases we have to resort to numerical methods that search the parameter space in a systematic way.

Common approaches. The most intuitive minimization technique is the method of steepest descent. The main idea is to change the current parameter estimates in the direction of the largest decrease in f. For a linear function, this direction is the direction of the negative of its gradient. We find the minimum value of the function along this direction and iterate the procedure until the function can no longer be decreased.

This method has some serious drawbacks, making it slow and inefficient. If we minimize f along a direction $\mathbf{u}$, the gradient at that minimum point will be orthogonal to $\mathbf{u}$. Therefore, the only search directions used in this procedure are the ones orthogonal to the gradient at the starting point. In the case where the minimum is not located in one of those directions from the current point, the procedure typically meanders in small steps along a valley, instead of making large steps in the direction of the minimum.

To improve convergence, we can make use of the fact that most functions are fairly well approximated by a quadratic function near their minima. A method that converges quickly to the minimum of a quadratic function will also quickly find the minimum of a general function. *Quickly* in this case means that the number of iterations is finite. Algorithms with this property are called *quadratically convergent*. We can classify these algorithms by the kind of information they use to find the minimum.

The first class of methods uses information on the partial derivatives of the function. Typically, during one iteration, information on the matrix of the second partial derivatives is built up and used to solve for the minimum of the quadratic. Algorithms using this approach have been proposed by Fletcher and Powell (1963) and Broyden (1967). Derivative methods are generally more efficient than the "direction-set" methods, which minimize f along a set of search directions chosen to make the algorithm quadratically convergent. This approach is taken in Brent's (1973) method, given below.

On the other hand, the direction-set methods are more general. They work even when derivatives do not exist. It should be noted that we cannot simply use the more efficient derivative methods by approximating derivatives numerically. The roundoff errors involved in the calculation of derivatives are significant, and the number of function evaluations necessary to do so diminishes the efficiency advantage. Furthermore, methods using derivatives do have problems in certain cases, which makes the derivative-free methods appear more stable and robust.

Finally, it should be pointed out that special algorithms exist for the case where the function to be minimized is a sum of squares $f = \sum_{i=1}^{N} f_i^2$. In this case, the information in the individual residuals can be used efficiently (see, e.g., Levenberg, 1944).

Brent's algorithm. PRAXIS is a modification of Powell's (1964) direction-set method. Powell devised an algorithm in which the set of search directions $\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(n)}$ (n is the number of parameters) is repeatedly updated until a set of conjugate directions (with respect to a quadratic form) is reached after n iterations. Therefore, at the next iteration, the minimum will be found if the function to be minimized was indeed quadratic. The algorithm proceeds as follows: Let $\mathbf{x}^{(0)}$ be an initial approximation to the minimum, and set the initial search directions $\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(n)}$ to the columns of the identity matrix. For all $i = 1, \ldots, n$, compute $\lambda^{(i)}$ to minimize $f(\mathbf{x}^{(i-1)} + \lambda^{(i)}\mathbf{u}^{(i)})$. Set $\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \lambda^{(i)}\mathbf{u}^{(i)}$. For all $i = 1, \ldots, n-1$, set $\mathbf{u}^{(i)} = \mathbf{u}^{(i+1)}$. Replace $\mathbf{u}^{(n)}$ by $\mathbf{x}^{(n)} - \mathbf{x}^{(0)}$. Compute λ to minimize $f(\mathbf{x}^{(0)} + \lambda\mathbf{u}^{(n)})$, and replace $\mathbf{x}^{(0)}$ by $\mathbf{x}^{(0)} + \lambda\mathbf{u}^{(n)}$. This procedure is iterated until the function minimum is reached according to some criterion. Brent gives a proof that the procedure will reach the minimum

Correspondence should be addressed to K. Gegenfurtner, Howard Hughes Medical Institute and Center for Neural Science, New York University, 4 Washington Place, New York, NY 10003.

of a quadratic function in n iterations. The algorithm is therefore quadratically convergent.

Powell's (1964) algorithm depends on keeping the $\mathbf{u}^{(i)}$ conjugate. If, however, one of the $\lambda^{(i)}$ above vanishes, the corresponding direction vector $\mathbf{u}^{(i)}$ will also vanish, and therefore the direction set will no longer span the entire parameter space and the correct minimum may not be found. Additional precautions have to be taken to ensure that the $\mathbf{u}^{(i)}$ remains conjugate. This could be done by resetting the $\mathbf{u}^{(i)}$ to the columns of the identity matrix after every n iterations, but this would throw away useful information about the function. In Brent's (1973) extension of Powell's method, therefore, the matrix of the search directions is replaced by its principal axes, thereby ensuring conjugacy and keeping previously obtained information. Other modifications include the option of automatic scaling of the different variables, the incorporation of random steps into the procedure to avoid "resolution ridge" problems, and the extrapolation of the minimization path along quadratic space curves. A detailed discussion of the algorithm is given in Brent.

It should be noted at this point that PRAXIS does not allow one to specify any constraints on the problem. For example, the range of values that a parameter can take cannot be restricted. This is not a big problem, however, since constrained problems can usually be converted into unconstrained ones by parameter transformations or simple penalty functions. Also, like almost all other methods, PRAXIS will search for local minima. For a discussion of global minimization, see Brent (1973). In practice, one would typically start the algorithm at a variety of different starting points. If it converges to the same solution every time, it is good evidence that the minimum found is global.

Usage. PRAXIS is declared as follows:

double *praxis* (double (**func*)(), double *x*[], int *n*),

where *func* is a pointer to a function returning a double, x is a pointer to an array of doubles holding the initial estimates of the minimum upon procedure entry and returning the final estimates on exit, and the integer variable n is the number of parameters. PRAXIS calls the function *func* repeatedly with arguments x and n until a minimum is found. PRAXIS returns the function value obtained at the minimum as a double.

There are various global variables to fine tune the algorithm:

prin: An integer variable that controls the printed output from the routine. A value of 0 suppresses all output, 1 only prints starting and final values, 2 (the default) prints a detailed map of the minimization process, and 3 prints eigenvalues and eigenvectors of the search directions as well.

tol: A double used for the stopping criterion. PRAXIS returns to the calling program if the criterion $2\,||x^{(k)}-x^{(k-1)}|| \leq \epsilon^{1/2}||x^{(k)}||\, + tol$ is fulfilled more than *ktm* times. ϵ is the machine precision, the smallest number so that $1+\epsilon>1$.

ktm: An integer value used above. The default for *ktm* is 1, which suffices for most cases. However, if PRAXIS indicates that the problem is difficult to minimize (see *illc* below) a higher value (e.g., 4) leads to a more conservative stopping criterion.

step: A double specifying the maximum step size. It should be set to the maximum expected distance between the initial guess and the solution. Exceptionally small or large values of *step* lead to slower convergence on the first few iterations.

illc: A logical variable specifying whether the problem is known to be difficult to minimize (ill conditioned). Ill-conditioned problems are prone to roundoff error, and convergence is typically slower. PRAXIS will set *illc* to a value of 1 if it finds the problem to be ill conditioned.

scbd: A double controlling the automatic scaling feature. *Scbd* acts as an upper limit on the scaling factor. Therefore, the default value of 1.0 disables automatic scaling. When the scaling is very different for the different parameters, the function is harder to minimize and *scbd* should be increased to a value of 10.

maxfun: An integer number specifying the maximum number of calls to *func*. PRAXIS will return after *maxfun* calls even when the minimum has not been found. A value of 0 indicates no limit on the number of calls.

Table 1
Data From a Brightness Discrimination Experiment

Stimulus	Intensity	"Yes" Responses	Comparisons
1	10	2	49
2	20	3	48
3	30	13	48
4	40	31	50
5	50	38	47
6	60	48	49

Example. As an example, we will use PRAXIS to fit a psychometric function to a set of data points. The data in Table 1 illustrate the outcome of a hypothetical experiment on brightness discrimination. On each experimental trial, two patches of light are presented to a subject: a reference light with an intensity of 35 units, and a variable intensity comparison light. The subject has to indicate which one of the two lights was brighter. The first column of the table gives X_i, the intensity of the comparison light in arbitrary units. The second column is the number of times that the subject judged the comparison to be brighter than the reference R_i. The third is the number of times, N_i, that each comparison was made.

We will not go into the theory behind the use of psychometric functions here. Details on that and on other fitting methods are given, for example, in Bock and Jones (1968). Suffice it to say that we want to fit the relative frequencies of "yes" responses R_i/N_i with a cumulative Gaussian probability distribution. We do so by the method of maximum likelihood. We search for parameters μ and σ of the Gaussian so that $L(\mu,\sigma;X)$ is maximal. We can use PRAXIS to minimize

$$-\ln L = \sum_i R_i \ln P((X_i-\mu)/\sigma) + (N_i-R_i)\ln(1-P((X_i-\mu)/\sigma)),$$

where $P(x)$ is the cumulative standard normal probability distribution.

The program in Listing 1 reads X_i, R_i, and N_i from its input, calls PRAXIS to minimize the negative of the log-likelihood function, and prints the final estimates of μ and and σ (in this case, $\mu = 37.1$, and $\sigma = 12.9$).

LISTING 1
PMF, a Program for the Maximum-Likelihood Estimation of the Parameters of a Psychometric Function

```
#include <stdio.h>
#include <math.h>
#define NMAX 1000
double     X[NMAX], /* comparison intensities */
           R[NMAX], /* number of correct responses */
           N[NMAX]; /* number of comparisons */
int        ncat; /* number of comparison stimuli */
extern     double     praxis();
extern     double     P(), /* cumulative Gauss. prob. */
                      L(); /* likelihood function */

main(argc, argv)
char *argv[];
{
    double mean,   /* estimate of the mean */
           sd,     /* estimate of the standard deviation */
           like,   /* log likelihood */
           x[2];   /* parameter vector */

    ncat = 0;
    mean = sd = 0.0;
/* read data */
    while (scanf("%lg %lg %lg", &X[ncat], &R[ncat], &N[ncat]) == 3) {
            mean += X[ncat];
            sd += pow(X[ncat], 2.0);
            ncat++;
    }
    mean /= ncat;
    sd = sqrt(sd /ncat - pow(mean, 2.0));
/* our initial parameter estimates are simply the */
/* mean and s.e. of the physical intensities */
    x[0] = mean;
    x[1] = sd;
    like = praxis(L, x, 2);
    mean = x[0];
    sd = x[1];
    printf("Mean = %g SD = %g -log(L) = %g\n", mean, sd, like);
}

double L(x, n)
double x[];
int n;
{
    double loglike, p;
    int i;

    loglike = 0.0;
    for (i=0; i<ncat; i++) {
            p = P((X[i] - x[0]) /x[1]);
            loglike += log(p) * R[i] + log(1.0-p) * (N[i] - R[i]);
    }
    return -loglike;
}
```

Table 2
Comparison of PRAXIS and STEPIT

	PRAXIS		STEPIT	
Function	No. Calls	Minimum	No. Calls	Minimum
Rosenbrock (1)	194	1.619e-17	456	0.149e-10
Rosenbrock (2)	183	1.154e-09	377	0.462e-07
Rosenbrock (3)	320	1.130e-15	592	0.142e-13
Helix	209	3.363e-17	231	0.140e-44
Cube	218	3.638e-15	529	0.309e-08
Beale	113	8.098e-14	157	0.364e-13
Singular	443	1.135e-16	929	0.285e-12

Note—The three different entries for Rosenbrock were obtained with different starting points.

Efficiency. To show the relative efficiency of PRAXIS, we will compare it with another minimization subroutine in widespread use, the FORTRAN routine STEPIT (Chandler, 1969). Table 2 gives the number of function evaluations and the obtained minima for a number of test functions from the literature. For a detailed description of the test functions, see Brent (1973). The test programs were straightforward translations between C and FORTRAN up to the STEPIT or PRAXIS function call.

Even though the different compilers used for the different routines make it hard to compare the numbers directly, it is indicative of the larger efficiency of PRAXIS that it consistently found a minimum in fewer function calls. In most cases, the minimum itself is smaller as well.

The one-dimensional case. Often all that is required is to minimize a function of one variable. In that case, it is more efficient to resort to a special routine than to apply a multidimensional minimizer. Routines generally fall into two classes. Bracketing routines systematically reduce the interval in which the minimum is to be found. Other routines approximate the function locally and then extrapolate. Whereas the first group has the best worst-case behavior, the second group of methods converges much quicker around the function minimum, the point where most functions are well approximated by parabolas. Brent (1973) gives a routine, *Localmin*, that combines the advantages of both methods. Interpolation steps are performed if they lead to significant improvement; otherwise a golden-section search is done. In a variety of tests, this method has been shown to be one of the most efficient in use. We give an implementation of the *Localmin* procedure from Brent's book in the C programming language.

Usage. The declaration for *Localmin* is as follows:

double *localmin* (double (**func*)(),
double **x*, double *a*, double *b*),

where *func* is a pointer to the function to be minimized. *x* holds a pointer to the one and only parameter to *func*, and *a* and *b* are the lower and the upper bounds for *x*. *Localmin* finds the minimum of *func* over the interval (*a*,*b*). Upon return to the calling program, *x* points to the value where *func* has a minimum, and the value of *func* at that point is returned. There are no global variables controlling execution.

LISTING 2
POLYC, a Program to Compute the Polychoric Correlation Coefficient from a $p\times q$ Table of Observed Frequencies

```
#include <stdio.h>
#include <math.h>
#define MAXR 10 /* maximum number of rows */
#define MAXC 10 /* maximum number of columns */
double      localmin(), /* univariate minimizer */
            L(), /* likelihood function */
            Zp(), /* inverse 1-dim gaussian */
            P(), /* 1-dim gaussian prob. distr */
            binorm(); /* 2-dim gaussian prob. distr */
int         data[MAXR][MAXC], /* observed frequencies */
            rmarg[MAXR], /* row marginals */
            cmarg[MAXC]; /* column marginals */
int         nr, /* number of rows */
            nc, /* number of columns */
            ntotal;
double      alpha[MAXR], /* row category boundaries */
            beta[MAXC]; /* column category boundaries */
double      Pd[MAXR][MAXC]; /* 2-dim gaussian volume */
double      Pc[MAXR][MAXC]; /* cumul. 2-dim gaussian volume */
double      rho = 0.0; /* polychoric coefficient */

main(argc, argv)
char *argv;
{
    int i, j;
    double fmin;

/* read number of rows and columns */
    scanf("%d %d", &nr, &nc);
    for (i=0; i<nr; i++) {
            for (j=0; j<nc; j++) {
                          /* read frequencies */
                         scanf("%d", &data[i][j]);
                         /* compute marginals */
                         rmarg[i] += data[i][j];
                         cmarg[j] += data[i][j];
                         ntotal += data[i][j];
            }
    }
/* cumulate marginals and fit univariate Gaussians to them */
    for (i=1; i<nr; i++) {                          /* rows */
            rmarg[i] += rmarg[i-1];
            alpha[i-1] = Zp((double)rmarg[i-1]/(double)ntotal);
    }
    alpha[nr-1] = 10.0;
    for (j=1; j<nc; j++) {                          /* columns */
            cmarg[j] += cmarg[j-1];
            beta[j-1] = Zp((double)cmarg[j-1]/(double)ntotal);
    }
    beta[nc-1] = 10.0;
/* maximize likelihood */
    fmin = localmin(L, &rho, -1.0, 1.0);
    printf("rho = %5.3f L = %g)\n", rho, fmin);
}

double L(rest)
double rest;             /* the current estimate of r */
{
    int i, j;
    double loglike = 0.0;

    for (i=0; i<nr; i++) {
            for (j=0; j<nc; j++) {
                         Pd[i][j] = binorm(alpha[i], beta[j], rest);
                         Pc[i][j] = Pd[i][j];
                         if (i > 0)
                                  Pd[i][j] -= Pc[i-1][j];
                         if (j > 0)
                                  Pd[i][j] -= Pc[i][j-1];
                         if (i > 0 && j > 0)
                                  Pd[i][j] += Pc[i-1][j-1];
                         loglike += data[i][j] * log(Pd[i][j]);
            }
    }
    return -loglike;
}
```

Example. As an example of how to use *Localmin*, we give a program that calculates the polychoric correlation coefficient ϱ from a $p\times q$ table of observed frequencies (see Drasgow, 1986). ϱ is a measure of bivariate association between two ordered categorical variables. It is assumed that the variables follow a bivariate normal distribution and that the categories arise from polychotomization. For the case of a 2×2 table, ϱ can easily be calculated from the observed frequencies. This coefficient, known as the tetrachoric correlation coefficient, was introduced by Pearson. For the general case of a $p\times q$ table, the estimation of ϱ is more difficult and has to be solved by iterative methods (Tallis, 1962). Not only ϱ is unknown, so are the category borders of the bivariate normal. To reduce the amount of calculation, we first fit univariate normals to the marginals of the table and then maximize the likelihood function with respect to ϱ only. This method is called the two-step method, and numerical simulations have shown that the estimates of ϱ are very close to those obtained by a full maximum-likelihood estimation. The program POLYC below implements this two-step method, using *Localmin* to find the minimum of the negative of the likelihood function. In this case

$$-\ln L = \sum_{i=1}^{p} \sum_{j=1}^{q} n_{ij} \ln(P_{ij}),$$

where n_{ij} is the number of observations in category i of the first variable and category j of the second. P_{ij} is the probability of an observation in that category, obtained by integrating over a rectangular area of a two-dimensional Gaussian probability distribution Φ enclosed by the category boundaries α_i and β_j:

$$P_{ij} = \int_{\beta_{j-1}}^{\beta_j} \int_{\alpha_{i-1}}^{\alpha_i} \Phi(x,y,\varrho)\,dy\,dx.$$

The program POLYC in Listing 2 reads from its input the number of rows and columns in the table, and then for each row the frequencies in each category. It then fits univariate Gaussians to the marginals of the table and calls *Localmin* to find the maximum-likelihood estimate of the polychoric correlation coefficient ϱ.

Availability. The C source code for all routines and programs (the PRAXIS and *Localmin* C functions, the POLYC and PMF programs, and various functions for calculating one- and two-dimensional cumulative normal probability functions, which are not given in the text) is available from the author at no cost. A TURBO-Pascal version of PRAXIS is also available. Send a 3.5-in. formatted MS-DOS floppy disk and a self-addressed stamped envelope. Alternatively, users with access to electronic mail can obtain the program by sending an e-mail note to karl@cns.nyu.edu, requesting the program.

REFERENCES

Bock, R. D., & Jones, L. V. (1968). *The measurement and prediction of judgment and choice*. San Francisco: Holden-Day.

Brent, R. P. (1973). *Algorithms for function minimization without derivatives*. Englewood Cliffs, NJ: Prentice-Hall.

Broyden, C. G. (1967). Quasi-Newton methods and their application to function minimization. *Mathematics of Computation*, **21**, 368-381.

Chandler, J. P. (1969). STEPIT: Finds local minima of a smooth function of several parameters. *Behavioral Science*, **14**, 81-82.

Drasgow, F. (1986). Polychoric and polyserial correlations. In S. Kotz, N. L. Johnson, & C. B. Read (Eds.), *Encyclopedia of statistical sciences: Vol. 7* (pp. 68-74). New York: Wiley.

Fletcher, R., & Powell, M. J. D. (1963). A rapidly convergent descent method for minimization. *Computer Journal*, **6**, 163-168.

Levenberg, K. A. (1944). A method for the solution of certain nonlinear problems in least squares. *Quarterly of Applied Mathematics*, **2**, 164-168.

Powell, M. J. D. (1964). An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Computer Journal*, **7**, 155-162.

Tallis, G. M. (1962). The maximum likelihood estimation of correlation from contingency tables. *Biometrics*, **18**, 342-353.

(Manuscript received September 3, 1991;
revision accepted for publication July 6, 1992.)