Algorithm 869: ODRPACK95: A Weighted Orthogonal Distance Regression Code with Bound Constraints

JASON W. ZWOLAK
Virginia Polytechnic Institute and State University
PAUL T. BOGGS
Sandia National Laboratories, Livermore
and
LAYNE T. WATSON
Virginia Polytechnic Institute and State University

ODRPACK (TOMS Algorithm 676) has provided a complete package for weighted orthogonal distance regression for many years. The code is complete with user selectable reporting facilities, numerical and analytic derivatives, derivative checking, and many more features. The foundation for the algorithm is a stable and efficient trust region Levenberg-Marquardt minimizer that exploits the structure of the orthogonal distance regression problem. ODRPACK95 was created to extend the functionality and usability of ODRPACK. ODRPACK95 adds bound constraints, uses the newer Fortran 95 language, and simplifies the interface to the user called subroutine.

Categories and Subject Descriptors: G.1.6 [**Numerical Analysis**]: Optimization—*Least squares methods*; G.3 [**Probability and Statistics**]—*Statistical software*

General Terms: Algorithms

Additional Key Words and Phrases: Errors in variables, measurement error models, nonlinear least squares, orthogonal distance regression, simple bounds, Fortran 95

ACM Reference Format:

Zwolak, J. W., Boggs, P. T., and Watson, L. T. 2007. Algorithm 869: ODRPACK95: A weighted orthogonal distance regression code with bound constraints. ACM Trans. Math. Softw., 33, 4, Article 27 (August 2007), 12 pages. DOI = 10.1145/1268776.1268782 http://doi.acm.org/10.1145/1268776.1268782

The work was supported in part by Air Force Office of Scientific Research grant F49620-01-2-0572, and National Science Foundation grant IBN-0219322.
Authors' addresses: J. Zwolak, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061-0106, jzwolak@vt.edu; P. Boggs, MS 9159, Sandia National Laboratories, Livermore, CA 94551, ptboggs@sandia.gov; L. Watson, Departments of Computer Science and Mathematics, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061-0106.

1. INTRODUCTION

Least squares is arguably the most common method for fitting data to a model when there are errors in the observations. For example, given the data pairs $(x_i, y_i),\ i = 1, \ldots, n$, where x_i is the independent variable and y_i is the dependent variable, suppose that x_i and y_i are related by a smooth, possibly nonlinear function f; that is,

$$y_i = f(x_i; \beta), \tag{1}$$

where $\beta \in \mathcal{R}^p$ is a vector of parameters to be determined. Equation (1) is meant to imply that if there are no errors in either x_i or y_i and if β is known exactly, then (1) holds. If there are errors in the data, then the true value of β can likely only be approximately obtained, unless the number of observations goes to infinity.

In classical least squares, it is assumed that x_i is known exactly and y_i is observed with error. Although it is often the case that the x_i have errors, these errors can be safely ignored if they are much smaller than the corresponding errors in the y_i. Thus, taking the error in y_i to be given by ϵ_i, write

$$y_i + \epsilon_i = f(x_i; \beta) \tag{2}$$

and approximate β by solving the classical, or ordinary, least squares problem [OLS] given by

$$\min_{\beta} \frac{1}{2} \sum_{i=1}^{n} [f(x_i; \beta) - y_i]^2.$$

This, of course, can be interpreted as minimizing the sum of the squares of the vertical distances from the data points to the curve $y = f(x; \beta)$.

If, however, the error in x_i cannot be ignored and δ_i denotes the error in x_i, then Equation (2) becomes

$$y_i + \epsilon_i = f(x_i + \delta_i; \beta),$$

and it is reasonable to approximate the parameter β by minimizing the sum of the squares of the orthogonal distances from the data points to the curve $y = f(x, \beta)$. As shown in Boggs et al. [1987] this gives rise to the *orthogonal distance regression* [ODR] problem given by

$$\min_{\beta,\delta} \frac{1}{2} \sum_{i=1}^{n} \left[(f(x_i + \delta_i; \beta) - y_i)^2 + \delta_i^2\right].$$

Note that ODR is easily seen to be equivalent to

$$\begin{aligned} &\min_{\beta,\delta,\epsilon} \frac{1}{2} \sum_{i=1}^{n} \left(\epsilon_i^2 + \delta_i^2\right) \\ &\text{subject to} \\ &y_i + \epsilon_i = f(x_i + \delta_i; \beta), \qquad i = 1, \ldots, n, \end{aligned} \tag{3}$$

from which it is easy to see that ODR is, indeed, minimizing the sum of the squares of the orthogonal distances. Note that in general x_i and $y_i + \epsilon_i$

$= f_i(x_i + \delta_i; \beta)$ are vectors, and the objective function in (1.3) takes the form

$$\min_{\beta,\delta,\epsilon} \frac{1}{2} \sum_{i=1}^{n} \left(\epsilon_i^t w_{\epsilon_i} \epsilon_i + \delta_i^t w_{\delta_i} \delta_i \right), \tag{4}$$

where the weights w_{ϵ_i} and w_{δ_i} are symmetric positive semidefinite matrices. Also in general the functional relationships $y_i + \epsilon_i = f_i(x_i + \delta_i; \beta)$ may vary between data points, hence the notation f_i.

A numerically stable and efficient algorithm for solving ODR is given in Boggs et al. [1987] and a detailed implementation, called ODRPACK, that provides a number of practical options and statistical output is given in Algorithm 676 [Boggs et al. 1989]. In Boggs et al. [1987], the authors show that the work per iteration for their algorithm for solving ODR is exactly the same as the work per iteration for solving OLS. An enhancement of ODRPACK is available from Netlib [Dongarra and Grosse 1987]. This is a Fortran 77 implementation that includes the ability to handle a general weighting scheme, allows x to be multidimensional, and contains a version to allow complex data. This code has been downloaded and used many times by scientists, engineers, and practitioners around the world; it is described in several textbooks, including Björck [1996] and Nocedal and Wright [1999]. ODR has important statistical applications; in the statistical literature it often goes by the name "errors in variables" (see, e.g., Fuller [1987] or Cheng and Van Ness [1999]).

Over the years, there have been occasional requests to implement a version of ODRPACK that allows explicit bounds on the values of β, but this was not done. The general form of the bound constrained problem (BC-ODR) can be expressed as

$$\min_{\beta,\delta} \frac{1}{2} \sum_{i=1}^{n} \left[(f(x_i + \delta_i; \beta) - y_i)^2 + \delta_i^2 \right] \qquad \text{subject to} \qquad L \le \beta \le U,$$

where L and U are vectors of length p that provide the lower and upper bounds on β, respectively. ODRPACK has some features that could be used to solve a bound constrained problem, but the resulting algorithm is not very efficient.

This paper has two goals. First, it addresses the issue of modifying the ODRPACK algorithm to handle bounds efficiently, and second, it describes the updates to ODRPACK by rewriting much of it in Fortran 95. The resulting code, called ODRPACK95, is thus much simpler to use because it takes advantage of Fortran 95 to do dynamic memory management and to allow much easier passing of parameters.

The paper is organized as follows. Section 2 reviews briefly ODRPACK and the algorithm given in Boggs et al. [1987]. Also reviewed are some of the features of ODRPACK that could be used to handle bounds and why these are not efficient. Section 3 describes and references a few applications of ODRPACK and motivates the need for bound constraints. Section 4 gives the modifications to the algorithm to handle these bounds efficiently. Sections 5 and 6 are a discussion of testing and performance, respectively.

2. ODRPACK DESCRIPTION

A brief description of the ODRPACK algorithm is provided here. Weights and multidimensional x_i, y_i are left out for simplicity and can be added at the expense of complexity and bookkeeping, but require no fundamental changes in the algorithm and pseudocode described here. ODRPACK is based on a trust region Levenberg-Marquardt algorithm with scaling and numeric or analytic derivatives and is described in detail in Boggs et al. [1987, 1989, 1992] and in general in Conn et al. [2000].

Building on the problem (BC-ODR) described earlier, the partials of the (weighted, in the general case) errors ϵ and δ are taken with respect to the parameters β and δ to give the Jacobian matrix

$$J = \begin{bmatrix} \frac{\partial \epsilon_1}{\partial \beta_1} & \cdots & \frac{\partial \epsilon_1}{\partial \beta_p} & \frac{\partial \epsilon_1}{\partial \delta_1} & \cdots & \frac{\partial \epsilon_1}{\partial \delta_n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \epsilon_n}{\partial \beta_1} & \cdots & \frac{\partial \epsilon_n}{\partial \beta_p} & \frac{\partial \epsilon_n}{\partial \delta_1} & \cdots & \frac{\partial \epsilon_n}{\partial \delta_n} \\ \frac{\partial \delta_1}{\partial \beta_1} & \cdots & \frac{\partial \delta_1}{\partial \beta_p} & \frac{\partial \delta_1}{\partial \delta_1} & \cdots & \frac{\partial \delta_1}{\partial \delta_n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \delta_n}{\partial \beta_1} & \cdots & \frac{\partial \delta_n}{\partial \beta_p} & \frac{\partial \delta_n}{\partial \delta_1} & \cdots & \frac{\partial \delta_n}{\partial \delta_n} \end{bmatrix}.$$

Note that the appearance of δ in both the errors and the parameters gives a special and exploitable structure to the Jacobian matrix. The Jacobian matrix can be divided into four quadrants based on the combinations of ϵ and δ with β and δ, where all but the upper left quadrant possess special structure. For convenience the quadrants are labeled

$$J = \begin{bmatrix} G & V \\ Z & D \end{bmatrix}.$$

G is the Jacobian matrix of ϵ with respect to β and has no special properties. V is the Jacobian matrix of ϵ with respect to δ and is a diagonal matrix. This property follows because each ϵ_i depends only on δ_i, see Equation (3). Z is the Jacobian matrix of δ with respect to β and is all zeros because δ is an independent variable. Finally, D is a constant diagonal matrix representing the Jacobian matrix of δ with respect to δ. (In the non-constant weighted case $D \neq I$.) In a naive algorithm for ODR each element in Z, D, and V must be calculated. The time complexity for this calculation is $3n^2$, where n is the number of experimental data points (the same as the number of δs and the number of ϵs). ODRPACK's exploitation of the structure of the problem ODR requires only the diagonals of V and D to be calculated; the time complexity with respect to n is reduced from quadratic ($3n^2$) to linear ($2n$). ODRPACK efficiently solves the orthogonal distance regression problem.

ODRPACK uses a trust region algorithm that minimizes a model of the objective function in a sufficiently small neighborhood of the current point in which the model is "trusted." In the case where a linear model is used, define

$$E(\beta,\delta) = \begin{pmatrix} \epsilon \\ \delta \end{pmatrix} = (f_1(x_1+\delta_1;\beta) - y_1, \dots, f_n(x_n+\delta_n;\beta) - y_n, \delta_1, \dots, \delta_n)^T, \tag{5}$$

so that

$$\|E(\beta,\delta)\|^2 = \sum_{i=1}^{n} \left[\| f_i(x_i+\delta_i;\beta) - y_i\|^2 + \|\delta_i\|^2 \right], \tag{6}$$

and let (β,δ) denote the current iterate in the code. The step composed of (s,t) (the increment to (β,δ)) is calculated by solving

$$\min_{s,t} \left\| E(\beta,\delta) + J(\beta,\delta)\begin{pmatrix} s \\ t \end{pmatrix} \right\|^2 \quad \text{subject to} \quad \left\| \begin{pmatrix} S & 0 \\ 0 & T \end{pmatrix} \begin{pmatrix} s \\ t \end{pmatrix} \right\| \le \tau, \tag{7}$$

where $J(\beta,\delta)$ is the Jacobian matrix of E evaluated at (β,δ), S and T are diagonal scaling matrices for s and t, respectively, and τ is the trust region radius. The scaling matrices can be chosen by the user (through SCLB and SCLD), or they can be set by default as described in the User's Guide [Boggs et al. 1992].

The following pseudocode gives an overview of the algorithm.

```
do until convergence
    Compute $G$, $V$, and $D$ as described above.
    $P = V^TV + D^2 + \alpha T^2$: $P$ is defined here to simplify the following equations.
        $\alpha$ is the Lagrange multiplier for Equation (7) and $T$ is a scaling matrix for
        $t$.
    Formulate the linear least squares problem (derived from the linearization
        of the objective function)
        $\min_s \left\| \left((I - VP^{-1}V^T)^{\frac{1}{2}}Gs\right) - (I - VP^{-1}V^T)^{-\frac{1}{2}}\left(-\epsilon + VP^{-1}(V^T\epsilon + D\delta)\right)\right\|^2$,
        and solve for $s$ with a QR factorization of the coefficient matrix of $s$.[1]
    $t = -P^{-1}(V^T\epsilon + D\delta + V^TGs)$;
    Use $s$ and $t$ to update $\beta$ and $\delta$, respectively.[2]
end do
```

[1]Note that Boggs et al. [1987] realized the problem ODR can be solved efficiently this way instead of solving the normal equations with the full Jacobian matrix J.

[2]The Levenberg-Marquardt method starts with the steepest decent method and smoothly changes to the Gauss-Newton method, where s and t are simply added to β and δ, as the solution is approached. ODRPACK uses a trust region implementation of the Levenberg-Marquardt method that reduces the step size based on the confidence in a model of the objective function. See Moré and Wright [1993] for details on how parameters are updated in the Levenberg-Marquardt algorithm.

ODRPACK has a simple method for handling invalid parameters. The user-supplied subroutine that calculates f can indicate invalid parameters by returning a flag to ODRPACK. ODRPACK then reverts to the last successful point in parameter space and reduces the step size until f can be evaluated. In the case where a user wishes parameters to be bound constrained this approach may cause ODRPACK to stall near a boundary but far from optimality even though a valid direction still exists that reduces the objective function. In this case a user can continue optimization using an active set strategy, keeping parameters in the active set fixed at their boundary values. This requires that ODRPACK be restarted with a new active set every time a parameter hits a boundary.

ODRPACK's effective use of the structure of the Jacobian matrix makes it an efficient algorithm for the unconstrained weighted orthogonal nonlinear least squares problem. However, for problems where β is constrained, ODRPACK's handling of invalid parameters can be inconvenient and slow [Zwolak et al. 2005a]. The next section describes the changes to ODRPACK, motivated by the need to directly support simple bound constraints.

3. APPLICATIONS

There are many applications of orthogonal distance regression. In general the technique is applicable in any field utilizing models with error in both dependent *and* independent variables. ODRPACK has been applied, for example, in ecology [Rothschild et al. 1997], astronomy [Feigelson and Babu 1992], and molecular biology [Zwolak et al. 2005b].

Feigelson and Babu [1992] give an overview of software for different types of regression problems occurring in astronomy. Regression problems, specifically ODR, occur frequently in observational astronomy. One example measures luminosity of an astronomical object with respect to distance of that object from the observer. The distance measurements contain error as they are also observations. Feigelson cites ODRPACK as one of two numerical packages solving ODR.

Rothschild et al. [1997] apply ODRPACK to estimating parameters of population models. Growth and mortality rates are estimated for each developmental stage of the organism (the stages for humans may be, for example, infant, child, and adult). The model uses ordinary differential equations to quantify the rates of growth, and mortality and simulations can be performed showing the population in each stage with respect to time. Rothschild et al. [1997] show that the ecology community can use ODRPACK to fit stage-structured population models to observed population numbers. Rothschild et al. [1997] note that "a drawback to this approach is that one cannot impose constraints ... representing maximum or minimum bounds on variations in population." Such a constraint can be represented by bound constraints on the parameters for growth and mortality.

Zwolak et al. [2005b] apply ODRPACK to estimating parameters of a molecular network model. This is a model of protein concentration for a few key proteins in cell division. The proteins undergo chemical changes that activate and inactivate their function. The chemical changes have rate constants, which are the model's parameters. Zwolak et al. [2005b] successfully estimate the rate constants with ODRPACK. Zwolak et al. [2005b] had to use ODRPACK's parameter fixing feature and ability to reject invalid parameters to manually constrain parameters to be positive. Negative parameters would constitute a qualitative change in the model (e.g., a reaction from A to B becomes a reaction from B to A) and is not desirable to the modeler.

These applications highlight a few areas where ODRPACK can be and has been used. They also show the need for bound constraints.

4. DIFFERENCES BETWEEN ODRPACK95 AND ODRPACK

The ODRPACK95 code contains the original ODRPACK code wherever possible. Deviations, rewrites, and additions made to the original code are described in this section. All changes fall into two major categories: those required to support bounds, and those required by or made possible by the conversion to Fortran 95.

4.1 Bound Constraints

The most important addition made in ODRPACK95 is the support for bound constraints on the parameter vector β: $L_i \leq \beta_i \leq U_i$, $i = 1, \ldots, p$. In some applications, evaluation of the function outside of the bounds is not possible. Thus the choice was made to implement a feasible-point algorithm and insist that all calls to the function evaluation routine be made with points that are feasible. This has particular consequences for the finite-difference gradient option, as described in the following.

The algorithm used for bound constraints is the same as that in LANCELOT [Conn et al. 1992], which calculates the projected step so as to always maintain a feasible β. Lin and Moré [1999] describe another algorithm for bound constraints that is the new default in LANCELOT and may be suited for a future version of ODRPACK95. With the former algorithm, when a parameter value is at its bound the corresponding numerical partial derivative will use a one sided finite difference approximation to avoid calculations with parameters outside the bounds. Lastly, the initialization in ODRPACK95 requires function evaluations at feasible parameter values. Thus the ODRPACK initialization algorithm has been modified to only use feasible β values. ODRPACK95 never calls the user supplied function with parameters outside the user supplied bounds and the final solution is guaranteed to be feasible.

The most important change added a restriction on β during each iteration before any function evaluations are made with β. The restriction ensures that the current function evaluation is made with feasible β. After β is updated with the step s, whose direction is a convex combination of the Gauss-Newton direction and the steepest descent direction, β will be projected into the hyperbox $[L, U]$, precisely:

$$\beta_i := \begin{cases} \beta_i, & \text{if } L_i \leq \beta_i \leq U_i, \\ L_i, & \text{if } \beta_i < L_i, \\ U_i, & \text{if } U_i < \beta_i, \end{cases} \qquad i = 1, \ldots, p.$$

Before β is updated and projected, numerical or analytic derivatives are calculated. If analytic derivatives are used, then no additional function evaluations are required; the derivatives are calculated at the current β. When forward or backward differences are used then the step h_i must obey $L_i \le \beta_i + h_i \le U_i$, for $i = 1, \dots, p$. The sign of h_i is changed if the bounds are violated, namely

$$h_i := \begin{cases} h_i, & \text{if } L_i \le \beta_i + h_i \le U_i; \\ -h_i, & \text{if } L_i > \beta_i + h_i \text{ or } \beta_i + h_i > U_i, \end{cases} \qquad i = 1, \dots, p.$$

It is possible that $\beta_i + |h_i| > U_i$ and $\beta_i - |h_i| < L_i$ for some i. ODRPACK95 avoids this situation by ensuring that $U_i - L_i \ge 2|h_i|$, for all i, where $h_i > 0$ is chosen with respect to the initial β during initialization. When central differences are used, then the points where the function is evaluated are shifted together until they are both within the bounds, precisely

$$(\beta_i^-, \beta_i^+) := \begin{cases} (\beta_i - h_i, \beta_i + h_i), & \text{if } L_i \le \beta_i - |h_i| < \beta_i + |h_i| \le U_i, \\ (L_i, L_i + 2|h_i|), & \text{if } L_i > \beta_i - |h_i|, \\ (U_i - 2|h_i|, U_i), & \text{if } U_i < \beta_i + |h_i|. \end{cases}$$

This has the effect of shifting the points β_i^- and β_i^+ such that they are always $2h_i$ apart and within the bounds. If, for example, β_i is $0.2h_i$ from U_i (such that $\beta_i + 0.2h_i = U_i$) then U_i and $U_i - 2h_i$ are used in the central difference formula instead of $\beta_i + h_i$ and $\beta_i - h_i$. Furthermore, if β_i is on a bound (e.g., $\beta_i = L_i$ or $\beta_i = U_i$) then the modified central difference method used here becomes a forward or backwards differentiation formula (depending on which bound β_i lies on). Again, ODRPACK95 ensures during initialization that $U_i - L_i \ge 2|h_i|$, for $i = 1, \dots, p$. It is possible that these modifications could cause some inconsistency in the finite difference approximations and thus some loss of final accuracy, but the testing here did not reveal any significant problems. Since the order of the finite difference formulas and the step size remain the same, significant numerical problems are unlikely. There remain, of course, functions for which finite difference approximations are notoriously bad, for example, functions based on the integration of an ODE.

During initialization function evaluations are required for derivative checking, the initial point, and prediction of the number of reliable digits (NDIGIT). The derivative checking uses the same code as the numerical derivatives to request function evaluations, and therefore does not evaluate the function outside the bounds. The initial point is verified to be within the bounds, and if it is not then the code returns with an appropriate error flag. Finally, prediction of the number of reliable digits in the objective function must be made. These calculations are used to determine the step size for the finite differences [Dennis and Schnabel 1983]. These calculations occur centered at the initial β. To ensure that these calculations occur only at feasible points, the center point used in the calculations is minimally adjusted from the initial β to be far enough from the bounds for the calculations to succeed (in a manner similar to that of central differences described above).

With these changes ODRPACK95 provides the same reliable and efficient optimization as ODRPACK, but with simple bound constraints. The original ODRPACK algorithm was shown to be convergent to a first-order critical point.

ODRPACK95 projects this direction onto the feasible region; if sufficient decrease (as measured by the true objective function) is not achieved, the trust region radius is reduced. Thus, in the limit, one would be projecting the gradient and convergence follows from the convergence of the projected gradient method. Although this argument does not constitute a formal proof of convergence, it does provide some justification.

4.2 Fortran 95

ODRPACK95 conforms to the Fortran 95 specification ISO/IEC 1539-1. The code was compiled, run, and tested with the SUN Solaris f95 (SUN Sparc), DEC f95 (Alpha), Intel ifort (Linux on an Intel Xeon), Lahey lf95 (Linux on an Intel Xeon), NAGWare f95 (Linux on an Intel Xeon), Intel Itanium compiler (SGI Altix), and IBM xlf (OS X on an Apple G5). The code uses the Fortran 95 fixed format to minimize changes from (the Fortran 77 fixed format code of) ODRPACK and the likelihood of bugs introduced into the code. The conversion to Fortran 95 facilitates a number of other significant improvements in ODRPACK95.

Among those improvements are optional arguments, use of Fortran 95 modules, automatic array allocation, and use of Fortran 95 intrinsic functions for machine constants. All nonessential arguments to ODRPACK95 are optional; this makes the ODRPACK95 interface considerably simpler and allows defaults to be set when arguments are not present. The call to ODRPACK95 was simplified from

```
      CALL DODRC(FCN,N,M,NP,NQ,BETA,Y,LDY,X,LDX,WE,LDWE1,LD2WE1, &
          WD,LDWD1,LD2WD1,IFIXB,IFIXX,LDIFX,JOB,NDIGIT,TAUFAC,    &
          SSTOL,PARTOL,MAXIT,IPRINT,LUNERR,LUNRPT,STPB,STPD,      &
          LDSTPD,SCLB,SCLD,LDSCLD,WORK,LWMIN,IWORK,LIWMIN,INFO)
```

to

```
      CALL ODR(FCN,N,M,NP,NQ,BETA,Y,X,LOWER=L,UPPER=U)
```

and there is only one interface to ODRPACK95 while ODRPACK has DODRC, DODR, SODRC, and SODR. These multiple interfaces to ODRPACK exist to allow short and long argument lists and single and double precision arithmetic. The user's calling program would contain the statement

```
USE ODRPACK95
```

giving their code access to the ODRPACK95 interface and aiding in compile time error checking. The array arguments in the interface will be automatically allocated if the user does not supply the (optional) argument or does not allocate the provided argument. The arrays will be deallocated only if the user does not provide an argument for them to be returned in. Lastly, all the calculations are done using the machine constants from the Fortran 95 intrinsics, eliminating the former need to supply a function to return machine constants. These changes all together make ODRPACK95 a substantial improvement over the original ODRPACK.

5. TESTING

ODRPACK95 was tested thoroughly with dozens of test cases and test problems. Some of these are distributed with the code and are described here. The test cases for ODRPACK are also distributed with ODRPACK95 and are described in Boggs et al. [1992]. The ODRPACK95 tests can be run with `make test.out`. The output file `test.out` contains the results of the tests and will end with "`ALL TESTS AGREE WITH EXPECTED RESULTS`" in the case that all tests passed. This is a way to ensure a properly installed and functioning ODRPACK95.

The model for the test problem new in ODRPACK95 is specially constructed to exercise many of the conditions that may arise in bound constrained optimization. The new test cases are listed below and numbered as they are seen in `test.f`.

(13) Parameters start on a boundary, move away from the boundary, hit a boundary, move away from the boundary, and stop at a minimum.
(14) Parameters start interior to the bounds, never hit a boundary, and stop at a minimum.
(15) Parameters start interior to the bounds and stop on a boundary.
(16) Parameters start outside the bounds, and ODRPACK95 returns an error flag.
(17) Bounds are ill defined ($L_i > U_i$ for some i), and ODRPACK95 returns an error flag.
(18) Central differences are used. Parameters start on a boundary, move away from the boundary, hit a boundary, move away from the boundary, and stop at a minimum.
(19) Bounds are well defined but slightly too close for ODRPACK95 to do any calculations. An error flag is returned.
(20) Bounds are well defined and slightly farther apart than the previous case. They are far enough apart to allow NDIGIT calculations but still too close for ODRPACK95 to proceed. An error flag is returned.
(21) Bounds are well defined and as close as the machine allows, and therefore too close for finite differences and NDIGIT calculations. ODRPACK95 returns an error flag.

The model used for all the bound constraint test cases above is

$$f(x;\beta) = \beta_1 e^{\beta_2 * x},$$

where the global minimum point (used to generate experimental data) is $\beta = (1, 1)^T$.

6. PERFORMANCE

The test cases documented in the previous section were also used to benchmark ODRPACK95. ODRPACK95 performs exactly like ODRPACK when $\beta + s$ remains feasible except that an additional IF-statement is executed to check that $\beta + s$ is feasible. When a boundary is crossed or reached, ODRPACK95 must execute many additional statements to ensure all function evaluations

Table I. The Setup and Timing Results for Benchmarks of ODRPACK95. Included is the test case number (Test #) as found in `test.f`; the lower and upper bounds of β (Lower and Upper); the initial β (β_0); the final β (β_{final}); the number of ODRPACK95 iterations while β was on a bound (#it$_B$) and the total number of iterations (#it$_T$); the run time measured from when `ODR` was called to when it finished (Time (ms)); number of function evaluations (#FEV); and the number of function evaluations per iteration (#FEV/#i). Case 18 was modified from `test.f` by changing its lower bound.

Test #	13	14	15	18
Lower	(0.10,0)	(0.00,0)	(1.10,0)	(0.01,0)
Upper	(200,5)	(400,6)	(400,6)	(200,5)
β_0	(200,5)	(200,5)	(200,3)	(200,5)
β_{final}	(1.0,1)	(1.0,1)	(1.1,1)	(1.0,1)
#it$_B$/#it$_T$	83/95	0/25	68/68	3/26
Time (ms)	9.765	3.907	5.859	3.907
#FEV	388	108	285	188
#FEV/#it$_T$	4.05	4.2	4.15	7.23

are performed with feasible parameters and that the resulting β is feasible. It is this case of active bound constraints that this section addresses.

The benchmarks were performed with a modified `test.f`. Lines were added before and after the call to `ODR` that read the time with the Fortran 95 `CPU_TIME` intrinsic function. In addition, the lower bound for Test Case 18 was modified. Without this modification Test Case 18 is the same as Test Case 13 except that central differences are used. This modification adds to the diversity of the benchmarks. Table I shows the setup for the benchmarks and timing results. After these modifications, the code was compiled, then run with the command

```
nice -n -20 ./a.out > stdout.txt
```

on a dual 2GHz Xeon machine running Linux. The `nice` command ensures the benchmark gets highest priority of the running processes (no other active processes were running during the benchmark, but many system processes were sleeping or waiting for interrupts). No options for the Intel Fortran 95 compiler were used for the benchmarks.

The number of iterations for the runs (see Table I) varies significantly. The variances can be attributed to ODRPACK/ODRPACK95's response to information about the objective function along the different paths through parameter space. (Since each test case took a different path through parameter space, the objective function will look different on those paths causing ODRPACK/ODRPACK95 to behave differently.) Reaching a bound increases the number of iterations, but the work per iteration is roughly the same whether β is on a bound or not.

A notable exception is when central differences are used. The use of central differences approximately doubles the number of function evaluations per iteration compared with forward and backward differences. On a boundary, central differences are replaced by forward or backward differences because the function cannot be evaluated outside the bounds. This explains the almost double number of function evaluations per iteration seen in Test Case 18 in Table I. It is always the case that use of central differences in ODRPACK95 will be cheaper per iteration when the parameters are on a boundary than when the parameters are away from all boundaries.

REFERENCES

Björck, Å. 1996. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, PA.

Boggs, P. T., Byrd, R. H., and Schnabel, R. B. 1987. A stable and efficient algorithm for nonlinear orthogonal distance regression. *SIAM J. Sci. Stat. Comput. 8*, 6 1052–1078.

Boggs, P. T., Donaldson, J. R., Byrd, R. H., and Schnabel, R. B. 1989. Algorithm 676: ODRPACK: Software for weighted orthogonal distance regression. *ACM Trans. Math. Soft. 15*, 4 348–364.

Boggs, P. T., Byrd, R. H., Rogers, J. E., and Schnabel, R. B. 1992. *User's Reference Guide for ODRPACK Version 2.01: Software for Weighted Orthogonal Distance Regression*. Center for Computing and Applied Mathematics, U.S. Department of Commerce, Gaithersburg, MD.

Cheng, C. L. and Van Ness, J. W. 1999. *Statistical Regression with Measurement Error.* Oxford University Press, Oxford, UK.

Conn, A. R., Gould, N. I. M., and Toint, Ph. L. 1992. *LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization (Release A)*. Springer-Verlag, Berlin, Germany.

Conn, A. R., Gould, N. I. M., and Toint, Ph. L. 2000. *Trust-Region Methods.* SIAM, Philadelphia.

Dennis, J. E. and Schnabel, R. B. 1983. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall, Englewood Cliffs, NJ.

Dongarra, J. J. and Grosse, E. H. 1987. Distribution of mathematical software via electronic mail. *Comm. ACM 30*, 403–407.

Feigelson, E. D. and Babu, G. J. 1992. Linear regression in astronomy II. *Astrophysical J. 397*, 55–67.

Fuller, W. A. 1987. *Measurement Error Models*. John Wiley, New York, NY.

Lin, C. J. and Moré, J. J. 1999. Newton's method for large bound-constrained optimization problems. *SIAM J. Optim. 9*, 4, 1100–27.

Moré, J. J. and Wright, S. J. 1993. *Optimization Software Guide.* SIAM, Philadelphia, PA.

Nocedal, J. and Wright, S. J. 1999. *Numerical Optimization.* Springer-Verlag, Berlin, Germany.

Rothschild, B. J., Sharov, A. F., Kearsley, A. J., and Bondarenko, A. S. 1997. Estimating growth and mortality in stage-structured populations. *J. Plankton Res. 19*, 12, 1913–28.

Zwolak, J. W., Tyson, J. J., and Watson, L. T. 2005a. Parameter estimation for a mathematical model of the cell cycle in frog eggs. *J. Comput. Biol. 12*, 1, 48–63.

Zwolak, J. W., Tyson, J. J., and Watson, L. T. 2005b. Globally optimized parameters for a model of mitotic control in frog egg extracts. *IEE Systems Biol. 152*, 2, 81–92.

Received December 2004; revised July 2006, December 2006; accepted January 2007