IMSL®

IMSL Fortran Library User's Guide MATH/LIBRARY Volume 2 of 2

Mathematical Functions in Fortran

Trusted For Over 30 Years

IMSL®

IMSL Fortran Library User's Guide MATH/LIBRARY Volume 2 of 2

Mathematical Functions in Fortran

Visual Numerics

P/N 7694

[www.vni.com]

Visual Numerics, Inc. – United States
Corporate Headquarters
2000 Crow Canyon Place, Suite 270
San Ramon, CA 94583
PHONE: 925-807-0138
FAX: 925-807-0145
e-mail: info@vni.com
Westminster, CO
PHONE: 303-379-3040

Houston, TX
PHONE: 713-784-3131

Visual Numerics International Ltd.
Sussex House
6 The Forbury
Reading, Berkshire RGI 3EJ
UNITED KINGDOM

PHONE: +44-1-189 25-3370
FAX: +44 –1-189-25-3371
e-mail: info@vniuk.co.uk
Support: support@vniuk.co.uk

Visual Numerics SARL
Immeuble le Wilson 1
70, avenue due General de Gaulle
F-92058 PARIS LA DEFENSE, Cedex
FRANCE

PHONE: +33-1-46-93-94-20
FAX: +33-1-46-93-94-39
e-mail: info@vni.paris.fr

Visual Numerics S. A. de C. V.
Florencia 57 Piso 10-01
Col. Juarez
Mexico D. F. C. P. 06600
MEXICO

PHONE: +52-55-514-9730 or 9628
FAX: +52-55-514-4873

Visual Numerics International GmbH
Zettachring 10
D-70567Stuttgart
GERMANY

PHONE: +49-711-13287-0
FAX: +49-711-13287-99
e-mail: vni@visual-numerics.de

Visual Numerics Japan, Inc.
GOBANCHO HIKARI BLDG. 4TH Floor
14 GOBAN-CHO CHIYODA-KU
TOKYO, JAPAN 102

PHONE: +81-3-5211-7760
FAX: +81-3-5211-7769
e-mail: vnijapan@vnij.co.jp

Visual Numerics, Inc.
7/F, #510, Sect. 5
Chung Hsiao E. Road
Taipei, Taiwan 110
ROC

PHONE: +(886) 2-2727-2255
FAX: +(886) 2-2727-6798
e-mail: info@vni.com.tw

Visual Numerics Korea, Inc.
HANSHIN BLDG. Room 801
136-1, MAPO-DONG, MAPO-GU
SEOUL, 121-050
KOREA SOUTH

PHONE: +82-2-3273-2632 or 2633
FAX: +82-2-3273--2634
e-mail: info@vni.co.kr

World Wide Web site: http://www.vni.com

IMSL Fortran, C, and Java
Application Development Tools

Contents

Chapter 8: Optimization

Routines

Usage Notes

Unconstrained Minimization

The unconstrained minimization problem can be stated as follows:

$$\min_{x \in \mathbf{R}^n} f(x)$$

where $f: \mathbf{R}^n \rightarrow \mathbf{R}$ is at least continuous. The routines for unconstrained minimization are grouped into three categories: univariate functions (`UV***`), multivariate functions (`UM***`), and nonlinear least squares (`UNLS*`).

For the univariate function routines, it is assumed that the function is unimodal within the specified interval. Otherwise, only a local minimum can be expected. For further discussion on unimodality, see Brent (1973).

A quasi-Newton method is used for the multivariate function routines `UMINF` (page 1196) and `UMING` (page 1202), whereas `UMIDH` (page 1208) and `UMIAH` (page 1213) use a modified Newton algorithm. The routines `UMCGF` (page 1219) and `UMCGG` (page 1223) make use of a conjugate gradient approach, and `UMPOL` (page 1227) uses a polytope method. For more details on these algorithms, see the documentation for the corresponding routines.

The nonlinear least squares routines use a modified Levenberg-Marquardt algorithm. If the nonlinear least squares problem is a nonlinear data-fitting problem, then software that is designed to deliver better statistical output may be useful; see IMSL (1991).

These routines are designed to find only a local minimum point. However, a function may have many local minima. It is often possible to obtain a better local solution by trying different initial points and intervals.

High precision arithmetic is recommended for the routines that use only function values. Also it is advised that the derivative-checking routines `CH***` be used to ensure the accuracy of the user-supplied derivative evaluation subroutines.

Minimization with Simple Bounds

The minimization with simple bounds problem can be stated as follows:

$$\min_{x \in \mathbf{R}^n} f(x)$$

$$\text{subject to } l_i \le x_i \le u_i, \text{ for } i = 1, 2, \ldots, n$$

where $f : \mathbf{R}^n \rightarrow \mathbf{R}$, and all the variables are not necessarily bounded.

The routines `BCO**` use the same algorithms as the routines `UMI**`, and the routines `BCLS*` are the corresponding routines of `UNLS*`. The only difference is that an active set strategy is used to ensure that each variable stays within its bounds. The routine `BCPOL` (page 1271) uses a function comparison method similar to the one used by `UMPOL` (page 1227). Convergence for these polytope methods is not guaranteed; therefore, these routines should be used as a last alternative.

Linearly Constrained Minimization

The linearly constrained minimization problem can be stated as follows:

$$\min_{x \in \mathbf{R}^n} f(x)$$

$$\text{subject to } Ax = b$$

where $f : \mathbf{R}^n \rightarrow \mathbf{R}$, A is an $m \times n$ coefficient matrix, and b is a vector of length m. If $f(x)$ is linear, then the problem is a linear programming problem; if $f(x)$ is quadratic, the problem is a quadratic programming problem.

The routine `DLPRS` (page 1297) uses a revised simplex method to solve small- to medium-sized linear programming problems. No sparsity is assumed since the coefficients are stored in full matrix form.

The routine `QPROG` (page 1307) is designed to solve convex quadratic programming problems using a dual quadratic programming algorithm. If the given Hessian is not positive definite, then `QPROG` modifies it to be positive definite. In this case, output should be interpreted with care.

The routines `LCONF` (page 1310) and `LCONG` (page 1316) use an iterative method to solve the linearly constrained problem with a general objective function. For a detailed description of the algorithm, see Powell (1988, 1989).

Nonlinearly Constrained Minimization

The nonlinearly constrained minimization problem can be stated as follows:

$$\min_{x \in \mathbf{R}^n} f(x)$$

$$\text{subject to } g_i(x) = 0, \text{ for } \quad i = 1, 2, \ldots, m_1$$

$$g_i(x) \ge 0, \text{ for } \quad i = m_1 + 1, \ldots, m$$

where $f : \mathbf{R}^n \rightarrow \mathbf{R}$ and $g_i : \mathbf{R}^n \rightarrow \mathbf{R}$, for $i = 1, 2, \ldots, m$

The routines NNLPF (page 1323) and NNLPG (page 1329) use a sequential equality constrained quadratic programming method. A more complete discussion of this algorithm can be found in the documentation.

Selection of Routines

The following general guidelines are provided to aid in the selection of the appropriate routine.

Unconstrained Minimization

1. For the univariate case, use UVMID (page 1189) when the gradient is available, and use UVMIF (page 1182) when it is not. If discontinuities exist, then use UVMGS (page 1193).

2. For the multivariate case, use UMCG* when storage is a problem, and use UMPOL (page 1227) when the function is nonsmooth. Otherwise, use UMI** depending on the availability of the gradient and the Hessian.

3. For least squares problems, use UNLSJ (page 1237) when the Jacobian is available, and use UNLSF (page 1231) when it is not.

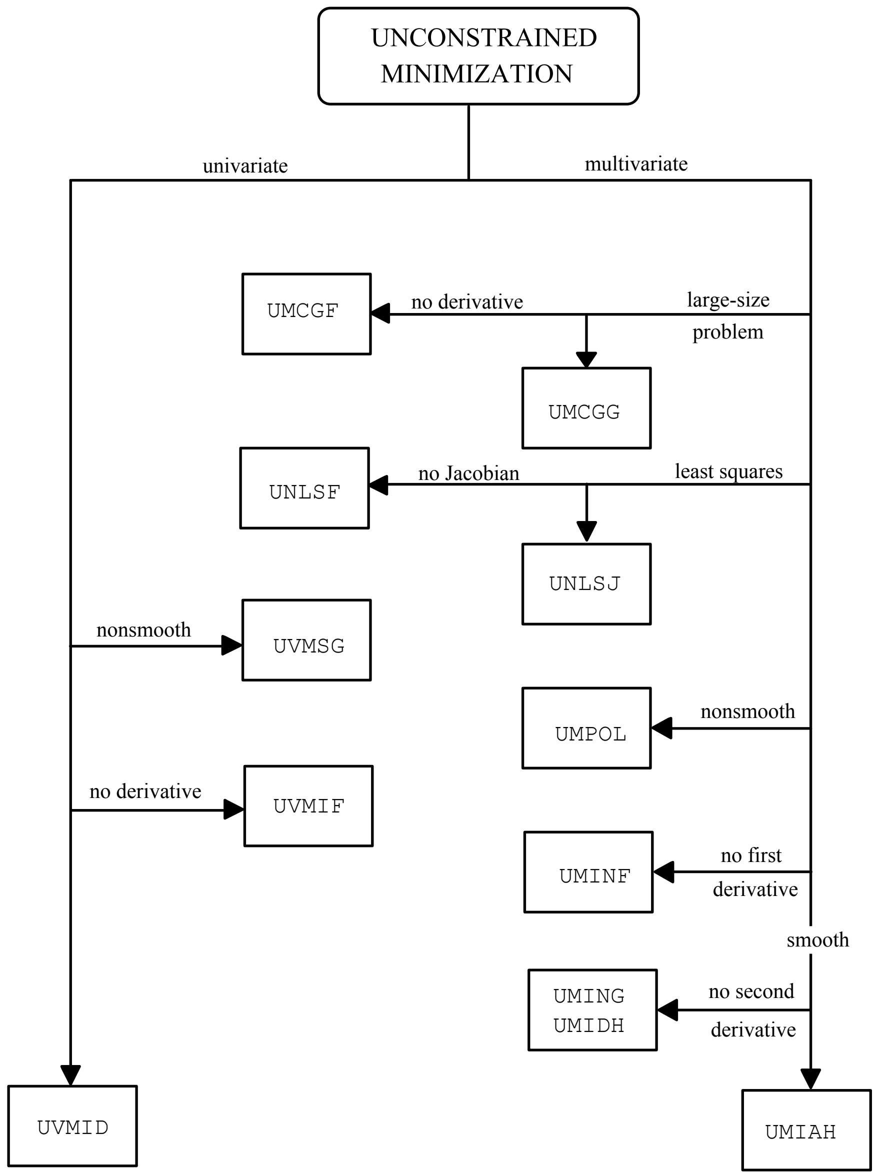
Minimization with Simple Bounds

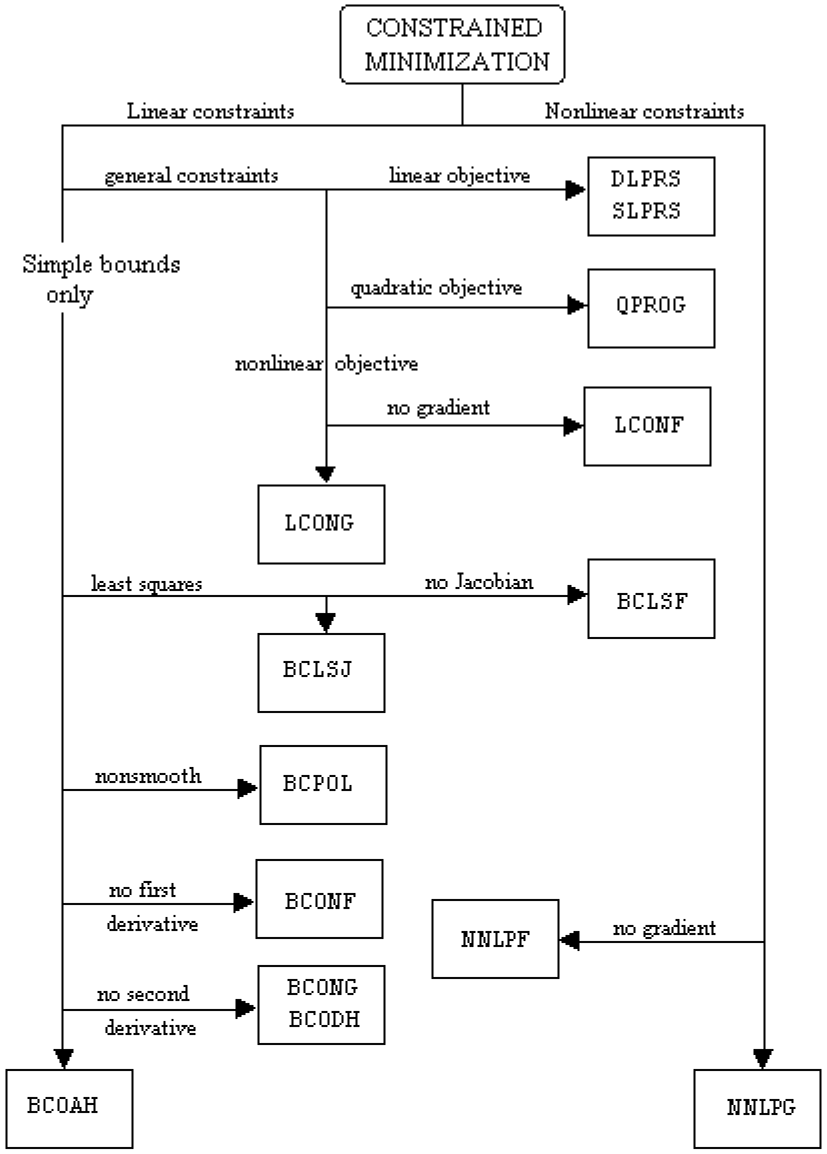
1. Use BCONF (page 1243) when only function values are available. When first derivatives are available, use either BCONG (page 1249) or BCODH (page 1257). If first and second derivatives are available, then use BCOAH (page 1263).

2. For least squares, use BCLSF (page 1274) or BCLSJ (page 1281) depending on the availability of the Jacobian.

3. Use BCPOL (page 1271) for nonsmooth functions that could not be solved satisfactorily by the other routines.

The following charts provide a quick reference to routines in this chapter:





UVMIF

Finds the minimum point of a smooth function of a single variable using only function evaluations.

Required Arguments

F — User-supplied `FUNCTION` to compute the value of the function to be minimized. The form is `F`(`X`), where
`X` – The point at which the function is evaluated. (Input)

`X` should not be changed by `F`.
`F` – The computed function value at the point `X`. (Output)

`F` must be declared `EXTERNAL` in the calling program.

XGUESS — An initial guess of the minimum point of `F`. (Input)

BOUND — A positive number that limits the amount by which `X` may be changed from its initial value. (Input)

X — The point at which a minimum value of `F` is found. (Output)

Optional Arguments

STEP — An order of magnitude estimate of the required change in `X`. (Input)
Default: `STEP` = 1.0.

XACC — The required absolute accuracy in the final value of `X`. (Input)
On a normal return there are points on either side of `X` within a distance `XACC` at which `F` is no less than `F(X)`.
Default: `XACC` = 1.e-4.

MAXFN — Maximum number of function evaluations allowed. (Input)
Default: `MAXFN` = 1000.

FORTRAN 90 Interface

Generic: `CALL UVMIF (F, XGUESS, BOUND, X [,…])`

Specific: The specific interface names are `S_UVMIF` and `D_UVMIF`.

FORTRAN 77 Interface

Single: `CALL UVMIF (F, XGUESS, STEP, BOUND, XACC, MAXFN, X)`

Double: The double precision name is `DUVMIF`.

Example

A minimum point of $e^x - 5x$ is found.

```
      USE UVMIF_INT
      USE UMACH_INT
!                                  Declare variables
      INTEGER    MAXFN, NOUT
      REAL       BOUND, F, FX, STEP, X, XACC, XGUESS
      EXTERNAL   F
!                                  Initialize variables
```

```
      XGUESS = 0.0
      XACC   = 0.001
      BOUND  = 100.0
      STEP   = 0.1
      MAXFN  = 50
!
!                                  Find minimum for F = EXP(X) - 5X
      CALL UVMIF (F, XGUESS, BOUND, X, STEP=STEP, XACC=XACC, MAXFN=MAXFN)
      FX = F(X)
!                                  Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, FX
!
99999 FORMAT ('   The minimum is at ', 7X, F7.3, //, '   The function ' &
            , 'value is ', F7.3)
!
      END
!                                  Real function: F = EXP(X) - 5.0*X
      REAL FUNCTION F (X)
      REAL       X
!
      REAL       EXP
      INTRINSIC  EXP
!
      F = EXP(X) - 5.0E0*X
!
      RETURN
      END
```

Output

```
The minimum is at          1.609

The function value is  -3.047
```

Comments

Informational errors

Type	Code	
3	1	Computer rounding errors prevent further refinement of X.
3	2	The final value of X is at a bound. The minimum is probably beyond the bound.
4	3	The number of function evaluations has exceeded MAXFN.

Description

The routine UVMIF uses a safeguarded quadratic interpolation method to find a minimum point of a univariate function. Both the code and the underlying algorithm are based on the routine ZXLSF written by M.J.D. Powell at the University of Cambridge.

The routine UVMIF finds the least value of a univariate function, *f*, that is specified by the function subroutine F. Other required data include an initial estimate of the solution, XGUESS , and a positive number BOUND. Let x_0 = XGUESS and b = BOUND, then x is restricted to the interval $[x_0 - b, x_0 + b]$. Usually, the algorithm begins the search by moving from x_0 to $x = x_0 + s$, where s = STEP is also provided by the user and may be positive or negative. The first two function evaluations indicate the direction to the minimum point, and the search strides out along this direction until a bracket on a minimum point is found or until x reaches one of the bounds $x_0 \pm b$. During this stage, the step length increases by a factor of between two and nine per function evaluation; the factor depends on the position of the minimum point that is predicted by quadratic interpolation of the three most recent function values.

When an interval containing a solution has been found, we will have three points, x_1, x_2, and x_3, with $x_1 < x_2 < x_3$ and $f(x_2) \le f(x_1)$ and $f(x_2) \le f(x_3)$. There are three main ingredients in the technique for choosing the new x from these three points. They are (i) the estimate of the minimum point that is given by quadratic interpolation of the three function values, (ii) a tolerance parameter ε, that depends on the closeness of f to a quadratic, and (iii) whether x_2 is near the center of the range between x_1 and x_3 or is relatively close to an end of this range. In outline, the new value of x is as near as possible to the predicted minimum point, subject to being at least ε from x_2, and subject to being in the longer interval between x_1 and x_2 or x_2 and x_3 when x_2 is particularly close to x_1 or x_3. There is some elaboration, however, when the distance between these points is close to the required accuracy; when the distance is close to the machine precision; or when ε is relatively large.

The algorithm is intended to provide fast convergence when f has a positive and continuous second derivative at the minimum and to avoid gross inefficiencies in pathological cases, such as

$$f(x) = x + 1.001|x|$$

The algorithm can make ε large automatically in the pathological cases. In this case, it is usual for a new value of x to be at the midpoint of the longer interval that is adjacent to the least calculated function value. The midpoint strategy is used frequently when changes to f are dominated by computer rounding errors, which will almost certainly happen if the user requests an accuracy that is less than the square root of the machine precision. In such cases, the routine claims to have achieved the required accuracy if it knows that there is a local minimum point within distance δ of x, where δ = XACC, even though the rounding errors in f may cause the existence of other local minimum points nearby. This difficulty is inevitable in minimization routines that use only function values, so high precision arithmetic is recommended.

UVMID

Finds the minimum point of a smooth function of a single variable using both function evaluations and first derivative evaluations.

Required Arguments

F — User-supplied FUNCTION to define the function to be minimized. The form is F(X), where

X — The point at which the function is to be evaluated. (Input)

F — The computed value of the function at X. (Output)

`F` must be declared `EXTERNAL` in the calling program.

G — User-supplied `FUNCTION` to compute the derivative of the function. The form is `G`(`X`), where

X — The point at which the derivative is to be computed. (Input)

G — The computed value of the derivative at X. (Output)

`G` must be declared `EXTERNAL` in the calling program.

A — `A` is the lower endpoint of the interval in which the minimum point of `F` is to be located. (Input)

B — `B` is the upper endpoint of the interval in which the minimum point of `F` is to be located. (Input)

X — The point at which a minimum value of `F` is found. (Output)

Optional Arguments

XGUESS — An initial guess of the minimum point of `F`. (Input)
Default: `XGUESS` = (`a` + `b`) / 2.0.

ERRREL — The required relative accuracy in the final value of `X`. (Input)
This is the first stopping criterion. On a normal return, the solution `X` is in an interval that contains a local minimum and is less than or equal to `MAX`(1.0, `ABS`(`X`)) `* ERRREL`. When the given `ERRREL` is less than machine epsilon, `SQRT`(machine epsilon) is used as `ERRREL`.
Default: `ERRREL` = 1.e-4.

GTOL — The derivative tolerance used to decide if the current point is a local minimum. (Input)
This is the second stopping criterion. `X` is returned as a solution when `GX` is less than or equal to `GTOL`. `GTOL` should be nonnegative, otherwise zero would be used.
Default: `GTOL` = 1.e-4.

MAXFN — Maximum number of function evaluations allowed. (Input)
Default: `MAXFN` = 1000.

FX — The function value at point `X`. (Output)

GX — The derivative value at point `X`. (Output)

FORTRAN 90 Interface

Generic: `CALL UVMID (F, G, A, B, X [,…])`

Specific: The specific interface names are `S_UVMID` and `D_UVMID`.

FORTRAN 77 Interface

Single: `CALL UVMID (F, G, XGUESS, ERRREL, GTOL, MAXFN, A, B, X, FX, GX)`

Double: The double precision name is `DUVMID`.

Example

A minimum point of $e^x - 5x$ is found.

```
      USE UVMID_INT
      USE UMACH_INT
!                                 Declare variables
      INTEGER    MAXFN, NOUT
      REAL       A, B, ERRREL, F, FX, G, GTOL, GX, X, XGUESS
      EXTERNAL   F, G
!                                 Initialize variables
      XGUESS = 0.0
!                                Set ERRREL to zero in order
!                                to use SQRT(machine epsilon)
!                                as relative error
      ERRREL = 0.0
      GTOL   = 0.0
      A      = -10.0
      B      = 10.0
      MAXFN  = 50
!
!                                Find minimum for F = EXP(X) - 5X
      CALL UVMID (F, G, A, B, X, XGUESS=XGUESS, ERRREL=ERRREL,  &
                 GTOL=FTOL, MAXFN=MAXFN, FX=FX, GX=GX)
!                                Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, FX, GX
!
99999 FORMAT ('   The minimum is at ', 7X, F7.3, //, '   The function ' &
            , 'value is ', F7.3, //, '   The derivative is ', F7.3)
!
      END
!                                Real function: F = EXP(X) - 5.0*X
      REAL FUNCTION F (X)
      REAL       X
!
      REAL       EXP
      INTRINSIC  EXP
!
      F = EXP(X) - 5.0E0*X
```

```
!
      RETURN
      END
!
      REAL FUNCTION G (X)
      REAL       X
!
      REAL       EXP
      INTRINSIC  EXP
!
      G = EXP(X) - 5.0E0
      RETURN
      END
```

Output

```
The minimum is at       1.609

The function value is  -3.047

The derivative is  -0.001
```

Comments

Informational errors

Type	Code	
3	1	The final value of `X` is at the lower bound. The minimum is probably beyond the bound.
3	2	The final value of `X` is at the upper bound. The minimum is probably beyond the bound.
4	3	The maximum number of function evaluations has been exceeded.

Description

The routine `UVMID` uses a descent method with either the secant method or cubic interpolation to find a minimum point of a univariate function. It starts with an initial guess and two endpoints. If any of the three points is a local minimum point and has least function value, the routine terminates with a solution. Otherwise, the point with least function value will be used as the starting point.

From the starting point, say x_c, the function value $f_c = f(x_c)$, the derivative value $g_c = g(x_c)$, and a new point x_n defined by $x_n = x_c - g_c$ are computed. The function $f_n = f(x_n)$, and the derivative $g_n = g(x_n)$ are then evaluated. If either $f_n \geq f_c$ or g_n has the opposite sign of g_c, then there exists a minimum point between x_c and x_n; and an initial interval is obtained. Otherwise, since x_c is kept as the point that has lowest function value, an interchange between x_n and x_c is performed. The secant method is then used to get a new point

$$x_s = x_c - g_c\left(\frac{g_n - g_c}{x_n - x_c}\right)$$

Let $x_n \leftarrow x_s$ and repeat this process until an interval containing a minimum is found or one of the convergence criteria is satisfied. The convergence criteria are as follows: Criterion 1:

$$\left|x_c - x_n\right| \le \varepsilon_c$$

Criterion 2:

$$\left|g_c\right| \le \varepsilon_g$$

where $\varepsilon_c = \max\{1.0, |x_c|\}\varepsilon$, ε is a relative error tolerance and ε_g is a gradient tolerance.

When convergence is not achieved, a cubic interpolation is performed to obtain a new point. Function and derivative are then evaluated at that point; and accordingly, a smaller interval that contains a minimum point is chosen. A safeguarded method is used to ensure that the interval reduces by at least a fraction of the previous interval. Another cubic interpolation is then performed, and this procedure is repeated until one of the stopping criteria is met.

UVMGS

Finds the minimum point of a nonsmooth function of a single variable.

Required Arguments

F — User-supplied `FUNCTION` to compute the value of the function to be minimized. The form is `F`(`X`), where

`X` – The point at which the function is evaluated. (Input)
`X` should not be changed by `F`.

F – The computed function value at the point `X`. (Output)

`F` must be declared `EXTERNAL` in the calling program.

A — On input, `A` is the lower endpoint of the interval in which the minimum of `F` is to be located. On output, `A` is the lower endpoint of the interval in which the minimum of `F` is located. (Input/Output)

B — On input, `B` is the upper endpoint of the interval in which the minimum of `F` is to be located. On output, `B` is the upper endpoint of the interval in which the minimum of `F` is located. (Input/Output)

XMIN — The approximate minimum point of the function `F` on the original interval (`A`, `B`). (Output)

Optional Arguments

TOL — The allowable length of the final subinterval containing the minimum point. (Input)
Default: `TOL` = 1.e-4.

FORTRAN 90 Interface

Generic: `CALL UVMGS (F, A, B, XMIN [,…])`

Specific: The specific interface names are `S_UVMGS` and `D_UVMGS`.

FORTRAN 77 Interface

Single: `CALL UVMGS (F, A, B, TOL, XMIN)`

Double: The double precision name is `DUVMGS`.

Example

A minimum point of $3x^2 - 2x + 4$ is found.

```
      USE UVMGS_INT
      USE UMACH_INT
!                                 Specification of variables
      INTEGER    NOUT
      REAL       A, B, FCN, FMIN, TOL, XMIN
      EXTERNAL   FCN
!                                 Initialize variables
      A   = 0.0E0
      B   = 5.0E0
      TOL = 1.0E-3
!                                 Minimize FCN
      CALL UVMGS (FCN, A, B, XMIN, TOL=TOL)
      FMIN = FCN(XMIN)
!                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) XMIN, FMIN, A, B
99999 FORMAT ('   The minimum is at ', F5.3, //, '   The ', &
            'function value is ', F5.3, //, '   The final ', &
            'interval is (', F6.4, ',', F6.4, ')', /)
!
      END
!
!                                 REAL FUNCTION: F = 3*X**2 - 2*X + 4
      REAL FUNCTION FCN (X)
      REAL       X
!
      FCN = 3.0E0*X*X - 2.0E0*X + 4.0E0
!
      RETURN
      END
```

Output

```
The minimum is at 0.333

The function value is 3.667

The final interval is (0.3331,0.3340)
```

Comments

1. Informational errors

Type	Code	
3	1	`TOL` is too small to be satisfied.
4	2	Due to rounding errors `F` does not appear to be unimodal.

2. On exit from `UVMGS` without any error messages, the following conditions hold: (`B`-`A`) ≤ `TOL`.
   `A` ≤ `XMIN` and `XMIN` ≤ `B`
   `F`(`XMIN`) ≤ `F`(`A`) and `F`(`XMIN`) ≤ `F`(`B`)

3. On exit from `UVMGS` with error code 2, the following conditions hold:
   `A` ≤ `XMIN` and `XMIN` ≤ `B`
   `F`(`XMIN`) ≥ `F`(`A`) and `F`(`XMIN`) ≥ `F`(`B`) (only one equality can hold).
   Further analysis of the function `F` is necessary in order to determine whether it is not unimodal in the mathematical sense or whether it appears to be not unimodal to the routine due to rounding errors in which case the `A`, `B`, and `XMIN` returned may be acceptable.

Description

The routine `UVMGS` uses the *golden section search* technique to compute to the desired accuracy the independent variable value that minimizes a unimodal function of one independent variable, where a known finite interval contains the minimum.

Let τ = `TOL`. The number of iterations required to compute the minimizing value to accuracy τ is the greatest integer less than or equal to

$$\frac{\ln\left(\tau/\left(b-a\right)\right)}{\ln\left(1-c\right)}+1$$

where a and b define the interval and

$$c=\left(3-\sqrt{5}\right)/2$$

The first two test points are v_1 and v_2 that are defined as

$$v_1 = a + c(b - a), \text{ and } v_2 = b - c(b - a)$$

If $f(v_1) < f(v_2)$, then the minimizing value is in the interval (a, v_2). In this case, $b \leftarrow v_2$, $v_2 \leftarrow v_1$, and $v_1 \leftarrow a + c(b - a)$. If $f(v_1) \geq f(v_2)$, the minimizing value is in (v_1, b). In this case, $a \leftarrow v_1$, $v_1 \leftarrow v_2$, and $v_2 \leftarrow b - c(b - a)$.

The algorithm continues in an analogous manner where only one new test point is computed at each step. This process continues until the desired accuracy τ is achieved. `XMIN` is set to the point producing the minimum value for the current iteration.

Mathematically, the algorithm always produces the minimizing value to the desired accuracy; however, numerical problems may be encountered. If f is too flat in part of the region of interest, the function may appear to be constant to the computer in that region. Error code 2 indicates that this problem has occurred. The user may rectify the problem by relaxing the requirement on τ, modifying (scaling, etc.) the form of f or executing the program in a higher precision.

UMINF

Minimizes a function of `N` variables using a quasi-Newton method and a finite-difference gradient.

Required Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (N, X, F)`, where

`N` – Length of `X`. (Input)

`X` – The point at which the function is evaluated. (Input)
`X` should not be changed by `FCN`.

`F` – The computed function value at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

X — Vector of length `N` containing the computed solution. (Output)

Optional Arguments

N — Dimension of the problem. (Input)
Default: `N` = size (`X`,1).

XGUESS — Vector of length `N` containing an initial guess of the computed solution. (Input)
Default: `XGUESS` = 0.0.

XSCALE — Vector of length `N` containing the diagonal scaling matrix for the variables. (Input)
`XSCALE` is used mainly in scaling the gradient and the distance between two points. In the absence of other information, set all entries to 1.0.
Default: `XSCALE` = 1.0.

FSCALE — Scalar containing the function scaling. (Input)
`FSCALE` is used mainly in scaling the gradient. In the absence of other information, set `FSCALE` to 1.0.
Default: `FSCALE` = 1.0.

IPARAM — Parameter vector of length 7. (Input/Output)
Set `IPARAM`(1) to zero for default values of `IPARAM` and `RPARAM`. See Comment 4.
Default: `IPARAM` = 0.

RPARAM — Parameter vector of length 7.(Input/Output)
See Comment 4.

FVALUE — Scalar containing the value of the function at the computed solution. (Output)

FORTRAN 90 Interface

Generic: `CALL UMINF (FCN, X [,…])`

Specific: The specific interface names are `S_UMINF` and `D_UMINF`.

FORTRAN 77 Interface

Single: `CALL UMINF (FCN, N, XGUESS, XSCALE, FSCALE, IPARAM, RPARAM, X, FVALUE)`

Double: The double precision name is `DUMINF`.

Example

The function

$$f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

is minimized.

```
      USE UMINF_INT
      USE U4INF_INT
      USE UMACH_INT
      INTEGER    N
      PARAMETER  (N=2)
!
      INTEGER    IPARAM(7), L, NOUT
      REAL       F, RPARAM(7), X(N), XGUESS(N), &
                XSCALE(N)
      EXTERNAL   ROSBRK
!
      DATA XGUESS/-1.2E0, 1.0E0/
!
!                                  Relax gradient tolerance stopping
!                                  criterion
      CALL U4INF (IPARAM, RPARAM)
      RPARAM(1) = 10.0E0*RPARAM(1)
!                                  Minimize Rosenbrock function using
!                                  initial guesses of -1.2 and 1.0
      CALL UMINF (ROSBRK, X, XGUESS=XGUESS, IPARAM=IPARAM, RPARAM=RPARAM, &
      FVALUE=F)
!                                  Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, F, (IPARAM(L),L=3,5)
!
99999 FORMAT ('  The solution is ', 6X, 2F8.3, //, '  The function ', &
            'value is ', F8.3, //, '  The number of iterations is ', &
            10X, I3, /, '  The number of function evaluations is ', &
            I3, /, '  The number of gradient evaluations is ', I3)
!
      END
!
      SUBROUTINE ROSBRK (N, X, F)
      INTEGER    N
      REAL       X(N), F
!
      F = 1.0E2*(X(2)-X(1)*X(1))**2 + (1.0E0-X(1))**2
!
      RETURN
      END
```

Output

```
The solution is          1.000   1.000

The function value is    0.000

The number of iterations is            15
The number of function evaluations is  40
The number of gradient evaluations is  19
```

Comments

1. Workspace may be explicitly provided, if desired, by use of `U2INF/DU2INF`. The reference is:

   ```
   CALL U2INF (FCN, N, XGUESS, XSCALE, FSCALE, IPARAM,
   RPARAM, X,FVALUE, WK)
   ```

   The additional argument is:

   WK — Work vector of length `N(N + 8)`. `WK` contains the following information on output: The second `N` locations contain the last step taken. The third `N` locations contain the last Newton step. The fourth `N` locations contain an estimate of the gradient at the solution. The final N^2 locations contain the Cholesky factorization of a `BFGS` approximation to the Hessian at the solution.

2. Informational errors

   Type Code

3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
4	2	The iterates appear to be converging to a noncritical point.
4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
4	5	Maximum number of gradient evaluations exceeded.
4	6	Five consecutive steps have been taken with the maximum step length.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or STEPTL is too big.
3	8	The last global step failed to locate a lower point than the current X value.

3. The first stopping criterion for UMINF occurs when the infinity norm of the scaled gradient is less than the given gradient tolerance (RPARAM(1)). The second stopping criterion for UMINF occurs when the scaled distance between the last two steps is less than the step tolerance (RPARAM(2)).

4. If the default parameters are desired for UMINF, then set IPARAM(1) to zero and call the routine UMINF. Otherwise, if any nondefault parameters are desired for IPARAM or RPARAM, then the following steps should be taken before calling UMINF:

CALL U4INF (IPARAM, RPARAM)
Set nondefault values for desired IPARAM, RPARAM elements.

Note that the call to U4INF will set IPARAM and RPARAM to their default values so only nondefault values need to be set above.

The following is a list of the parameters and the default values:

IPARAM — Integer vector of length 7.
IPARAM(1) = Initialization flag.

IPARAM(2) = Number of good digits in the function.
Default: Machine dependent.

IPARAM(3) = Maximum number of iterations.
Default: 100.

IPARAM(4) = Maximum number of function evaluations.
Default: 400.

IPARAM(5) = Maximum number of gradient evaluations.
Default: 400.

IPARAM(6) = Hessian initialization parameter.

If IPARAM(6) = 0, the Hessian is initialized to the identity matrix; otherwise, it is initialized to a diagonal matrix containing

$$\max\left(\left|f\left(t\right)\right|, f_s\right) * s_i^2$$

on the diagonal where t = XGUESS, f_S = FSCALE, and s = XSCALE.

Default: 0.

IPARAM(7) = Maximum number of Hessian evaluations.

Default: Not used in UMINF.

RPARAM — Real vector of length 7.

RPARAM(1) = Scaled gradient tolerance.

The i-th component of the scaled gradient at x is calculated as

$$\frac{\left|g_i\right| * \max\left(\left|x_i\right|, 1/s_i\right)}{\max\left(\left|f\left(x\right)\right|, f_s\right)}$$

where $g = \nabla f(x)$, s = XSCALE, and f_S = FSCALE.

Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

RPARAM(2) = Scaled step tolerance. (STEPTL)

The i-th component of the scaled step between two points x and y is computed as

$$\frac{\left|x_i - y_i\right|}{\max\left(\left|x_i\right|, 1/s_i\right)}$$

where s = XSCALE.

Default: ε2/3 where ε is the machine precision.

RPARAM(3) = Relative function tolerance.

Default: $\max(10^{-10}, \varepsilon^{2/3})$, $\max(10^{-20}, \varepsilon^{2/3})$ in double where ε is the machine precision.

RPARAM(4) = Absolute function tolerance.

Default: Not used in UMINF.

RPARAM(5) = False convergence tolerance.

Default: Not used in UMINF.

`RPARAM`(6) = Maximum allowable step size.
Default: 1000 max(ε_1, ε_2) where

$$\varepsilon_1 = \sqrt{\sum_{i=1}^{n} \left(s_i t_i\right)^2}, \; \varepsilon_2 = \|s\|_2, \; s = \text{XSCALE}, \text{ and } t = \text{XGUESS}$$

`RPARAM`(7) = Size of initial trust region radius.
Default: Not used in `UMINF`.

If double precision is required, then `DU4INF` is called, and `RPARAM` is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to "Error Handling" in the Introduction.

Description

The routine `UMINF` uses a quasi-Newton method to find the minimum of a function $f(x)$ of n variables. Only function values are required. The problem is stated as follows:

$$\min_{x \in \mathbf{R}^n} f(x)$$

Given a starting point x_c, the search direction is computed according to the formula

$$d = -B^{-1} g_c$$

where B is a positive definite approximation of the Hessian and g_c is the gradient evaluated at x_c. A line search is then used to find a new point

$$x_n = x_c + \lambda d, \lambda > 0$$

such that

$$f(x_n) \le f(x_c) + \alpha g^T d, \quad \alpha \in (0, 0.5)$$

Finally, the optimality condition $\|g(x)\| = \varepsilon$ is checked where ε is a gradient tolerance.

When optimality is not achieved, B is updated according to the `BFGS` formula

$$B \leftarrow B - \frac{Bss^T B}{s^T Bs} + \frac{yy^T}{y^T s}$$

where $s = x_n - x_c$ and $y = g_n - g_c$. Another search direction is then computed to begin the next iteration. For more details, see Dennis and Schnabel (1983, Appendix A).

Since a finite-difference method is used to estimate the gradient, for some single precision calculations, an inaccurate estimate of the gradient may cause the algorithm to terminate at a noncritical point. In such cases, high precision arithmetic is recommended. Also, whenever the exact gradient can be easily provided, IMSL routine `UMING` (page 1202) should be used instead.

UMING

Minimizes a function of N variables using a quasi-Newton method and a user-supplied gradient.

Required Arguments

FCN — User-supplied SUBROUTINE to evaluate the function to be minimized. The usage is CALL FCN (N, X, F), where

N – Length of X. (Input)

X – Vector of length N at which point the function is evaluated. (Input)
X should not be changed by FCN.

F – The computed function value at the point X. (Output)

FCN must be declared EXTERNAL in the calling program.

GRAD — User-supplied SUBROUTINE to compute the gradient at the point X. The usage is CALL GRAD (N, X, G), where

N – Length of X and G. (Input)
X – Vector of length N at which point the function is evaluated. (Input)
X should not be changed by GRAD .
G – The gradient evaluated at the point X. (Output)

GRAD must be declared EXTERNAL in the calling program.

X — Vector of length N containing the computed solution. (Output)

Optional Arguments

N — Dimension of the problem. (Input)
Default: N = size (X,1).

XGUESS — Vector of length N containing the initial guess of the minimum. (Input)
Default: XGUESS = 0.0.

XSCALE — Vector of length N containing the diagonal scaling matrix for the variables. (Input)
XSCALE is used mainly in scaling the gradient and the distance between two points. In the absence of other information, set all entries to 1.0.
Default: XSCALE = 1.0.

FSCALE — Scalar containing the function scaling. (Input)
FSCALE is used mainly in scaling the gradient. In the absence of other information, set

`FSCALE` to 1.0.
Default: `FSCALE` = 1.0.

IPARAM — Parameter vector of length 7. (Input/Output)
Set `IPARAM`(1) to zero for default values of `IPARAM` and `RPARAM`. See Comment 4.
Default: `IPARAM` = 0.

RPARAM — Parameter vector of length 7. (Input/Output)
See Comment 4.

FVALUE — Scalar containing the value of the function at the computed solution. (Output)

FORTRAN 90 Interface

Generic: `CALL UMING (FCN, GRAD, X [,…])`

Specific: The specific interface names are `S_UMING` and `D_UMING`.

FORTRAN 77 Interface

Single: `CALL UMING (FCN, GRAD, N, XGUESS, XSCALE, FSCALE, IPARAM, RPARAM, X, FVALUE)`

Double: The double precision name is `DUMING`.

Example

The function

$$f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

is minimized. Default values for parameters are used.

```
      USE UMING_INT
      USE UMACH_INT
      INTEGER    N
      PARAMETER  (N=2)
!
      INTEGER    IPARAM(7), L, NOUT
      REAL       F, X(N), XGUESS(N)
      EXTERNAL   ROSBRK, ROSGRD
!
      DATA XGUESS/-1.2E0, 1.0E0/
!
      IPARAM(1) = 0
!                                 Minimize Rosenbrock function using
!                                 initial guesses of -1.2 and 1.0
      CALL UMING (ROSBRK, ROSGRD, X, XGUESS=XGUESS, IPARAM=IPARAM, FVALUE=F)
!                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, F, (IPARAM(L),L=3,5)
```

```
!
99999 FORMAT ('  The solution is ', 6X, 2F8.3, //, '  The function ', &
            'value is ', F8.3, //, '  The number of iterations is ', &
            10X, I3, /, '  The number of function evaluations is ', &
            I3, /, '  The number of gradient evaluations is ', I3)
!
      END
!
      SUBROUTINE ROSBRK (N, X, F)
      INTEGER    N
      REAL       X(N), F
!
      F = 1.0E2*(X(2)-X(1)*X(1))**2 + (1.0E0-X(1))**2
!
      RETURN
      END
!
      SUBROUTINE ROSGRD (N, X, G)
      INTEGER    N
      REAL       X(N), G(N)
!
      G(1) = -4.0E2*(X(2)-X(1)*X(1))*X(1) - 2.0E0*(1.0E0-X(1))
      G(2) = 2.0E2*(X(2)-X(1)*X(1))
!
      RETURN
      END
```

Output

```
The solution is          1.000   1.000

The function value is    0.000

The number of iterations is            18
The number of function evaluations is  31
The number of gradient evaluations is  22
```

Comments

1. Workspace may be explicitly provided, if desired, by use of `U2ING/DU2ING`. The reference is:

   `CALL U2ING (FCN, GRAD, N, XGUESS, XSCALE, FSCALE, IPARAM, RPARAM, X, FVALUE, WK)`

   The additional argument is

   WK — Work vector of length `N * (N + 8)`. `WK` contains the following information on output: The second `N` locations contain the last step taken. The third `N` locations contain the last Newton step. The fourth `N` locations contain an estimate of the gradient at the solution. The final N^2 locations contain the Cholesky factorization of a `BFGS` approximation to the Hessian at the solution.

2. Informational errors

Type	Code	
3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
4	2	The iterates appear to be converging to a noncritical point.
4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
4	5	Maximum number of gradient evaluations exceeded.
4	6	Five consecutive steps have been taken with the maximum step length.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or `STEPTL` is too big.
3	8	The last global step failed to locate a lower point than the current `X` value.

3. The first stopping criterion for `UMING` occurs when the infinity norm of the scaled gradient is less than the given gradient tolerance (`RPARAM`(1)). The second stopping criterion for `UMING` occurs when the scaled distance between the last two steps is less than the step tolerance (`RPARAM`(2)).

4. If the default parameters are desired for `UMING`, then set `IPARAM`(1) to zero and call routine `UMING` (page 1202). Otherwise, if any nondefault parameters are desired for `IPARAM` or `RPARAM`, then the following steps should be taken before calling `UMING`:

   `CALL U4INF` (`IPARAM`, `RPARAM`)
   Set nondefault values for desired `IPARAM`, `RPARAM` elements.

   Note that the call to `U4INF` will set `IPARAM` and `RPARAM` to their default values so only nondefault values need to be set above.

The following is a list of the parameters and the default values:

IPARAM — Integer vector of length 7.

`IPARAM`(1) = Initialization flag.

`IPARAM`(2) = Number of good digits in the function.
Default: Machine dependent.

`IPARAM`(3) = Maximum number of iterations.
Default: 100.

`IPARAM`(4) = Maximum number of function evaluations.
Default: 400.

`IPARAM`(5) = Maximum number of gradient evaluations.
Default: 400.

IPARAM(6) = Hessian initialization parameter

If IPARAM(6) = 0, the Hessian is initialized to the identity matrix; otherwise, it is initialized to a diagonal matrix containing

$$\max\left(\left|f(t)\right|, f_s\right) * s_i^2$$

on the diagonal where t = XGUESS, f_s = FSCALE, and s = XSCALE.

Default: 0.

IPARAM(7) = Maximum number of Hessian evaluations.

Default: Not used in UMING.

RPARAM — Real vector of length 7.

RPARAM(1) = Scaled gradient tolerance.

The i-th component of the scaled gradient at x is calculated as

$$\frac{\left|g_i\right| * \max\left(\left|x_i\right|, 1/s_i\right)}{\max\left(\left|f(x)\right|, f_s\right)}$$

where $g = \nabla f(x)$, s = XSCALE, and f_s = FSCALE.

Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

RPARAM(2) = Scaled step tolerance. (STEPTL)

The i-th component of the scaled step between two points x and y is computed as

$$\frac{\left|x_i - y_i\right|}{\max\left(\left|x_i\right|, 1/s_i\right)}$$

where s = XSCALE.

Default: $\varepsilon^{2/3}$ where ε is the machine precision.

RPARAM(3) = Relative function tolerance.

Default: $\max(10^{-10}, \varepsilon^{2/3})$, $\max(10^{-20}, \varepsilon^{2/3})$ in double where ε is the machine precision.

RPARAM(4) = Absolute function tolerance.

Default: Not used in UMING.

`RPARAM`(5) = False convergence tolerance.
Default: Not used in `UMING`.

`RPARAM`(6) = Maximum allowable step size.
Default: 1000 max(ε_1, ε_2) where

$$\varepsilon_1 = \sqrt{\sum_{i=1}^{n}(s_i t_i)^2}$$

$$\varepsilon_2 = \| s \|_2, \; s = \texttt{XSCALE}, \text{ and } t = \texttt{XGUESS}.$$

`RPARAM`(7) = Size of initial trust region radius.
Default: Not used in `UMING`.

If double precision is required, then `DU4INF` is called, and `RPARAM` is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to “Error Handling” in the Introduction.

Description

The routine `UMING` uses a quasi-Newton method to find the minimum of a function $f(x)$ of n variables. Function values and first derivatives are required. The problem is stated as follows:

$$\min_{x \in \mathbf{R}^n} f(x)$$

Given a starting point x_c, the search direction is computed according to the formula

$$d = -B^{-1} g_c$$

where B is a positive definite approximation of the Hessian and g_c is the gradient evaluated at x_c. A line search is then used to find a new point

$$x_n = x_c + \lambda d, \; \lambda > 0$$

such that

$$f(x_n) \le f(x_c) + \alpha g^T d, \quad \alpha \in (0, 0.5)$$

Finally, the optimality condition $\|g(x)\| = \varepsilon$ is checked where ε is a gradient tolerance.

When optimality is not achieved, B is updated according to the BFGS formula

$$B \leftarrow B - \frac{Bss^T B}{s^T Bs} + \frac{yy^T}{y^T s}$$

where $s = x_n - x_c$ and $y = g_n - g_c$. Another search direction is then computed to begin the next iteration. For more details, see Dennis and Schnabel (1983, Appendix A).

UMIDH

Minimizes a function of `N` variables using a modified Newton method and a finite-difference Hessian.

Required Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (N, X, F)`, where

`N` – Length of `X`. (Input)

`X` – Vector of length N at which point the function is evaluated. (Input)
`X` should not be changed by `FCN`.

`F` – The computed function value at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

GRAD — User-supplied `SUBROUTINE` to compute the gradient at the point `X`. The usage is `CALL GRAD (N, X, G)`, where

`N` – Length of `X` and `G`. (Input)

`X` – The point at which the gradient is evaluated. (Input)
`X` should not be changed by `GRAD`.

`G` – The gradient evaluated at the point `X`. (Output)

`GRAD` must be declared `EXTERNAL` in the calling program.

X — Vector of length `N` containing the computed solution. (Output)

Optional Arguments

N — Dimension of the problem. (Input)
Default: `N` = size (`X`,1).

XGUESS — Vector of length `N` containing initial guess. (Input)
Default: `XGUESS` = 0.0.

XSCALE — Vector of length `N` containing the diagonal scaling matrix for the variables. (Input)
`XSCALE` is used mainly in scaling the gradient and the distance between two points. In the absence of other information, set all entries to 1.0.
Default: `XSCALE` = 1.0.

FSCALE — Scalar containing the function scaling. (Input)
`FSCALE` is used mainly in scaling the gradient. In the absence of other information, set `FSCALE` to 1.0.
Default: `FSCALE` = 1.0.

IPARAM — Parameter vector of length 7. (Input/Output)
Set `IPARAM`(1) to zero for default values of `IPARAM` and `RPARAM`. See Comment 4.
Default: `IPARAM` = 0.

RPARAM — Parameter vector of length 7. (Input/Output)
See Comment 4.

FVALUE — Scalar containing the value of the function at the computed solution. (Output)

FORTRAN 90 Interface

Generic: `CALL UMIDH (FCN, GRAD, X [,…])`

Specific: The specific interface names are `S_UMIDH` and `D_UMIDH`.

FORTRAN 77 Interface

Single: `CALL UMIDH (FCN, GRAD, N, XGUESS, XSCALE, FSCALE, IPARAM, RPARAM, X, FVALUE)`

Double: The double precision name is `DUMIDH`.

Example

The function

$$f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

is minimized. Default values for parameters are used.

```
      USE UMIDH_INT
      USE UMACH_INT
      INTEGER    N
      PARAMETER  (N=2)
!
      INTEGER    IPARAM(7), L, NOUT
      REAL       F, X(N), XGUESS(N)
      EXTERNAL   ROSBRK, ROSGRD
!
      DATA XGUESS/-1.2E0, 1.0E0/
!
      IPARAM(1) = 0
!                                 Minimize Rosenbrock function using
!                                 initial guesses of -1.2 and 1.0
      CALL UMIDH (ROSBRK, ROSGRD, X, XGUESS=XGUESS, IPARAM=IPARAM, FVALUE=F)
!                                 Print results
```

```
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, F, (IPARAM(L),L=3,5), IPARAM(7)
!
99999 FORMAT ('  The solution is ', 6X, 2F8.3, //, '  The function ', &
            'value is ', F8.3, //, '  The number of iterations is ', &
            10X, I3, /, '  The number of function evaluations is ', &
            I3, /, '  The number of gradient evaluations is ', I3, /, &
            '  The number of Hessian evaluations is  ', I3)
!
      END
!
      SUBROUTINE ROSBRK (N, X, F)
      INTEGER    N
      REAL       X(N), F
!
      F = 1.0E2*(X(2)-X(1)*X(1))**2 + (1.0E0-X(1))**2
!
      RETURN
      END
!
      SUBROUTINE ROSGRD (N, X, G)
      INTEGER    N
      REAL       X(N), G(N)
!
      G(1) = -4.0E2*(X(2)-X(1)*X(1))*X(1) - 2.0E0*(1.0E0-X(1))
      G(2) = 2.0E2*(X(2)-X(1)*X(1))
!
      RETURN
      END
```

Output

```
The solution is          1.000   1.000

The function value is    0.000

The number of iterations is            21
The number of function evaluations is  30
The number of gradient evaluations is  22
The number of Hessian evaluations is   21
```

Comments

1. Workspace may be explicitly provided, if desired, by use of U2IDH/DU2IDH. The reference is:

   ```
   1CALL U2IDH (FCN, GRAD, N, XGUESS, XSCALE, FSCALE, IPARAM,
   RPARAM, X, FVALUE, WK)
   ```

   The additional argument is:

   WK — Work vector of length N * (N + 9). WK contains the following information on output: The second N locations contain the last step taken. The third N locations contain the last Newton step. The fourth N locations contain an estimate of the gradient at the solution. The final N^2 locations contain the Hessian at the approximate solution.

2. Informational errors

Type	Code	
3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
4	2	The iterates appear to be converging to a noncritical point.
4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
4	5	Maximum number of gradient evaluations exceeded.
4	6	Five consecutive steps have been taken with the maximum step length.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or STEPTL is too big.
4	7	Maximum number of Hessian evaluations exceeded.
3	8	The last global step failed to locate a lower point than the current X value.

3. The first stopping criterion for UMIDH occurs when the norm of the gradient is less than the given gradient tolerance (RPARAM(1)). The second stopping criterion for UMIDH occurs when the scaled distance between the last two steps is less than the step tolerance (RPARAM(2)).

4. If the default parameters are desired for UMIDH, then set IPARAM(1) to zero and call routine UMIDH. Otherwise, if any nondefault parameters are desired for IPARAM or RPARAM, then the following steps should be taken before calling UMIDH:

   CALL U4INF (IPARAM, RPARAM)

   Set nondefault values for desired IPARAM, RPARAM elements.

   Note that the call to U4INF will set IPARAM and RPARAM to their default values so only nondefault values need to be set above.

   The following is a list of the parameters and the default values:

   IPARAM — Integer vector of length 7.
   IPARAM(1) = Initialization flag.

   IPARAM(2) = Number of good digits in the function.
   Default: Machine dependent.

   IPARAM(3) = Maximum number of iterations.
   Default: 100.

IPARAM(4) = Maximum number of function evaluations.
Default: 400.

IPARAM(5) = Maximum number of gradient evaluations.
Default: 400.

IPARAM(6) = Hessian initialization parameter
Default: Not used in UMIDH.

IPARAM(7) = Maximum number of Hessian evaluations.
Default:100

RPARAM — Real vector of length 7.

RPARAM(1) = Scaled gradient tolerance.
The *i*-th component of the scaled gradient at *x* is calculated as

$$\frac{|g_i| * \max(|x_i|, 1/s_i)}{\max(|f(x)|, f_s)}$$

where $g = \nabla f(x)$, s = XSCALE, and f_s = FSCALE.
Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

RPARAM(2) = Scaled step tolerance. (STEPTL)

The *i*-th component of the scaled step between two points *x* and *y* is computed as

$$\frac{|x_i - y_i|}{\max(|x_i|, 1/s_i)}$$

where s = XSCALE.
Default: $\varepsilon^{2/3}$ where ε is the machine precision.

RPARAM(3) = Relative function tolerance.

Default: $\max(10^{-10}, \varepsilon^{2/3})$, $\max(10^{-20}, \varepsilon^{2/3})$ in double where ε is the machine precision.

RPARAM(4) = Absolute function tolerance.

Default: Not used in UMIDH.

RPARAM(5) = False convergence tolerance.

Default: 100ε where ε is the machine precision.

RPARAM(6) = Maximum allowable step size.

Default: 1000 max(ε_1, ε_2) where

$$\varepsilon_1 = \sqrt{\sum_{i=1}^{n}(s_i t_i)^2}$$

$\varepsilon_2 = \| s \|_2$, s = XSCALE, and t = XGUESS.

RPARAM(7) = Size of initial trust region radius.

Default: Based on initial scaled Cauchy step.

If double precision is required, then DU4INF is called, and RPARAM is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to “Error Handling” in the Introduction.

Description

The routine UMIDH uses a modified Newton method to find the minimum of a function $f(x)$ of n variables. First derivatives must be provided by the user. The algorithm computes an optimal locally constrained step (Gay 1981) with a trust region restriction on the step. It handles the case that the Hessian is indefinite and provides a way to deal with negative curvature. For more details, see Dennis and Schnabel (1983, Appendix A) and Gay (1983).

Since a finite-difference method is used to estimate the Hessian for some single precision calculations, an inaccurate estimate of the Hessian may cause the algorithm to terminate at a noncritical point. In such cases, high precision arithmetic is recommended. Also, whenever the exact Hessian can be easily provided, IMSL routine UMIAH (page 1213) should be used instead.

UMIAH

Minimizes a function of N variables using a modified Newton method and a user-supplied Hessian.

Required Arguments

FCN — User-supplied SUBROUTINE to evaluate the function to be minimized. The usage is CALL FCN (N, X, F), where

N – Length of X. (Input)

`X` – Vector of length `N` at which point the function is evaluated. (Input)
`X` should not be changed by `FCN`.

`F` – The computed function value at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

GRAD — User-supplied `SUBROUTINE` to compute the gradient at the point `X`. The usage is `CALL GRAD (N, X, G)`, where

`N` – Length of `X` and `G`. (Input)

`X` – Vector of length N at which point the gradient is evaluated. (Input)
`X` should not be changed by `GRAD`.

`G` – The gradient evaluated at the point `X`. (Output)

`GRAD` must be declared `EXTERNAL` in the calling program.

HESS — User-supplied `SUBROUTINE` to compute the Hessian at the point `X`. The usage is `CALL HESS (N, X, H, LDH)`, where

`N` – Length of `X`. (Input)

`X` – Vector of length N at which point the Hessian is evaluated. (Input)
`X` should not be changed by `HESS`.

`H` – The Hessian evaluated at the point `X`. (Output)

`LDH` – Leading dimension of `H` exactly as specified in the dimension statement of the calling program. `LDH` must be equal to `N` in this routine. (Input)

`HESS` must be declared `EXTERNAL` in the calling program.

X — Vector of length `N` containing the computed solution. (Output)

Optional Arguments

N — Dimension of the problem. (Input)
Default: `N` = size `(X`,1).

XGUESS — Vector of length `N` containing initial guess. (Input)
Default: `XGUESS` = 0.0.

XSCALE — Vector of length `N` containing the diagonal scaling matrix for the variables. (Input)
`XSCALE` is used mainly in scaling the gradient and the distance between two points. In

the absence of other information, set all entries to 1.0.
Default: `XSCALE` = 1.0.

FSCALE — Scalar containing the function scaling. (Input)
`FSCALE` is used mainly in scaling the gradient. In the absence of other information, set `FSCALE` to 1.0.
Default: `FSCALE` = 1.0.

IPARAM — Parameter vector of length 7. (Input/Output)
Set `IPARAM`(1) to zero for default values of `IPARAM` and `RPARAM`. See Comment 4.
Default: `IPARAM` = 0.

RPARAM — Parameter vector of length 7. (Input/Output)
See Comment 4.

FVALUE — Scalar containing the value of the function at the computed solution. (Output)

FORTRAN 90 Interface

Generic: `CALL UMIAH (FCN, GRAD, HESS, X, [,…])`

Specific: The specific interface names are `S_UMIAH` and `D_UMIAH`.

FORTRAN 77 Interface

Single:
```
CALL UMIAH (FCN, GRAD, HESS, N, XGUESS, XSCALE, FSCALE,
IPARAM, RPARAM, X, FVALUE)
```

Double: The double precision name is `DUMIAH`.

Example

The function

$$f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

is minimized. Default values for parameters are used.

```
      USE UMIAH_INT
      USE UMACH_INT
      INTEGER    N
      PARAMETER  (N=2)
!
      INTEGER    IPARAM(7), L, NOUT
      REAL       F, FSCALE, RPARAM(7), X(N), &
                XGUESS(N), XSCALE(N)
      EXTERNAL   ROSBRK, ROSGRD, ROSHES
!
      DATA XGUESS/-1.2E0, 1.0E0/, XSCALE/1.0E0, 1.0E0/, FSCALE/1.0E0/
!
```

```
      IPARAM(1) = 0
!                                  Minimize Rosenbrock function using
!                                  initial guesses of -1.2 and 1.0
       CALL UMIAH (ROSBRK, ROSGRD, ROSHES, X, XGUESS=XGUESS, IPARAM=IPARAM, &
                   FVALUE=F)
!                                  Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, F, (IPARAM(L),L=3,5), IPARAM(7)
!
99999 FORMAT (’  The solution is ’, 6X, 2F8.3, //, ’  The function ’, &
            ’value is ’, F8.3, //, ’  The number of iterations is ’, &
            10X, I3, /, ’  The number of function evaluations is ’, &
            I3, /, ’  The number of gradient evaluations is ’, I3, /, &
            ’  The number of Hessian evaluations is  ’, I3)
!
      END
!
      SUBROUTINE ROSBRK (N, X, F)
      INTEGER    N
      REAL       X(N), F
!
      F = 1.0E2*(X(2)-X(1)*X(1))**2 + (1.0E0-X(1))**2
!
      RETURN
      END
!
      SUBROUTINE ROSGRD (N, X, G)
      INTEGER    N
      REAL       X(N), G(N)
!
      G(1) = -4.0E2*(X(2)-X(1)*X(1))*X(1) - 2.0E0*(1.0E0-X(1))
      G(2) = 2.0E2*(X(2)-X(1)*X(1))
!
      RETURN
      END
!
      SUBROUTINE ROSHES (N, X, H, LDH)
      INTEGER    N, LDH
      REAL       X(N), H(LDH,N)
!
      H(1,1) = -4.0E2*X(2) + 1.2E3*X(1)*X(1) + 2.0E0
      H(2,1) = -4.0E2*X(1)
      H(1,2) = H(2,1)
      H(2,2) = 2.0E2
!
      RETURN
      END
```

Output

```
The solution is          1.000   1.000

The function value is    0.000

The number of iterations is            21
The number of function evaluations is  31
The number of gradient evaluations is  22
The number of Hessian evaluations is   21
```

Comments

1. Workspace may be explicitly provided, if desired, by use of `U2IAH/DU2IAH`. The reference is:

   ```
   CALL U2IAH (FCN, GRAD, HESS, N, XGUESS, XSCALE, FSCALE, IPARAM,
   RPARAM, X, FVALUE, WK)
   ```

   The additional argument is:

   WK — Work vector of length `N * (N + 9)`. `WK` contains the following information on output: The second `N` locations contain the last step taken. The third `N` locations contain the last Newton step. The fourth `N` locations contain an estimate of the gradient at the solution. The final N^2 locations contain the Hessian at the approximate solution.

2. Informational errors

Type	Code	
3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
4	2	The iterates appear to be converging to a noncritical point.
4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
4	5	Maximum number of gradient evaluations exceeded.
4	6	Five consecutive steps have been taken with the maximum step length.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or `STEPTL` is too big.
4	7	Maximum number of Hessian evaluations exceeded.
3	8	The last global step failed to locate a lower point than the current `X` value.

3. The first stopping criterion for `UMIAH` occurs when the norm of the gradient is less than the given gradient tolerance (`RPARAM`(1)). The second stopping criterion for `UMIAH` occurs when the scaled distance between the last two steps is less than the step tolerance (`RPARAM`(2)).

4. If the default parameters are desired for `UMIAH`, then set `IPARAM`(1) to zero and call the routine `UMIAH`. Otherwise, if any nondefault parameters are desired for `IPARAM` or `RPARAM`, then the following steps should be taken before calling `UMIAH`:

   `CALL U4INF` (`IPARAM`, `RPARAM`)

   Set nondefault values for desired `IPARAM`, `RPARAM` elements.

Note that the call to U4INF will set IPARAM and RPARAM to their default values so only nondefault values need to be set above.

The following is a list of the parameters and the default values:

IPARAM — Integer vector of length 7.
IPARAM(1) = Initialization flag.

IPARAM(2) = Number of good digits in the function.
Default: Machine dependent.

IPARAM(3) = Maximum number of iterations.
Default: 100.

IPARAM(4) = Maximum number of function evaluations.
Default: 400.

IPARAM(5) = Maximum number of gradient evaluations.
Default: 400.

IPARAM(6) = Hessian initialization parameter
Default: Not used in UMIAH.

IPARAM(7) = Maximum number of Hessian evaluations.
Default: 100.

RPARAM — Real vector of length 7.
RPARAM(1) = Scaled gradient tolerance.
The i-th component of the scaled gradient at x is calculated as

$$\frac{|g_i| * \max(|x_i|, 1/s_i)}{\max(|f(x)|, f_s)}$$

where $g = \nabla f(x)$, s = XSCALE, and f_s = FSCALE.
Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

RPARAM(2) = Scaled step tolerance. (STEPTL)
The i-th component of the scaled step between two points x and y is computed as

$$\frac{|x_i - y_i|}{\max(|x_i|, 1/s_i)}$$

where s = `XSCALE`.
Default: $\varepsilon^{2/3}$ where ε is the machine precision.

`RPARAM`(3) = Relative function tolerance.
Default: $\max(10^{-10}, \varepsilon^{2/3})$, $\max(10^{-20}, \varepsilon^{2/3})$ in double where ε is the machine precision.

`RPARAM`(4) = Absolute function tolerance.
Default: Not used in `UMIAH`.

`RPARAM`(5) = False convergence tolerance.
Default: 100ε where ε is the machine precision.

`RPARAM`(6) = Maximum allowable step size.
Default: 1000 $\max(\varepsilon_1, \varepsilon_2)$ where

$$\varepsilon_1 = \sqrt{\sum_{i=1}^{n} \left(s_i t_i\right)^2}$$

$\varepsilon_2 = \| s \|_2$, s = `XSCALE`, and t = `XGUESS`.

`RPARAM`(7) = Size of initial trust region radius.
Default: based on the initial scaled Cauchy step.

If double precision is required, then `DU4INF` is called, and `RPARAM` is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to “Error Handling” in the Introduction.

Description

The routine `UMIAH` uses a modified Newton method to find the minimum of a function $f(x)$ of n variables. First and second derivatives must be provided by the user. The algorithm computes an optimal locally constrained step (Gay 1981) with a trust region restriction on the step. This algorithm handles the case where the Hessian is indefinite and provides a way to deal with negative curvature. For more details, see Dennis and Schnabel (1983, Appendix A) and Gay (1983).

UMCGF

Minimizes a function of `N` variables using a conjugate gradient algorithm and a finite-difference gradient.

Required Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (N, X, F)`, where

`N` – Length of `X`. (Input)

`X` – The point at which the function is evaluated. (Input)
`X` should not be changed by `FCN`.

`F` – The computed function value at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

DFPRED — A rough estimate of the expected reduction in the function. (Input)
`DFPRED` is used to determine the size of the initial change to `X`.

X — Vector of length `N` containing the computed solution. (Output)

Optional Arguments

N — Dimension of the problem. (Input)
Default: `N` = size (`X`,1).

XGUESS — Vector of length `N` containing the initial guess of the minimum. (Input)
Default: `XGUESS` = 0.0.

XSCALE — Vector of length `N` containing the diagonal scaling matrix for the variables. (Input)
Default: `XSCALE` = 1.0.

GRADTL — Convergence criterion. (Input)
The calculation ends when the sum of squares of the components of `G` is less than `GRADTL`.
Default: `GRADTL` = 1.e-4.

MAXFN — Maximum number of function evaluations. (Input)
If `MAXFN` is set to zero, then no restriction on the number of function evaluations is set.
Default: `MAXFN` = 0.

G — Vector of length `N` containing the components of the gradient at the final parameter estimates. (Output)

FVALUE — Scalar containing the value of the function at the computed solution. (Output)

FORTRAN 90 Interface

Generic: `CALL UMCGF (FCN, DFPRED, X [,…])`

Specific: The specific interface names are `S_UMCGF` and `D_UMCGF`.

FORTRAN 77 Interface

Single: `CALL UMCGF (FCN, N, XGUESS, XSCALE, GRADTL, MAXFN, DFPRED, X, G, FVALUE)`

Double: The double precision name is `DUMCGF`.

Example

The function

$$f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

is minimized and the solution is printed.

```
      USE UMCGF_INT
      USE UMACH_INT
!                                 Declaration of variables
      INTEGER    N
      PARAMETER  (N=2)
!
      INTEGER    I, MAXFN, NOUT
      REAL       DFPRED, FVALUE, G(N), GRADTL, X(N), XGUESS(N)
      EXTERNAL   ROSBRK
!
      DATA XGUESS/-1.2E0, 1.0E0/
!
      DFPRED = 0.2
      GRADTL = 1.0E-6
      MAXFN  = 100
!                                 Minimize the Rosenbrock function
      CALL UMCGF (ROSBRK, DFPRED, X, XGUESS=XGUESS, GRADTL=GRADTL, &
                 G=G, FVALUE=FVALUE)
!                                 Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) (X(I),I=1,N), FVALUE, (G(I),I=1,N)
99999 FORMAT ('  The solution is ', 2F8.3, //, '  The function ', &
            'evaluated at the solution is ', F8.3, //, '  The ', &
            'gradient is ', 2F8.3, /)
!
      END
!
      SUBROUTINE ROSBRK (N, X, F)
      INTEGER    N
      REAL       X(N), F
!
      F = 1.0E2*(X(2)-X(1)*X(1))**2 + (1.0E0-X(1))**2
      RETURN
      END
```

Output

```
The solution is    0.999   0.998

The function evaluated at the solution is    0.000

The gradient is   -0.001   0.000
```

Comments

1. Workspace may be explicitly provided, if desired, by use of `U2CGF/DU2CGF`. The reference is:

   ```
   CALL U2CGF (FCN, N, XGUESS, XSCALE, GRADTL, MAXFN, DFPRED, X, G,
   FVALUE, S, RSS, RSG, GINIT, XOPT, GOPT)
   ```

   The additional arguments are as follows:

   S — Vector of length `N` used for the search direction in each iteration.

   RSS — Vector of length `N` containing conjugacy information.

   RSG — Vector of length `N` containing conjugacy information.

   GINIT — Vector of length `N` containing the gradient values at the start of an iteration.

   XOPT — Vector of length `N` containing the parameter values that yield the least calculated value for `FVALUE`.

   GOPT — Vector of length `N` containing the gradient values that yield the least calculated value for `FVALUE`.

2. Informational errors

Type	Code	
4	1	The line search of an integration was abandoned. This error may be caused by an error in gradient.
4	2	The calculation cannot continue because the search is uphill.
4	3	The iteration was terminated because `MAXFN` was exceeded.
3	4	The calculation was terminated because two consecutive iterations failed to reduce the function.

3. Because of the close relation between the conjugate-gradient method and the method of steepest descent, it is very helpful to choose the scale of the variables in a way that balances the magnitudes of the components of a typical gradient vector. It can be particularly inefficient if a few components of the gradient are much larger than the rest.

4. If the value of the parameter `GRADTL` in the argument list of the routine is set to zero, then the subroutine will continue its calculation until it stops reducing the objective function. In this case, the usual behavior is that changes in the objective function become dominated by computer rounding errors before precision is lost in the gradient

vector. Therefore, because the point of view has been taken that the user requires the least possible value of the function, a value of the objective function that is small due to computer rounding errors can prevent further progress. Hence, the precision in the final values of the variables may be only about half the number of significant digits in the computer arithmetic, but the least value of `FVALUE` is usually found to be quite accurate.

Description

The routine `UMCGF` uses a conjugate gradient method to find the minimum of a function $f(x)$ of n variables. Only function values are required.

The routine is based on the version of the conjugate gradient algorithm described in Powell (1977). The main advantage of the conjugate gradient technique is that it provides a fast rate of convergence without the storage of any matrices. Therefore, it is particularly suitable for unconstrained minimization calculations where the number of variables is so large that matrices of dimension n cannot be stored in the main memory of the computer. For smaller problems, however, a routine such as routine `UMINF` (page 1196), is usually more efficient because each iteration makes use of additional information from previous iterations.

Since a finite-difference method is used to estimate the gradient for some single precision calculations, an inaccurate estimate of the gradient may cause the algorithm to terminate at a noncritical point. In such cases, high precision arithmetic is recommended. Also, whenever the exact gradient can be easily provided, routine `UMCGG` (page 1223) should be used instead.

UMCGG

Minimizes a function of `N` variables using a conjugate gradient algorithm and a user-supplied gradient.

Required Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (N, X, F)`, where

`N` – Length of `X`. (Input)

`X` – The point at which the function is evaluated. (Input)
`X` should not be changed by `FCN`.

`F` – The computed function value at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

GRAD — User-supplied `SUBROUTINE` to compute the gradient at the point `X`. The usage is `CALL GRAD (N, X, G)`, where

`N` – Length of `X` and `G`. (Input)

`X` – The point at which the gradient is evaluated. (Input)
`X` should not be changed by `GRAD`.

`G` – The gradient evaluated at the point `X`. (Output)

`GRAD` must be declared `EXTERNAL` in the calling program.

DFPRED — A rough estimate of the expected reduction in the function. (Input) `DFPRED` is used to determine the size of the initial change to `X`.

X — Vector of length `N` containing the computed solution. (Output)

Optional Arguments

N — Dimension of the problem. (Input)
Default: `N` = size (`X`,1).

XGUESS — Vector of length `N` containing the initial guess of the minimum. (Input)
Default: `XGUESS` = 0.0.

GRADTL — Convergence criterion. (Input)
The calculation ends when the sum of squares of the components of `G` is less than `GRADTL`.
Default: `GRADTL` = 1.e-4.

MAXFN — Maximum number of function evaluations. (Input)
Default: `MAXFN` = 100.

G — Vector of length `N` containing the components of the gradient at the final parameter estimates. (Output)

FVALUE — Scalar containing the value of the function at the computed solution. (Output)

FORTRAN 90 Interface

Generic: `CALL UMCGG (FCN, GRAD, DFPRED, X [,…])`

Specific: The specific interface names are `S_UMCGG` and `D_UMCGG`.

FORTRAN 77 Interface

Single:
```
CALL UMCGG (FCN, GRAD, N, XGUESS, GRADTL, MAXFN, DFPRED, X,
G, FVALUE)
```

Double: The double precision name is `DUMCGG`.

Example

The function

$$f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

is minimized and the solution is printed.

```
      USE UMCGG_INT
      USE UMACH_INT
!                                 Declaration of variables
      INTEGER    N
      PARAMETER  (N=2)
!
      INTEGER    I, NOUT
      REAL       DFPRED, FVALUE, G(N), GRADTL, X(N), &
                XGUESS(N)
      EXTERNAL   ROSBRK, ROSGRD
!
      DATA XGUESS/-1.2E0, 1.0E0/
!
      DFPRED = 0.2
      GRADTL = 1.0E-7
!                                 Minimize the Rosenbrock function
      CALL UMCGG (ROSBRK, ROSGRD, DFPRED, X, XGUESS=XGUESS, &
                 GRADTL=GRADTL, G=G, FVALUE=FVALUE)
!                                 Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) (X(I),I=1,N), FVALUE, (G(I),I=1,N)
99999 FORMAT ('  The solution is ', 2F8.3, //, '  The function ', &
            'evaluated at the solution is ', F8.3, //, '  The ', &
            'gradient is ', 2F8.3, /)
!
      END
!
      SUBROUTINE ROSBRK (N, X, F)
      INTEGER    N
      REAL       X(N), F
!
      F = 1.0E2*(X(2)-X(1)*X(1))**2 + (1.0E0-X(1))**2
      RETURN
      END
!
      SUBROUTINE ROSGRD (N, X, G)
      INTEGER    N
      REAL       X(N), G(N)
!
      G(1) = -4.0E2*(X(2)-X(1)*X(1))*X(1) - 2.0E0*(1.0E0-X(1))
      G(2) = 2.0E2*(X(2)-X(1)*X(1))
!
      RETURN
      END
```

Output

```
The solution is    1.000   1.000

The function evaluated at the solution is    0.000

The gradient is    0.000   0.000
```

Comments

1. Workspace may be explicitly provided, if desired, by use of `U2CGG/DU2CGG`. The reference is:

   ```
   CALL U2CGG (FCN, GRAD, N, XGUESS, GRADTL, MAXFN, DFPRED, X, G,
   FVALUE, S, RSS, RSG, GINIT, XOPT, GOPT)
   ```

   The additional arguments are as follows:

   S — Vector of length `N` used for the search direction in each iteration.

   RSS — Vector of length `N` containing conjugacy information.

   RSG — Vector of length `N` containing conjugacy information.

   GINIT — Vector of length `N` containing the gradient values at the start on an iteration.

   XOPT — Vector of length `N` containing the parameter values which yield the least calculated value for `FVALUE`.

   GOPT — Vector of length N containing the gradient values which yield the least calculated value for `FVALUE`.

2. Informational errors

Type	Code	
4	1	The line search of an integration was abandoned. This error may be caused by an error in gradient.
4	2	The calculation cannot continue because the search is uphill.
4	3	The iteration was terminated because `MAXFN` was exceeded.
3	4	The calculation was terminated because two consecutive iterations failed to reduce the function.

3. The routine includes no thorough checks on the part of the user program that calculates the derivatives of the objective function. Therefore, because derivative calculation is a frequent source of error, the user should verify independently the correctness of the derivatives that are given to the routine.

4. Because of the close relation between the conjugate-gradient method and the method of steepest descent, it is very helpful to choose the scale of the variables in a way that balances the magnitudes of the components of a typical gradient vector. It can be particularly inefficient if a few components of the gradient are much larger than the rest.

5. If the value of the parameter `GRADTL` in the argument list of the routine is set to zero, then the subroutine will continue its calculation until it stops reducing the objective function. In this case, the usual behavior is that changes in the objective function become dominated by computer rounding errors before precision is lost in the gradient vector. Therefore, because the point of view has been taken that the user requires the least possible value of the function, a value of the objective function that is small due to computer rounding errors can prevent further progress. Hence, the precision in the final values of the variables may be only about half the number of significant digits in the computer arithmetic, but the least value of `FVALUE` is usually found to be quite accurate.

Description

The routine `UMCGG` uses a conjugate gradient method to find the minimum of a function $f(x)$ of n variables. Function values and first derivatives are required.

The routine is based on the version of the conjugate gradient algorithm described in Powell (1977). The main advantage of the conjugate gradient technique is that it provides a fast rate of convergence without the storage of any matrices. Therefore, it is particularly suitable for unconstrained minimization calculations where the number of variables is so large that matrices of dimension n cannot be stored in the main memory of the computer. For smaller problems, however, a subroutine such as IMSL routine `UMING` (page 1202), is usually more efficient because each iteration makes use of additional information from previous iterations.

UMPOL

Minimizes a function of `N` variables using a direct search polytope algorithm.

Required Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (N, X, F)`, where

`N` – Length of `X`. (Input)

`X` – Vector of length `N` at which point the function is evaluated. (Input)
`X` should not be changed by `FCN`.

F – The computed function value at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

X — Real vector of length `N` containing the best estimate of the minimum found. (Output)

Optional Arguments

N — Dimension of the problem. (Input)
Default: `N` = size (`X`,1).

XGUESS — Real vector of length `N` which contains an initial guess to the minimum. (Input)
Default: `XGUESS` = 0.0.

S — On input, real scalar containing the length of each side of the initial simplex. (Input/Output)
If no reasonable information about `S` is known, `S` could be set to a number less than or equal to zero and `UMPOL` will generate the starting simplex from the initial guess with a random number generator. On output, the average distance from the vertices to the centroid that is taken to be the solution; see Comment 4.
Default: `S` = 0.0.

FTOL — First convergence criterion. (Input)
The algorithm stops when a relative error in the function values is less than `FTOL`, i.e. when (`F`(worst) – `F`(best)) < `FTOL` * (1 + `ABS`(`F`(best))) where `F`(worst) and `F`(best) are the function values of the current worst and best points, respectively. Second convergence criterion. The algorithm stops when the standard deviation of the function values at the `N` + 1 current points is less than `FTOL`. If the subroutine terminates prematurely, try again with a smaller value for `FTOL`.
Default: `FTOL` = 1.e-7.

MAXFCN — On input, maximum allowed number of function evaluations. (Input/ Output)
On output, actual number of function evaluations needed.
Default: `MAXFCN` = 200.

FVALUE — Function value at the computed solution. (Output)

FORTRAN 90 Interface

Generic: `CALL UMPOL (FCN, X [,…])`

Specific: The specific interface names are `S_UMPOL` and `D_UMPOL`.

FORTRAN 77 Interface

Single: `CALL UMPOL (FCN, N, XGUESS, S, FTOL, MAXFCN, X, FVALUE)`

Double: The double precision name is `DUMPOL`.

Example

The function

$$f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

is minimized and the solution is printed.

```
      USE UMPOL_INT
      USE UMACH_INT
!                                 Variable declarations
```

```
      INTEGER    N
      PARAMETER  (N=2)
!
      INTEGER    K, NOUT
      REAL       FTOL, FVALUE, S, X(N), XGUESS(N)
      EXTERNAL   FCN
!
!                                  Initializations
!                                  XGUESS = ( -1.2, 1.0)
!
      DATA XGUESS/-1.2, 1.0/
!
      FTOL   = 1.0E-10
      S      = 1.0
!
      CALL UMPOL (FCN, X, XGUESS=XGUESS, S=S, FTOL=FTOL,&
                  FVALUE=FVALUE)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) (X(K),K=1,N), FVALUE
99999 FORMAT ('  The best estimate for the minimum value of the', /, &
            '  function is X = (', 2(2X,F4.2), ')', /, '  with ', &
            'function value FVALUE = ', E12.6)
!
      END
!                                  External function to be minimized
      SUBROUTINE FCN (N, X, F)
      INTEGER    N
      REAL       X(N), F
!
      F = 100.0*(X(1)*X(1)-X(2))**2 + (1.0-X(1))**2
      RETURN
      END
```

Output

```
The best estimate for the minimum value of the
function is X = (  1.00  1.00)
with function value FVALUE = 0.502496E-10
```

Comments

1. Workspace may be explicitly provided, if desired, by use of `U2POL/DU2POL`. The reference is:

   ```
   CALL U2POL (FCN, N, XGUESS, S, FTOL, MAXFCN, X,
   FVALUE, WK)
   ```

   The additional argument is:

   WK — Real work vector of length `N**2 + 5 * N + 1`.

2. Informational error

   Type   Code

4	1	Maximum number of function evaluations exceeded.

3. Since `UMPOL` uses only function value information at each step to determine a new approximate minimum, it could be quite ineficient on smooth problems compared to other methods such as those implemented in routine `UMINF` that takes into account derivative information at each iteration. Hence, routine `UMPOL` should only be used as a last resort. Briefly, a set of `N` + 1 points in an `N`-dimensional space is called a simplex. The minimization process iterates by replacing the point with the largest function value by a new point with a smaller function value. The iteration continues until all the points cluster sufficiently close to a minimum.

4. The value returned in `S` is useful for assessing the flatness of the function near the computed minimum. The larger its value for a given value of `FTOL`, the flatter the function tends to be in the neighborhood of the returned point.

Description

The routine `UMPOL` uses the polytope algorithm to find a minimum point of a function $f(x)$ of n variables. The polytope method is based on function comparison; no smoothness is assumed. It starts with $n + 1$ points $x_1, x_2, \ldots, x_{n+1}$. At each iteration, a new point is generated to replace the worst point x_j, which has the largest function value among these $n + 1$ points. The new point is constructed by the following formula:

$$x_k = c + \alpha(c - x_j)$$

where

$$c = \frac{1}{n}\sum_{i \neq j} x_i$$

and α ($\alpha > 0$) is the *reflection coefficient.*

When x_k is a best point, that is $f(x_k) \le f(x_i)$ for $i = 1, \ldots, n + 1$, an expansion point is computed $x_e = c + \beta(x_k - c)$ where $\beta(\beta > 1)$ is called the *expansion coefficient*. If the new point is a worst point, then the polytope would be contracted to get a better new point. If the contraction step is unsuccessful, the polytope is shrunk by moving the vertices halfway toward current best point. This procedure is repeated until one of the following stopping criteria is satisfied:

Criterion 1:

$$f_{best} - f_{worst} \le \varepsilon_f(1. + |f_{best}|)$$

Criterion 2:

$$\sum_{i=1}^{n+1}\left(f_i - \frac{\sum_{j=1}^{n+1} f_j}{n+1}\right)^2 \le \varepsilon_f$$

where $f_i = f(x_i)$, $f_j = f(x_j)$, and ε_f is a given tolerance. For a complete description, see Nelder and Mead (1965) or Gill et al. (1981).

UNLSF

Solves a nonlinear least-squares problem using a modified Levenberg-Marquardt algorithm and a finite-difference Jacobian.

Required Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function that defines the least-squares problem. The usage is `CALL FCN (M, N, X, F)`, where

`M` – Length of F. (Input)

`N` – Length of X. (Input)

`X` – Vector of length `N` at which point the function is evaluated. (Input)
X should not be changed by `FCN`.

`F` – Vector of length `M` containing the function values at `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

M — Number of functions. (Input)

X — Vector of length `N` containing the approximate solution. (Output)

Optional Arguments

N — Number of variables. `N` must be less than or equal to `M`. (Input)
Default: `N` = size (`X`,1).

XGUESS — Vector of length `N` containing the initial guess. (Input)
Default: `NDEG` = size (`COEFF`,1) – 1.

XSCALE — Vector of length `N` containing the diagonal scaling matrix for the variables. (Input)
`XSCALE` is used mainly in scaling the gradient and the distance between two points. By default, the values for `XSCALE` are set internally. See `IPARAM`(6) in Comment 4.
Default: `XSCALE` = 1.0.

FSCALE — Vector of length `M` containing the diagonal scaling matrix for the functions. (Input)
`FSCALE` is used mainly in scaling the gradient. In the absence of other information, set all entries to 1.0.
Default: `FSCALE` = 1.0.

IPARAM — Parameter vector of length 6.   (Input/Output)
Set `IPARAM`(1) to zero for default values of `IPARAM` and `RPARAM`. See Comment 4.
Default: `IPARAM` = 0.

RPARAM — Parameter vector of length 7.   (Input/Output)
See Comment 4.

FVEC — Vector of length `M` containing the residuals at the approximate solution.   (Output)

FJAC — `M` by `N` matrix containing a finite difference approximate Jacobian at the approximate solution.   (Output)

LDFJAC — Leading dimension of `FJAC` exactly as specified in the dimension statement of the calling program.   (Input)
Default: `LDFJAC` = size (`FJAC`,1).

FORTRAN 90 Interface

Generic: `CALL UNLSF (FCN, M, X [,…])`

Specific: The specific interface names are `S_UNLSF` and `D_UNLSF`.

FORTRAN 77 Interface

Single:
```
CALL UNLSF (FCN, M, N, XGUESS, XSCALE, FSCALE, IPARAM,
RPARAM, X, FVEC, FJAC, LDFJAC)
```

Double: The double precision name is `DUNLSF`.

Example

The nonlinear least squares problem

$$\min_{x \in \mathbf{R}^2} \frac{1}{2} \sum_{i=1}^{2} f_i(x)^2$$

where

$$f_1(x) = 10\left(x_2 - x_1^2\right) \text{ and } f_2(x) = \left(1 - x_1\right)$$

is solved. `RPARAM`(4) is changed to a non-default value.

```
      USE UNLSF_INT
      USE UMACH_INT
      USE U4LSF_INT
!                                  Declaration of variables
      INTEGER    LDFJAC, M, N
      PARAMETER  (LDFJAC=2, M=2, N=2)
!
      INTEGER    IPARAM(6), NOUT
      REAL       FVEC(M), RPARAM(7),X(N), XGUESS(N)
      EXTERNAL   ROSBCK
!                                 Compute the least squares for the
!                                 Rosenbrock function.
      DATA XGUESS/-1.2E0, 1.0E0/
!
!                                 Relax the first stopping criterion by
!                                 calling U4LSF and scaling the
!                                 absolute function tolerance by 10.
      CALL U4LSF (IPARAM, RPARAM)
      RPARAM(4) = 10.0E0*RPARAM(4)
!
      CALL UNLSF (ROSBCK, M, X,XGUESS=XGUESS, IPARAM=IPARAM, &
                 RPARAM=RPARAM, FVEC=FVEC)
!                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, FVEC, IPARAM(3), IPARAM(4)
!
99999 FORMAT ('  The solution is ', 2F9.4, //, '  The function ', &
            'evaluated at the solution is ', /, 18X, 2F9.4, //, &
            '  The number of iterations is ', 10X, I3, /, '  The ', &
            'number of function evaluations is ', I3, /)
      END
!
      SUBROUTINE ROSBCK (M, N, X, F)
      INTEGER    M, N
      REAL       X(N), F(M)
!
      F(1) = 10.0E0*(X(2)-X(1)*X(1))
      F(2) = 1.0E0 - X(1)
      RETURN
      END
```

Output

```
The solution is    1.0000   1.0000

The function evaluated at the solution is
0.0000   0.0000

The number of iterations is           24
The number of function evaluations is  33
```

Comments

1. Workspace may be explicitly provided, if desired, by use of U2LSF/DU2LSF. The reference is:

   ```
   CALL U2LSF (FCN, M, N, XGUESS, XSCALE, FSCALE, IPARAM, RPARAM,
   X, FVEC, FJAC, LDFJAC, WK, IWK)
   ```

   The additional arguments are as follows:

   WK — Real work vector of length 9 * N + 3 * M – 1. WK contains the following information on output: The second N locations contain the last step taken. The third `N` locations contain the last Gauss-Newton step. The fourth `N` locations contain an estimate of the gradient at the solution.

IWK — Integer work vector of length `N` containing the permutations used in the `QR` factorization of the Jacobian at the solution.

2. Informational errors

Type	Code	
3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
3	2	The iterates appear to be converging to a noncritical point.
4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
3	6	Five consecutive steps have been taken with the maximum step length.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or `STEPTL` is too big.

3. The first stopping criterion for `UNLSF` occurs when the norm of the function is less than the absolute function tolerance (`RPARAM`(4)). The second stopping criterion occurs when the norm of the scaled gradient is less than the given gradient tolerance (`RPARAM`(1)). The third stopping criterion for `UNLSF` occurs when the scaled distance between the last two steps is less than the step tolerance (`RPARAM`(2)).

4. If the default parameters are desired for `UNLSF`, then set `IPARAM`(1) to zero and call the routine `UNLSF`. Otherwise, if any nondefault parameters are desired for `IPARAM` or `RPARAM`, then the following steps should be taken before calling `UNLSF`:

`CALL U4LSF (IPARAM, RPARAM)`
Set nondefault values for desired `IPARAM`, `RPARAM` elements.

Note that the call to `U4LSF` will set `IPARAM` and `RPARAM` to their default values so only nondefault values need to be set above.

The following is a list of the parameters and the default values:

IPARAM — Integer vector of length 6.

`IPARAM`(1) = Initialization flag.

`IPARAM`(2) = Number of good digits in the function.
Default: Machine dependent.

`IPARAM`(3) = Maximum number of iterations.
Default: 100.

`IPARAM`(4) = Maximum number of function evaluations.
Default: 400.

`IPARAM`(5) = Maximum number of Jacobian evaluations.
Default: Not used in `UNLSF`.

`IPARAM`(6) = Internal variable scaling flag.
If `IPARAM`(6) = 1, then the values for `XSCALE` are set internally.
Default: 1.

RPARAM — Real vector of length 7.
`RPARAM`(1) = Scaled gradient tolerance.
The *i*-th component of the scaled gradient at x is calculated as

$$\frac{|g_i| * \max\left(|x_i|, 1/s_i\right)}{\left\|F(x)\right\|_2^2}$$

where

$$g_i = \left(J(x)^T F(x)\right)_i * \left(f_s\right)_i^2$$

$J(x)$ is the Jacobian, s = `XSCALE`, and f_s = `FSCALE`.
Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

`RPARAM`(2) = Scaled step tolerance. (`STEPTL`)
The *i*-th component of the scaled step between two points *x* and *y* is computed as

$$\frac{|x_i - y_i|}{\max\left(|x_i|, 1/s_i\right)}$$

where s = `XSCALE`.
Default: $\varepsilon^{2/3}$ where ε is the machine precision.

`RPARAM`(3) = Relative function tolerance.
Default: $\max(10^{-10}, \varepsilon^{2/3})$, $\max(10^{-20}, \varepsilon^{2/3})$ in double where ε is the machine precision.

`RPARAM`(4) = Absolute function tolerance.
Default: $\max(10^{-20}, \varepsilon^{2})$, $\max(10^{-40}, \varepsilon^{2})$ in double where ε is the machine precision.

`RPARAM`(5) = False convergence tolerance.
Default: 100ε where ε is the machine precision.

`RPARAM`(6) = Maximum allowable step size.
Default: 1000 max(ε_1, ε_2) where

$$\varepsilon_1 = \sqrt{\sum_{i=1}^{n} \left(s_i t_i\right)^2}$$

$\varepsilon_2 = \| s \|_2$, s = `XSCALE`, and t = `XGUESS`.

`RPARAM`(7) = Size of initial trust region radius.
Default: based on the initial scaled Cauchy step.

If double precision is desired, then `DU4LSF` is called and `RPARAM` is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to "Error Handling" in the Introduction.

Description

The routine `UNLSF` is based on the MINPACK routine `LMDIF` by Moré et al. (1980). It uses a modified Levenberg-Marquardt method to solve nonlinear least squares problems. The problem is stated as follows:

$$\min_{x \in \mathbf{R}^n} \frac{1}{2} F(x)^T F(x) = \frac{1}{2} \sum_{i=1}^{m} f_i(x)^2$$

where $m \ge n$, $F : \mathbf{R}^n \rightarrow \mathbf{R}^m$, and $f_i(x)$ is the i-th component function of $F(x)$. From a current point, the algorithm uses the trust region approach:

$$\min_{x_n \in \mathbf{R}^n} \left\| F(x_c) + J(x_c)(x_n - x_c) \right\|_2$$

$$\text{subject to } \|x_n - x_c\|_2 \le \delta_c$$

to get a new point x_n, which is computed as

$$x_n = x_c - \left(J(x_c)^T J(x_c) + \mu_c I \right)^{-1} J(x_c)^T F(x_c)$$

where $\mu_c = 0$ if $\delta_c \ge \|(J(x_c)^T J(x_c))^{-1} J(x_c)^T F(x_c)\|_2$ and $\mu_c > 0$ otherwise. $F(x_c)$ and $J(x_c)$ are the function values and the Jacobian evaluated at the current point x_c. This procedure is repeated until the stopping criteria are satisfied. For more details, see Levenberg (1944), Marquardt (1963), or Dennis and Schnabel (1983, Chapter 10).

Since a finite-difference method is used to estimate the Jacobian for some single precision calculations, an inaccurate estimate of the Jacobian may cause the algorithm to terminate at a noncritical point. In such cases, high precision arithmetic is recommended. Also, whenever the exact Jacobian can be easily provided, routine `UNLSJ` (page 1237) should be used instead.

UNLSJ

Solves a nonlinear least squares problem using a modified Levenberg-Marquardt algorithm and a user-supplied Jacobian.

Required Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function which defines the least-squares problem. The usage is `CALL FCN (M, N, X, F)`, where

`M` – Length of `F`. (Input)
`N` – Length of `X`. (Input)
`X` – Vector of length `N` at which point the function is evaluated. (Input)
`X` should not be changed by `FCN`.
`F` – Vector of length `M` containing the function values at `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

JAC — User-supplied `SUBROUTINE` to evaluate the Jacobian at a point `X`. The usage is `CALL JAC (M, N, X, FJAC, LDFJAC)`, where

`M` – Length of `F`. (Input)
`N` – Length of `X`. (Input)
`X` – Vector of length `N` at which point the Jacobian is evaluated. (Input)
`X` should not be changed by `JAC`.
`FJAC` – The computed `M` by `N` Jacobian at the point `X`. (Output)
`LDFJAC` – Leading dimension of `FJAC`. (Input)

`JAC` must be declared `EXTERNAL` in the calling program.

M — Number of functions. (Input)

X — Vector of length `N` containing the approximate solution. (Output)

Optional Arguments

N — Number of variables. `N` must be less than or equal to `M`. (Input)
Default: `N` = size (`X`,1).

XGUESS — Vector of length `N` containing the initial guess. (Input)
Default: `XGUESS` = 0.0.

XSCALE — Vector of length `N` containing the diagonal scaling matrix for the variables. (Input)
`XSCALE` is used mainly in scaling the gradient and the distance between two points. By default, the values for `XSCALE` are set internally. See `IPARAM`(6) in Comment 4.
Default: `XSCALE` = 1.0.

FSCALE — Vector of length `M` containing the diagonal scaling matrix for the functions. (Input)
`FSCALE` is used mainly in scaling the gradient. In the absence of other information, set all entries to 1.0.
Default: `FSCALE` = 1.0.

IPARAM — Parameter vector of length 6. (Input/Output)
Set `IPARAM`(1) to zero for default values of `IPARAM` and `RPARAM`. See Comment 4.
Default: `IPARAM` = 0.

RPARAM — Parameter vector of length 7. (Input/Output)
See Comment 4.

FVEC — Vector of length `M` containing the residuals at the approximate solution. (Output)

FJAC — `M` by `N` matrix containing a finite-difference approximate Jacobian at the approximate solution. (Output)

LDFJAC — Leading dimension of `FJAC` exactly as specified in the dimension statement of the calling program. (Input)
Default: `LDFJAC` = size (`FJAC`,1).

FORTRAN 90 Interface

Generic: `CALL UNLSJ (FCN, JAC, M, X [,…])`

Specific: The specific interface names are `S_UNLSJ` and `D_UNLSJ`.

FORTRAN 77 Interface

Single:
```
CALL UNLSJ (FCN, JAC, M, N, XGUESS, XSCALE, FSCALE, IPARAM,
RPARAM, X, FVEC, FJAC, LDFJAC)
```

Double: The double precision name is `DUNLSJ`.

Example

The nonlinear least-squares problem

$$\min_{x \in \mathbf{R}^2} \frac{1}{2} \sum_{i=1}^{2} f_i(x)^2$$

where

$$f_1(x) = 10\left(x_2 - x_1^2\right) \text{ and } f_2(x) = \left(1 - x_1\right)$$

is solved; default values for parameters are used.

```
      USE UNLSJ_INT
      USE UMACH_INT
!                                  Declaration of variables
      INTEGER    LDFJAC, M, N
      PARAMETER  (LDFJAC=2, M=2, N=2)
!
      INTEGER    IPARAM(6), NOUT
      REAL       FVEC(M), X(N), XGUESS(N)
      EXTERNAL   ROSBCK, ROSJAC
!                                  Compute the least squares for the
!                                  Rosenbrock function.
      DATA XGUESS/-1.2E0, 1.0E0/
      IPARAM(1) = 0
!
      CALL UNLSJ (ROSBCK, ROSJAC, M, X, XGUESS=XGUESS, &
                 IPARAM=IPARAM, FVEC=FVEC)
!                                  Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, FVEC, IPARAM(3), IPARAM(4), IPARAM(5)
!
99999 FORMAT ('  The solution is ', 2F9.4, //, '  The function ', &
            'evaluated at the solution is ', /, 18X, 2F9.4, //, &
            '  The number of iterations is ', 10X, I3, /, '  The ', &
            'number of function evaluations is ', I3, /, '  The ', &
            'number of Jacobian evaluations is ', I3, /)
      END
!
      SUBROUTINE ROSBCK (M, N, X, F)
      INTEGER    M, N
      REAL       X(N), F(M)
!
      F(1) = 10.0E0*(X(2)-X(1)*X(1))
      F(2) = 1.0E0 - X(1)
      RETURN
      END
!
      SUBROUTINE ROSJAC (M, N, X, FJAC, LDFJAC)
      INTEGER    M, N, LDFJAC
      REAL       X(N), FJAC(LDFJAC,N)
!
      FJAC(1,1) = -20.0E0*X(1)
      FJAC(2,1) = -1.0E0
      FJAC(1,2) = 10.0E0
      FJAC(2,2) = 0.0E0
      RETURN
      END
```

Output

```
The solution is    1.0000   1.0000

The function evaluated at the solution is
0.0000   0.0000

The number of iterations is            23
The number of function evaluations is  32
The number of Jacobian evaluations is  24
```

Comments

1. Workspace may be explicitly provided, if desired, by use of `U2LSJ/DU2LSJ`. The reference is:

   ```
   CALL U2LSJ (FCN, JAC, M, N, XGUESS, XSCALE, FSCALE, IPARAM,
   RPARAM, X, FVEC, FJAC, LDFJAC, WK, IWK)
   ```

   The additional arguments are as follows:

   WK — Work vector of length 9 * `N` + 3 * `M` – 1. `WK` contains the following information on output: The second `N` locations contain the last step taken. The third `N` locations contain the last Gauss-Newton step. The fourth `N` locations contain an estimate of the gradient at the solution.

   IWK — Work vector of length `N` containing the permutations used in the `QR` factorization of the Jacobian at the solution.

2. Informational errors

Type	Code	
3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
3	2	The iterates appear to be converging to a noncritical point.
4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
4	5	Maximum number of Jacobian evaluations exceeded.
3	6	Five consecutive steps have been taken with the maximum step length.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or `STEPTL` is too big.

3. The first stopping criterion for `UNLSJ` occurs when the norm of the function is less than the absolute function tolerance (`RPARAM`(4)). The second stopping criterion occurs when the norm of the scaled gradient is less than the given gradient tolerance (`RPARAM`(1)). The third stopping criterion for `UNLSJ` occurs when the scaled distance between the last two steps is less than the step tolerance (`RPARAM`(2)).

4. If the default parameters are desired for `UNLSJ`, then set `IPARAM`(1) to zero and call the routine `UNLSJ`. Otherwise, if any nondefault parameters are desired for `IPARAM` or `RPARAM`, then the following steps should be taken before calling `UNLSJ`:

   `CALL U4LSF (IPARAM, RPARAM)`
   Set nondefault values for desired `IPARAM`, `RPARAM` elements.

Note that the call to `U4LSF` will set `IPARAM` and `RPARAM` to their default values, so only nondefault values need to be set above.

The following is a list of the parameters and the default values:

IPARAM — Integer vector of length 6.
`IPARAM`(1) = Initialization flag.

`IPARAM`(2) = Number of good digits in the function.
Default: Machine dependent.

`IPARAM`(3) = Maximum number of iterations.
Default: 100.

`IPARAM`(4) = Maximum number of function evaluations.
Default: 400.

`IPARAM`(5) = Maximum number of Jacobian evaluations.
Default: 100.

`IPARAM`(6) = Internal variable scaling flag.
If `IPARAM`(6) = 1, then the values for `XSCALE` are set internally.
Default: 1.

RPARAM — Real vector of length 7.

`RPARAM`(1) = Scaled gradient tolerance.
The *i*-th component of the scaled gradient at x is calculated as

$$\frac{|g_i| * \max(|x_i|, 1/s_i)}{\|F(x)\|_2^2}$$

where

$$g_i = \left(J(x)^T F(x)\right)_i * (f_s)_i^2$$

J(*x*) is the Jacobian, *s* = `XSCALE`, and f_s = `FSCALE`.
Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

`RPARAM`(2) = Scaled step tolerance. (`STEPTL`)
The *i*-th component of the scaled step between two points *x* and *y* is computed as

$$\frac{|x_i - y_i|}{\max\left(|x_i|, 1/s_i\right)}$$

where s = `XSCALE`.
Default: $\varepsilon^{2/3}$ where ε is the machine precision.

`RPARAM`(3) = Relative function tolerance.
Default: max(10^{-10}, $\varepsilon^{2/3}$), max (10^{-20}, $\varepsilon^{2/3}$) in double where ε is the machine precision.

`RPARAM`(4) = Absolute function tolerance.
Default: max (10^{-20}, ε^2), max(10^{-40}, ε^2) in double where ε is the machine precision.

`RPARAM`(5) = False convergence tolerance.
Default: 100ε where ε is the machine precision.

`RPARAM`(6) = Maximum allowable step size.
Default: 1000 max(ε_1, ε_2) where

$$\varepsilon_1 \sqrt{\sum\nolimits_{i=1}^{n} \left(s_i t_i\right)^2}$$

$\varepsilon_2 = \| s \|_2$, s = `XSCALE`, and t = `XGUESS`.

`RPARAM`(7) = Size of initial trust region radius.
Default: based on the initial scaled Cauchy step.

If double precision is desired, then `DU4LSF` is called and `RPARAM` is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to “Error Handling” in the Introduction.

Description

The routine `UNLSJ` is based on the MINPACK routine `LMDER` by Moré et al. (1980). It uses a modified Levenberg-Marquardt method to solve nonlinear least squares problems. The problem is stated as follows:

$$\min_{x \in \mathbf{R}^n} \frac{1}{2} F(x)^T F(x) = \frac{1}{2} \sum_{i=1}^{m} f_i(x)^2$$

where $m \geq n$, $F : \mathbf{R}^n \rightarrow \mathbf{R}^m$, and $f_i(x)$ is the i-th component function of $F(x)$. From a current point, the algorithm uses the trust region approach:

$$\min_{x_n \in \mathbf{R}^n} \left\| F(x_c) + J(x_c)(x_n - x_c) \right\|_2$$

$$\text{subject to } \|x_n - x_c\|_2 \le \delta_c$$

to get a new point x_n, which is computed as

$$x_n = x_c - \left(J(x_c)^T J(x_c) + \mu_c I \right)^{-1} J(x_c)^T F(x_c)$$

where $\mu_c = 0$ if $\delta_c \ge \|(J(x_c)^T J(x_c))^{-1} J(x_c)^T F(x_c)\|_2$ and $\mu_c > 0$ otherwise. $F(x_c)$ and $J(x_c)$ are the function values and the Jacobian evaluated at the current point x_c. This procedure is repeated until the stopping criteria are satisfied. For more details, see Levenberg (1944), Marquardt(1963), or Dennis and Schnabel (1983, Chapter 10).

BCONF

Minimizes a function of N variables subject to bounds on the variables using a quasi-Newton method and a finite-difference gradient.

Required Arguments

FCN — User-supplied SUBROUTINE to evaluate the function to be minimized. The usage is CALL FCN (N, X, F), where

N – Length of X. (Input)

X – Vector of length N at which point the function is evaluated. (Input)
X should not be changed by FCN.

F – The computed function value at the point X. (Output)

FCN must be declared EXTERNAL in the calling program.

IBTYPE — Scalar indicating the types of bounds on variables. (Input)

IBTYPE	**Action**
0	User will supply all the bounds.
1	All variables are nonnegative.
2	All variables are nonpositive.
3	User supplies only the bounds on 1st variable, all other variables will have the same bounds.

XLB — Vector of length N containing the lower bounds on variables. (Input, if IBTYPE = 0; output, if IBTYPE = 1 or 2; input/output, if IBTYPE = 3)

XUB — Vector of length N containing the upper bounds on variables. (Input, if IBTYPE = 0; output, if IBTYPE = 1 or 2; input/output, if IBTYPE = 3)

X — Vector of length N containing the computed solution. (Output)

Optional Arguments

N — Dimension of the problem. (Input)
Default: N = size (X,1).

XGUESS — Vector of length N containing an initial guess of the computed solution. (Input)
Default: XGUESS = 0.0.

XSCALE — Vector of length N containing the diagonal scaling matrix for the variables. (Input)
XSCALE is used mainly in scaling the gradient and the distance between two points. In the absence of other information, set all entries to 1.0.
Default: XSCALE = 1.0.

FSCALE — Scalar containing the function scaling. (Input)
FSCALE is used mainly in scaling the gradient. In the absence of other information, set FSCALE to 1.0.
Default: FSCALE = 1.0.

IPARAM — Parameter vector of length 7. (Input/Output)
Set IPARAM(1) to zero for default values of IPARAM and RPARAM. See Comment 4.
Default: IPARAM = 0.

RPARAM — Parameter vector of length 7. (Input/Output)
See Comment 4.

FVALUE — Scalar containing the value of the function at the computed solution. (Output)

FORTRAN 90 Interface

Generic: `CALL BCONF (FCN, IBTYPE, XLB, XUB, X [,…])`

Specific: The specific interface names are S_BCONF and D_BCONF.

FORTRAN 77 Interface

Single:
```
CALL BCONF (FCN, N, XGUESS, IBTYPE, XLB, XUB, XSCALE,
 FSCALE, IPARAM, RPARAM, X, FVALUE)
```

Double: The double precision name is DBCONF.

Example

The problem

$$\min f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

$$\text{subject to}\quad -2 \le x_1 \le 0.5$$

$$-1 \le x_2 \le 2$$

is solved with an initial guess (−1.2, 1.0) and default values for parameters.

```
      USE BCONF_INT
      USE UMACH_INT
      INTEGER    N
      PARAMETER  (N=2)
!
      INTEGER    IPARAM(7), ITP, L, NOUT
      REAL       F, FSCALE, RPARAM(7), X(N), XGUESS(N), &
                XLB(N), XSCALE(N), XUB(N)
      EXTERNAL   ROSBRK
!
      DATA XGUESS/-1.2E0, 1.0E0/
      DATA XLB/-2.0E0, -1.0E0/, XUB/0.5E0, 2.0E0/
!                                All the bounds are provided
      ITP = 0
!                                Default parameters are used
      IPARAM(1) = 0
!                                Minimize Rosenbrock function using
!                                initial guesses of -1.2 and 1.0
      CALL BCONF (ROSBRK, ITP, XLB, XUB, X, XGUESS=XGUESS,  &
                 IPARAM=IPARAM, FVALUE=F)
!                                Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, F, (IPARAM(L),L=3,5)
!
99999 FORMAT ('  The solution is ', 6X, 2F8.3, //, '  The function ', &
            'value is ', F8.3, //, '  The number of iterations is ', &
            10X, I3, /, '  The number of function evaluations is ', &
            I3, /, '  The number of gradient evaluations is ', I3)
!
      END
!
      SUBROUTINE ROSBRK (N, X, F)
      INTEGER    N
      REAL       X(N), F
!
      F = 1.0E2*(X(2)-X(1)*X(1))**2 + (1.0E0-X(1))**2
!
      RETURN
      END
```

Output

```
The solution is          0.500   0.250

The function value is    0.250
The number of iterations is           24
The number of function evaluations is  34
The number of gradient evaluations is  26
```

Comments

1. Workspace may be explicitly provided, if desired, by use of `B2ONF/DB2ONF`. The reference is:

   ```
   CALL B2ONF (FCN, N, XGUESS, IBTYPE, XLB, XUB,
   XSCALE, FSCALE, IPARAM, RPARAM, X, FVALUE, WK, IWK)
   ```

   The additional arguments are as follows:

   WK — Real work vector of length `N * (2 * N + 8)`. `WK` contains the following information on output: The second `N` locations contain the last step taken. The third `N` locations contain the last Newton step. The fourth `N` locations contain an estimate of the gradient at the solution. The final N^2 locations contain a `BFGS` approximation to the Hessian at the solution.

   IWK — Work vector of length `N` stored in column order. Only the lower triangular portion of the matrix is stored in `WK`. The values returned in the upper triangle should be ignored.

2. Informational errors

Type	Code	
3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
4	2	The iterates appear to be converging to a noncritical point.
4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
4	5	Maximum number of gradient evaluations exceeded.
4	6	Five consecutive steps have been taken with the maximum step length.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or `STEPTL` is too big.
3	8	The last global step failed to locate a lower point than the current `X` value.

3. The first stopping criterion for `BCONF` occurs when the norm of the gradient is less than the given gradient tolerance (`RPARAM`(1)). The second stopping criterion for `BCONF` occurs when the scaled distance between the last two steps is less than the step tolerance (`RPARAM`(2)).

4. If the default parameters are desired for `BCONF`, then set `IPARAM`(1) to zero and call the routine `BCONF`. Otherwise, if any nondefault parameters are desired for `IPARAM` or `RPARAM`, then the following steps should be taken before calling `BCONF`:

CALL U4INF (IPARAM, RPARAM)

Set nondefault values for desired IPARAM, RPARAM elements.

Note that the call to U4INF will set IPARAM and RPARAM to their default values so only nondefault values need to be set above.

The following is a list of the parameters and the default values:

IPARAM — Integer vector of length 7.
IPARAM(1) = Initialization flag.

IPARAM(2) = Number of good digits in the function.
Default: Machine dependent.

IPARAM(3) = Maximum number of iterations.
Default: 100.

IPARAM(4) = Maximum number of function evaluations.
Default: 400.

IPARAM(5) = Maximum number of gradient evaluations.
Default: 400.

IPARAM(6) = Hessian initialization parameter.
If IPARAM(6) = 0, the Hessian is initialized to the identity matrix; otherwise, it is initialized to a diagonal matrix containing

$$\max\left(\left|f\left(t\right)\right|, f_s\right) * s_i^2$$

on the diagonal where t = XGUESS, f_s = FSCALE, and s = XSCALE.
Default: 0.

IPARAM(7) = Maximum number of Hessian evaluations.
Default: Not used in BCONF.

RPARAM — Real vector of length 7.
RPARAM(1) = Scaled gradient tolerance.
The *i*-th component of the scaled gradient at x is calculated as

$$\frac{\left|g_i\right| * \max\left(\left|x_i\right|, 1/s_i\right)}{\max\left(\left|f\left(x\right)\right|, f_s\right)}$$

where $g = \nabla f(x)$, s = XSCALE, and f_s = FSCALE.
Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

`RPARAM`(2) = Scaled step tolerance. (`STEPTL`)
The *i*-th component of the scaled step between two points *x* and *y* is computed as

$$\frac{|x_i - y_i|}{\max\left(|x_i|, 1/s_i\right)}$$

where s = `XSCALE`.
Default: $\varepsilon^{2/3}$ where ε is the machine precision.

`RPARAM`(3) = Relative function tolerance.
Default: max(10^{-10}, $\varepsilon^{2/3}$), max (10^{-20}, $\varepsilon^{2/3}$) in double where ε is the machine precision.

`RPARAM`(4) = Absolute function tolerance.
Default: Not used in `BCONF`.

`RPARAM`(5) = False convergence $^{\tau o\lambda\varepsilon\rho\alpha\nu\chi\varepsilon}$.
Default: 100ε where ε is the machine precision.

`RPARAM`(6) = Maximum allowable step size.
Default: 1000 max(ε_1, ε_2) where

$$\varepsilon_1 \sqrt{\sum_{i=1}^{n} \left(s_i t_i\right)^2}$$

$\varepsilon_2 = \| s \|_2$, s = `XSCALE`, and t = `XGUESS`.

`RPARAM`(7) = Size of initial trust region radius.
Default: based on the initial scaled Cauchy step.

If double precision is required, then `DU4INF` is called and `RPARAM` is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to "Error Handling" in the Introduction.

Description

The routine `BCONF` uses a quasi-Newton method and an active set strategy to solve minimization problems subject to simple bounds on the variables. The problem is stated as follows:

$$\min_{x \in \mathbf{R}^n} f(x)$$

$$\text{subject to } l \le x \le u$$

From a given starting point x^c, an active set IA, which contains the indices of the variables at their bounds, is built. A variable is called a "free variable" if it is not in the active set. The routine then computes the search direction for the free variables according to the formula

$$d = -B^{-1} g^c$$

where B is a positive definite approximation of the Hessian and g^c is the gradient evaluated at x^c; both are computed with respect to the free variables. The search direction for the variables in IA is set to zero. A line search is used to find a new point x^n ,

$$x^n = x^c + \lambda d, \lambda \in (0, 1]$$

such that

$$f(x^n) \le f(x^c) + \alpha g^T d, \qquad \alpha \in (0, 0.5)$$

Finally, the optimality conditions

$$\|g(x_i)\| \le \varepsilon, \; l_i < x_i < u_i$$

$$g(x_i) < 0, \; x_i = u_i$$

$$g(x_i) > 0, \; x_i = l_i$$

are checked, where ε is a gradient tolerance. When optimality is not achieved, B is updated according to the BFGS formula:

$$B \leftarrow B - \frac{Bss^T B}{s^T Bs} + \frac{yy^T}{y^T s}$$

where $s = x^n - x^c$ and $y = g^n - g^c$. Another search direction is then computed to begin the next iteration.

The active set is changed only when a free variable hits its bounds during an iteration or the optimality condition is met for the free variables but not for all variables in IA, the active set. In the latter case, a variable that violates the optimality condition will be dropped out of IA. For more details on the quasi-Newton method and line search, see Dennis and Schnabel (1983). For more detailed information on active set strategy, see Gill and Murray (1976).

Since a finite-difference method is used to estimate the gradient for some single precision calculations, an inaccurate estimate of the gradient may cause the algorithm to terminate at a noncritical point. In such cases, high precision arithmetic is recommended. Also, whenever the exact gradient can be easily provided, routine `BCONG` (page 1249) should be used instead.

BCONG

Minimizes a function of `N` variables subject to bounds on the variables using a quasi-Newton method and a user-supplied gradient.

Required Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (N, X, F)`, where

`N` – Length of `X`. (Input)

`X` – Vector of length `N` at which point the function is evaluated. (Input)
`X` should not be changed by `FCN`.

`F` – The computed function value at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

GRAD — User-supplied `SUBROUTINE` to compute the gradient at the point `X`. The usage is `CALL GRAD (N, X, G)`, where

`N` – Length of `X` and `G`. (Input)

`X` – Vector of length `N` at which point the gradient is evaluated. (Input)
`X` should not be changed by `GRAD`.

`G` – The gradient evaluated at the point `X`. (Output)

`GRAD` must be declared `EXTERNAL` in the calling program.

IBTYPE — Scalar indicating the types of bounds on variables. (Input)

IBTYPE	Action
0	User will supply all the bounds.
1	All variables are nonnegative.
2	All variables are nonpositive.
3	User supplies only the bounds on 1st variable, all other variables will have the same bounds.

XLB — Vector of length N containing the lower bounds on variables. (Input, if `IBTYPE` = 0; output, if IBTYPE = 1 or 2; input/output, if `IBTYPE` = 3)

XUB — Vector of length N containing the upper bounds on variables. (Input, if `IBTYPE` = 0; output, if IBTYPE = 1 or 2; input/output, if `IBTYPE` = 3)

X — Vector of length N containing the computed solution. (Output)

Optional Arguments

N — Dimension of the problem. (Input)
Default: N = size (X,1).

XGUESS — Vector of length N containing the initial guess of the minimum. (Input)
Default: XGUESS = 0.0.

XSCALE — Vector of length N containing the diagonal scaling matrix for the variables. (Input)
XSCALE is used mainly in scaling the gradient and the distance between two points. In the absence of other information, set all entries to 1.0.
Default: XSCALE = 1.0.

FSCALE — Scalar containing the function scaling. (Input)
FSCALE is used mainly in scaling the gradient. In the absence of other information, set FSCALE to 1.0.
Default: FSCALE = 1.0.

IPARAM — Parameter vector of length 7. (Input/Output)
Set IPARAM(1) to zero for default values of IPARAM and RPARAM. See Comment 4.
Default: IPARAM = 0.

RPARAM — Parameter vector of length 7. (Input/Output)
See Comment 4.

FVALUE — Scalar containing the value of the function at the computed solution. (Output)

FORTRAN 90 Interface

Generic: CALL BCONG (FCN, GRAD, IBTYPE, XLB, XUB, X [,…])

Specific: The specific interface names are S_BCONG and D_BCONG.

FORTRAN 77 Interface

Single: CALL BCONG (FCN, GRAD, N, XGUESS, IBTYPE, XLB, XUB, XSCALE, FSCALE, IPARAM, RPARAM, X, FVALUE)

Double: The double precision name is DBCONG.

Example

The problem

$$\min f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$
$$\text{subject to} \quad -2 \le x_1 \le 0.5$$
$$-1 \le x_2 \le 2$$

is solved with an initial guess (–1.2, 1.0), and default values for parameters.

```
      USE BCONG_INT
      USE UMACH_INT
      INTEGER    N
      PARAMETER  (N=2)
!
      INTEGER    IPARAM(7), ITP, L, NOUT
      REAL       F, X(N), XGUESS(N), XLB(N), XUB(N)
      EXTERNAL   ROSBRK, ROSGRD
!
      DATA XGUESS/-1.2E0, 1.0E0/
      DATA XLB/-2.0E0, -1.0E0/, XUB/0.5E0, 2.0E0/
!                                 All the bounds are provided
      ITP = 0
!                                 Default parameters are used
      IPARAM(1) = 0
!                                 Minimize Rosenbrock function using
!                                 initial guesses of -1.2 and 1.0
      CALL BCONG (ROSBRK, ROSGRD, ITP, XLB, XUB, X, XGUESS=XGUESS, &
                 IPARAM=IPARAM, FVALUE=F)
!                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, F, (IPARAM(L),L=3,5)
!
99999 FORMAT ('  The solution is ', 6X, 2F8.3, //, '  The function ', &
            'value is ', F8.3, //, '  The number of iterations is ', &
            10X, I3, /, '  The number of function evaluations is ', &
            I3, /, '  The number of gradient evaluations is ', I3)
!
      END
!
      SUBROUTINE ROSBRK (N, X, F)
      INTEGER    N
      REAL       X(N), F
!
      F = 1.0E2*(X(2)-X(1)*X(1))**2 + (1.0E0-X(1))**2
!
      RETURN
      END
!
      SUBROUTINE ROSGRD (N, X, G)
      INTEGER    N
      REAL       X(N), G(N)
!
      G(1) = -4.0E2*(X(2)-X(1)*X(1))*X(1) - 2.0E0*(1.0E0-X(1))
      G(2) = 2.0E2*(X(2)-X(1)*X(1))
!
      RETURN
      END
```

Output

```
The solution is          0.500   0.250

The function value is    0.250

The number of iterations is            22
The number of function evaluations is  32
The number of gradient evaluations is  23
```

Comments

1. Workspace may be explicitly provided, if desired, by use of B2ONG/DB2ONG. The reference is:

   CALL B2ONG (FCN, GRAD, N, XGUESS, IBTYPE, XLB, XUB, XSCALE, FSCALE, IPARAM, RPARAM, X, FVALUE, WK, IWK)

   The additional arguments are as follows:

   WK — Real work vector of length N * (2 * N + 8). WK contains the following information on output: The second N locations contain the last step taken. The third N locations contain the last Newton step. The fourth N locations contain an estimate of the gradient at the solution. The final N^2 locations contain a BFGS approximation to the Hessian at the solution.

   IWK — Work vector of length N stored in column order. Only the lower triangular portion of the matrix is stored in WK. The values returned in the upper triangle should be ignored.

2. Informational errors

Type	Code	
3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
4	2	The iterates appear to be converging to a noncritical point.
4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
4	5	Maximum number of gradient evaluations exceeded.
4	6	Five consecutive steps have been taken with the maximum step length.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or STEPTL is too big.

3 8 The last global step failed to locate a lower point than the current X value.

3. The first stopping criterion for BCONG occurs when the norm of the gradient is less than the given gradient tolerance (RPARAM(1)). The second stopping criterion for BCONG occurs when the scaled distance between the last two steps is less than the step tolerance (RPARAM(2)).

4. If the default parameters are desired for BCONG, then set IPARAM (1) to zero and call the routine BCONG. Otherwise, if any nondefault parameters are desired for IPARAM or RPARAM, then the following steps should be taken before calling BCONG:

```
CALL U4INF (IPARAM, RPARAM)
```

Set nondefault values for desired IPARAM, RPARAM elements.

Note that the call to U4INF will set IPARAM and RPARAM to their default values so only nondefault values need to be set above.

The following is a list of the parameters and the default values:

IPARAM — Integer vector of length 7.
IPARAM(1) = Initialization flag.

IPARAM(2) = Number of good digits in the function.
Default: Machine dependent.

IPARAM(3) = Maximum number of iterations.
Default: 100.

IPARAM(4) = Maximum number of function evaluations.
Default: 400.

IPARAM(5) = Maximum number of gradient evaluations.
Default: 400.

IPARAM(6) = Hessian initialization parameter.
If IPARAM(6) = 0, the Hessian is initialized to the identity matrix; otherwise, it is initialized to a diagonal matrix containing

$$\max\left(\left|f\left(t\right)\right|, f_s\right) * s_i^2$$

on the diagonal where t = XGUESS, fs = FSCALE, and s = XSCALE.
Default: 0.

IPARAM(7) = Maximum number of Hessian evaluations.
Default: Not used in BCONG.

RPARAM — Real vector of length 7.

RPARAM(1) = Scaled gradient tolerance.

The *i*-th component of the scaled gradient at x is calculated as

$$\frac{|g_i| * \max(|x_i|, 1/s_i)}{\max(|f(x)|, f_s)}$$

where $g = \nabla f(x)$, s = XSCALE, and f_s = FSCALE.

Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

RPARAM(2) = Scaled step tolerance. (STEPTL)

The *i*-th component of the scaled step between two points *x* and *y* is computed as

$$\frac{|x_i - y_i|}{\max(|x_i|, 1/s_i)}$$

where s = XSCALE.

Default: $\varepsilon^{2/3}$ where ε is the machine precision.

RPARAM(3) = Relative function tolerance.

Default: $\max(10^{-10}, \varepsilon^{2/3})$, $\max(10^{-20}, \varepsilon^{2/3})$ in double where ε is the machine precision.

RPARAM(4) = Absolute function tolerance.

Default: Not used in BCONG.

RPARAM(5) = False convergence tolerance.

Default: 100ε where ε is the machine precision.

RPARAM(6) = Maximum allowable step size.

Default: $1000 \max(\varepsilon_1, \varepsilon_2)$ where

$$\varepsilon_1 \sqrt{\sum_{i=1}^{n} (s_i t_i)^2}$$

$\varepsilon_2 = \| s \|_2$, s = XSCALE, and t = XGUESS.

RPARAM(7) = Size of initial trust region radius.

Default: based on the initial scaled Cauchy step.

If double precision is required, then DU4INF is called and RPARAM is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to “Error Handling” in the Introduction.

Description

The routine `BCONG` uses a quasi-Newton method and an active set strategy to solve minimization problems subject to simple bounds on the variables. The problem is stated as follows:

$$\min_{x \in \mathbf{R}^n} f(x)$$

$$\text{subject to } \; l \le x \le u$$

From a given starting point x^c, an active set IA, which contains the indices of the variables at their bounds, is built. A variable is called a “free variable” if it is not in the active set. The routine then computes the search direction for the free variables according to the formula

$$d = -B^{-1} g^c$$

where B is a positive definite approximation of the Hessian and g^c is the gradient evaluated at x^c; both are computed with respect to the free variables. The search direction for the variables in IA is set to zero. A line search is used to find a new point x^n ,

$$x^n = x^c + \lambda d, \lambda \in (0, 1]$$

such that

$$f(x^n) \le f(x^c) + \alpha g^T d, \qquad \alpha \in (0, 0.5)$$

Finally, the optimality conditions

$$\|g(x_i)\| \le \varepsilon, \; l_i < x_i < u_i$$

$$g(x_i) < 0, \; x_i = u_i$$

$$g(x_i) > 0, \; x_i = l_i$$

are checked, where ε is a gradient tolerance. When optimality is not achieved, B is updated according to the BFGS formula:

$$B \leftarrow B - \frac{Bss^T B}{s^T Bs} + \frac{yy^T}{y^T s}$$

where $s = x^n - x^c$ and $y = g^n - g^c$. Another search direction is then computed to begin the next iteration.

The active set is changed only when a free variable hits its bounds during an iteration or the optimality condition is met for the free variables but not for all variables in IA, the active set. In the latter case, a variable that violates the optimality condition will be dropped out of IA. For more details on the quasi-Newton method and line search, see Dennis and Schnabel (1983). For more detailed information on active set strategy, see Gill and Murray (1976).

BCODH

Minimizes a function of N variables subject to bounds on the variables using a modified Newton method and a finite-difference Hessian.

Required Arguments

FCN — User-supplied SUBROUTINE to evaluate the function to be minimized. The usage is CALL FCN (N, X, F), where

N – Length of X. (Input)

X – Vector of length N at which point the function is evaluated. (Input)
X should not be changed by FCN.

F – The computed function value at the point X. (Output)

FCN must be declared EXTERNAL in the calling program.

GRAD — User-supplied SUBROUTINE to compute the gradient at the point X. The usage is CALL GRAD (N, X, G), where

N – Length of X and G. (Input)

X – Vector of length N at which point the gradient is evaluated. (Input)
X should not be changed by GRAD.

G – The gradient evaluated at the point X. (Output)

GRAD must be declared EXTERNAL in the calling program.

IBTYPE — Scalar indicating the types of bounds on variables. (Input)

IBTYPE	**Action**
0	User will supply all the bounds.
1	All variables are nonnegative.
2	All variables are nonpositive.
3	User supplies only the bounds on 1st variable, all other variables will have the same bounds.

XLB — Vector of length N containing the lower bounds on the variables. (Input)

XUB — Vector of length N containing the upper bounds on the variables. (Input)

X — Vector of length `N` containing the computed solution. (Output)

Optional Arguments

N — Dimension of the problem. (Input)
Default: `N` = size (`X`,1).

XGUESS — Vector of length `N` containing the initial guess of the minimum. (Input)
Default: `XGUESS` = 0.0.

XSCALE — Vector of length `N` containing the diagonal scaling matrix for the variables. (Input)
`XSCALE` is used mainly in scaling the gradient and the distance between two points. In the absence of other information, set all entries to 1.0.
Default: `XSCALE` = 1.0.

FSCALE — Scalar containing the function scaling. (Input)
`FSCALE` is used mainly in scaling the gradient. In the absence of other information, set `FSCALE` to 1.0.
Default: `FSCALE` = 1.0.

IPARAM — Parameter vector of length 7. (Input/Output)
Set `IPARAM`(1) to zero for default values of `IPARAM` and `RPARAM`. See Comment 4.
Default: `IPARAM` = 0.

RPARAM — Parameter vector of length 7. (Input/Output)
See Comment 4.

FVALUE — Scalar containing the value of the function at the computed solution. (Output)

FORTRAN 90 Interface

Generic: `CALL BCODH (FCN, GRAD, IBTYPE, XLB, XUB, X [,…])`

Specific: The specific interface names are `S_BCODH` and `D_BCODH`.

FORTRAN 77 Interface

Single:
```
CALL BCODH (FCN, GRAD, N, XGUESS, IBTYPE, XLB, XUB, XSCALE,
FSCALE, IPARAM, RPARAM, X, FVALUE)
```

Double: The double precision name is `DBCODH`.

Example

The problem

$$\min f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$
$$\text{subject to} \quad -2 \le x_1 \le 0.5$$
$$-1 \le x_2 \le 2$$

is solved with an initial guess (−1.2, 1.0), and default values for parameters.

```
      USE BCODH_INT
      USE UMACH_INT
      INTEGER    N
      PARAMETER  (N=2)
!
      INTEGER    IP, IPARAM(7), L, NOUT
      REAL       F, X(N), XGUESS(N), XLB(N), XUB(N)
      EXTERNAL   ROSBRK, ROSGRD
!
      DATA XGUESS/-1.2E0, 1.0E0/
      DATA XLB/-2.0E0, -1.0E0/, XUB/0.5E0, 2.0E0/
!
      IPARAM(1) = 0
      IP        = 0
!                                 Minimize Rosenbrock function using
!                                 initial guesses of -1.2 and 1.0
      CALL BCODH (ROSBRK, ROSGRD, IP, XLB, XUB, X, XGUESS=XGUESS, &
                 IPARAM=IPARAM, FVALUE=F)
!                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, F, (IPARAM(L),L=3,5)
!
99999 FORMAT ('  The solution is ', 6X, 2F8.3, //, '  The function ', &
            'value is ', F8.3, //, '  The number of iterations is ', &
            10X, I3, /, '  The number of function evaluations is ', &
            I3, /, '  The number of gradient evaluations is ', I3)
!
      END
!
      SUBROUTINE ROSBRK (N, X, F)
      INTEGER    N
      REAL       X(N), F
!
      F = 1.0E2*(X(2)-X(1)*X(1))**2 + (1.0E0-X(1))**2
!
      RETURN
      END
      SUBROUTINE ROSGRD (N, X, G)
      INTEGER    N
      REAL       X(N), G(N)
!
      G(1) = -4.0E2*(X(2)-X(1)*X(1))*X(1) - 2.0E0*(1.0E0-X(1))
      G(2) = 2.0E2*(X(2)-X(1)*X(1))
!
      RETURN
      END
```

Output

```
The solution is          0.500   0.250

The function value is    0.250

The number of iterations is            17
The number of function evaluations is  26
The number of gradient evaluations is  18
```

Comments

1. Workspace may be explicitly provided, if desired, by use of `B2ODH/DB2ODH`. The reference is:

   ```
   CALL B2ODH (FCN, GRAD, N, XGUESS, IBTYPE, XLB, XUB, XSCALE,
   FSCALE, IPARAM, RPARAM, X, FVALUE, WK, IWK)
   ```

   The additional arguments are as follows:

   WK — Real work vector of length `N * (N + 8)`. `WK` contains the following information on output: The second `N` locations contain the last step taken. The third `N` locations contain the last Newton step. The fourth `N` locations contain an estimate of the gradient at the solution. The final N^2 locations contain the Hessian at the approximate solution.

   IWK — Integer work vector of length `N`.

2. Informational errors

Type	Code	
3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
4	2	The iterates appear to be converging to a noncritical point.
4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
4	5	Maximum number of gradient evaluations exceeded.
4	6	Five consecutive steps have been taken with the maximum step length.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or `STEPTL` is too big.
4	7	Maximum number of Hessian evaluations exceeded.

3. The first stopping criterion for `BCODH` occurs when the norm of the gradient is less than the given gradient tolerance (`RPARAM`(1)). The second stopping criterion for `BCODH` occurs when the scaled distance between the last two steps is less than the step tolerance (`RPARAM`(2)).

4. If the default parameters are desired for `BCODH`, then set `IPARAM`(1) to zero and call the routine `BCODH`. Otherwise, if any nondefault parameters are desired for `IPARAM` or `RPARAM`; then the following steps should be taken before calling `BCODH`:

CALL U4INF (IPARAM, RPARAM)
Set nondefault values for desired IPARAM, RPARAM elements.

Note that the call to U4INF will set IPARAM and RPARAM to their default values so only nondefault values need to be set above.

The following is a list of the parameters and the default values:

IPARAM — Integer vector of length 7.
IPARAM(1) = Initialization flag.

IPARAM(2) = Number of good digits in the function.
Default: Machine dependent.

IPARAM(3) = Maximum number of iterations.
Default: 100.

IPARAM(4) = Maximum number of function evaluations.
Default: 400.

IPARAM(5) = Maximum number of gradient evaluations.
Default: 400.

IPARAM(6) = Hessian initialization parameter.
Default: Not used in BCODH.

IPARAM(7) = Maximum number of Hessian evaluations.
Default: 100.

RPARAM — Real vector of length 7.
RPARAM(1) = Scaled gradient tolerance.
The *i*-th component of the scaled gradient at x is calculated as

$$\frac{|g_i| * \max\left(|x_i|, 1/s_i\right)}{\max\left(|f(x)|, f_s\right)}$$

where $g = \nabla f(x)$, s = XSCALE, and f_s = FSCALE.
Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

RPARAM(2) = Scaled step tolerance. (STEPTL)
The *i*-th component of the scaled step between two points *x* and *y* is computed as

$$\frac{|x_i - y_i|}{\max\left(|x_i|, 1/s_i\right)}$$

where s = `XSCALE`.
Default: $\varepsilon^{2/3}$ where ε is the machine precision.

`RPARAM`(3) = Relative function tolerance.
Default: $\max(10^{-10}, \varepsilon^{2/3})$, $\max(10^{-20}, \varepsilon^{2/3})$ in double where ε is the machine precision.

`RPARAM`(4) = Absolute function tolerance.
Default: Not used in `BCODH`.

`RPARAM`(5) = False convergence tolerance.
Default: 100ε where ε is the machine precision.

`RPARAM`(6) = Maximum allowable step size.
Default: $1000 \max(\varepsilon_1, \varepsilon_2)$ where

$$\varepsilon_1 \sqrt{\sum_{i=1}^{n} \left(s_i t_i\right)^2}$$

$\varepsilon_2 = \| s \|_2$, s = `XSCALE`, and t = `XGUESS`.

`RPARAM`(7) = Size of initial trust region radius.
Default: based on the initial scaled Cauchy step.

If double precision is required, then `DU4INF` is called and `RPARAM` is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to “Error Handling” in the Introduction.

Description

The routine `BCODH` uses a modified Newton method and an active set strategy to solve minimization problems subject to simple bounds on the variables. The problem is stated as

$$\min_{x \in \mathbf{R}^n} f(x)$$

$$\text{subject to } l \le x \le u$$

From a given starting point x^c, an active set IA, which contains the indices of the variables at their bounds, is built. A variable is called a “free variable” if it is not in the active set. The routine then computes the search direction for the free variables according to the formula

$$d = -H^{-1} g^c$$

where H is the Hessian and g^c is the gradient evaluated at x^c; both are computed with respect to the free variables. The search direction for the variables in IA is set to zero. A line search is used to find a new point x^n ,

$$x^n = x^c + \lambda d, \lambda \in (0, 1]$$

such that

$$f(x^n) \leq f(x^c) + \alpha g^T d, \qquad \alpha \in (0, 0.5)$$

Finally, the optimality conditions

$$\|g(x_i)\| \leq \varepsilon, l_i < x_i < u_i$$

$$g(x_i) < 0, x_i = u_i$$

$$g(x_i) > 0, x_i = l_i$$

are checked where ε is a gradient tolerance. When optimality is not achieved, another search direction is computed to begin the next iteration. This process is repeated until the optimality criterion is met.

The active set is changed only when a free variable hits its bounds during an iteration or the optimality condition is met for the free variables but not for all variables in IA, the active set. In the latter case, a variable that violates the optimality condition will be dropped out of IA. For more details on the modified Newton method and line search, see Dennis and Schnabel (1983). For more detailed information on active set strategy, see Gill and Murray (1976).

Since a finite-difference method is used to estimate the Hessian for some single precision calculations, an inaccurate estimate of the Hessian may cause the algorithm to terminate at a noncritical point. In such cases, high precision arithmetic is recommended. Also, whenever the exact Hessian can be easily provided, routine `BCOAH` (page 1263) should be used instead.

BCOAH

Minimizes a function of `N` variables subject to bounds on the variables using a modified Newton method and a user-supplied Hessian.

Required Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (N, X, F)`, where

`N` – Length of `X`. (Input)

`X` – Vector of length `N` at which point the function is evaluated. (Input)
`X` should not be changed by `FCN`.

F – The computed function value at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

GRAD — User-supplied `SUBROUTINE` to compute the gradient at the point `X`. The usage is `CALL GRAD (N, X, G)`, where

`N` – Length of `X` and `G`. (Input)

`X` – Vector of length `N` at which point the gradient is evaluated. (Input) `X` should not be changed by `GRAD`.

`G` – The gradient evaluated at the point `X`. (Output)

`GRAD` must be declared `EXTERNAL` in the calling program.

HESS — User-supplied `SUBROUTINE` to compute the Hessian at the point `X`. The usage is `CALL HESS (N, X, H, LDH)`, where

`N` – Length of `X`. (Input)

`X` – Vector of length `N` at which point the Hessian is evaluated. (Input) `X` should not be changed by `HESS`.

`H` – The Hessian evaluated at the point `X`. (Output)

`LDH` – Leading dimension `of H` exactly as specified in the dimension statement of the calling program. (Input)

`HESS` must be declared `EXTERNAL` in the calling program.

IBTYPE — Scalar indicating the types of bounds on variables. (Input)

IBTYPE	**Action**
0	User will supply all the bounds.
1	All variables are nonnegative.
2	All variables are nonpositive.
3	User supplies only the bounds on 1st variable, all other variables will have the same bounds.

XLB — Vector of length `N` containing the lower bounds on the variables. (Input)

XUB — Vector of length `N` containing the upper bounds on the variables. (Input)

X — Vector of length `N` containing the computed solution. (Output)

Optional Arguments

N — Dimension of the problem. (Input)
Default: `N` = size (`X`,1).

XGUESS — Vector of length `N` containing the initial guess. (Input)
Default: `XGUESS` = 0.0.

XSCALE — Vector of length `N` containing the diagonal scaling matrix for the variables. (Input)
`XSCALE` is used mainly in scaling the gradient and the distance between two points. In the absence of other information, set all entries to 1.0.
Default: `XSCALE` = 1.0.

FSCALE — Scalar containing the function scaling. (Input)
`FSCALE` is used mainly in scaling the gradient. In the absence of other information, set `FSCALE` to 1.0.
Default: `FSCALE` = 1.0.

IPARAM — Parameter vector of length 7. (Input/Output)
Set `IPARAM`(1) to zero for default values of `IPARAM` and `RPARAM`. See Comment 4.
Default: `IPARAM` = 0.

RPARAM — Parameter vector of length 7. (Input/Output)
See Comment 4.

FVALUE — Scalar containing the value of the function at the computed solution. (Output)

FORTRAN 90 Interface

Generic: `CALL BCOAH (FCN, GRAD, HESS, IBTYPE, XLB, XUB, X [,…])`

Specific: The specific interface names are `S_BCOAH` and `D_BCOAH`.

FORTRAN 77 Interface

Single: `CALL BCOAH (FCN, GRAD, HESS, N, XGUESS, IBTYPE, XLB, XUB, XSCALE, FSCALE, IPARAM, RPARAM, X, FVALUE)`

Double: The double precision name is `DBCOAH`.

Example

The problem

$$\min f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$
$$\text{subject to} \quad -2 \le x_1 \le 0.5$$
$$-1 \le x_2 \le 2$$

is solved with an initial guess (–1.2, 1.0), and default values for parameters.

```
      USE BCOAH_INT
      USE UMACH_INT
      INTEGER    N
      PARAMETER  (N=2)
!
      INTEGER    IP, IPARAM(7), L, NOUT
      REAL       F, X(N), XGUESS(N), XLB(N), XUB(N)
      EXTERNAL   ROSBRK, ROSGRD, ROSHES
!
      DATA XGUESS/-1.2E0, 1.0E0/
      DATA XLB/-2.0E0, -1.0E0/, XUB/0.5E0, 2.0E0/
!
      IPARAM(1) = 0
      IP        = 0
!                                Minimize Rosenbrock function using
!                                initial guesses of -1.2 and 1.0
      CALL BCOAH (ROSBRK, ROSGRD, ROSHES, IP, XLB, XUB, X, &
                 XGUESS=XGUESS,IPARAM=IPARAM, FVALUE=F)
!                                Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, F, (IPARAM(L),L=3,5), IPARAM(7)
!
99999 FORMAT ('  The solution is ', 6X, 2F8.3, //, '  The function ', &
            'value is ', F8.3, //, '  The number of iterations is ', &
            10X, I3, /, '  The number of function evaluations is ', &
            I3, /, '  The number of gradient evaluations is ', I3, /, &
            '  The number of Hessian evaluations is  ', I3)
!
      END
!
      SUBROUTINE ROSBRK (N, X, F)
      INTEGER    N
      REAL       X(N), F
!
      F = 1.0E2*(X(2)-X(1)*X(1))**2 + (1.0E0-X(1))**2
!
      RETURN
      END
!
      SUBROUTINE ROSGRD (N, X, G)
      INTEGER    N
      REAL       X(N), G(N)
!
      G(1) = -4.0E2*(X(2)-X(1)*X(1))*X(1) - 2.0E0*(1.0E0-X(1))
      G(2) = 2.0E2*(X(2)-X(1)*X(1))
!
      RETURN
      END
```

```
!
      SUBROUTINE ROSHES (N, X, H, LDH)
      INTEGER    N, LDH
      REAL       X(N), H(LDH,N)
!
      H(1,1) = -4.0E2*X(2) + 1.2E3*X(1)*X(1) + 2.0E0
      H(2,1) = -4.0E2*X(1)
      H(1,2) = H(2,1)
      H(2,2) = 2.0E2
!
      RETURN
      END
```

Output

```
The solution is          0.500   0.250

The function value is    0.250

The number of iterations is            18
The number of function evaluations is  29
The number of gradient evaluations is  19
The number of Hessian evaluations is   18
```

Comments

1. Workspace may be explicitly provided, if desired, by use of `B2OAH/DB2OAH`. The reference is:

   ```
   CALL B2OAH (FCN, GRAD, HESS, N, XGUESS, IBTYPE, XLB,
          XUB, XSCALE, FSCALE, IPARAM, RPARAM, X,
          FVALUE, WK, IWK)
   ```

   The additional arguments are as follows:

   WK — Work vector of length `N * (N + 8)`. `WK` contains the following information on output: The second `N` locations contain the last step taken. The third `N` locations contain the last Newton step. The fourth `N` locations contain an estimate of the gradient at the solution. The final N^2 locations contain the Hessian at the approximate solution.

   IWK — Work vector of length `N`.

2. Informational errors

Type	Code	
3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
4	2	The iterates appear to be converging to a noncritical point.

4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
4	5	Maximum number of gradient evaluations exceeded.
4	6	Five consecutive steps have been taken with the maximum step length.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or `STEPTL` is too big.
4	7	Maximum number of Hessian evaluations exceeded.
3	8	The last global step failed to locate a lower point than the current `X` value.

3. The first stopping criterion for `BCOAH` occurs when the norm of the gradient is less than the given gradient tolerance (`RPARAM`(1)). The second stopping criterion for `BCOAH` occurs when the scaled distance between the last two steps is less than the step tolerance (`RPARAM`(2)).

4. If the default parameters are desired for `BCOAH`, then set `IPARAM`(1) to zero and call the routine `BCOAH`. Otherwise, if any nondefault parameters are desired for `IPARAM` or `RPARAM`, then the following steps should be taken before calling `BCOAH`:

   `CALL U4INF` (`IPARAM`, `RPARAM`)
   Set nondefault values for desired `IPARAM`, `RPARAM` elements.

   Note that the call to `U4INF` will set `IPARAM` and `RPARAM` to their default values so only nondefault values need to be set above.

   The following is a list of the parameters and the default values:

   IPARAM — Integer vector of length 7.
   `IPARAM`(1) = Initialization flag.

   `IPARAM`(2) = Number of good digits in the function.
   Default: Machine dependent.

   `IPARAM`(3) = Maximum number of iterations.
   Default: 100.

   `IPARAM`(4) = Maximum number of function evaluations.
   Default: 400.

   `IPARAM`(5) = Maximum number of gradient evaluations.
   Default: 400.

   `IPARAM`(6) = Hessian initialization parameter.
   Default: Not used in `BCOAH`.

IPARAM(7) = Maximum number of Hessian evaluations.
Default: 100.

RPARAM — Real vector of length 7.
RPARAM(1) = Scaled gradient tolerance.
The *i*-th component of the scaled gradient at x is calculated as

$$\frac{|g_i| * \max(|x_i|, 1/s_i)}{\max(|f(x)|, f_s)}$$

where $g = \nabla f(x)$, s = XSCALE, and f_s = FSCALE.
Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

RPARAM(2) = Scaled step tolerance. (STEPTL)
The *i*-th component of the scaled step between two points *x* and *y* is computed as

$$\frac{|x_i - y_i|}{\max(|x_i|, 1/s_i)}$$

where s = XSCALE.
Default: $\varepsilon^{2/3}$ where ε is the machine precision.

RPARAM(3) = Relative function tolerance.
Default: $\max(10^{-10}, \varepsilon^{2/3})$, $\max(10^{-20}, \varepsilon^{2/3})$ in double where ε is the machine precision.

RPARAM(4) = Absolute function tolerance.
Default: Not used in BCOAH.

RPARAM(5) = False convergence tolerance.
Default: 100ε where ε is the machine precision.

RPARAM(6) = Maximum allowable step size.
Default: 1000 $\max(\varepsilon_1, \varepsilon_2)$ where

$$\varepsilon_1 \sqrt{\sum_{i=1}^{n} (s_i t_i)^2}$$

$\varepsilon_2 = \| s \|_2$, s = XSCALE, and t = XGUESS.

RPARAM(7) = Size of initial trust region radius.
Default: based on the initial scaled Cauchy step.

If double precision is required, then DU4INF is called and RPARAM is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to “Error Handling” in the Introduction.

Description

The routine BCOAH uses a modified Newton method and an active set strategy to solve minimization problems subject to simple bounds on the variables. The problem is stated as follows:

$$\min_{x \in \mathbf{R}^n} f(x)$$

$$\text{subject to } l \le x \le u$$

From a given starting point x^c, an active set IA, which contains the indices of the variables at their bounds, is built. A variable is called a “free variable” if it is not in the active set. The routine then computes the search direction for the free variables according to the formula

$$d = -H^{-1} g^c$$

where H is the Hessian and g^c is the gradient evaluated at x^c; both are computed with respect to the free variables. The search direction for the variables in IA is set to zero. A line search is used to find a new point x^n ,

$$x^n = x^c + \lambda d, \lambda \in (0, 1]$$

such that

$$f(x^n) \le f(x^c) + \alpha g^T d, \; \alpha \in (0, 0.5)$$

Finally, the optimality conditions

$$\|g(x_i)\| \le \varepsilon, \; l_i < x_i < u_i$$

$$g(x_i) < 0, \; x_i = u_i$$

$$g(x_i) > 0, \; x_i = l_i$$

are checked where ε is a gradient tolerance. When optimality is not achieved, another search direction is computed to begin the next iteration. This process is repeated until the optimality criterion is met.

The active set is changed only when a free variable hits its bounds during an iteration or the optimality condition is met for the free variables but not for all variables in IA, the active set. In the latter case, a variable that violates the optimality condition will be dropped out of IA. For more details on the modified Newton method and line search, see Dennis and Schnabel (1983). For more detailed information on active set strategy, see Gill and Murray (1976).

BCPOL

Minimizes a function of `N` variables subject to bounds on the variables using a direct search complex algorithm.

Required Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (N, X, F)`, where

`N` – Length of `X`. (Input)

`X` – Vector of length `N` at which point the function is evaluated. (Input)
`X` should not be changed by `FCN`.

`F` – The computed function value at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

IBTYPE — Scalar indicating the types of bounds on variables. (Input)

IBTYPE	**Action**
0	User will supply all the bounds.
1	All variables are nonnegative.
2	All variables are nonpositive.
3	User supplies only the bounds on the first, variable. All other variables will have the same bounds.

XLB — Vector of length `N` containing the lower bounds on the variables. (Input, if `IBTYPE` = 0; output, if `IBTYPE` = 1 or 2; input/output, if `IBTYPE` = 3)

XUB — Vector of length `N` containing the upper bounds on the variables. (Input, if `IBTYPE` = 0; output, if `IBTYPE` = 1 or 2; input/output, if `IBTYPE` = 3)

X — Real vector of length `N` containing the best estimate of the minimum found. (Output)

Optional Arguments

N — The number of variables. (Input)
Default: `N` = size (`XGUESS`,1).

XGUESS — Real vector of length `N` that contains an initial guess to the minimum. (Input)
Default: `XGUESS` = 0.0.

FTOL — First convergence criterion. (Input)
The algorithm stops when a relative error in the function values is less than `FTOL`, i.e. when (`F`(worst) – `F`(best)) < `FTOL * ` (1 + `ABS`(`F`(best))) where `F`(worst) and `F`(best) are the function values of the current worst and best point, respectively. Second convergence criterion. The algorithm stops when the standard deviation of the function values at the 2 `* N` current points is less than `FTOL`. If the subroutine terminates prematurely, try again with a smaller value `FTOL`.
Default: `FTOL` = 1.0e-4 for single and 1.0d-8 for double precision.

MAXFCN — On input, maximum allowed number of function evaluations. (Input/ Output)
On output, actual number of function evaluations needed.
Default: `MAXFCN` = 300.

FVALUE — Function value at the computed solution. (Output)

FORTRAN 90 Interface

Generic: `CALL BCPOL (FCN, IBTYPE, XLB, XUB, X [,…])`

Specific: The specific interface names are `S_BCPOL` and `D_BCPOL`.

FORTRAN 77 Interface

Single: `CALL BCPOL (FCN, N, XGUESS, IBTYPE, XLB, XUB, FTOL, MAXFCN, X, FVALUE)`

Double: The double precision name is `DBCPOL`.

Example

The problem

$$\begin{aligned} \min f(x) &= 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2 \\ \text{subject to} \quad & -2 \le x_1 \le 0.5 \\ & -1 \le x_2 \le 2 \end{aligned}$$

is solved with an initial guess (–1.2, 1.0), and the solution is printed.

```
      USE BCPOL_INT
      USE UMACH_INT
!                                  Variable declarations
      INTEGER    N
      PARAMETER  (N=2)
!
      INTEGER    IBTYPE, K, NOUT
      REAL       FTOL, FVALUE, X(N), XGUESS(N), XLB(N), XUB(N)
      EXTERNAL   FCN
```

```
!
!                                  Initializations
!                                  XGUESS = (-1.2,  1.0)
!                                  XLB    = (-2.0, -1.0)
!                                  XUB    = ( 0.5,  2.0)
      DATA  XGUESS/-1.2, 1.0/, XLB/-2.0E0, -1.0E0/, XUB/0.5E0, 2.0E0/
!
      FTOL   = 1.0E-5
      IBTYPE = 0
!
      CALL BCPOL (FCN, IBTYPE, XLB, XUB, X, XGUESS=XGUESS, FTOL=FTOL, &
                 FVALUE=FVALUE)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) (X(K),K=1,N), FVALUE
99999 FORMAT ('  The best estimate for the minimum value of the', /, &
             '  function is X = (', 2(2X,F4.2), ')', /, '  with ', &
             'function value FVALUE = ', E12.6)
!
      END
!                                  External function to be minimized
      SUBROUTINE FCN (N, X, F)
      INTEGER    N
      REAL       X(N), F
!
      F = 100.0*(X(2)-X(1)*X(1))**2 + (1.0-X(1))**2
      RETURN
      END
```

Output

```
The best estimate for the minimum value of the
function is X = (  0.50  0.25)
with function value FVALUE = 0.250002E+00
```

Comments

1. Workspace may be explicitly provided, if desired, by use of B2POL/DB2POL. The reference is:

   ```
   CALL B2POL (FCN, N, XGUESS, IBTYPE, XLB, XUB, FTOL,
          MAXFCN, X, FVALUE, WK)
   ```

   The additional argument is:

   WK — Real work vector of length `2 * N**2 + 5 * N`

2. Informational error

Type	Code	
3	1	The maximum number of function evaluations is exceeded.

3. Since BCPOL uses only function-value information at each step to determine a new approximate minimum, it could be quite inefficient on smooth problems compared to other methods such as those implemented in routine BCONF (page 1243), which takes

into account derivative information at each iteration. Hence, routine `BCPOL` should only be used as a last resort. Briefly, a set of 2 `*` `N` points in an `N`-dimensional space is called a complex. The minimization process iterates by replacing the point with the largest function value by a new point with a smaller function value. The iteration continues until all the points cluster sufficiently close to a minimum.

Description

The routine `BCPOL` uses the complex method to find a minimum point of a function of n variables. The method is based on function comparison; no smoothness is assumed. It starts with $2n$ points $x_1, x_2, \ldots, x_{2n}$. At each iteration, a new point is generated to replace the worst point x_j, which has the largest function value among these $2n$ points. The new point is constructed by the following formula:

$$x_k = c + \alpha(c - x_j)$$

where

$$c = \frac{1}{2n-1} \sum_{i \neq j} x_i$$

and α ($\alpha > 0$) is the *reflection coefficient*.

When x_k is a best point, that is, when $f(x_k) \leq f(x_i)$ for $i = 1, \ldots, 2n$, an expansion point is computed $x_e = c + \beta(x_k - c)$, where $\beta(\beta > 1)$ is called the *expansion coefficient*. If the new point is a worst point, then the complex would be contracted to get a better new point. If the contraction step is unsuccessful, the complex is shrunk by moving the vertices halfway toward the current best point. Whenever the new point generated is beyond the bound, it will be set to the bound. This procedure is repeated until one of the following stopping criteria is satisfied:

Criterion 1:

$$f_{best} - f_{worst} \leq \varepsilon_f(1. + |f_{best}|)$$

Criterion 2:

$$\sum_{i=1}^{2n} \left(f_i - \frac{\sum_{j=1}^{2n} f_j}{2n}\right)^2 \leq \varepsilon_f$$

where $f_i = f(x_i)$, $f_j = f(x_j)$, and ε_f is a given tolerance. For a complete description, see Nelder and Mead (1965) or Gill et al. (1981).

BCLSF

Solves a nonlinear least squares problem subject to bounds on the variables using a modified Levenberg-Marquardt algorithm and a finite-difference Jacobian.

Required Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (M, N, X, F)`, where

`M` – Length of `F`. (Input)

`N` – Length of `X`. (Input)

`X` – The point at which the function is evaluated. (Input)
`X` should not be changed by `FCN`.

`F` – The computed function at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

M — Number of functions. (Input)

IBTYPE — Scalar indicating the types of bounds on variables. (Input)

IBTYPE	**Action**
0	User will supply all the bounds.
1	All variables are nonnegative.
2	All variables are nonpositive.
3	User supplies only the bounds on 1st variable, all other variables will have the same bounds.

XLB — Vector of length `N` containing the lower bounds on variables. (Input, if `IBTYPE` = 0; output, if `IBTYPE` = 1 or 2; input/output, if `IBTYPE` = 3)

XUB — Vector of length `N` containing the upper bounds on variables. (Input, if `IBTYPE` = 0; output, if `IBTYPE` = 1 or 2; input/output, if `IBTYPE` = 3)

X — Vector of length `N` containing the approximate solution. (Output)

Optional Arguments

N — Number of variables. (Input)
`N` must be less than or equal to `M`.
Default: `N` = size (`X`,1).

XGUESS — Vector of length `N` containing the initial guess. (Input)
Default: `XGUESS` = 0.0.

XSCALE — Vector of length `N` containing the diagonal scaling matrix for the variables. (Input)
`XSCALE` is used mainly in scaling the gradient and the distance between two points. By default, the values for `XSCALE` are set internally. See `IPARAM`(6) in Comment 4.

FSCALE — Vector of length `M` containing the diagonal scaling matrix for the functions. (Input)
`FSCALE` is used mainly in scaling the gradient. In the absence of other information, set all entries to 1.0.
Default: `FSCALE` = 1.0.

IPARAM — Parameter vector of length 6. (Input/Output)
Set `IPARAM`(1) to zero for default values of `IPARAM` and `RPARAM`. See Comment 4.
Default: `IPARAM`= 0.

RPARAM — Parameter vector of length 7. (Input/Output)
See Comment 4.

FVEC — Vector of length `M` containing the residuals at the approximate solution. (Output)

FJAC — `M` by `N` matrix containing a finite difference approximate Jacobian at the approximate solution. (Output)

LDFJAC — Leading dimension of `FJAC` exactly as specified in the dimension statement of the calling program. (Input)
Default: `LDFJAC` = size (`FJAC` ,1).

FORTRAN 90 Interface

Generic: `CALL BCLSF (FCN, M, IBTYPE, XLB, XUB, X [,…])`

Specific: The specific interface names are `S_BCLSF` and `D_BCLSF`.

FORTRAN 77 Interface

Single:
```
CALL BCLSF (FCN, M, N, XGUESS, IBTYPE, XLB, XUB, XSCALE,
FSCALE, IPARAM, RPARAM, X, FVEC, FJAC, LDFJAC)
```

Double: The double precision name is `DBCLSF`.

Example

The nonlinear least squares problem

$$\min_{x \in \mathbf{R}^2} \frac{1}{2} \sum_{i=1}^{2} f_i(x)^2$$

$$\text{subject to } -2 \le x_1 \le 0.5$$

$$-1 \le x_2 \le 2$$

where

$$f_1(x) = 10(x_2 - x_1^2) \text{ and } f_2(x) = (1 - x_1)$$

is solved with an initial guess (−1.2, 1.0) and default values for parameters.

```
      USE BCLSF_INT
      USE UMACH_INT
!                                 Declaration of variables
      INTEGER    M, N
      PARAMETER  (M=2, N=2)
!
      INTEGER    IPARAM(7), ITP, NOUT
      REAL       FSCALE(M), FVEC(M), X(N), XGUESS(N), XLB(N), XS(N), XUB(N)
      EXTERNAL   ROSBCK
!                                 Compute the least squares for the
!                                 Rosenbrock function.
      DATA XGUESS/-1.2E0, 1.0E0/
      DATA XLB/-2.0E0, -1.0E0/, XUB/0.5E0, 2.0E0/
!                                 All the bounds are provided
      ITP = 0
!                                 Default parameters are used
      IPARAM(1) = 0
!
      CALL BCLSF (ROSBCK, M, ITP, XLB, XUB, X, XGUESS=XGUESS, &
                 IPARAM=IPARAM, FVEC=FVEC)
!                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, FVEC, IPARAM(3), IPARAM(4)
!
99999 FORMAT ('  The solution is ', 2F9.4, //, '  The function ', &
            'evaluated at the solution is ', /, 18X, 2F9.4, //, &
            '  The number of iterations is ', 10X, I3, /, '  The ', &
            'number of function evaluations is ', I3, /)
      END
!
      SUBROUTINE ROSBCK (M, N, X, F)
      INTEGER    M, N
      REAL       X(N), F(M)
!
      F(1) = 1.0E1*(X(2)-X(1)*X(1))
      F(2) = 1.0E0 - X(1)
      RETURN
      END
```

Output

```
The solution is    0.5000   0.2500

The function evaluated at the solution is
0.0000   0.5000

The number of iterations is            15
The number of function evaluations is  20
```

Comments

1. Workspace may be explicitly provided, if desired, by use of B2LSF/DB2LSF. The reference is:

   ```
   CALL B2LSF (FCN, M, N, XGUESS, IBTYPE, XLB, XUB, XSCALE, FSCALE,
   IPARAM, RPARAM, X, FVEC, FJAC, LDFJAC, WK, IWK)
   ```

   The additional arguments are as follows:

   WK — Work vector of length 11 * N + 3 * M − 1. WK contains the following information on output: The second N locations contain the last step taken. The third N locations contain the last Gauss-Newton step. The fourth N locations contain an estimate of the gradient at the solution.

   IWK — Work vector of length 2 * N containing the permutations used in the QR factorization of the Jacobian at the solution.

2. Informational errors

Type	Code	
3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
3	2	The iterates appear to be converging to a noncritical point.
4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
3	6	Five consecutive steps have been taken with the maximum step length.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or STEPTL is too big.

3. The first stopping criterion for BCLSF occurs when the norm of the function is less than the absolute function tolerance. The second stopping criterion occurs when the norm of the scaled gradient is less than the given gradient tolerance. The third stopping criterion for BCLSF occurs when the scaled distance between the last two steps is less than the step tolerance.

4. If the default parameters are desired for BCLSF, then set IPARAM(1) to zero and call the routine BCLSF. Otherwise, if any nondefault parameters are desired for IPARAM or RPARAM, then the following steps should be taken before calling BCLSF:

   CALL U4LSF (IPARAM, RPARAM)
   Set nondefault values for desired IPARAM, RPARAM elements.

   Note that the call to U4LSF will set IPARAM and RPARAM to their default values so only nondefault values need to be set above.

   The following is a list of the parameters and the default values:

IPARAM — Integer vector of length 6.
`IPARAM`(1) = Initialization flag.

`IPARAM`(2) = Number of good digits in the function.
Default: Machine dependent.

`IPARAM`(3) = Maximum number of iterations.
Default: 100.

`IPARAM`(4) = Maximum number of function evaluations.
Default: 400.

`IPARAM`(5) = Maximum number of Jacobian evaluations.
Default: 100.

`IPARAM`(6) = Internal variable scaling flag.
If `IPARAM`(6) = 1, then the values for `XSCALE` are set internally.
Default: 1.

RPARAM — Real vector of length 7.
`RPARAM`(1) = Scaled gradient tolerance.
The *i*-th component of the scaled gradient at x is calculated as

$$\frac{|g_i| * \max\left(|x_i|, 1/s_i\right)}{\left\|F(x)\right\|_2^2}$$

where

$$g_i = \left(J(x)^T F(x)\right)_i * \left(f_s\right)_i^2$$

$J(x)$ is the Jacobian, s = `XSCALE`, and f_s = `FSCALE`.
Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

`RPARAM`(2) = Scaled step tolerance. (`STEPTL`)
The i-th component of the scaled step between two points x and y is computed as

$$\frac{|x_i - y_i|}{\max\left(|x_i|, 1/s_i\right)}$$

where s = `XSCALE`.

Default: $\varepsilon^{2/3}$ where ε is the machine precision.

`RPARAM`(3) = Relative function tolerance.
Default: max(10^{-10}, $\varepsilon^{2/3}$), max(10^{-20}, $\varepsilon^{2/3}$) in double where ε is the machine precision.

`RPARAM`(4) = Absolute function tolerance.
Default: max (10^{-20}, ε^2), max(10^{-40}, ε^2) in double where ε is the machine precision.

`RPARAM`(5) = False convergence tolerance.
Default: 100 ε where ε is the machine precision.

`RPARAM`(6) = Maximum allowable step size.
Default: 1000 max(ε_1, ε_2) where

$$\varepsilon_1 = \sqrt{\sum_{i=1}^{n} (s_i t_i)^2}$$

$\varepsilon_2 = \|s\|_2$, s = `XSCALE`, and t = `XGUESS`.

`RPARAM`(7) = Size of initial trust region radius.
Default: based on the initial scaled Cauchy step.

If double precision is desired, then `DU4LSF` is called and `RPARAM` is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to “Error Handling” in the Introduction.

Description

The routine `BCLSF` uses a modified Levenberg-Marquardt method and an active set strategy to solve nonlinear least squares problems subject to simple bounds on the variables. The problem is stated as follows:

$$\min_{x \in \mathbf{R}^n} \frac{1}{2} F(x)^T F(x) = \frac{1}{2} \sum_{i=1}^{m} f_i(x)^2$$

$$\text{subject to } l \le x \le u$$

where $m \ge n$, $F : \mathbf{R}^n \to \mathbf{R}^m$, and $f_i(x)$ is the i-th component function of $F(x)$. From a given starting point, an active set IA, which contains the indices of the variables at their bounds, is built. A variable is called a “free variable” if it is not in the active set. The routine then computes the search direction for the free variables according to the formula

$$d = -(J^T J + \mu I)^{-1} J^T F$$

where μ is the Levenberg-Marquardt parameter, $F = F(x)$, and J is the Jacobian with respect to the free variables. The search direction for the variables in IA is set to zero. The trust region approach discussed by Dennis and Schnabel (1983) is used to find the new point. Finally, the optimality conditions are checked. The conditions are

$$\|g(x_i)\| \le \varepsilon,\ l_i < x_i < u_i$$

$$g(x_i) < 0,\ \ x_i = u_i$$

$$g(x_i) > 0,\ x_i = l_i$$

where ε is a gradient tolerance. This process is repeated until the optimality criterion is achieved.

The active set is changed only when a free variable hits its bounds during an iteration or the optimality condition is met for the free variables but not for all variables in IA, the active set. In the latter case, a variable that violates the optimality condition will be dropped out of IA. For more detail on the Levenberg-Marquardt method, see Levenberg (1944), or Marquardt (1963). For more detailed information on active set strategy, see Gill and Murray (1976).

Since a finite-difference method is used to estimate the Jacobian for some single precision calculations, an inaccurate estimate of the Jacobian may cause the algorithm to terminate at a noncritical point. In such cases, high precision arithmetic is recommended. Also, whenever the exact Jacobian can be easily provided, routine `BCLSJ` (page 1281) should be used instead.

BCLSJ

Solves a nonlinear least squares problem subject to bounds on the variables using a modified Levenberg-Marquardt algorithm and a user-supplied Jacobian.

Required Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (M, N, X, F)`, where

`M` – Length of `F`. (Input)

`N` – Length of `X`. (Input)

`X` – The point at which the function is evaluated. (Input)
`X` should not be changed by `FCN`.

`F` – The computed function at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

JAC — User-supplied `SUBROUTINE` to evaluate the Jacobian at a point `X`. The usage is `CALL JAC (M, N, X, FJAC, LDFJAC)`, where

`M` – Length of `F`. (Input)

`N` – Length of `X`. (Input)

X – The point at which the function is evaluated. (Input)
X should not be changed by FCN.

FJAC – The computed M by N Jacobian at the point X. (Output)

LDFJAC – Leading dimension of FJAC. (Input)

JAC must be declared EXTERNAL in the calling program.

M — Number of functions. (Input)

IBTYPE — Scalar indicating the types of bounds on variables. (Input)

IBTYPE	**Action**
0	User will supply all the bounds.
1	All variables are nonnegative.
2	All variables are nonpositive.
3	User supplies only the bounds on 1st variable, all other variables will have the same bounds.

XLB — Vector of length N containing the lower bounds on variables. (Input, if IBTYPE = 0; output, if IBTYPE = 1 or 2; input/output, if IBTYPE = 3)

XUB — Vector of length N containing the upper bounds on variables. (Input, if IBTYPE = 0; output, if IBTYPE = 1 or 2; input/output, if IBTYPE = 3)

X — Vector of length N containing the approximate solution. (Output)

Optional Arguments

N — Number of variables. (Input)
N must be less than or equal to M.
Default: N = size (X,1).

XGUESS — Vector of length N containing the initial guess. (Input)
Default: XGUESS = 0.0.

XSCALE — Vector of length N containing the diagonal scaling matrix for the variables. (Input)
XSCALE is used mainly in scaling the gradient and the distance between two points. By default, the values for XSCALE are set internally. See IPARAM(6) in Comment 4.

FSCALE — Vector of length `M` containing the diagonal scaling matrix for the functions. (Input)
`FSCALE` is used mainly in scaling the gradient. In the absence of other information, set all entries to 1.0.
Default: `FSCALE` = 1.0.

IPARAM — Parameter vector of length 6. (Input/Output)
Set `IPARAM`(1) to zero for default values of `IPARAM` and `RPARAM`. See Comment 4.
Default: `IPARAM`= 0.

RPARAM — Parameter vector of length 7. (Input/Output)
See Comment 4.

FVEC — Vector of length `M` containing the residuals at the approximate solution. (Output)

FJAC — `M` by `N` matrix containing a finite difference approximate Jacobian at the approximate solution. (Output)

LDFJAC — Leading dimension of `FJAC` exactly as specified in the dimension statement of the calling program. (Input)
Default: `LDFJAC` size = (`FJAC`,1).

FORTRAN 90 Interface

Generic: `CALL BCLSJ (FCN, JAC, M, IBTYPE, XLB, XUB, X [,…])`

Specific: The specific interface names are `S_BCLSJ` and `D_BCLSJ`.

FORTRAN 77 Interface

Single:
```
CALL BCLSJ (FCN, JAC, M, N, XGUESS, IBTYPE, XLB, XUB,
XSCALE, FSCALE, IPARAM, RPARAM, X, FVEC, FJAC,
LDFJAC)
```

Double: The double precision name is `DBCLSJ`.

Example

The nonlinear least squares problem

$$\min_{x\in\mathbf{R}^2} \frac{1}{2}\sum_{i=1}^{2} f_i(x)^2$$

$$\text{subject to } -2 \le x_1 \le 0.5$$

$$-1 \le x_2 \le 2$$

where

$$f_1(x) = 10\left(x_2 - x_1^2\right) \text{ and } f_2(x) = \left(1 - x_1\right)$$

is solved with an initial guess (−1.2, 1.0) and default values for parameters.

```
      USE BCLSJ_INT
      USE UMACH_INT
!                                 Declaration of variables
      INTEGER    LDFJAC, M, N
      PARAMETER  (LDFJAC=2, M=2, N=2)
!
      INTEGER    IPARAM(7), ITP, NOUT
      REAL       FVEC(M), RPARAM(7), X(N), XGUESS(N), XLB(N), XUB(N)
      EXTERNAL   ROSBCK, ROSJAC
!                                 Compute the least squares for the
!                                 Rosenbrock function.
      DATA XGUESS/-1.2E0, 1.0E0/
      DATA XLB/-2.0E0, -1.0E0/, XUB/0.5E0, 2.0E0/
!                                 All the bounds are provided
      ITP = 0
!                                 Default parameters are used
      IPARAM(1) = 0
!
      CALL BCLSJ (ROSBCK,ROSJAC,M,ITP,XLB,XUB,X,XGUESS=XGUESS, &
                IPARAM=IPARAM, FVEC=FVEC)
!                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, FVEC, IPARAM(3), IPARAM(4)
!
99999 FORMAT ('  The solution is ', 2F9.4, //, '  The function ', &
            'evaluated at the solution is ', /, 18X, 2F9.4, //, &
            '  The number of iterations is ', 10X, I3, /, '  The ', &
            'number of function evaluations is ', I3, /)
      END
!
      SUBROUTINE ROSBCK (M, N, X, F)
      INTEGER    M, N
      REAL       X(N), F(M)
!
      F(1) = 1.0E1*(X(2)-X(1)*X(1))
      F(2) = 1.0E0 - X(1)
      RETURN
      END
!
      SUBROUTINE ROSJAC (M, N, X, FJAC, LDFJAC)
      INTEGER    M, N, LDFJAC
      REAL       X(N), FJAC(LDFJAC,N)
!
      FJAC(1,1) = -20.0E0*X(1)
      FJAC(2,1) = -1.0E0
      FJAC(1,2) = 10.0E0
      FJAC(2,2) = 0.0E0
      RETURN
      END
```

Output

```
The solution is    0.5000   0.2500

The function evaluated at the solution is
0.0000   0.5000

The number of iterations is            13
The number of function evaluations is  21
```

Comments

1. Workspace may be explicitly provided, if desired, by use of `B2LSJ/DB2LSJ`. The reference is:

   ```
   CALL B2LSJ (FCN, JAC, M, N, XGUESS, IBTYPE, XLB, XUB, XSCALE,
   FSCALE, IPARAM, RPARAM, X, FVEC, FJAC, LDFJAC, WK, IWK)
   ```

   The additional arguments are as follows:

   WK — Work vector of length 11 * `N` + 3 * `M` – 1. `WK` contains the following information on output: The second `N` locations contain the last step taken. The third `N` locations contain the last Gauss-Newton step. The fourth `N` locations contain an estimate of the gradient at the solution.

   IWK — Work vector of length 2 * `N` containing the permutations used in the `QR` factorization of the Jacobian at the solution.

2. Informational errors

Type	Code	
3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
3	2	The iterates appear to be converging to a noncritical point.
4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
3	6	Five consecutive steps have been taken with the maximum step length.
4	5	Maximum number of Jacobian evaluations exceeded.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or `STEPTL` is too big.

3. The first stopping criterion for `BCLSJ` occurs when the norm of the function is less than the absolute function tolerance. The second stopping criterion occurs when the norm of the scaled gradient is less than the given gradient tolerance. The third stopping criterion for `BCLSJ` occurs when the scaled distance between the last two steps is less than the step tolerance.

4. If the default parameters are desired for `BCLSJ`, then set `IPARAM`(1) to zero and call the routine `BCLSJ`. Otherwise, if any nondefault parameters are desired for `IPARAM` or `RPARAM`, then the following steps should be taken before calling `BCLSJ`:

CALL U4LSF (IPARAM, RPARAM)
Set nondefault values for desired IPARAM, RPARAM elements.

Note that the call to U4LSF will set IPARAM and RPARAM to their default values so only nondefault values need to be set above.

The following is a list of the parameters and the default values:

IPARAM — Integer vector of length 6.
IPARAM(1) = Initialization flag.

IPARAM(2) = Number of good digits in the function.
Default: Machine dependent.

IPARAM(3) = Maximum number of iterations.
Default: 100.

IPARAM(4) = Maximum number of function evaluations.
Default: 400.

IPARAM(5) = Maximum number of Jacobian evaluations.
Default: 100.

IPARAM(6) = Internal variable scaling flag.

If IPARAM(6) = 1, then the values for XSCALE are set internally.
Default: 1.

RPARAM — Real vector of length 7.
RPARAM(1) = Scaled gradient tolerance.
The *i*-th component of the scaled gradient at x is calculated as

$$\frac{|g_i| * \max(|x_i|, 1/s_i)}{\|F(x)\|_2^2}$$

where

$$g_i = \left(J(x)^T F(x)\right)_i * (f_s)_i^2$$

$J(x)$ is the Jacobian, s = XSCALE, and f_s = FSCALE.
Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

RPARAM(2) = Scaled step tolerance. (STEPTL)
The *i*-th component of the scaled step
between two points *x* and *y* is computed as

$$\frac{\left|x_i - y_i\right|}{\max\left(\left|x_i\right|, 1/s_i\right)}$$

where *s* = XSCALE.

Default: $\varepsilon^{2/3}$ where ε is the machine precision.

RPARAM(3) = Relative function tolerance.
Default: $\max(10^{-10}, \varepsilon^{2/3})$, $\max(10^{-20}, \varepsilon^{2/3})$ in double where ε is the machine precision.

RPARAM(4) = Absolute function tolerance.
Default: $\max(10^{-20}, \varepsilon^{2})$, $\max(10^{-40}, \varepsilon^{2})$ in double where ε is the machine precision.

RPARAM(5) = False convergence tolerance.
Default: 100ε where ε is the machine precision.

RPARAM(6) = Maximum allowable step size.
Default: 1000 $\max(\varepsilon_1, \varepsilon_2)$ where

$$\varepsilon_1 = \sqrt{\sum\nolimits_{i=1}^{n} \left(s_i t_i\right)^2}$$

$\varepsilon_2 = \|s\|_2$, *s* = XSCALE, and *t* = XGUESS.

RPARAM(7) = Size of initial trust region radius.
Default: based on the initial scaled Cauchy step.

If double precision is desired, then DU4LSF is called and RPARAM is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to ERROR HANDLING in the Introduction.

Description

The routine BCLSJ uses a modified Levenberg-Marquardt method and an active set strategy to solve nonlinear least squares problems subject to simple bounds on the variables. The problem is stated as follows:

$$\min_{x \in \mathbf{R}^n} \frac{1}{2} F(x)^T F(x) = \frac{1}{2} \sum_{i=1}^{m} f_i(x)^2$$

subject to $l \le x \le u$

where $m \geq n$, $F : \mathbf{R}^n \rightarrow \mathbf{R}^m$, and $f_i(x)$ is the i-th component function of $F(x)$. From a given starting point, an active set IA, which contains the indices of the variables at their bounds, is built. A variable is called a "free variable" if it is not in the active set. The routine then computes the search direction for the free variables according to the formula

$$d = -\left(J^T J + \mu I\right)^{-1} J^T F$$

where is the Levenberg-Marquardt parameter, $F = F(x)$, and J is the Jacobian with respect to the free variables. The search direction for the variables in IA is set to zero. The trust region approach discussed by Dennis and Schnabel (1983) is used to find the new point. Finally, the optimality conditions are checked. The conditions are

$$\|g(x_i)\| \leq \varepsilon, \; l_i < x_i < u_i$$

$$g(x_i) < 0, \;\; x_i = u_i$$

$$g(x_i) > 0, \; x_i = l_i$$

where ε is a gradient tolerance. This process is repeated until the optimality criterion is achieved.

The active set is changed only when a free variable hits its bounds during an iteration or the optimality condition is met for the free variables but not for all variables in IA, the active set. In the latter case, a variable that violates the optimality condition will be dropped out of IA. For more detail on the Levenberg-Marquardt method, see Levenberg (1944) or Marquardt (1963). For more detailed information on active set strategy, see Gill and Murray (1976).

BCNLS

Solves a nonlinear least-squares problem subject to bounds on the variables and general linear constraints.

Required Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is
`CALL FCN (M, N, X, F)`, where
`M` – Number of functions. (Input)
`N` – Number of variables. (Input)
`X` – Array of length `N` containing the point at which the function will be evaluated. (Input)
`F` – Array of length `M` containing the computed function at the point `X`. (Output)
The routine `FCN` must be declared `EXTERNAL` in the calling program.

M — Number of functions. (Input)

C — `MCON` × `N` matrix containing the coefficients of the `MCON` general linear constraints. (Input)

BL — Vector of length `MCON` containing the lower limit of the general constraints. (Input).

BU — Vector of length `MCON` containing the upper limit of the general constraints. (Input).

IRTYPE — Vector of length `MCON` indicating the types of general constraints in the matrix `C`. (Input)
Let `R(I) = C(I, 1)*X(1) + … + C(I, N)*X(N)`. Then the value of `IRTYPE(I)` signifies the following:

IRTYPE(I)	**I-th CONSTRAINT**
0	`BL(I).EQ.R(I).EQ.BU(I)`
1	`R(I).LE.BU(I)`
2	`R(I).GE.BL(I)`
3	`BL(I).LE.R(I).LE.BU(I)`

XLB — Vector of length `N` containing the lower bounds on variables; if there is no lower bound on a variable, then 1.0E30 should be set as the lower bound. (Input)

XUB — Vector of length `N` containing the upper bounds on variables; if there is no upper bound on a variable, then −1.0E30 should be set as the upper bound. (Input)

X — Vector of length `N` containing the approximate solution. (Output)

Optional Arguments

N — Number of variables. (Input)
Default: `N` = size (`C`,2).

MCON — The number of general linear constraints for the system, not including simple bounds. (Input)
Default: `MCON` = size (`C`,1).

LDC — Leading dimension of `C` exactly as specified in the dimension statement of the calling program. (Input)
`LDC` must be at least `MCON`.
Default: `LDC` = size (`C`,1).

XGUESS — Vector of length `N` containing the initial guess. (Input)
Default: `XGUESS` = 0.0.

RNORM — The Euclidean length of components of the function $f(x)$ after the approximate solution has been found. (Output).

ISTAT — Scalar indicating further information about the approximate solution `X`. (Output)
See the Comments section for a description of the tolerances and the vectors `IPARAM` and `RPARAM`.

ISTAT **Meaning**

1 The function $f(x)$ has a length less than `TOLF` = `RPARAM`(1). This is the expected value for `ISTAT` when an actual zero value of $f(x)$ is anticipated.

2 The function $f(x)$ has reached a local minimum. This is the expected value for `ISTAT` when a nonzero value of $f(x)$ is anticipated.

3 A small change (absolute) was noted for the vector x. A full model problem step was taken. The condition for `ISTAT` = 2 may also be satisfied, so that a minimum has been found. However, this test is made before the test for `ISTAT` = 2.

4 A small change (relative) was noted for the vector x. A full model problem step was taken. The condition for `ISTAT` = 2 may also be satisfied, so that a minimum has been found. However, this test is made before the test for `ISTAT` = 2.

5 The number of terms in the quadratic model is being restricted by the amount of storage allowed for that purpose. It is suggested, but not required, that additional storage be given for the quadratic model parameters. This is accessed through the vector
IPARAM, documented below.

6 Return for evaluation of function and Jacobian if reverse communication is desired. See the Comments below.

FORTRAN 90 Interface

Generic: `CALL BCNLS (FCN, M, C, BL, BU, IRTYPE, XLB, XUB, X [,…])`

Specific: The specific interface names are `S_BCNLS` and `D_BCNLS`.

FORTRAN 77 Interface

Single:
```
CALL BCNLS (FCN, M, N, MCON, C, LDC, BL, BU, IRTYPE,
XLB, XUB, XGUESS, X, RNORM, ISTAT)
```

Double: The double precision name is `DBCNLS`.

Example 1

This example finds the four variables x_1, x_2, x_3, x_4 that are in the model function

$$h(t) = x_1 e^{x_2 t} + x_3 e^{x_4 t}$$

There are values of *h(t)* at five values of *t*.

$h(0.05) = 2.206$

$h(0.1) = 1.994$

$h(0.4) = 1.35$

$h(0.5) = 1.216$

$h(1.0) = 0.7358$

There are also the constraints that $x_2, x_4 \le 0$, $x_1, x_3 \ge 0$, and x_2 and x_4 must be separated by at least 0.05. Nothing more about the values of the parameters is known so the initial guess is 0.

```
      USE BCNLS_INT
      USE UMACH_INT
      USE WRRRN_INT
      INTEGER    MCON, N
      PARAMETER  (MCON=1, N=4)
!                                 SPECIFICATIONS FOR PARAMETERS
      INTEGER    LDC, M
      PARAMETER  (M=5, LDC=MCON)
!                                 SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    IRTYPE(MCON), NOUT
      REAL       BL(MCON), C(MCON,N), RNORM, X(N), XLB(N), &
                XUB(N)
!                                 SPECIFICATIONS FOR SUBROUTINES
!                                 SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   FCN
!
      CALL UMACH (2, NOUT)
!                                 Define the separation between x(2)
!                                 and x(4)
      C(1,1) = 0.0
      C(1,2) = 1.0
      C(1,3) = 0.0
      C(1,4) = -1.0
      BL(1) = 0.05
      IRTYPE(1) = 2
!                                 Set lower bounds on variables
      XLB(1) = 0.0
      XLB(2) = 1.0E30
      XLB(3) = 0.0
      XLB(4) = 1.0E30
!                                 Set upper bounds on variables
      XUB(1) = -1.0E30
      XUB(2) = 0.0
      XUB(3) = -1.0E30
      XUB(4) = 0.0
!
      CALL BCNLS (FCN, M, C, BL, BL, IRTYPE, XLB, XUB, X, RNORM=RNORM)
      CALL WRRRN ('X', X, 1, N, 1)
      WRITE (NOUT,99999) RNORM
99999 FORMAT (/, 'rnorm = ', E10.5)
      END
!
      SUBROUTINE FCN (M, N, X, F)
!                                 SPECIFICATIONS FOR ARGUMENTS
      INTEGER    M, N
      REAL       X(*), F(*)
!                                 SPECIFICATIONS FOR LOCAL VARIABLES
```

```
      INTEGER    I
!                                 SPECIFICATIONS FOR SAVE VARIABLES
      REAL       H(5), T(5)
      SAVE       H, T
!                                 SPECIFICATIONS FOR INTRINSICS
      INTRINSIC  EXP
      REAL       EXP
!
      DATA T/0.05, 0.1, 0.4, 0.5, 1.0/
      DATA H/2.206, 1.994, 1.35, 1.216, 0.7358/
!
      DO 10  I=1, M
         F(I) = X(1)*EXP(X(2)*T(I)) + X(3)*EXP(X(4)*T(I)) - H(I)
   10 CONTINUE
      RETURN
      END
```

Output

```
                  X
     1       2       3       4
   1.999  -1.000   0.500  -9.954
rnorm = .42425E-03
```

Comments

1. Workspace may be explicitly provided, if desired, by use of B2NLS/DB2NLS. The reference is:

   ```
   CALL B2NLS (FCN, M, N, MCON, C, LDC, BL, BU, IRTYPE, XLB, XUB,
   XGUESS, X, RNORM,ISTAT, IPARAM, RPARAM, JAC, F, FJ, LDFJ,
   IWORK, LIWORK, WORK, LWORK)
   ```

   The additional arguments are as follows:

   IPARAM — Integer vector of length six used to change certain default attributes of BCNLS. (Input).
   If the default parameters are desired for BCNLS, set IPARAM(1) to zero.
   Otherwise, if any nondefault parameters are desired for IPARAM or RPARAM, the following steps should be taken before calling B2NLS:

   CALL B7NLS (IPARAM, RPARAM)
   Set nondefault values for IPARAM and RPARAM.

   If double precision is being used, DB7NLS should be called instead. Following is a list of parameters and the default values.

   IPARAM(1) = Initialization flag.

   IPARAM(2) = ITMAX, the maximum number of iterations allowed.
   Default: 75

IPARAM(3) = a flag that suppresses the use of the quadratic model in the inner loop. If set to one, then the quadratic model is never used. Otherwise use the quadratic model where appropriate. This option decreases the amount of workspace as well as the computing overhead required. A user may wish to determine if the application really requires the use of the quadratic model.
Default: 0

IPARAM(4) = NTERMS, one more than the maximum number of terms used in the quadratic model.
Default: 5

IPARAM(5) = RCSTAT, a flag that determines whether forward or reverse communication is used. If set to zero, forward communication through functions FCN and JAC is used. If set to one, reverse communication is used, and the dummy routines B10LS/DB10LS and B11LS/DB11LS may be used in place of FCN and JAC, respectively. When BCNLS returns with ISTAT = 6, arrays F and FJ are filled with $f(x)$ and the Jacobian of $f(x)$, respectively. BCNLS is then called again.
Default: 0

IPARAM(6) = a flag that determines whether the analytic Jacobian, as supplied in JAC, is used, or if a finite difference approximation is computed. If set to zero, JAC is not accessed and finite differences are used. If set to one, JAC is used to compute the Jacobian.
Default: 0

RPARAM — Real vector of length 7 used to change certain default attributes of BCNLS. (Input)

For the description of RPARAM, we make the following definitions:

FC	current value of the length of $f(x)$
FB	best value of length of $f(x)$
FL	value of length of $f(x)$ at the previous step
PV	predicted value of length of $f(x)$, after the step is taken, using the approximating model
ε	machine epsilon = amach(4)

The conditions |FB − PV| ≤ TOLSNR*FB and |FC − PV| ≤ TOLP*FB and |FC − FL| ≤ TOLSNR*FB together with taking a full model step, must be satisfied before the condition ISTAT = 2 is returned. (Decreasing any of the values for TOLF, TOLD, TOLX, TOLSNR, or TOLP will likely increase the number of iterations required for convergence.)
RPARAM(1) = TOLF, tolerance used for stopping when FC ≤ TOLF.
Default : $\min(1.\text{E}-5, \sqrt{\varepsilon})$

RPARAM(2) = TOLX, tolerance for stopping when change to x values has length less than or equal to TOLX*length of x values.
Default : $\min(1.\mathrm{E}-5, \sqrt{\varepsilon})$

RPARAM(3) = TOLD, tolerance for stopping when change to x values has length less than pr equal to TOLD.
Default : $\min(1.\mathrm{E}-5, \sqrt{\varepsilon})$

RPARAM(4) = TOLSNR, tolerance used in stopping condition ISTAT = 2.
Default: 1.E−5

RPARAM(5) = TOLP, tolerance used in stopping condition ISTAT = 2.
Default: 1.E−5

RPARAM(6) = TOLUSE, tolerance used to avoid values of x in the quadratic model's interpolation of previous points. Decreasing this value may result in more terms being included in the quadratic model.
Default : $\sqrt{\varepsilon}$

RPARAM(7) = COND, largest condition number to allow when solving for the quadratic model coefficients. Increasing this value may result in more terms being included in the quadratic model.
Default: 30

JAC — User-supplied SUBROUTINE to evaluate the Jacobian. The usage is
CALL JAC(M, N, X, FJAC, LDFJAC), where
M – Number of functions. (Input)
N – Number of variables. (Input)
X – Array of length N containing the point at which the Jacobian will be evaluated. (Input)
FJAC – The computed M × N Jacobian at the point X. (Output)
LDFJAC – Leading dimension of the array FJAC. (Input)
The routine JAC must be declared EXTERNAL in the calling program.

F — Real vector of length N used to pass $f(x)$ if reverse communication (IPARAM(4)) is enabled. (Input)

FJ — Real array of size M × N used to store the Jacobian matrix of $f(x)$ if reverse communication (IPARAM(4)) is enabled. (Input)
Specifically,

$$FJ(i,j) = \frac{\partial f_i}{\partial x_j}$$

LDFJ — Leading dimension of `FJ` exactly as specified in the dimension statement of the calling program. (Input)

IWORK — Integer work vector of length `LIWORK`.

LIWORK — Length of work vector `IWORK`. `LIWORK` must be at least
`5MCON` + `12N` + 47 + `MAX`(`M`, `N`)

WORK — Real work vector of length `LWORK`

LWORK — Length of work vector `WORK`. `LWORK` must be at least `41N` + `6M` + `11MCON` + (`M` + `MCON)(N + 1) + NA(NA + 7) + 8 MAX(M, N) + 99. Where NA = MCON + 2N + 6.`

2. Informational errors

Type	Code	
3	1	The function $f(x)$ has reached a value that may be a local minimum. However, the bounds on the trust region defining the size of the step are being hit at each step. Thus, the situation is suspect. (Situations of this type can occur when the solution is at infinity at some of the components of the unknowns, x).
3	2	The model problem solver has noted a value for the linear or quadratic model problem residual vector length that is greater than or equal to the current value of the function, i.e. the Euclidean length of $f(x)$. This situation probably means that the evaluation of $f(x)$ has more uncertainty or noise than is possible to account for in the tolerances used to not a local minimum. The value of x is suspect, but a minimum has probably been found.
3	3	More than `ITMAX` iterations were taken to obtain the solution. The value obtained for x is suspect, although it is the best set of x values that occurred in the entire computation. The value of `ITMAX` can be increased though the `IPARAM` vector.

Description

The routine `BCNLS` solves the nonlinear least squares problem

$$\min \sum_{i=1}^{m} f_i\left(x\right)^2$$

subject to

$$b_l \le Cx \le b_u$$
$$x_l \le x \le x_u$$

`BCNLS` is based on the routine `DQED` by R.J. Hanson and F.T. Krogh. The section of `BCNLS` that approximates, using finite differences, the Jacobian of *f(x)* is a modification of `JACBF` by D.E. Salane.

Example 2

This example solves the same problem as the last example, but reverse communication is used to evaluate *f*(*x*) and the Jacobian of *f*(*x*). The use of the quadratic model is turned off.

```
      USE B2NLS_INT
      USE UMACH_INT
      USE WRRRN_INT
      INTEGER    LDC, LDFJ, M, MCON, N
      PARAMETER  (M=5, MCON=1, N=4, LDC=MCON, LDFJ=M)
!                                 Specifications for local variables
      INTEGER    I, IPARAM(6), IRTYPE(MCON), ISTAT, IWORK(1000), &
                LIWORK, LWORK, NOUT
      REAL       BL(MCON), C(MCON,N), F(M), FJ(M,N), RNORM, RPARAM(7), &
                WORK(1000), X(N), XGUESS(N), XLB(N), XUB(N)
      REAL       H(5), T(5)
      SAVE       H, T
      INTRINSIC  EXP
      REAL       EXP
!                                 Specifications for subroutines
      EXTERNAL   B7NLS
!                                 Specifications for functions
      EXTERNAL   B10LS, B11LS
!
      DATA T/0.05, 0.1, 0.4, 0.5, 1.0/
      DATA H/2.206, 1.994, 1.35, 1.216, 0.7358/
!
      CALL UMACH (2, NOUT)
!                                 Define the separation between x(2)
!                                 and x(4)
      C(1,1)    = 0.0
      C(1,2)    = 1.0
      C(1,3)    = 0.0
      C(1,4)    = -1.0
      BL(1)     = 0.05
      IRTYPE(1) = 2
!                                 Set lower bounds on variables
      XLB(1) = 0.0
      XLB(2) = 1.0E30
      XLB(3) = 0.0
      XLB(4) = 1.0E30
!                                 Set upper bounds on variables
      XUB(1) = -1.0E30
      XUB(2) = 0.0
      XUB(3) = -1.0E30
      XUB(4) = 0.0
!                                 Set initial guess to 0.0
      XGUESS = 0.0E0
!                                 Call B7NLS to set default parameters
      CALL B7NLS (IPARAM, RPARAM)
!                                 Suppress the use of the quadratic
!                                 model, evaluate functions and
!                                 Jacobian by reverse communication
      IPARAM(3) = 1
      IPARAM(5) = 1
      IPARAM(6) = 1
      LWORK     = 1000
      LIWORK    = 1000
!                                  Specify dummy routines for FCN
!                                  and JAC since we are using reverse
!                                  communication
   10 CONTINUE
      CALL B2NLS (B10LS, M, N, MCON, C, LDC, BL, BL, IRTYPE, XLB, &
                 XUB, XGUESS, X, RNORM, ISTAT, IPARAM, RPARAM, &
                 B11LS, F, FJ, LDFJ, IWORK, LIWORK, WORK, LWORK)
!
!                                  Evaluate functions if the routine
!                                  returns with ISTAT = 6
      IF (ISTAT .EQ. 6) THEN
         DO 20  I=1, M
            FJ(I,1) = EXP(X(2)*T(I))
            FJ(I,2) = T(I)*X(1)*FJ(I,1)
            FJ(I,3) = EXP(X(4)*T(I))
            FJ(I,4) = T(I)*X(3)*FJ(I,3)
            F(I) = X(1)*FJ(I,1) + X(3)*FJ(I,3) - H(I)
   20    CONTINUE
         GO TO 10
      END IF
!
      CALL WRRRN ('X', X, 1, N, 1)
      WRITE (NOUT,99999) RNORM
99999 FORMAT (/, 'rnorm = ', E10.5)
      END
```

Output

```
                  X
      1       2       3       4
   1.999  -1.000   0.500  -9.954
rnorm = .42413E-03
```

DLPRS

Solves a linear programming problem via the revised simplex algorithm.

Required Arguments

A — `M` by `NVAR` matrix containing the coefficients of the `M` constraints. (Input)

BL — Vector of length `M` containing the lower limit of the general constraints; if there is no lower limit on the `I`-th constraint, then `BL(I)` is not referenced. (Input)

BU — Vector of length `M` containing the upper limit of the general constraints; if there is no upper limit on the `I`-th constraint, then `BU(I)` is not referenced; if there are no range constraints, `BL` and `BU` can share the same storage locations. (Input)

C — Vector of length `NVAR` containing the coefficients of the objective function. (Input)

IRTYPE — Vector of length `M` indicating the types of general constraints in the matrix `A`. (Input)
Let `R(I)` = `A(I, 1) * XSOL(1)` + … + `A(I, NVAR) * XSOL(NVAR)`. Then, the value of `IRTYPE(I)` signifies the following:

IRTYPE(I)	**I-th Constraint**
0	`BL(I).EQ.R(I).EQ.BU(I)`
1	`R(I).LE.BU(I)`
2	`R(I).GE.BL(I)`
3	`BL(I).LE.R(I).LE.BU(I)`

OBJ — Value of the objective function. (Output)

XSOL — Vector of length `NVAR` containing the primal solution. (Output)

DSOL — Vector of length `M` containing the dual solution. (Output)

Optional Arguments

M — Number of constraints. (Input)
Default: `M` = size (`A`,1).

NVAR — Number of variables. (Input)
Default: `NVAR` = size (`A`,2).

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)
`LDA` must be at least `M`.
Default: `LDA` = size (`A`,1).

XLB — Vector of length `NVAR` containing the lower bound on the variables; if there is no lower bound on a variable, then 1.0E30 should be set as the lower bound. (Input)
Default: `XLB` = 0.0.

XUB — Vector of length `NVAR` containing the upper bound on the variables; if there is no upper bound on a variable, then −1.0E30 should be set as the upper bound. (Input)
Default: `XUB` = 3.4e38 for single precision and 1.79d + 308 for double precision.

FORTRAN 90 Interface

Generic: `CALL DLPRS (A, BL, BU, C, IRTYPE, OBJ, XSOL, DSOL [,…])`

Specific: The specific interface names are `S_DLPRS` and `D_DLPRS`.

FORTRAN 77 Interface

Single: `CALL DLPRS (M, NVAR, A, LDA, BL, BU, C, IRTYPE, XLB, XUB, OBJ, XSOL, DSOL)`

Double: The double precision name is `DDLPRS`.

Example

A linear programming problem is solved.

```
      USE DLPRS_INT
      USE UMACH_INT
      USE SSCAL_INT
      INTEGER    LDA, M, NVAR
      PARAMETER  (M=2, NVAR=2, LDA=M)
!                                 M = number of constraints
!                                 NVAR = number of variables
!
      INTEGER    I, IRTYPE(M), NOUT
      REAL       A(LDA,NVAR), B(M), C(NVAR), DSOL(M), OBJ, XLB(NVAR), &
                XSOL(NVAR), XUB(NVAR)
!
!                                 Set values for the following problem
!
!                                 Max 1.0*XSOL(1) + 3.0*XSOL(2)
!
!                                 XSOL(1) + XSOL(2) .LE. 1.5
!                                 XSOL(1) + XSOL(2) .GE. 0.5
!
!                                 0 .LE. XSOL(1) .LE. 1
!                                 0 .LE. XSOL(2) .LE. 1
!
      DATA XLB/2*0.0/, XUB/2*1.0/
      DATA A/4*1.0/, B/1.5, .5/, C/1.0, 3.0/
      DATA IRTYPE/1, 2/
!                                 To maximize, C must be multiplied by
!                                 -1.
      CALL SSCAL (NVAR, -1.0E0, C, 1)
!                                 Solve the LP problem.  Since there is
!                                 no range constraint, only B is
!                                 needed.
      CALL DLPRS (A, B, B, C, IRTYPE, OBJ, XSOL, DSOL, &
                 XUB=XUB)
!                                 OBJ must be multiplied by -1 to get
!                                 the true maximum.
      OBJ = -OBJ
!                                 DSOL must be multiplied by -1 for
!                                 maximization.
      CALL SSCAL (M, -1.0E0, DSOL, 1)
!                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) OBJ, (XSOL(I),I=1,NVAR), (DSOL(I),I=1,M)
!
99999 FORMAT (//, '   Objective       = ', F9.4, //, '   Primal ',&
             'Solution =', 2F9.4, //, '   Dual solution   =', 2F9.4)
!
      END
```

Output

```
Objective       =    3.5000

Primal Solution =   0.5000   1.0000

Dual solution   =   1.0000   0.0000
```

Comments

1. Workspace may be explicitly provided, if desired, by use of D2PRS/DD2PRS. The reference is:

   ```
   CALL D2PRS (M, NVAR, A, LDA, BL, BU, C, IRTYPE, XLB, XUB, OBJ,
   XSOL, DSOL, AWK, LDAWK, WK, IWK)
   ```

   The additional arguments are as follows:

   AWK — Real work array of dimension 1 by 1. (AWK is not used in the new implementation of the revised simplex algorithm. It is retained merely for calling sequence consistency.)

   LDAWK — Leading dimension of AWK exactly as specified in the dimension statement of the calling program. LDAWK should be 1. (LDAWK is not used in the new implementation of the revised simplex algorithm. It is retained merely for calling sequence consistency.)

   WK — Real work vector of length M * (M + 28).

   IWK — Integer work vector of length 29 * M + 3 * NVAR.

2. Informational errors

Type	Code	
3	1	The problem is unbounded.
4	2	Maximum number of iterations exceeded.
3	3	The problem is infeasible.
4	4	Moved to a vertex that is poorly conditioned; using double precision may help.
4	5	The bounds are inconsistent.

Description

The routine `DLPRS` uses a revised simplex method to solve linear programming problems, i.e., problems of the form

$$\min_{x \in \mathbf{R}^n} c^T x$$

$$\text{subject to } b_l \le Ax \le b_u$$

$$x_l \le x \le x_u$$

where c is the objective coefficient vector, A is the coefficient matrix, and the vectors b_l, b_u, x_l and x_u are the lower and upper bounds on the constraints and the variables, respectively.

For a complete description of the revised simplex method, see Murtagh (1981) or Murty (1983).

SLPRS

Solves a sparse linear programming problem via the revised simplex algorithm.

Required Arguments

A — Vector of length `NZ` containing the coefficients of the `M` constraints. (Input)

IROW — Vector of length `NZ` containing the row numbers of the corresponding element in `A`. (Input)

JCOL — Vector of length `NZ` containing the column numbers of the corresponding elements in *A*. (Input)

BL — Vector of length `M` containing the lower limit of the general constraints; if there is no lower limit on the `I`-th constraint, then `BL(I)` is not referenced. (Input)

BU — Vector of length `M` containing the upper lower limit of the general constraints; if there is no upper limit on the `I`-th constraint, then `BU(I)` is not referenced. (Input)

C — Vector of length `NVAR` containing the coefficients of the objective function. (Input)

IRTYPE — Vector of length `M` indicating the types of general constraints in the matrix `A`. (Input)
Let `R(I) = A(I, 1)*XSOL(1) + … + A(I, NVAR)*XSOL(NVAR)`

IRTYPE(I)	**I-th CONSTRAINT**
0	`BL(I)` = `R(I)` = `BU(I)`
1	`R(I)` ≤ `BU(I)`
2	`R(I)` ≥ `BL(I)`
3	`BL(I)` ≤ `R(I)` ≤ `BU(I)`

OBJ — Value of the objective function. (Output)

XSOL — Vector of length `NVAR` containing the primal solution. (Output)

DSOL — Vector of length `M` containing the dual solution. (Output)

Optional Arguments

M — Number of constraints. (Input)
Default: `M` = size (`IRTYPE`,1).

NVAR — Number of variables. (Input)
Default: `NVAR` = size (`C`,1).

NZ — Number of nonzero coefficients in the matrix *A*. (Input)
Default: `NZ` = size (`A`,1).

XLB — Vector of length `NVAR` containing the lower bound on the variables; if there is no lower bound on a variable, then 1.0E30 should be set as the lower bound. (Input)
Default: `XLB` = 0.0.

XUB — Vector of length `NVAR` containing the upper bound on the variables; if there is no upper bound on a variable, then −1.0E30 should be set as the upper bound. (Input)
Default: `XLB` = 3.4e38 for single precision and 1.79d + 308 for double precision.

FORTRAN 90 Interface

Generic:
```
CALL SLPRS (A, IROW, JCOL, BL, BU, C, IRTYPE,
OBJ, XSOL, DSOL [,…])
```

Specific: The specific interface names are `S_SLPRS` and `D_SLPRS`.

FORTRAN 77 Interface

Single:
```
CALL SLPRS (M, NVAR, NZ, A, IROW, JCOL, BL, BU, C, IRTYPE,
XLB, XUB, OBJ, XSOL, DSOL)
```

Double: The double precision name is `DSLPRS`.

Example

Solve a linear programming problem, with

$$A = \begin{bmatrix} 0 & 0.5 & & & \\ & 1 & 0.5 & & \\ & & 1 & \ddots & \\ & & & \ddots & 0.5 \\ & & & & 1 \end{bmatrix}$$

defined in sparse coordinate format.

```
      USE SLPRS_INT
      USE UMACH_INT
      INTEGER    M, NVAR
      PARAMETER  (M=200, NVAR=200)
!                                 Specifications for local variables
      INTEGER    INDEX, IROW(3*M), J, JCOL(3*M), NOUT, NZ
      REAL       A(3*M), DSOL(M), OBJ, XSOL(NVAR)
      INTEGER    IRTYPE(M)
      REAL       B(M), C(NVAR), XL(NVAR), XU(NVAR)
!                                 Specifications for subroutines
      DATA B/199*1.7, 1.0/
      DATA C/-1.0, -2.0, -3.0, -4.0, -5.0, -6.0, -7.0, -8.0, -9.0, &
      -10.0, 190*-1.0/
      DATA XL/200*0.1/
      DATA XU/200*2.0/
      DATA IRTYPE/200*1/
!
      CALL UMACH (2, NOUT)
!                                 Define A
      INDEX = 1
      DO 10  J=2, M
!                                 Superdiagonal element
         IROW(INDEX) = J - 1
         JCOL(INDEX) = J
         A(INDEX)    = 0.5
!                                 Diagonal element
         IROW(INDEX+1) = J
         JCOL(INDEX+1) = J
         A(INDEX+1) = 1.0
         INDEX      = INDEX + 2
   10 CONTINUE
      NZ = INDEX - 1
!
!
      XL(4) = 0.2
      CALL SLPRS (A, IROW, JCOL, B, B, C, IRTYPE, OBJ, XSOL, DSOL, &
                 NZ=NZ, XLB=XL, XUB=XU)
!
      WRITE (NOUT,99999) OBJ
!
99999 FORMAT (/, 'The value of the objective function is ', E12.6)
!
      END
```

Output

```
The value of the objective function is -.280971E+03
```

Comments

Workspace may be explicitly provided, if desired, by use of S2PRS/DS2PRS. The reference is:

```
CALL S2PRS (M, NVAR, NZ, A, IROW, JCOL, BL, BU, C,
       IRTYPE, XLB, XUB, OBJ, XSOL, DSOL,
       IPARAM, RPARAM, COLSCL, ROWSCL, WORK,
       LW, IWORK, LIW)
```

The additional arguments are as follows:

IPARAM — Integer parameter vector of length 12. If the default parameters are desired for `SLPRS`, then set `IPARAM`(1) to zero and call the routine `SLPRS`. Otherwise, if any nondefault parameters are desired for `IPARAM` or `RPARAM`, then the following steps should be taken before calling `SLPRS`:

`CALL S5PRS` (`IPARAM`, `RPARAM`)
Set nondefault values for `IPARAM` and `RPARAM`.

Note that the call to `S5PRS` will set `IPARAM` and `RPARAM` to their default values so only nondefault values need to be set above.

`IPARAM`(1) = 0 indicates that a minimization problem is solved. If set to 1, a maximization problem is solved.
Default: 0

`IPARAM`(2) = switch indicating the maximum number of iterations to be taken before returning to the user. If set to zero, the maximum number of iterations taken is set to 3*(`NVARS`+`M`). If positive, that value is used as the iteration limit.
Default: `IPARAM`(2) = 0

`IPARAM`(3) = indicator for choosing how columns are selected to enter the basis. If set to zero, the routine uses the steepest edge pricing strategy which is the best local move. If set to one, the minimum reduced cost pricing strategy is used. The steepest edge pricing strategy generally uses fewer iterations than the minimum reduced cost pricing, but each iteration costs more in terms of the amount of calculation performed. However, this is very problem-dependent.
Default: `IPARAM`(3) = 0

`IPARAM`(4) = `MXITBR`, the number of iterations between recalculating the error in the primal solution is used to monitor the error in solving the linear system. This is an expensive calculation and every tenth iteration is generally enough.
Default: `IPARAM`(4) = 10

`IPARAM`(5) = `NPP`, the number of negative reduced costs (at most) to be found at each iteration of choosing a variable to enter the basis. If set to zero, `NPP` = `NVARS` will be used, implying that all of the reduced costs are computed at each such step. This “Partial pricing” may increase the total number of iterations required. However, it decreases the number of calculation required at each iteration. The effect on overall efficiency is very problem-dependent. If set to some positive number, that value is used as `NPP`.
Default: `IPARAM`(5) = 0

IPARAM(6) = IREDFQ, the number of steps between basis matrix redecompositions. Redecompositions also occur whenever the linear systems for the primal and dual systems have lost half their working precision.
Default: IPARAM(6) = 50

IPARAM(7) = LAMAT, the length of the portion of WORK that is allocated to sparse matrix storage and decomposition. LAMAT must be greater than NZ + NVARS + 4.
Default: LAMAT = NZ + NVARS + 5

IPARAM(8) = LBM, then length of the portion of IWORK that is allocated to sparse matrix storage and decomposition. LBM must be positive.
Default: LBM = 8*M

IPARAM(9) = switch indicating that partial results should be saved after the maximum number of iterations, IPARAM(2), or at the optimum. If IPARAM(9) is not zero, data essential to continuing the calculation is saved to a file, attached to unit number IPARAM(9). The data saved includes all the information about the sparse matrix A and information about the current basis. If IPARAM(9) is set to zero, partial results are not saved. It is the responsibility of the calling program to open the output file.

IPARAM(10) = switch indicating that partial results have been computed and stored on unit number IPARAM(10), if greater than zero. If IPARAM(10) is zero, a new problem is started.
Default: IPARAM(10) = 0

IPARAM(11) = switch indicating that the user supplies scale factors for the columns of the matrix *A*. If IPARAM(11) = 0, SLPRS computes the scale factors as the reciprocals of the max norm of each column. If IPARAM(11) is set to one, element I of the vector COLSCL is used as the scale factor for column I of the matrix *A*. The scaling is implicit, so no input data is actually changed.
Default: IPARAM(11) = 0

IPARAM(12) = switch indicating that the user supplied scale factors for the rows of the matrix *A*. If IPARAM(12) is set to zero, no row scaling is one. If IPARAM(12) is set to 1, element I of the vector ROWSCL is used as the scale factor for row I of the matrix *A*. The scaling is implicit, so no input data is actually changed.
Default: IPARAM(12) = 0

RPARAM — Real parameter vector of length 7.

RPARAM(1) = COSTSC, a scale factor for the vector of costs. Normally SLPRS computes this scale factor to be the reciprocal of the max norm if the vector costs after the column scaling has been applied. If RPARAM(1) is zero, SLPRS compute COSTSC.
Default: RPARAM(1) = 0.0

RPARAM(2) = ASMALL, the smallest magnitude of nonzero entries in the matrix *A*. If RPARAM(2) is nonzero, checking is done to ensure that all elements of *A* are at least as

large as RPARAM(2). Otherwise, no checking is done.
Default: RPARAM(2) = 0.0

RPARAM(3) = ABIG, the largest magnitude of nonzero entries in the matrix *A*. If RPARAM(3) is nonzero, checking is done to ensure that all elements of *A* are no larger than RPARAM(3). Otherwise, no checking is done.
Default: RPARAM(3) = 0.0

RPARAM(4) = TOLLS, the relative tolerance used in checking if the residuals are feasible. RPARAM(4) is nonzero, that value is used as TOLLS, otherwise the default value is used.
Default: TOLLS = 1000.0*amach(4)

RPARAM(5) = PHI, the scaling factor used to scale the reduced cost error estimates. In some environments, it may be necessary to reset PHI to the range [0.01, 0.1], particularly on machines with short word length and working precision when solving a large problem. If RPARAM(5) is nonzero, that value is used as PHI, otherwise the default value is used.
Default: PHI = 1.0

RPARAM(6) = TOLABS, an absolute error test on feasibility. Normally a relative test is used with TOLLS (see RPARAM(4)). If this test fails, an absolute test will be applied using the value TOLABS.
Default: TOLABS = 0.0

RPARAM(7) = pivot tolerance of the underlying sparse factorization routine. If RPARAM(7) is set to zero, the default pivot tolerance is used, otherwise, the RPARAM(7) is used.
Default: RPARAM(7) = 0.1

COLSCL — Array of length NVARS containing column scale factors for the matrix *A*. (Input).
COLSCL is not used if IPARAM(11) is set to zero.

ROWSCL — Array of length M containing row scale factors for the matrix *A*. (Input)
ROWSCL is not used if IPARAM(12) is set to zero.

WORK — Work array of length LW.

LW — Length of real work array. LW must be at least
2 + 2NZ + 9NVAR + 27M + MAX(NZ + NVAR + 8, 4NVAR + 7).

IWORK — Integer work array of length LIW.

LIW — Length of integer work array. LIW must be at least
1 + 3NVAR + 41M + MAX(NZ + NVAR + 8, 4NVAR + 7).

Description

This subroutine solves problems of the form

$$\min c^T x$$

subject to

$$b_l \le Ax \le b_u,$$
$$x_l \le x \le x_u$$

where c is the objective coefficient vector, A is the coefficient matrix, and the vectors b_l, b_u, x_l, and x_u are the lower and upper bounds on the constraints and the variables, respectively. `SLPRS` is designed to take advantage of sparsity in A. The routine is based on `DPLO` by Hanson and Hiebert.

QPROG

Solves a quadratic programming problem subject to linear equality/inequality constraints.

Required Arguments

NEQ — The number of linear equality constraints. (Input)

A — `NCON` by `NVAR` matrix. (Input)
The matrix contains the equality contraints in the first `NEQ` rows followed by the inequality constraints.

B — Vector of length `NCON` containing right-hand sides of the linear constraints. (Input)

G — Vector of length `NVAR` containing the coefficients of the linear term of the objective function. (Input)

H — `NVAR` by `NVAR` matrix containing the Hessian matrix of the objective function. (Input)
`H` should be symmetric positive definite; if `H` is not positive definite, the algorithm attempts to solve the QP problem with `H` replaced by a `H` + `DIAGNL * I` such that `H` + `DIAGNL * I` is positive definite. See Comment 3.

SOL — Vector of length `NVAR` containing solution. (Output)

Optional Arguments

NVAR — The number of variables. (Input)
Default: `NVAR` = size (`A`,2).

NCON — The number of linear constraints. (Input)
Default: `NCON` = size (`A`,1).

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)
Default: `LDA` = size (`A`,1).

LDH — Leading dimension of `H` exactly as specified in the dimension statement of the calling program. (Input)
Default: `LDH` = size (`H`,1).

DIAGNL — Scalar equal to the multiple of the identity matrix added to `H` to give a positive definite matrix. (Output)

NACT — Final number of active constraints. (Output)

IACT — Vector of length `NVAR` containing the indices of the final active constraints in the first `NACT` positions. (Output)

ALAMDA — Vector of length `NVAR` containing the Lagrange multiplier estimates of the final active constraints in the first `NACT` positions. (Output)

FORTRAN 90 Interface

Generic: `CALL QPROG (NEQ, A, B, G, H, SOL [,…])`

Specific: The specific interface names are `S_QPROG` and `D_QPROG`.

FORTRAN 77 Interface

Single:
```
CALL QPROG (NVAR, NCON, NEQ, A, LDA, B, G, H, LDH, DIAGNL,
SOL, NACT, IACT, ALAMDA)
```

Double: The double precision name is `DQPROG`.

Example

The quadratic programming problem

$$\min f(x) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 2x_2x_3 - 2x_4x_5 - 2x_1$$
$$\text{subject to} \quad x_1 + x_2 + x_3 + x_4 + x_5 = 5$$
$$x_3 - 2x_4 - 2x_5 = -3$$

is solved.

```
      USE QPROG_INT
      USE UMACH_INT
!                                 Declare variables
      INTEGER    LDA, LDH, NCON, NEQ, NVAR
      PARAMETER  (NCON=2, NEQ=2, NVAR=5, LDA=NCON, LDH=NVAR)
```

```
!
      INTEGER    K, NACT, NOUT
      REAL       A(LDA,NVAR), ALAMDA(NVAR), B(NCON), G(NVAR), &
                H(LDH,LDH), SOL(NVAR)
!
!                                Set values of A, B, G and H.
!                                A = ( 1.0  1.0  1.0  1.0  1.0)
!                                    ( 0.0  0.0  1.0 -2.0 -2.0)
!
!                                B = ( 5.0 -3.0)
!
!                                G = (-2.0  0.0  0.0  0.0  0.0)
!
!                                H = ( 2.0  0.0  0.0  0.0  0.0)
!                                    ( 0.0  2.0 -2.0  0.0  0.0)
!                                    ( 0.0 -2.0  2.0  0.0  0.0)
!                                    ( 0.0  0.0  0.0  2.0 -2.0)
!                                    ( 0.0  0.0  0.0 -2.0  2.0)
!
      DATA A/1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, -2.0, 1.0, -2.0/
      DATA B/5.0, -3.0/
      DATA G/-2.0, 4*0.0/
      DATA H/2.0, 5*0.0, 2.0, -2.0, 3*0.0, -2.0, 2.0, 5*0.0, 2.0, &
          -2.0, 3*0.0, -2.0, 2.0/
!
      CALL QPROG (NEQ, A, B, G, H, SOL)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) (SOL(K),K=1,NVAR)
99999 FORMAT ('  The solution vector is', /, '  SOL = (', 5F6.1, &
            '  )')
!
      END
```

Output

```
The solution vector is
SOL = (   1.0   1.0   1.0   1.0   1.0  )
```

Comments

1. Workspace may be explicitly provided, if desired, by use of Q2ROG/DQ2ROG. The reference is:

   CALL Q2ROG (NVAR, NCON, NEQ, A, LDA, B, G, H, LDH, DIAGNL, SOL, NACT, IACT, ALAMDA, WK)

   The additional argument is:

   WK — Work vector of length (3 * NVAR**2 + 11 * NVAR)/2 + NCON.

2. Informational errors

   Type Code

3	1	Due to the effect of computer rounding error, a change in the variables fail to improve the objective function value; usually the solution is close to optimum.
4	2	The system of equations is inconsistent. There is no solution.

3. If a perturbation of `H`, `H + DIAGNL * I`, was used in the `QP` problem, then `H + DIAGNL * I` should also be used in the definition of the Lagrange multipliers.

Description

The routine `QPROG` is based on M.J.D. Powell's implementation of the Goldfarb and Idnani (1983) dual quadratic programming (QP) algorithm for convex QP problems subject to general linear equality/inequality constraints, i.e., problems of the form

$$\min_{x\in\mathbf{R}^n} g^T x + \frac{1}{2} x^T H x$$

$$\text{subject to} \quad A_1 x = b_1$$

$$A_2 x \ge b_2$$

given the vectors b_1, b_2, and g and the matrices H, A_1, and A_2. H is required to be positive definite. In this case, a unique x solves the problem or the constraints are inconsistent. If H is not positive definite, a positive definite perturbation of H is used in place of H. For more details, see Powell (1983, 1985).

LCONF

Minimizes a general objective function subject to linear equality/inequality constraints.

Required Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (N, X, F)`, where

`N` – Value of `NVAR`. (Input)

`X` – Vector of length `N` at which point the function is evaluated. (Input) `X` should not be changed by `FCN`.

`F` – The computed function value at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

NEQ — The number of linear equality constraints. (Input)

A — `NCON` by `NVAR` matrix. (Input)
The matrix contains the equality constraint gradients in the first `NEQ` rows, followed by the inequality constraint gradients.

B — Vector of length NCON containing right-hand sides of the linear constraints. (Input)
Specifically, the constraints on the variables X(I), I = 1, …, NVAR are A(K, 1) * X(1) + … + A(K, NVAR) * X(NVAR).EQ.B(K), K = 1, …, NEQ.A(K, 1) * X(1) + … + A(K, NVAR) * X(NVAR).LE.B(K), K = NEQ + 1, …, NCON. Note that the data that define the equality constraints come before the data of the inequalities.

XLB — Vector of length NVAR containing the lower bounds on the variables; choose a very large negative value if a component should be unbounded below or set XLB(I) = XUB(I) to freeze the I-th variable. (Input)
Specifically, these simple bounds are XLB(I).LE.X(I), I = 1, …, NVAR.

XUB — Vector of length NVAR containing the upper bounds on the variables; choose a very large positive value if a component should be unbounded above. (Input)
Specifically, these simple bounds are X(I).LE.XUB(I), I = 1, …, NVAR.

SOL — Vector of length NVAR containing solution. (Output)

Optional Arguments

NVAR — The number of variables. (Input)
Default: NVAR = size (A,2).

NCON — The number of linear constraints (excluding simple bounds). (Input)
Default: NCON = size (A,1).

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)
Default: LDA = size (A,1).

XGUESS — Vector of length NVAR containing the initial guess of the minimum. (Input)
Default: XGUESS = 0.0.

ACC — The nonnegative tolerance on the first order conditions at the calculated solution. (Input)
Default: ACC = 1.e-4 for single precision and 1.d-8 for double precision.

MAXFCN — On input, maximum number of function evaluations allowed. (Input/ Output)
On output, actual number of function evaluations needed.
Default: MAXFCN = 400.

OBJ — Value of the objective function. (Output)

NACT — Final number of active constraints. (Output)

IACT — Vector containing the indices of the final active constraints in the first NACT positions. (Output)
Its length must be at least NCON + 2 * NVAR.

ALAMDA — Vector of length `NVAR` containing the Lagrange multiplier estimates of the final active constraints in the first `NACT` positions. (Output)

FORTRAN 90 Interface

Generic: `CALL LCONF (FCN, NEQ, A, B, XLB, XUB, SOL [,…])`

Specific: The specific interface names are `S_LCONF` and `D_LCONF`.

FORTRAN 77 Interface

Single:
```
CALL LCONF (FCN, NVAR, NCON, NEQ, A, LDA, B, XLB, XUB,
XGUESS, ACC, MAXFCN, SOL, OBJ, NACT, IACT,
ALAMDA)
```

Double: The double precision name is `DLCONF`.

Example

The problem from Schittkowski (1987)

$$\min f(x) = -x_1 x_2 x_3$$

$$\text{subject to} \quad -x_1 - 2x_2 - 2x_3 \le 0$$

$$x_1 + 2x_2 + 2x_3 \le 72$$

$$0 \le x_1 \le 20$$

$$0 \le x_2 \le 11$$

$$0 \le x_3 \le 42$$

is solved with an initial guess $x_1 = 10$, $x_2 = 10$ and $x_3 = 10$.

```
      USE LCONF_INT
      USE UMACH_INT
!                                 Declaration of variables
      INTEGER    NCON, NEQ, NVAR
      PARAMETER  (NCON=2, NEQ=0, NVAR=3)
!
      INTEGER    MAXFCN, NOUT
      REAL       A(NCON,NVAR), ACC, B(NCON), OBJ, &
                SOL(NVAR), XGUESS(NVAR), XLB(NVAR), XUB(NVAR)
      EXTERNAL   FCN
!
!                                 Set values for the following problem.
!
!                                 Min  -X(1)*X(2)*X(3)
!
!                                 -X(1) - 2*X(2) - 2*X(3)  .LE.   0
!                                  X(1) + 2*X(2) + 2*X(3)  .LE.  72
!
```

```
!                                  0  .LE.  X(1)  .LE.  20
!                                  0  .LE.  X(2)  .LE.  11
!                                  0  .LE.  X(3)  .LE.  42
!
      DATA A/-1.0, 1.0, -2.0, 2.0, -2.0, 2.0/, B/0.0, 72.0/
      DATA XLB/3*0.0/, XUB/20.0, 11.0, 42.0/, XGUESS/3*10.0/
      DATA ACC/0.0/, MAXFCN/400/
!
      CALL UMACH (2, NOUT)
!
      CALL LCONF (FCN, NEQ, A, B, XLB, XUB, SOL, XGUESS=XGUESS,  &
                 MAXFCN=MAXFCN, ACC=ACC, OBJ=OBJ)
!
      WRITE (NOUT,99998) 'Solution:'
      WRITE (NOUT,99999) SOL
      WRITE (NOUT,99998) 'Function value at solution:'
      WRITE (NOUT,99999) OBJ
      WRITE (NOUT,99998) 'Number of function evaluations:', MAXFCN
      STOP
99998 FORMAT (//, ' ', A, I4)
99999 FORMAT (1X, 5F16.6)
      END
!
      SUBROUTINE FCN (N, X, F)
      INTEGER    N
      REAL       X(*), F
!
      F = -X(1)*X(2)*X(3)
      RETURN
      END
```

Output

```
Solution:
 20.000000       11.000000       15.000000

Function value at solution:
-3300.000000

Number of function evaluations:   5
```

Comments

1. Workspace may be explicitly provided, if desired, by use of `L2ONF/DL2ONF`. The reference is:

   ```
   CALL L2ONF (FCN, NVAR, NCON, NEQ, A, LDA, B, XLB, XUB, XGUESS,
   ACC, MAXFCN, SOL, OBJ, NACT, IACT, ALAMDA, IPRINT, INFO, WK)
   ```

   The additional arguments are as follows:

   IPRINT — Print option (see Comment 3). (Input)

   INFO — Informational flag (see Comment 3). (Output)

WK — Real work vector of length NVAR**2 + 11 * NVAR + NCON.

2. Informational errors

Type	Code	
4	4	The equality constraints are inconsistent.
4	5	The equality constraints and the bounds on the variables are found to be inconsistent.
4	6	No vector X satisfies all of the constraints. In particular, the current active constraints prevent any change in X that reduces the sum of constraint violations.
4	7	Maximum number of function evaluations exceeded.
4	9	The variables are determined by the equality constraints.

3. The following are descriptions of the arguments IPRINT and INFO:

IPRINT — This argument must be set by the user to specify the frequency of printing during the execution of the routine LCONF. There is no printed output if IPRINT = 0. Otherwise, after ensuring feasibility, information is given every IABS(IPRINT) iterations and whenever a parameter called TOL is reduced. The printing provides the values of X(.), F(.) and G(.) = GRAD(F) if IPRINT is positive. If IPRINT is negative, this information is augmented by the current values of IACT(K) K = 1, …, NACT, PAR(K) K = 1, …, NACT and RESKT(I) I = 1, …, N. The reason for returning to the calling program is also displayed when IPRINT is nonzero.

INFO — On exit from L2ONF, INFO will have one of the following integer values to indicate the reason for leaving the routine:

INFO = 1 SOL is feasible, and the condition that depends on ACC is satisfied.

INFO = 2 SOL is feasible, and rounding errors are preventing further progress.

INFO = 3 SOL is feasible, but the objective function fails to decrease although a decrease is predicted by the current gradient vector.

INFO = 4 In this case, the calculation cannot begin because LDA is less than NCON or because the lower bound on a variable is greater than the upper bound.

INFO = 5 This value indicates that the equality constraints are inconsistent. These constraints include any components of X(.) that are frozen by setting XL(I) = XU(I).

INFO = 6 In this case there is an error return because the equality constraints and the bounds on the variables are found to be inconsistent.

INFO = 7 This value indicates that there is no vector of variables that satisfies all of the constraints. Specifically, when this return or an INFO = 6 return occurs, the current active constraints (whose indices are IACT(K), K = 1, …, NACT) prevent any change in `X(.)` that reduces the sum of constraint violations. Bounds are only included in this sum if `INFO` = 6.

`INFO` = 8 Maximum number of function evaluations exceeded.

`INFO` = 9 The variables are determined by the equality constraints.

Description

The routine `LCONF` is based on M.J.D. Powell's `TOLMIN`, which solves linearly constrained optimization problems, i.e., problems of the form

$$\min_{x \in \mathbf{R}^n} f(x)$$

$$\text{subject to} \quad A_1 x = b_1$$

$$A_2 x \le b_2$$

$$x_l \le x \le x_u$$

given the vectors b_1, b_2, x_l and x_u and the matrices A_1, and A_2.

The algorithm starts by checking the equality constraints for inconsistency and redundancy. If the equality constraints are consistent, the method will revise x^0, the initial guess provided by the user, to satisfy

$$A_1 x = b_1$$

Next, x^0 is adjusted to satisfy the simple bounds and inequality constraints. This is done by solving a sequence of quadratic programming subproblems to minimize the sum of the constraint or bound violations.

Now, for each iteration with a feasible x^k, let J_k be the set of indices of inequality constraints that have small residuals. Here, the simple bounds are treated as inequality constraints. Let I_k be the set of indices of active constraints. The following quadratic programming problem

$$\min f\left(x^k\right) + d^T \nabla f\left(x^k\right) + \frac{1}{2} d^T B^k d$$

$$\text{subject to} \quad a_j d = 0 \quad j \in I_k$$

$$a_j d \le 0 \quad j \in J_k$$

is solved to get (d^k, λ^k) where a_j is a row vector representing either a constraint in A_1 or A_2 or a bound constraint on x. In the latter case, the $a_j = e_i$ for the bound constraint $x_i \le (x_u)_i$ and $a_j = -e_i$ for the constraint $-x_i \le (-x_l)_i$. Here, e_i is a vector with a 1 as the i-th component, and zeroes elsewhere. λ^k are the Lagrange multipliers, and B^k is a positive definite approximation to the second derivative $\nabla^2 f(x^k)$.

After the search direction d^k is obtained, a line search is performed to locate a better point. The new point $x^{k+1} = x^k + \alpha^k d^k$ has to satisfy the conditions

$$f\left(x^k + \alpha^k d^k\right) \le f\left(x^k\right) + 0.1\alpha^k \left(d^k\right)^T \nabla f\left(x^k\right)$$

and

$$\left(d^k\right)^T \nabla f\left(x^k + \alpha^k d^k\right) \ge 0.7\left(d^k\right)^T \nabla f\left(x^k\right)$$

The main idea in forming the set J_k is that, if any of the inequality constraints restricts the step-length α^k, then its index is not in J_k. Therefore, small steps are likely to be avoided.

Finally, the second derivative approximation, B^k , is updated by the BFGS formula, if the condition

$$\left(d^k\right)^T \nabla f\left(x^k + \alpha^k d^k\right) - \nabla f\left(x^k\right) > 0$$

holds. Let $x^k \leftarrow x^{k+1}$, and start another iteration.

The iteration repeats until the stopping criterion

$$\left\| \nabla f\left(x^k\right) - A^k \lambda^k \right\|_2 \le \tau$$

is satisfied; here, τ is a user-supplied tolerance. For more details, see Powell (1988, 1989).

Since a finite-difference method is used to estimate the gradient for some single precision calculations, an inaccurate estimate of the gradient may cause the algorithm to terminate at a noncritical point. In such cases, high precision arithmetic is recommended. Also, whenever the exact gradient can be easily provided, routine `LCONG` (page 1316) should be used instead.

LCONG

Minimizes a general objective function subject to linear equality/inequality constraints.

Required Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (N, X, F)`, where

`N` – Value of `NVAR`. (Input)

`X` – Vector of length `N` at which point the function is evaluated. (Input)
`X` should not be changed by `FCN`.

F – The computed function value at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

GRAD — User-supplied SUBROUTINE to compute the gradient at the point X. The usage is CALL GRAD (N, X, G), where

N – Value of NVAR. (Input)

X – Vector of length N at which point the function is evaluated. (Input)
X should not be changed by GRAD.

G – Vector of length N containing the values of the gradient of the objective function evaluated at the point X. (Output)

GRAD must be declared EXTERNAL in the calling program.

NEQ — The number of linear equality constraints. (Input)

A — NCON by NVAR matrix. (Input)
The matrix contains the equality constraint gradients in the first NEQ rows, followed by the inequality constraint gradients.

B — Vector of length NCON containing right-hand sides of the linear constraints. (Input)
Specifically, the constraints on the variables X(I), I = 1, …, NVAR are A(K, 1) * X(1) + … + A(K, NVAR) * X(NVAR).EQ.B(K), K = 1, …, NEQ.A(K, 1) * X(1) + … + A(K, NVAR) * X(NVAR).LE.B(K), K = NEQ + 1, …, NCON. Note that the data that define the equality constraints come before the data of the inequalities.

XLB — Vector of length NVAR containing the lower bounds on the variables; choose a very large negative value if a component should be unbounded below or set XLB(I) = XUB(I) to freeze the I-th variable. (Input)
Specifically, these simple bounds are XLB(I).LE.X(I), I = 1, …, NVAR.

XUB — Vector of length NVAR containing the upper bounds on the variables; choose a very large positive value if a component should be unbounded above. (Input)
Specifically, these simple bounds are X(I).LE. XUB(I), I = 1, …, NVAR.

SOL — Vector of length NVAR containing solution. (Output)

Optional Arguments

NVAR — The number of variables. (Input)
Default: NVAR = size (A,2).

NCON — The number of linear constraints (excluding simple bounds). (Input)
Default: NCON = size (A,1).

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)
Default: LDA = size (A,1).

XGUESS — Vector of length `NVAR` containing the initial guess of the minimum. (Input)
Default: `XGUESS` = 0.0.

ACC — The nonnegative tolerance on the first order conditions at the calculated solution. (Input)
Default: `ACC` = 1.e-4 for single precision and 1.d-8 for double precision.

MAXFCN — On input, maximum number of function evaluations allowed.(Input/ Output)
On output, actual number of function evaluations needed.
Default: `MAXFCN` = 400.

OBJ — Value of the objective function. (Output)

NACT — Final number of active constraints. (Output)

IACT — Vector containing the indices of the final active constraints in the first `NACT` positions. (Output)
Its length must be at least `NCON` + 2 `*` `NVAR`.

ALAMDA — Vector of length `NVAR` containing the Lagrange multiplier estimates of the final active constraints in the first `NACT` positions. (Output)

FORTRAN 90 Interface

Generic: `CALL LCONG (FCN, GRAD, NEQ, A, B, XLB, XUB, SOL [,…])`

Specific: The specific interface names are `S_LCONG` and `D_LCONG`.

FORTRAN 77 Interface

Single:
```
CALL LCONG (FCN, GRAD, NVAR, NCON, NEQ, A, LDA, B, XLB,
XUB, XGUESS, ACC, MAXFCN, SOL, OBJ, NACT, IACT,
ALAMDA)
```

Double: The double precision name is `DLCONG`.

Example

The problem from Schittkowski (1987)

$$\min f(x) = -x_1 x_2 x_3$$

$$\text{subject to} \quad -x_1 - 2x_2 - 2x_3 \le 0$$

$$x_1 + 2x_2 + 2x_3 \le 72$$

$$0 \le x_1 \le 20$$

$$0 \le x_2 \le 11$$

$$0 \le x_3 \le 42$$

is solved with an initial guess $x_1 = 10$, $x_2 = 10$ and $x_3 = 10$.

```
      USE LCONG_INT
      USE UMACH_INT
!                                 Declaration of variables
      INTEGER    NCON, NEQ, NVAR
      PARAMETER  (NCON=2, NEQ=0, NVAR=3)
!
      INTEGER    MAXFCN, NOUT
      REAL       A(NCON,NVAR), ACC, B(NCON), OBJ, &
                 SOL(NVAR), XGUESS(NVAR), XLB(NVAR), XUB(NVAR)
      EXTERNAL   FCN, GRAD
!
!                                 Set values for the following problem.
!
!                                 Min  -X(1)*X(2)*X(3)
!
!                                 -X(1) - 2*X(2) - 2*X(3)  .LE.   0
!                                  X(1) + 2*X(2) + 2*X(3)  .LE.  72
!
!                                 0  .LE.  X(1)  .LE.  20
!                                 0  .LE.  X(2)  .LE.  11
!                                 0  .LE.  X(3)  .LE.  42
!
      DATA A/-1.0, 1.0, -2.0, 2.0, -2.0, 2.0/, B/0.0, 72.0/
      DATA XLB/3*0.0/, XUB/20.0, 11.0, 42.0/, XGUESS/3*10.0/
      DATA ACC/0.0/, MAXFCN/400/
!
      CALL UMACH (2, NOUT)
!
      CALL LCONG (FCN, GRAD, NEQ, A, B, XLB, XUB, SOL, XGUESS=XGUESS, &
                  ACC=ACC, MAXFCN=MAXFCN, OBJ=OBJ)
!
      WRITE (NOUT,99998) 'Solution:'
      WRITE (NOUT,99999) SOL
      WRITE (NOUT,99998) 'Function value at solution:'
      WRITE (NOUT,99999) OBJ
      WRITE (NOUT,99998) 'Number of function evaluations:', MAXFCN
      STOP
99998 FORMAT (//, ' ', A, I4)
99999 FORMAT (1X, 5F16.6)
      END
!
      SUBROUTINE FCN (N, X, F)
      INTEGER    N
      REAL       X(*), F
!
      F = -X(1)*X(2)*X(3)
      RETURN
      END
!
      SUBROUTINE GRAD (N, X, G)
      INTEGER    N
      REAL       X(*), G(*)
```

```
!
      G(1) = -X(2)*X(3)
      G(2) = -X(1)*X(3)
      G(3) = -X(1)*X(2)
      RETURN
      END
```

Output

```
Solution:
20.000000       11.000000       15.000000

Function value at solution:
-3300.000000

Number of function evaluations:   5
```

Comments

1. Workspace may be explicitly provided, if desired, by use of L2ONG/DL2ONG. The reference is:

   ```
   CALL L2ONG (FCN, GRAD, NVAR, NCON, NEQ, A, LDA, B, XLB, XUB,
   XGUESS, ACC, MAXFCN, SOL, OBJ, NACT, IACT, ALAMDA, IPRINT,
   INFO, WK)
   ```

   The additional arguments are as follows:

   IPRINT — Print option (see Comment 3). (Input)

   INFO — Informational flag (see Comment 3). (Output)

   WK — Real work vector of length NVAR**2 + 11 * NVAR + NCON.

2. Informational errors

Type	Code	
4	4	The equality constraints are inconsistent.
4	5	The equality constraints and the bounds on the variables are found to be inconsistent.
4	6	No vector X satisfies all of the constraints. In particular, the current active constraints prevent any change in X that reduces the sum of constraint violations.
4	7	Maximum number of function evaluations exceeded.
4	9	The variables are determined by the equality constraints.

3. The following are descriptions of the arguments IPRINT and INFO:

   IPRINT — This argument must be set by the user to specify the frequency of printing during the execution of the routine LCONG. There is no printed output if IPRINT = 0. Otherwise, after ensuring feasibility, information is given every IABS(IPRINT) iterations and whenever a parameter called TOL is reduced. The printing provides the values of X(.), F(.) and G(.) = GRAD(F) if IPRINT is positive. If `IPRINT` is negative, this information is augmented by the current values of `IACT(K)` `K` = 1, …, `NACT`, `PAR(K)` `K` = 1, …, `NACT` and `RESKT(I)` I = 1, …, `N`. The reason for returning to the calling program is also displayed when `IPRINT` is nonzero.

INFO —	On exit from `L2ONG`, `INFO` will have one of the following integer values to indicate the reason for leaving the routine:
`INFO` = 1	`SOL` is feasible and the condition that depends on `ACC` is satisfied.
`INFO` = 2	`SOL` is feasible and rounding errors are preventing further progress.
`INFO` = 3	`SOL` is feasible but the objective function fails to decrease although a decrease is predicted by the current gradient vector.
`INFO` = 4	In this case, the calculation cannot begin because `LDA` is less than `NCON` or because the lower bound on a variable is greater than the upper bound.
`INFO` = 5	This value indicates that the equality constraints are inconsistent. These constraints include any components of `X(.)` that are frozen by setting `XL(I)` = `XU(I)`.
`INFO` = 6	In this case, there is an error return because the equality constraints and the bounds on the variables are found to be inconsistent.
`INFO` = 7	This value indicates that there is no vector of variables that satisfies all of the constraints. Specifically, when this return or an `INFO` = 6 return occurs, the current active constraints (whose indices are `IACT(K)`, `K` = 1, …, `NACT`) prevent any change in `X(.)` that reduces the sum of constraint violations, where only bounds are included in this sum if `INFO` = 6.
`INFO` = 8	Maximum number of function evaluations exceeded.
`INFO` = 9	The variables are determined by the equality constraints.

Description

The routine `LCONG` is based on M.J.D. Powell's `TOLMIN`, which solves linearly constrained optimization problems, i.e., problems of the form

$$\min_{x \in \mathbf{R}^n} f(x)$$

$$\text{subject to} \quad A_1 x = b_1$$

$$A_2 x \le b_2$$

$$x_l \le x \le x_u$$

given the vectors b_1, b_2, x_l and x_u and the matrices A_1, and A_2.

The algorithm starts by checking the equality constraints for inconsistency and redundancy. If the equality constraints are consistent, the method will revise x^0, the initial guess provided by the user, to satisfy

$$A_1 x = b_1$$

Next, x^0 is adjusted to satisfy the simple bounds and inequality constraints. This is done by solving a sequence of quadratic programming subproblems to minimize the sum of the constraint or bound violations.

Now, for each iteration with a feasible x_k, let J_k be the set of indices of inequality constraints that have small residuals. Here, the simple bounds are treated as inequality constraints. Let I_k be the set of indices of active constraints. The following quadratic programming problem

$$\min f\left(x^k\right) + d^T \nabla f\left(x^k\right) + \frac{1}{2} d^T B^k d$$

$$\text{subject to} \quad a_j d = 0 \quad j \in I_k$$

$$a_j d \le 0 \quad j \in J_k$$

is solved to get (d^k, λ^k) where a_j is a row vector representing either a constraint in A_1 or A_2 or a bound constraint on x. In the latter case, the $a_j = e_i$ for the bound constraint $x_i \le (x_u)_i$ and $a_j = -e_i$ for the constraint $-x_i \le (-x_l)_i$. Here, e_i is a vector with a 1 as the i-th component, and zeroes elsewhere. λ^k are the Lagrange multipliers, and B^k is a positive definite approximation to the second derivative $\nabla^2 f(x^k)$.

After the search direction d^k is obtained, a line search is performed to locate a better point. The new point $x^{k+1} = x^k + \alpha^k d^k$ has to satisfy the conditions

$$f\left(x^k + \alpha^k d^k\right) \le f\left(x^k\right) + 0.1\alpha^k \left(d^k\right)^T \nabla f\left(x^k\right)$$

and

$$\left(d^k\right)^T \nabla f\left(x^k + \alpha^k d^k\right) \ge 0.7\left(d^k\right)^T \nabla f\left(x^k\right)$$

The main idea in forming the set J_k is that, if any of the inequality constraints restricts the step-length α^k, then its index is not in J_k. Therefore, small steps are likely to be avoided.

Finally, the second derivative approximation, B^k, is updated by the BFGS formula, if the condition

$$\left(d^k\right)^T \nabla f\left(x^k + \alpha^k d^k\right) - \nabla f\left(x^k\right) > 0$$

holds. Let $x^k \leftarrow x^{k+1}$, and start another iteration.

The iteration repeats until the stopping criterion

$$\left\|\nabla f\left(x^k\right) - A^k \lambda^k\right\|_2 \le \tau$$

is satisfied; here, τ is a user-supplied tolerance. For more details, see Powell (1988, 1989).

NNLPF

Solves a general nonlinear programming problem using a sequential equality constrained quadratic programming method.

Required Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the objective function and constraints at a given point. The internal usage is `CALL FCN (X, IACT, RESULT, IERR)`, where

X – The point at which the objective function or constraint is evaluated. (Input)

IACT – Integer indicating whether evaluation of the objective function is requested or evaluation of a constraint is requested. If `IACT` is zero, then an objective function evaluation is requested. If `IACT` is nonzero then the value if `IACT` indicates the index of the constraint to evaluate. (Input)

RESULT – If `IACT` is zero, then `RESULT` is the computed function value at the point `X`. If `IACT` is nonzero, then `RESULT` is the computed constraint value at the point `X`. (Output)

IERR – Logical variable. On input `IERR` is set to `.FALSE.` If an error or other undesirable condition occurs during evaluation, then `IERR` should be set to `.TRUE.` Setting `IERR` to `.TRUE.` will result in the step size being reduced and the step being tried again. (If `IERR` is set to `.TRUE.` for `XGUESS`, then an error is issued.)

The routine `FCN` must be use-associated in a user module that uses `NNLPF_INT`, or else declared `EXTERNAL` in the calling program. If `FCN` is a separately compiled routine, not in a module, then it must be declared `EXTERNAL.`

M — Total number of constraints. (Input)

ME — Number of equality constraints. (Input)

IBTYPE — Scalar indicating the types of bounds on variables. (Input)

IBTYPE	**Action**
0	User will supply all the bounds.

1	All variables are nonnegative.
2	All variables are nonpositive.
3	User supplies only the bounds on 1st variable; all other variables will have the same bounds.

XLB — Vector of length `N` containing the lower bounds on variables. (Input, if `IBTYPE` = 0; output, if `IBTYPE` = 1 or 2; input/output, if `IBTYPE` = 3)
If there is no lower bound for a variable, then the corresponding `XLB` value should be set to −`Huge`(`X`(`1`)).

XUB — Vector of length `N` containing the upper bounds on variables. (Input, if `IBTYPE` = 0; output, if `IBTYPE` = 1 or 2; input/output, if `IBTYPE` = 3).
If there is no upper bound for a variable, then the corresponding `XUB` value should be set to `Huge`(`X`(`1`)).

X — Vector of length `N` containing the computed solution. (Output)

Optional Arguments

N — Number of variables. (Input)
Default: `N` = `size`(`X`).

XGUESS — Vector of length `N` containing an initial guess of the solution. (Input)
Default: `XGUESS` = `X`, (with the smallest value of $\|X\|_2$) that satisfies the bounds.

XSCALE — Vector of length `N` setting the internal scaling of the variables. The initial value given and the objective function and gradient evaluations however are always in the original unscaled variables. The first internal variable is obtained by dividing values `X(I)` by `XSCALE(I)`. (Input)
In the absence of other information, set all entries to 1.0.
Default: `XSCALE(:)` = 1.0.

IPRINT — Parameter indicating the desired output level. (Input)

`IPRINT`	**Action**
0	No output printed.
1	One line of intermediate results is printed in each iteration.
2	Lines of intermediate results summarizing the most important data for each step are printed.

3	Lines of detailed intermediate results showing all primal and dual variables, the relevant values from the working set, progress in the backtracking and etc are printed
4	Lines of detailed intermediate results showing all primal and dual variables, the relevant values from the working set, progress in the backtracking, the gradients in the working set, the quasi-Newton updated and etc are printed.

Default: `IPRINT` = 0.

MAXITN — Maximum number of iterations allowed. (Input)
Default: `MAXITN` = 200.

EPSDIF — Relative precision in gradients. (Input)
Default: `EPSDIF` = `epsilon`(`x`(1))

TAU0 — A universal bound describing how much the unscaled penalty-term may deviate from zero. (Input)
`NNLPF` assumes that within the region described by

$$\sum_{i=1}^{M_e} \left|g_i\left(x\right)\right| - \sum_{i=M_e+1}^{M} \min\left(0, g_i\left(x\right)\right) \le \texttt{TAU0}$$

all functions may be evaluated safely. The initial guess, however, may violate these requirements. In that case an initial feasibility improvement phase is run by `NNLPF` until such a point is found. A small `TAU0` diminishes the efficiency of `NNLPF`, because the iterates then will follow the boundary of the feasible set closely. Conversely, a large `TAU0` may degrade the reliability of the code.
Default `TAU0` = 1.E0

DEL0 — In the initial phase of minimization a constraint is considered binding if

$$\frac{g_i\left(x\right)}{\max\left(1, \left\|\nabla g_i\left(x\right)\right\|\right)} \le \texttt{DEL0} \qquad i = M_e + 1, \ldots, M$$

Good values are between .01 and 1.0. If `DEL0` is chosen too small then identification of the correct set of binding constraints may be delayed. Contrary, if `DEL0` is too large, then the method will often escape to the full regularized SQP method, using individual slack variables for any active constraint, which is quite costly. For well-scaled problems `DEL0=1.0` is reasonable. (Input)
Default: `DEL0` = .5*`TAU0`

EPSFCN – Relative precision of the function evaluation routine. (Input)
Default: `EPSFCN` = `epsilon`(`x`(1))

IDTYPE – Type of numerical differentiation to be used. (Input)
Default: `IDTYPE` = 1

`IDTYPE`	**Action**
1	Use a forward difference quotient with discretization stepsize 0.1 (`EPSFCN`$^{1/2}$) componentwise relative.
2	Use the symmetric difference quotient with discretization stepsize 0.1 (`EPSFCN`$^{1/3}$) componentwise relative
3	Use the sixth order approximation computing a Richardson extrapolation of three symmetric difference quotient values. This uses a discretization stepsize 0.01(`EPSFCN`$^{1/7}$)

TAUBND – Amount by which bounds may be violated during numerical differentiation. Bounds are violated by `TAUBND` (at most) only if a variable is on a bound and finite differences are taken for gradient evaluations. (Input)
Default: `TAUBND` = 1.E0

SMALLW — Scalar containing the error allowed in the multipliers. For example, a negative multiplier of an inequality constraint is accepted (as zero) if its absolute value is less than `SMALLW`. (Input)
Default: `SMALLW` = exp(2*log(epsilon(x(1)/3)))

DELMIN — Scalar which defines allowable constraint violations of the final accepted result. Constraints are satisfied if $|g_i(x)| \le$ `DELMIN` , and $g_j(x) \ge$ (-`DELMIN`) respectively. (Input)
Default: `DELMIN =` min(DEL0/10, max(EPSDIF, min(DEL0/10, max(1.E-6*DEL0, SMALLW))

SCFMAX — Scalar containing the bound for the internal automatic scaling of the objective function. (Intput)
Default: `SCFMAX` = 1.0E4

FVALUE — Scalar containing the value of the objective function at the computed solution. (Output)

FORTRAN 90 Interface

Generic: `CALL NNLPF (FCN, M, ME, IBTYPE, XLB, XUB, X [,…])`

Specific: The specific interface names are `S_NNLPF` and `D_NNLPF` .

Example

The problem

$$\min F(x) = (x_1 - 2)^2 + (x_2 - 1)^2$$
$$\text{subject to} \quad g_1(x) = x_1 - 2x_2 + 1 = 0$$
$$g_2(x) = -x_1^2 / 4 - x_2^2 + 1 \geq 0$$

is solved.

```
      USE NNLPF_INT
      USE WRRRN_INT
      INTEGER    IBTYPE, M, ME
      PARAMETER  (IBTYPE=0, M=2, ME=1)
!
      REAL(KIND(1E0)) FVALUE, X(2), XGUESS(2), XLB(2), XUB(2)
      EXTERNAL FCN, GRAD
!
      XLB = -HUGE(X(1))
      XUB = HUGE(X(1))
!
      CALL NNLPF (FCN, M, ME, IBTYPE, XLB, XUB, X)
!
      CALL WRRRN ('The solution is', X)
      END

      SUBROUTINE FCN (X, IACT, RESULT, IERR)
      INTEGER    IACT
      REAL(KIND(1E0)) X(*), RESULT
      LOGICAL IERR
!
      SELECT CASE (IACT)
      CASE(0)
         RESULT = (X(1)-2.0E0)**2 + (X(2)-1.0E0)**2
      CASE(1)
         RESULT = X(1) - 2.0E0*X(2) + 1.0E0
      CASE(2)
         RESULT = -(X(1)**2)/4.0E0 - X(2)**2 + 1.0E0
      END SELECT
      RETURN
      END
```

Output

```
The solution is
 1   0.8229
 2   0.9114
```

Comments

1. Informational errors

Type	Code	
4	1	Constraint evaluation returns an error with current point.
4	2	Objective evaluation returns an error with current point.
4	3	Working set is singular in dual extended QP.
4	4	QP problem is seemingly infeasible.
4	5	A stationary point located.

4	6	A stationary point located or termination criteria too strong.
4	7	Maximum number of iterations exceeded.
4	8	Stationary point not feasible.
4	9	Very slow primal progress.
4	10	The problem is singular.
4	11	Matrix of gradients of binding constraints is singular or very ill-conditioned.
4	12	Small changes in the penalty function.

Description

The routine `NNLPF` provides an interface to a licensed version of subroutine `DONLP2`, a FORTRAN code developed by Peter Spellucci (1998). It uses a sequential equality constrained quadratic programming method with an active set technique, and an alternative usage of a fully regularized mixed constrained subproblem in case of nonregular constraints (i.e. linear dependent gradients in the "working sets"). It uses a slightly modified version of the Pantoja-Mayne update for the Hessian of the Lagrangian, variable dual scaling and an improved Armjijo-type stepsize algorithm. Bounds on the variables are treated in a gradient-projection like fashion. Details may be found in the following two papers:

P. Spellucci: *An SQP method for general nonlinear programs using only equality constrained subproblems*. Math. Prog. 82, (1998), 413-448.

P. Spellucci: *A new technique for inconsistent problems in the SQP method.* Math. Meth. of Oper. Res. 47, (1998), 355-500. (published by Physica Verlag, Heidelberg, Germany).

The problem is stated as follows:

$$\min_{x \in \mathbf{R}^n} f(x)$$

$$\text{subject to} \quad \begin{array}{ll} g_j(x) = 0, \text{ for} & j = 1, \ldots, m_e \\ g_j(x) \ge 0, \text{ for} & j = m_e + 1, \ldots, m \\ x_l \le x \le x_u & \end{array}$$

Although default values are provided for optional input arguments, it may be necessary to adjust these values for some problems. Through the use of optional arguments, NNLPF allows for several parameters of the algorithm to be adjusted to account for specific characteristics of problems. The DONLP2 Users Guide provides detailed descriptions of these parameters as well as strategies for maximizing the perfomance of the algorithm. The DONLP2 Users Guide is available in the "*help*" subdirectory of the main IMSL product installation directory. In addition, the following are a number of guidelines to consider when using NNLPF.

- A good initial starting point is very problem specific and should be provided by the calling program whenever possible. See optional argument `XGUESS`.
- Gradient approximation methods can have an effect on the success of NNLPF. Selecting a higher order appoximation method may be necessary for some problems. See optional argument `IDTYPE`.

- If a two sided constraint $l_i \le g_i(x) \le u_i$ is transformed into two constraints $g_{2i}(x) \ge 0$ and $g_{2i+1}(x) \ge 0$, then choose $\text{DEL0} < \frac{1}{2}(u_i - l_i) / max\{1, \|\nabla g_i(x)\|\}$, or at least try to provide an estimate for that value. This will increase the efficiency of the algorithm. See optional argument `DEL0`.
- The parameter `IERR` provided in the interface to the user supplied function `FCN` can be very useful in cases when evaluation is requested at a point that is not possible or reasonable. For example, if evaluation at the requested point would result in a floating point exception, then setting `IERR` to `.TRUE.` and returning without performing the evaluation will avoid the exception. NNLPF will then reduce the stepsize and try the step again. Note, if `IERR` is set to `.TRUE.` for the initial guess, then an error is issued.

NNLPG

Solves a general nonlinear programming problem using a sequential equality constrained quadratic programming method with user supplied gradients.

Required Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the objective function and constraints at a given point. The internal usage is `CALL FCN (X, IACT, RESULT, IERR)`, where

X – The point at which the objective function or constraint is evaluated. (Input)

IACT – Integer indicating whether evaluation of the objective function is requested or evaluation of a constraint is requested. If `IACT` is zero, then an objective function evaluation is requested. If `IACT` is nonzero then the value if `IACT` indicates the index of the constraint to evaluate. (Input)

RESULT – If `IACT` is zero, then `RESULT` is the computed objective function value at the point `X`. If `IACT` is nonzero, then `RESULT` is the computed constraint value at the point `X`. (Output)

IERR – Logical variable. On input `IERR` is set to `.FALSE.` If an error or other undesirable condition occurs during evaluation, then `IERR` should be set to `.TRUE.` Setting `IERR` to `.TRUE.` will result in the step size being reduced and the step being tried again. (If `IERR` is set to `.TRUE.` for `XGUESS`, then an error is issued.)

The routine `FCN` must be use-associated in a user module that uses `NNLPG_INT`, or else declared `EXTERNAL` in the calling program. If `FCN` is a separately compiled routine, not in a module, then it must be declared `EXTERNAL`.

GRAD — User-supplied `SUBROUTINE` to evaluate the gradients at a given point. The usage is `CALL GRAD (X, IACT, RESULT)`, where

X – The point at which the gradient of the objective function or gradient of a constraint is evaluated. (Input)

IACT – Integer indicating whether evaluation of the function gradient is requested or evaluation of a constraint gradient is requested. If `IACT` is zero, then an objective function gradient evaluation is requested. If `IACT` is nonzero then the value if `IACT` indicates the index of the constraint gradient to evaluate. (Input)***RESULT*** – If `IACT` is zero, then `RESULT` is the computed gradient of the objective function at the point `X`. If `IACT` is nonzero, then `RESULT` is the computed gradient of the requested constraint value at the point `X`. (Output)

The routine `GRAD` must be use-associated in a user module that uses `NNLPG_INT`, or else declared `EXTERNAL` in the calling program. If `GRAD` is a separately compiled routine, not in a module, then is must be declared `EXTERNAL`

M — Total number of constraints. (Input)

ME — Number of equality constraints. (Input)

IBTYPE — Scalar indicating the types of bounds on variables. (Input)

IBTYPE	**Action**
0	User will supply all the bounds.
1	All variables are nonnegative.
2	All variables are nonpositive.
3	User supplies only the bounds on 1st variable, all other variables will have the same bounds.

XLB — Vector of length `N` containing the lower bounds on the variables. (Input, if `IBTYPE` = 0; output, if `IBTYPE` = 1 or 2; input/output, if `IBTYPE` = 3) If there is no lower bound on a variable, then the corresponding `XLB` value should be set to `−huge(x(1))`.

XUB — Vector of length `N` containing the upper bounds on the variables. (Input, if `IBTYPE` = 0; output, if `IBTYPE` = 1 or 2; input/output, if `IBTYPE` = 3) If there is no upper bound on a variable, then the corresponding `XUB` value should be set to `huge(x(1))`.

X — Vector of length `N` containing the computed solution. (Output)

Optional Arguments

N — Number of variables. (Input)
Default: `N` = `size(X)`.

IPRINT — Parameter indicating the desired output level. (Input)

IPRINT	**Action**
0	No output printed.
1	One line of intermediate results is printed in each iteration.
2	Lines of intermediate results summarizing the most important data for each step are printed.
3	Lines of detailed intermediate results showing all primal and dual variables, the relevant values from the working set, progress in the backtracking and etc are printed
4	Lines of detailed intermediate results showing all primal and dual variables, the relevant values from the working set, progress in the backtracking, the gradients in the working set, the quasi-Newton updated and etc are printed.

Default: `IPRINT` = 0.

MAXITN — Maximum number of iterations allowed. (Input)
Default: `MAXITN` = 200.

XGUESS — Vector of length `N` containing an initial guess of the solution. (Input)
Default: `XGUESS` = `X`, (with the smallest value of $\|X\|_2$) that satisfies the bounds.

TAU0 — A universal bound describing how much the unscaled penalty-term may deviate from zero. (Input)
`NNLPG` assumes that within the region described by

$$\sum_{i=1}^{M_e} |g_i(x)| - \sum_{i=M_e+1}^{M} \min(0, g_i(x)) \le \texttt{TAU0}$$

all functions may be evaluated safely. The initial guess however, may violate these requirements. In that case an initial feasibility improvement phase is run by `NNLPG` until such a point is found. A small `TAU`0 diminishes the efficiency of `NNLPG`, because the iterates then will follow the boundary of the feasible set closely. Conversely, a large `TAU0` may degrade the reliability of the code.
Default: `TAU0` = 1.E0

DEL0 — In the initial phase of minimization a constraint is considered binding if

$$\frac{g_i(x)}{\max(1, \|\nabla g_i(x)\|)} \le \texttt{DEL0} \qquad i = M_e + 1, \ldots, M$$

Good values are between .01 and 1.0. If DEL0 is chosen too small then identification of the correct set of binding constraints may be delayed. Contrary, if DEL0 is too large, then the method will often escape to the full regularized SQP method, using individual slack variables for any active constraint, which is quite costly. For well-scaled problems DEL0=1.0 is reasonable. (Input)
Default: DEL0 = .5*TAU0

SMALLW — Scalar containing the error allowed in the multipliers. For example, a negative multiplier of an inequality constraint is accepted (as zero) if its absolute value is less than SMALLW. (Input)
Default: SMALLW = exp(2*log(epsilon(x(1)/3)))

DELMIN — Scalar which defines allowable constraint violations of the final accepted result. Constraints are satisfied if $|g_i(x)| \le$ DELMIN , and $g_j(x) \ge$ (-DELMIN) respectively. (Input)
Default: DELMIN = min(DEL0/10, max(EPSDIF, min(DEL0/10, max(1.E-6*DEL0, SMALLW))

SCFMAX — Scalar containing the bound for the internal automatic scaling of the objective function. (Intput)
Default: SCFMAX = 1.0E4

FVALUE — Scalar containing the value of the objective function at the computed solution. (Output)

FORTRAN 90 Interface

Generic: CALL NNLPG (FCN, GRAD, M, ME, IBTYPE, XLB, XUB, X [,…])

Specific: The specific interface names are S_NNLPG and D_NNLPG.

Example 1

The problem

$$\begin{aligned} \min F(x) &= (x_1 - 2)^2 + (x_2 - 1)^2 \\ \text{subject to} \quad g_1(x) &= x_1 - 2x_2 + 1 = 0 \\ g_2(x) &= -x_1^2/4 - x_2^2 + 1 \ge 0 \end{aligned}$$

is solved.

```
      USE NNLPG_INT
      USE WRRRN_INT
      INTEGER    IBTYPE, M, ME
      PARAMETER  (IBTYPE=0, M=2, ME=1)
!
      REAL(KIND(1E0)) FVALUE, X(2), XGUESS(2), XLB(2), XUB(2)
      EXTERNAL FCN, GRAD
!
```

```
      XLB = -HUGE(X(1))
      XUB = HUGE(X(1))
!
      CALL NNLPG (FCN, GRAD, M, ME, IBTYPE, XLB, XUB, X)
!
      CALL WRRRN ('The solution is', X)
      END

      SUBROUTINE FCN (X, IACT, RESULT, IERR)
      INTEGER    IACT
      REAL(KIND(1E0)) X(*), RESULT
      LOGICAL IERR
!
      SELECT CASE (IACT)
      CASE(0)
         RESULT = (X(1)-2.0E0)**2 + (X(2)-1.0E0)**2
      CASE(1)
         RESULT = X(1) - 2.0E0*X(2) + 1.0E0
      CASE(2)
         RESULT = -(X(1)**2)/4.0E0 - X(2)**2 + 1.0E0
      END SELECT
      RETURN
      END

      SUBROUTINE GRAD (X, IACT, RESULT)
      INTEGER    IACT
      REAL(KIND(1E0)) X(*),RESULT(*)
!
      SELECT CASE (IACT)
      CASE(0)
         RESULT (1) = 2.0E0*(X(1)-2.0E0)
         RESULT (2) = 2.0E0*(X(2)-1.0E0)
      CASE(1)
         RESULT (1) = 1.0E0
         RESULT (2) = -2.0E0
      CASE(2)
         RESULT (1) = -0.5E0*X(1)
         RESULT (2) = -2.0E0*X(2)
      END SELECT
      RETURN
      END
```

Output

```
The solution is
1   0.8229
2   0.9114
```

Comments

1. Informational errors

Type	Code	
4	1	Constraint evaluation returns an error with current point.
4	2	Objective evaluation returns an error with current point.

4	3	Working set is singular in dual extended QP.
4	4	QP problem is seemingly infeasible.
4	5	A stationary point located.
4	6	A stationary point located or termination criteria too strong.
4	7	Maximum number of iterations exceeded.
4	8	Stationary point not feasible.
4	9	Very slow primal progress.
4	10	The problem is singular.
4	11	Matrix of gradients of binding constraints is singular or very ill-conditioned.
4	12	Small changes in the penalty function.

.

Description

The routine `NNLPG` provides an interface to a licensed version of subroutine `DONLP2`, a FORTRAN code developed by Peter Spellucci (1998). It uses a sequential equality constrained quadratic programming method with an active set technique, and an alternative usage of a fully regularized mixed constrained subproblem in case of nonregular constraints (i.e. linear dependent gradients in the “working sets”). It uses a slightly modified version of the Pantoja-Mayne update for the Hessian of the Lagrangian, variable dual scaling and an improved Armjijo-type stepsize algorithm. Bounds on the variables are treated in a gradient-projection like fashion. Details may be found in the following two papers:

P. Spellucci: *An SQP method for general nonlinear programs using only equality constrained subproblems*. Math. Prog. 82, (1998), 413-448.

P. Spellucci: *A new technique for inconsistent problems in the SQP method.* Math. Meth. of Oper. Res. 47, (1998), 355-500. (published by Physica Verlag, Heidelberg, Germany).

The problem is stated as follows:

$$\min_{x \in \mathbf{R}^n} f(x)$$

$$\begin{array}{lll} \text{subject to} & g_j(x) = 0, \text{ for} & j = 1, \ldots, m_e \\ & g_j(x) \ge 0, \text{ for} & j = m_e + 1, \ldots, m \\ & x_l \le x \le x_u & \end{array}$$

Although default values are provided for optional input arguments, it may be necessary to adjust these values for some problems. Through the use of optional arguments, NNLPG allows for several parameters of the algorithm to be adjusted to account for specific characteristics of problems. The DONLP2 Users Guide provides detailed descriptions of these parameters as well as strategies for maximizing the perfomance of the algorithm. The DONLP2 Users Guide is available in the “*help*” subdirectory of the main IMSL product installation directory. In addition, the following are a number of guidelines to consider when using NNLPG.

- A good initial starting point is very problem specific and should be provided by the calling program whenever possible. See optional argument `XGUESS`.

- If a two sided constraint $l_i \le g_i(x) \le u_i$ is transformed into two constraints $g_{2i}(x) \ge 0$ and $g_{2i+1}(x) \ge 0$, then choose $\mathrm{DEL0} < \frac{1}{2}(u_i - l_i) / max\{1, \|\nabla g_i(x)\|\}$, or at least try to provide an estimate for that value. This will increase the efficiency of the algorithm. See optional argument `DEL0`.
- The parameter `IERR` provided in the interface to the user supplied function `FCN` can be very useful in cases when evaluation is requested at a point that is not possible or reasonable. For example, if evaluation at the requested point would result in a floating point exception, then setting `IERR` to `.TRUE.` and returning without performing the evaluation will avoid the exception. NNLPG will then reduce the stepsize and try the step again. Note, if `IERR` is set to `.TRUE.` for the initial guess, then an error is issued.

Example 2

The same problem from Example 1 is solved, but here we use central differences to compute the gradient of the first constraint. This example demonstrates how `NNLPG` can be used in cases when analytic gradients are known for only a portion of the constraints and/or objective function. The subroutine `CDGRD` is used to compute an approximation to the gradient of the first constraint.

```
      USE NNLPG_INT
      USE CDGRD_INT
      USE WRRRN_INT
      INTEGER    IBTYPE, M, ME
      PARAMETER  (IBTYPE=0, M=2, ME=1)
!
      REAL(KIND(1E0)) FVALUE, X(2), XGUESS(2), XLB(2), XUB(2)
      EXTERNAL FCN, GRAD
!
      XLB = -HUGE(X(1))
      XUB = HUGE(X(1))
!
      CALL NNLPG (FCN, GRAD, M, ME, IBTYPE, XLB, XUB, X)
!
      CALL WRRRN ('The solution is', X)
      END

      SUBROUTINE FCN (X, IACT, RESULT, IERR)
      INTEGER    IACT
      REAL(KIND(1E0)) X(2), RESULT
      LOGICAL IERR
      EXTERNAL CONSTR1
!
      SELECT CASE (IACT)
      CASE(0)
         RESULT = (X(1)-2.0E0)**2 + (X(2)-1.0E0)**2
      CASE(1)
         CALL CONSTR1(2, X, RESULT)
      CASE(2)
         RESULT = -(X(1)**2)/4.0E0 - X(2)**2 + 1.0E0
      END SELECT
      RETURN
      END

      SUBROUTINE GRAD (X, IACT, RESULT)
      USE CDGRD_INT
      INTEGER    IACT
      REAL(KIND(1E0)) X(2),RESULT(2)
      EXTERNAL CONSTR1
!
      SELECT CASE (IACT)
      CASE(0)
         RESULT (1) = 2.0E0*(X(1)-2.0E0)
         RESULT (2) = 2.0E0*(X(2)-1.0E0)
      CASE(1)
         CALL CDGRD(CONSTR1, X, RESULT)
      CASE(2)
         RESULT (1) = -0.5E0*X(1)
         RESULT (2) = -2.0E0*X(2)
      END SELECT
      RETURN
      END

      SUBROUTINE CONSTR1 (N, X, RESULT)
      INTEGER N
      REAL(KIND(1E0)) X(*), RESULT
      RESULT = X(1) - 2.0E0*X(2) + 1.0E0
      RETURN
      END
```

Output

```
The solution is
1   0.8229
2   0.9114
```

CDGRD

Approximates the gradient using central differences.

Required Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (N, X, F)`, where

`N` – Length of `X`.   (Input)

`X` – The point at which the function is evaluated.   (Input)
`X` should not be changed by `FCN`.

`F` – The computed function value at the point `X`.   (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

XC — Vector of length `N` containing the point at which the gradient is to be estimated. (Input)

GC — Vector of length `N` containing the estimated gradient at `XC`. (Output)

Optional Arguments

N — Dimension of the problem. (Input)
Default: `N` = size (`XC`,1).

XSCALE — Vector of length `N` containing the diagonal scaling matrix for the variables. (Input)
In the absence of other information, set all entries to 1.0.
Default: `XSCALE` = 1.0.

EPSFCN — Estimate for the relative noise in the function. (Input)
`EPSFCN` must be less than or equal to 0.1. In the absence of other information, set `EPSFCN` to 0.0.
Default: `EPSFCN` = 0.0.

FORTRAN 90 Interface

Generic: `CALL CDGRD (FCN, XC, GC [,…])`

Specific: The specific interface names are `S_CDGRD` and `D_CDGRD`.

FORTRAN 77 Interface

Single: `CALL CDGRD (FCN, N, XC, XSCALE, EPSFCN, GC)`

Double: The double precision name is `DCDGRD`.

Example

In this example, the gradient of $f(x) = x_1 - x_1 x_2 - 2$ is estimated by the finite-difference method at the point (1.0, 1.0).

```
      USE CDGRD_INT
      USE UMACH_INT
      INTEGER    I, N, NOUT
      PARAMETER  (N=2)
      REAL       EPSFCN, GC(N), XC(N)
      EXTERNAL   FCN
!                                  Initialization.
      DATA XC/2*1.0E0/
!                                  Set function noise.
      EPSFCN = 0.01
!
      CALL CDGRD (FCN, XC, GC, EPSFCN=EPSFCN)
```

```
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) (GC(I),I=1,N)
99999 FORMAT ('  The gradient is', 2F8.2, /)
!
      END
!
      SUBROUTINE FCN (N, X, F)
      INTEGER    N
      REAL       X(N), F
!
      F = X(1) - X(1)*X(2) - 2.0E0
!
      RETURN
      END
```

Output

```
The gradient is    0.00   -1.00
```

Comments

This is Description A5.6.4, Dennis and Schnabel, 1983, page 323.

Description

The routine `CDGRD` uses the following finite-difference formula to estimate the gradient of a function of *n* variables at *x*:

$$\frac{f\left(x+h_i e_i\right)-f\left(x-h_i e_i\right)}{2h_i} \quad \text{for } i = 1, \ldots, n$$

where $h_i = \varepsilon^{1/2} \max\{|x_i|, 1/s_i\} \operatorname{sign}(x_i)$, ε is the machine epsilon, s_i is the scaling factor of the *i*-th variable, and e_i is the *i*-th unit vector. For more details, see Dennis and Schnabel (1983).

Since the finite-difference method has truncation error, cancellation error, and rounding error, users should be aware of possible poor performance. When possible, high precision arithmetic is recommended.

FDGRD

Approximates the gradient using forward differences.

Required Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (N, X, F)`, where

N – Length of X.   (Input)

X – The point at which the function is evaluated.   (Input)
X should not be changed by FCN.

F – The computed function value at the point X. (Output)

FCN must be declared EXTERNAL in the calling program.

XC — Vector of length N containing the point at which the gradient is to be estimated. (Input)

FC — Scalar containing the value of the function at XC. (Input)

GC — Vector of length N containing the estimated gradient at XC. (Output)

Optional Arguments

N — Dimension of the problem. (Input)
Default: N = size (XC,1).

XSCALE — Vector of length N containing the diagonal scaling matrix for the variables. (Input)
In the absence of other information, set all entries to 1.0.
Default: XSCALE = 1.0.

EPSFCN — Estimate of the relative noise in the function. (Input)
EPSFCN must be less than or equal to 0.1. In the absence of other information, set EPSFCN to 0.0.
Default: EPSFCN = 0.0.

FORTRAN 90 Interface

Generic: CALL FDGRD (FCN, XC, FC, GC [,…])

Specific: The specific interface names are S_FDGRD and D_FDGRD.

FORTRAN 77 Interface

Single: CALL FDGRD (FCN, XC, FC, GC, N, XSCALE, EPSFCN)

Double: The double precision name is DFDGRD.

Example

In this example, the gradient of $f(x) = x_1 - x_1x_2 - 2$ is estimated by the finite-difference method at the point (1.0, 1.0).

```
USE FDGRD_INT
USE UMACH_INT
INTEGER    I, N, NOUT
PARAMETER  (N=2)
REAL       EPSFCN, FC, GC(N), XC(N)
```

```
      EXTERNAL   FCN
!                                    Initialization.
      DATA XC/2*1.0E0/
!                                    Set function noise.
      EPSFCN = 0.01
!                                    Get function value at current
!                                    point.
      CALL FCN (N, XC, FC)
!
      CALL FDGRD (FCN, XC, FC, GC, EPSFCN=EPSFCN)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) (GC(I),I=1,N)
99999 FORMAT ('  The gradient is', 2F8.2, /)
!
      END
!
      SUBROUTINE FCN (N, X, F)
      INTEGER    N
      REAL       X(N), F
!
      F = X(1) - X(1)*X(2) - 2.0E0
!
      RETURN
      END
```

Output

```
The gradient is    0.00   -1.00
```

Comments

This is Description A5.6.3, Dennis and Schnabel, 1983, page 322.

Description

The routine FDGRD uses the following finite-difference formula to estimate the gradient of a function of n variables at x:

$$\frac{f(x+h_ie_i)-f(x)}{h_i} \quad \text{for } i=1,\ldots,n$$

where $h_i = \varepsilon^{1/2} \max\{|x_i|, 1/s_i\} \operatorname{sign}(x_i)$, ε is the machine epsilon, e_i is the i-th unit vector, and s_i is the scaling factor of the i-th variable. For more details, see Dennis and Schnabel (1983).

Since the finite-difference method has truncation error, cancellation error, and rounding error, users should be aware of possible poor performance. When possible, high precision arithmetic is recommended. When accuracy of the gradient is important, IMSL routine CDGRD (page 1336) should be used.

FDHES

Approximates the Hessian using forward differences and function values.

Required Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (N, X, F)`, where

`N` – Length of `X`. (Input)

`X` – The point at which the function is evaluated. (Input)
`X` should not be changed by `FCN`.

`F` – The computed function value at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

XC — Vector of length `N` containing the point at which the Hessian is to be approximated. (Input)

FC — Function value at `XC`. (Input)

H — `N` by `N` matrix containing the finite difference approximation to the Hessian in the lower triangle. (Output)

Optional Arguments

N — Dimension of the problem. (Input)
Default: `N` = size (`XC`,1).

XSCALE — Vector of length `N` containing the diagonal scaling matrix for the variables. (Input)
In the absence of other information, set all entries to 1.0.
Default: `XSCALE` = 1.0.

EPSFCN — Estimate of the relative noise in the function. (Input)
`EPSFCN` must be less than or equal to 0.1. In the absence of other information, set `EPSFCN` to 0.0.
Default: `EPSFCN` = 0.0.

LDH — Row dimension of `H` exactly as specified in the dimension statement of the calling program. (Input)
Default: `LDH` = size (`H`,1).

FORTRAN 90 Interface

Generic: `CALL FDHES (FCN, XC, FC, H [,…])`

Specific: The specific interface names are `S_FDHES` and `D_FDHES`.

FORTRAN 77 Interface

Single: `CALL FDHES (FCN, N, XC, XSCALE, FC, EPSFCN, H, LDH)`

Double: The double precision name is `DFDHES`.

Example

The Hessian is estimated for the following function at (1, −1)

$$f(x) = x_1^2 - x_1x_2 - 2$$

```
      USE FDHES_INT
      USE UMACH_INT
!                                 Declaration of variables
      INTEGER    N, LDHES, NOUT
      PARAMETER  (N=2, LDHES=2)
      REAL       XC(N), FVALUE, HES(LDHES,N), EPSFCN
      EXTERNAL   FCN
!                                   Initialization
      DATA XC/1.0E0,-1.0E0/
!                                   Set function noise
      EPSFCN = 0.001
!                                   Evaluate the function at
!                                   current point
      CALL FCN (N, XC, FVALUE)
!                                 Get Hessian forward difference
!                                 approximation
      CALL FDHES (FCN, XC, FVALUE, HES, EPSFCN=EPSFCN)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) ((HES(I,J),J=1,I),I=1,N)
99999 FORMAT ('  The lower triangle of the Hessian is', /,&
               5X,F10.2,/,5X,2F10.2,/)
!
      END
!
      SUBROUTINE FCN (N, X, F)
!                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER N
      REAL    X(N), F
!
      F = X(1)*(X(1) - X(2)) - 2.0E0
!
      RETURN
      END
```

Output

```
 The lower triangle of the Hessian is
  2.00
 -1.00      0.00
```

Comments

1. Workspace may be explicitly provided, if desired, by use of `F2HES/DF2HES`. The reference is:

   ```
   CALL F2HES (FCN, N, XC, XSCALE, FC, EPSFCN, H, LDH, WK1, WK2)
   ```

   The additional arguments are as follows:

   WK1 — Real work vector of length `N`.

   WK2 — Real work vector of length `N`.

2. This is Description A5.6.2 from Dennis and Schnabel, 1983; page 321.

Description

The routine `FDHES` uses the following finite-difference formula to estimate the Hessian matrix of function *f* at *x*:

$$\frac{f\left(x+h_ie_i+h_je_j\right)-f\left(x+h_ie_i\right)-f\left(x+h_je_j\right)+f\left(x\right)}{h_ih_j}$$

where $h_i = \varepsilon^{1/3} \max\{|x_i|, 1/s_i\} \operatorname{sign}(x_i)$, $h_j = \varepsilon^{1/3} \max\{|x_j|, 1/s_i\} \operatorname{sign}(x_j)$, ε is the machine epsilon or user-supplied estimate of the relative noise, s_i and s_j are the scaling factors of the *i*-th and *j*-th variables, and e_i and e_j are the *i*-th and *j*-th unit vectors, respectively. For more details, see Dennis and Schnabel (1983).

Since the finite-difference method has truncation error, cancellation error, and rounding error, users should be aware of possible poor performance. When possible, high precision arithmetic is recommended.

GDHES

Approximates the Hessian using forward differences and a user-supplied gradient.

Required Arguments

GRAD — User-supplied `SUBROUTINE` to compute the gradient at the point `X`. The usage is `CALL GRAD (N, X, G)`, where

`N` – Length of `X` and `G`. (Input)

`X` – The point at which the gradient is evaluated. (Input)
`X` should not be changed by `GRAD`.

`G` – The gradient evaluated at the point `X`. (Output)

GRAD must be declared EXTERNAL in the calling program.

XC — Vector of length N containing the point at which the Hessian is to be estimated. (Input)

GC — Vector of length N containing the gradient of the function at XC. (Input)

H — N by N matrix containing the finite-difference approximation to the Hessian in the lower triangular part and diagonal. (Output)

Optional Arguments

N — Dimension of the problem. (Input)
Default: N = size (XC,1).

XSCALE — Vector of length N containing the diagonal scaling matrix for the variables. (Input)
In the absence of other information, set all entries to 1.0.
Default: XSCALE = 1.0.

EPSFCN — Estimate of the relative noise in the function. (Input)
EPSFCN must be less than or equal to 0.1. In the absence of other information, set EPSFCN to 0.0.
Default: EPSFCN = 0.0.

LDH — Leading dimension of H exactly as specified in the dimension statement of the calling program. (Input)
Default: LDH = size (H,1).

FORTRAN 90 Interface

Generic: `CALL GDHES (GRAD, XC, GC, H [,…])`

Specific: The specific interface names are S_GDHES and D_GDHES.

FORTRAN 77 Interface

Single: `CALL GDHES (GRAD, N, XC, XSCALE, GC, EPSFCN, H, LDH)`

Double: The double precision name is DGDHES.

Example

The Hessian is estimated by the finite-difference method at point (1.0, 1.0) from the following gradient functions:

$$g_1 = 2x_1x_2 - 2$$
$$g_2 = x_1x_1 + 1$$

```
      USE GDHES_INT
      USE UMACH_INT
!                                 Declaration of variables
      INTEGER    N, LDHES, NOUT
      PARAMETER  (N=2, LDHES=2)
      REAL       XC(N), GC(N), HES(LDHES,N)
      EXTERNAL   GRAD
!
      DATA XC/2*1.0E0/
!                                 Set function noise
!                                 Evaluate the gradient at the
!                                 current point
      CALL GRAD (N, XC, GC)
!                                 Get Hessian forward-difference
!                                 approximation
      CALL GDHES (GRAD, XC, GC, HES)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) ((HES(I,J),J=1,N),I=1,N)
99999 FORMAT ('  THE HESSIAN IS', /, 2(5X,2F10.2,/),/)
!
      END
!
      SUBROUTINE GRAD (N, X, G)
!                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER N
      REAL    X(N), G(N)
!
      G(1) = 2.0E0*X(1)*X(2) - 2.0E0
      G(2) = X(1)*X(1) + 1.0E0
!
      RETURN
      END
```

Output

```
 THE HESSIAN IS
 2.00      2.00
 2.00      0.00
```

Comments

1. Workspace may be explicitly provided, if desired, by use of `G2HES/DG2HES`. The reference is:

   `CALL G2HES (GRAD, N, XC, XSCALE, GC, EPSFCN, H, LDH, WK)`

   The additional argument is

   WK — Work vector of length `N`.

2. This is Description A5.6.1, Dennis and Schnabel, 1983; page 320.

Description

The routine `GDHES` uses the following finite-difference formula to estimate the Hessian matrix of function *F* at *x*:

$$\frac{g\left(x+h_j e_j\right)-g\left(x\right)}{h_j}$$

where $h_j = \varepsilon^{1/2} \max\{|x_j|, 1/s_j\} \operatorname{sign}(x_j)$, ε is the machine epsilon, s_j is the scaling factor of the *j*-th variable, *g* is the analytic gradient of *F* at *x*, and e_j is the *j*-th unit vector. For more details, see Dennis and Schnabel (1983).

Since the finite-difference method has truncation error, cancellation error, and rounding error, users should be aware of possible poor performance. When possible, high precision arithmetic is recommended.

FDJAC

Approximates the Jacobian of `M` functions in `N` unknowns using forward differences.

Required Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (M, N, X, F)`, where

`M` – Length of `F`. (Input)

`N` – Length of `X`. (Input)

`X` – The point at which the function is evaluated. (Input)
`X` should not be changed by `FCN`.

F – The computed function at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

XC — Vector of length `N` containing the point at which the gradient is to be estimated. (Input)

FC — Vector of length `M` containing the function values at `XC`. (Input)

FJAC — `M` by `N` matrix containing the estimated Jacobian at `XC`. (Output)

Optional Arguments

M — The number of functions. (Input)
Default: `M` = size (`FC`,1).

N — The number of variables. (Input)
Default: `N` = size (`XC`,1).

XSCALE — Vector of length `N` containing the diagonal scaling matrix for the variables. (Input)
In the absence of other information, set all entries to 1.0.
Default: `XSCALE` = 1.0.

EPSFCN — Estimate for the relative noise in the function. (Input)
`EPSFCN` must be less than or equal to 0.1. In the absence of other information, set `EPSFCN` to 0.0.
Default: `EPSFCN` = 0.0.

LDFJAC — Leading dimension of `FJAC` exactly as specified in the dimension statement of the calling program. (Input)
Default: `LDFJAC` = size (`FJAC`,1).

FORTRAN 90 Interface

Generic: `CALL FDJAC (FCN, XC, FC, FJAC [,…])`

Specific: The specific interface names are `S_FDJAC` and `D_FDJAC`.

FORTRAN 77 Interface

Single: `CALL FDJAC (FCN, M, N, XC, XSCALE, FC, EPSFCN, FJAC, LDFJAC)`

Double: The double precision name is `DFDJAC`.

Example

In this example, the Jacobian matrix of

$$f_1(x) = x_1x_2 - 2$$
$$f_2(x) = x_1 - x_1x_2 + 1$$

is estimated by the finite-difference method at the point (1.0, 1.0).

```
      USE FDJAC_INT
      USE UMACH_INT
!                                 Declaration of variables
      INTEGER   N, M, LDFJAC, NOUT
      PARAMETER (N=2, M=2, LDFJAC=2)
      REAL      FJAC(LDFJAC,N), XC(N), FC(M), EPSFCN
      EXTERNAL  FCN
!
      DATA XC/2*1.0E0/
!                                 Set function noise
      EPSFCN = 0.01
```

```
!                                 Evaluate the function at the
!                                 current point
      CALL FCN (M, N, XC, FC)
!                                 Get Jacobian forward-difference
!                                 approximation
      CALL FDJAC (FCN, XC, FC, FJAC, EPSFCN=EPFSCN)
!                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) ((FJAC(I,J),J=1,N),I=1,M)
99999 FORMAT ('  The Jacobian is', /, 2(5X,2F10.2,/),/)
!
      END
!
      SUBROUTINE FCN (M, N, X, F)
!                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER M, N
      REAL    X(N), F(M)
!
      F(1) = X(1)*X(2) - 2.0E0
      F(2) = X(1) - X(1)*X(2) + 1.0E0
!
      RETURN
      END
```

Output

```
 The Jacobian is
 1.00      1.00
 0.00     -1.00
```

Comments

1. Workspace may be explicitly provided, if desired, by use of `F2JAC/DF2JAC`. The reference is:

   `CALL F2JAC (FCN, M, N, XC, XSCALE, FC, EPSFCN, FJAC, LDFJAC, WK)`

   The additional argument is:

   WK — Work vector of length M.

2. This is Description A5.4.1, Dennis and Schnabel, 1983, page 314.

Description

The routine `FDJAC` uses the following finite-difference formula to estimate the Jacobian matrix of function *f* at *x*:

$$\frac{f\left(x+h_j e_j\right)-f\left(x\right)}{h_j}$$

where e_j is the *j*-th unit vector, $h_j = \varepsilon^{1/2} \max\{|x_j|, 1/s_j\} \operatorname{sign}(x_j)$, ε is the machine epsilon, and s_j is the scaling factor of the *j*-th variable. For more details, see Dennis and Schnabel (1983).

Since the finite-difference method has truncation error, cancellation error, and rounding error, users should be aware of possible poor performance. When possible, high precision arithmetic is recommended.

CHGRD

Checks a user-supplied gradient of a function.

Required Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function of which the gradient will be checked. The usage is `CALL FCN (N, X, F)`, where

`N` – Length of `X`. (Input)

`X` – The point at which the function is evaluated. (Input)
`X` should not be changed by `FCN`.

`F` – The computed function value at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

GRAD — Vector of length `N` containing the estimated gradient at `X`. (Input)

X — Vector of length `N` containing the point at which the gradient is to be checked. (Input)

INFO — Integer vector of length `N`. (Output)

`INFO`(`I`) = 0 means the user-supplied gradient is a poor estimate of the numerical gradient at the point `X`(`I`).

`INFO`(`I`) = 1 means the user-supplied gradient is a good estimate of the numerical gradient at the point `X`(`I`).

`INFO`(`I`) = 2 means the user-supplied gradient disagrees with the numerical gradient at the point `X`(`I`), but it might be impossible to calculate the numerical gradient.

`INFO`(`I`) = 3 means the user-supplied gradient and the numerical gradient are both zero at `X`(`I`), and, therefore, the gradient should be rechecked at a different point.

Optional Arguments

N — Dimension of the problem. (Input)
Default: `N` = size (`X`,1).

FORTRAN 90 Interface

Generic: `CALL CHGRD (FCN, GRAD, X, INFO [,…])`

Specific: The specific interface names are `S_CHGRD` and `D_CHGRD`.

FORTRAN 77 Interface

Single: `CALL CHGRD (FCN, GRAD, N, X, INFO)`

Double: The double precision name is `DCHGRD`.

Example

The user-supplied gradient of

$$f(x) = x_i + x_2 e^{-(t-x_3)2/x_4}$$

at (625, 1, 3.125, 0.25) is checked where $t = 2.125$.

```
      USE CHGRD_INT
      USE WRIRN_INT
!                                 Declare variables
      INTEGER    N
      PARAMETER  (N=4)
!
      INTEGER    INFO(N)
      REAL       GRAD(N), X(N)
      EXTERNAL   DRIV, FCN
!
!                                 Input values for point X
!                                 X = (625.0, 1.0, 3.125, .25)
!
      DATA X/625.0E0, 1.0E0, 3.125E0, 0.25E0/
!
      CALL DRIV (N, X, GRAD)
!
      CALL CHGRD (FCN, GRAD, X, INFO)
      CALL WRIRN ('The information vector', INFO, 1, N, 1)
!
      END
!
      SUBROUTINE FCN (N, X, FX)
      INTEGER    N
      REAL       X(N), FX
!
      REAL       EXP
      INTRINSIC  EXP
!
      FX = X(1) + X(2)*EXP(-1.0E0*(2.125E0-X(3))**2/X(4))
      RETURN
      END
!
      SUBROUTINE DRIV (N, X, GRAD)
      INTEGER    N
      REAL       X(N), GRAD(N)
!
      REAL       EXP
      INTRINSIC  EXP
!
      GRAD(1) = 1.0E0
      GRAD(2) = EXP(-1.0E0*(2.125E0-X(3))**2/X(4))
      GRAD(3) = X(2)*EXP(-1.0E0*(2.125E0-X(3))**2/X(4))*2.0E0/X(4)* &
               (2.125-X(3))
      GRAD(4) = X(2)*EXP(-1.0E0*(2.125E0-X(3))**2/X(4))* &
               (2.125E0-X(3))**2/(X(4)*X(4))
      RETURN
      END
```

Output

```
The information vector
1   2   3   4
1   1   1   1
```

Comments

1. Workspace may be explicitly provided, if desired, by use of C2GRD/DC2GRD. The reference is:

   CALL C2GRD (FCN, GRAD, N, X, INFO, FX, XSCALE, EPSFCN, XNEW)

   The additional arguments are as follows:

   FX — The functional value at X.

   XSCALE — Real vector of length N containing the diagonal scaling matrix.

   EPSFCN — The relative "noise" of the function FCN.

   XNEW — Real work vector of length N.

2. Informational errors

Type	Code	
4	1	The user-supplied gradient is a poor estimate of the numerical gradient.

Description

The routine CHGRD uses the following finite-difference formula to estimate the gradient of a function of n variables at x:

$$g_i(x) = \frac{f(x + h_i e_i) - f(x)}{h_i} \qquad \text{for } i=1, \ldots, n$$

where $h_i = \varepsilon^{1/2} \max\{|x_i|, 1/s_i\} \operatorname{sign}(x_i)$, ε is the machine epsilon, e_i is the *i*-th unit vector, and s_i is the scaling factor of the *i*-th variable.

The routine `CHGRD` checks the user-supplied gradient $\nabla f(x)$ by comparing it with the finite-difference gradient $g(x)$. If

$$\left|g_i(x) - (\nabla f(x))_i\right| < \tau\left|(\nabla f(x))_i\right|$$

where $\tau = \varepsilon^{1/4}$, then $(\nabla f(x))_i$, which is the *i*-th element of $\nabla f(x)$, is declared correct; otherwise, `CHGRD` computes the bounds of calculation error and approximation error. When both bounds are too small to account for the difference, $(\nabla f(x))_i$ is reported as incorrect. In the case of a large error bound, `CHGRD` uses a nearly optimal stepsize to recompute $g_i(x)$ and reports that $(\nabla f(x))_i$ is correct if

$$\left|g_i(x) - (\nabla f(x))_i\right| < 2\tau\left|(\nabla f(x))_i\right|$$

Otherwise, $(\nabla f(x))_i$ is considered incorrect unless the error bound for the optimal step is greater than $\tau\,|(\nabla f(x))_i|$. In this case, the numeric gradient may be impossible to compute correctly. For more details, see Schnabel (1985).

CHHES

Checks a user-supplied Hessian of an analytic function.

Required Arguments

GRAD — User-supplied `SUBROUTINE` to compute the gradient at the point `X`. The usage is `CALL GRAD (N, X, G)`, where

`N` – Length of X and G. (Input)

`X` – The point at which the gradient is evaluated. X should not be changed by GRAD. (Input)

`G` – The gradient evaluated at the point X. (Output)

`GRAD` must be declared `EXTERNAL` in the calling program.

HESS — User-supplied `SUBROUTINE` to compute the Hessian at the point `X`. The usage is `CALL HESS (N, X, H, LDH)`, where

`N` – Length of X. (Input)

`X` – The point at which the Hessian is evaluated. (Input)
X should not be changed by HESS.

`H` – The Hessian evaluated at the point X. (Output)

`LDH` – Leading dimension of H exactly as specified in in the dimension statement of the calling program. (Input)

`HESS` must be declared `EXTERNAL` in the calling program.

X — Vector of length `N` containing the point at which the Hessian is to be checked. (Input)

INFO — Integer matrix of dimension `N` by `N`. (Output)

`INFO`(`I`, `J`) = 0 means the Hessian is a poor estimate for function `I` at the point `X`(`J`).

`INFO`(`I`, `J`) = 1 means the Hessian is a good estimate for function `I` at the point `X`(`J`).

`INFO`(`I`, `J`) = 2 means the Hessian disagrees with the numerical Hessian for function `I` at the point `X`(`J`), but it might be impossible to calculate the numerical Hessian.

`INFO`(`I`, `J`) = 3 means the Hessian for function `I` at the point `X`(`J`) and the numerical Hessian are both zero, and, therefore, the gradient should be rechecked at a different point.

Optional Arguments

N — Dimension of the problem. (Input)
Default: `N` = size (`X`,1).

LDINFO — Leading dimension of `INFO` exactly as specified in the dimension statement of the calling program. (Input)
Default: `LDINFO` = size (`INFO`,1).

FORTRAN 90 Interface

Generic: `CALL CHHES (GRAD, HESS, X, INFO [,…])`

Specific: The specific interface names are `S_CHHES` and `D_CHHES`.

FORTRAN 77 Interface

Single: `CALL CHHES (GRAD, HESS, N, X, INFO, LDINFO)`

Double: The double precision name is `DCHHES`.

Example

The user-supplied Hessian of

$$f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

at (−1.2, 1.0) is checked, and the error is found.

```
      USE CHHES_INT
      INTEGER    LDINFO, N
      PARAMETER  (N=2, LDINFO=N)
!
      INTEGER    INFO(LDINFO,N)
      REAL       X(N)
      EXTERNAL   GRD, HES
!
!                                Input values for X
!                                  X = (-1.2, 1.0)
!
      DATA X/-1.2, 1.0/
!
      CALL CHHES (GRD, HES, X, INFO)
!
      END
!
      SUBROUTINE GRD (N, X, UG)
      INTEGER    N
      REAL       X(N), UG(N)
!
      UG(1) = -400.0*X(1)*(X(2)-X(1)*X(1)) + 2.0*X(1) - 2.0
      UG(2) = 200.0*X(2) - 200.0*X(1)*X(1)
      RETURN
      END
!
      SUBROUTINE HES (N, X, HX, LDHS)
      INTEGER    N, LDHS
      REAL       X(N), HX(LDHS,N)
!
      HX(1,1) = -400.0*X(2) + 1200.0*X(1)*X(1) + 2.0
      HX(1,2) = -400.0*X(1)
      HX(2,1) = -400.0*X(1)
!                                 A sign change is made to HX(2,2)
!
      HX(2,2) = -200.0
      RETURN
      END
```

Output

```
*** FATAL    ERROR 1 from CHHES.  The Hessian evaluation with respect to
***          X(2) and X(2) is a poor estimate.
```

Comments

Workspace may be explicitly provided, if desired, by use of `C2HES`/`DC2HES`. The reference is

```
CALL C2HES (GRAD, HESS, N, X, INFO, LDINFO, G, HX, HS,
XSCALE, EPSFCN, INFT, NEWX)
```

The additional arguments are as follows:

G — Vector of length `N` containing the value of the gradient `GRD` at `X`.

HX — Real matrix of dimension `N` by `N` containing the Hessian evaluated at `X`.

HS — Real work vector of length `N`.

XSCALE — Vector of length `N` used to store the diagonal scaling matrix for the variables.

EPSFCN — Estimate of the relative noise in the function.

INFT — Vector of length `N`. For `I` = 1 through `N`, `INFT` contains information about the Jacobian.

NEWX — Real work array of length `N`.

Description

The routine `CHHES` uses the following finite-difference formula to estimate the Hessian of a function of n variables at x:

$$B_{ij}\left(x\right)=\left(g_i\left(x+h_je_j\right)-g_i\left(x\right)\right)/h_j \quad \text{for } j=1,\ldots,n$$

where $h_j = \varepsilon^{1/2}\max\{|x_j|, 1/s_j\}\ \mathrm{sign}(x_j)$, ε is the machine epsilon, e_j is the j-th unit vector, s_j is the scaling factor of the j-th variable, and $g_i(x)$ is the gradient of the function with respect to the i-th variable.

Next, `CHHES` checks the user-supplied Hessian $H(x)$ by comparing it with the finite difference approximation $B(x)$. If

$$|B_{ij}(x) - H_{ij}(x)| < \tau\ |H_{ij}(x)|$$

where $\tau = \varepsilon^{1/4}$, then $H_{ij}(x)$ is declared correct; otherwise, `CHHES` computes the bounds of calculation error and approximation error. When both bounds are too small to account for the difference, $H_{ij}(x)$ is reported as incorrect. In the case of a large error bound, `CHHES` uses a nearly optimal stepsize to recompute $B_{ij}(x)$ and reports that $B_{ij}(x)$ is correct if

$$|B_{ij}(x) - H_{ij}(x)| < 2\tau\ |H_{ij}(x)|$$

Otherwise, $H_{ij}(x)$ is considered incorrect unless the error bound for the optimal step is greater than $\tau\ |H_{ij}(x)|$. In this case, the numeric approximation may be impossible to compute correctly. For more details, see Schnabel (1985).

CHJAC

Checks a user-supplied Jacobian of a system of equations with `M` functions in `N` unknowns.

Required Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (M, N, X, F)`, where

M – Length of F. (Input)

N – Length of X. (Input)

X – The point at which the function is evaluated. (Input)
X should not be changed by FCN.

F – The computed function value at the point X. (Output)

FCN must be declared EXTERNAL in the calling program.

JAC — User-supplied SUBROUTINE to evaluate the Jacobian at a point X. The usage is CALL JAC (M, N, X, FJAC, LDFJAC), where

M – Length of F. (Input)

N – Length of X. (Input)

X – The point at which the function is evaluated. (Input)
X should not be changed by FCN.

FJAC – The computed M by N Jacobian at the point X. (Output)

LDFJAC – Leading dimension of FJAC. (Input)

JAC must be declared EXTERNAL in the calling program.

X — Vector of length N containing the point at which the Jacobian is to be checked. (Input)

INFO — Integer matrix of dimension M by N. (Output)

INFO(I, J) = 0 means the user-supplied Jacobian is a poor estimate for function I at the point X(J).

INFO(I, J) = 1 means the user-supplied Jacobian is a good estimate for function I at the point X(J).

INFO(I, J) = 2 means the user-supplied Jacobian disagrees with the numerical Jacobian for function I at the point X(J), but it might be impossible to calculate the numerical Jacobian.

INFO(I, J) = 3 means the user-supplied Jacobian for function I at the point X(J) and the numerical Jacobian are both zero. Therefore, the gradient should be rechecked at a different point.

Optional Arguments

M — The number of functions in the system of equations. (Input)
Default: `M` = size (`INFO`,1).

N — The number of unknowns in the system of equations. (Input)
Default: `N` = size (`X`,1).

LDINFO — Leading dimension of `INFO` exactly as specified in the dimension statement of the calling program. (Input)
Default: `LDINFO` = size (`INFO`,1).

FORTRAN 90 Interface

Generic: `CALL CHJAC (FCN, JAC, X, INFO [,…])`

Specific: The specific interface names are `S_CHJAC` and `D_CHJAC`.

FORTRAN 77 Interface

Single: `CALL CHJAC (FCN, JAC, M, N, X, INFO, LDINFO)`

Double: The double precision name is `DCHJAC`.

Example

The user-supplied Jacobian of

$$f_1 = 1 - x_1$$
$$f_2 = 10\left(x_2 - x_1^2\right)$$

at (−1.2, 1.0) is checked.

```
      USE CHJAC_INT
      USE WRIRN_INT
      INTEGER    LDINFO, N
      PARAMETER  (M=2,N=2,LDINFO=M)
!
      INTEGER    INFO(LDINFO,N)
      REAL       X(N)
      EXTERNAL   FCN, JAC
!
!                                 Input value for X
!                                    X = (-1.2, 1.0)
!
      DATA X/-1.2, 1.0/
!
      CALL CHJAC (FCN, JAC, X, INFO)
      CALL WRIRN ('The information matrix', INFO)
!
      END
```

```
!
      SUBROUTINE FCN (M, N, X, F)
      INTEGER    M, N
      REAL       X(N), F(M)
!
      F(1) = 1.0 - X(1)
      F(2) = 10.0*(X(2)-X(1)*X(1))
      RETURN
      END
!
      SUBROUTINE JAC (M, N, X, FJAC, LDFJAC)
      INTEGER    M, N, LDFJAC
      REAL       X(N), FJAC(LDFJAC,N)
!
      FJAC(1,1) = -1.0
      FJAC(1,2) = 0.0
      FJAC(2,1) = -20.0*X(1)
      FJAC(2,2) = 10.0
      RETURN
      END
```

Output

```
*** WARNING  ERROR 2 from C2JAC.  The numerical value of the Jacobian
***          evaluation for function 1 at the point X(2) = 1.000000E+00 and
***          the user-supplied value are both zero.  The Jacobian for this
***          function should probably be re-checked at another value for
***          this point.

The information matrix
    1   2
1   1   3
2   1   1
```

Comments

1. Workspace may be explicitly provided, if desired, by use of C2JAC/DC2JAC. The reference is:

   ```
   CALL C2JAC (FCN, JAC, N, X, INFO, LDINFO, FX, FJAC,
   GRAD, XSCALE, EPSFCN, INFT, NEWX)
   ```

   The additional arguments are as follows:

   FX — Vector of length M containing the value of each function in FCN at X.

   FJAC — Real matrix of dimension M by N containing the Jacobian of FCN evaluated at X.

   GRAD — Real work vector of length N used to store the gradient of each function in FCN.

   XSCALE — Vector of length N used to store the diagonal scaling matrix for the variables.

EPSFCN — Estimate of the relative noise in the function.

INFT — Vector of length `N`. For I = 1 through `N`, `INFT` contains information about the Jacobian.

NEWX — Real work array of length `N`.

2. Informational errors

Type	Code	
4	1	The user-supplied Jacobian is a poor estimate of the numerical Jacobian.

Description

The routine `CHJAC` uses the following finite-difference formula to estimate the gradient of the *i*-th function of *n* variables at *x*:

$$g_{ij}(x) = (f_i(x + h_j e_j) - f_i(x))/h_j \quad \text{for } j = 1, \ldots, n$$

where $h_j = \varepsilon^{1/2} \max\{|x_j|, 1/s_j\} \operatorname{sign}(x_j)$, ε is the machine epsilon, e_j is the j-th unit vector, and s_j is the scaling factor of the j-th variable.

Next, `CHJAC` checks the user-supplied Jacobian $J(x)$ by comparing it with the finite difference gradient $g_i(x)$. If

$$|g_{ij}(x) - J_{ij}(x)| < \tau\, |J_{ij}(x)|$$

where $\tau = \varepsilon^{1/4}$, then $\mathrm{J}_{ij}(x)$ is declared correct; otherwise, `CHJAC` computes the bounds of calculation error and approximation error. When both bounds are too small to account for the difference, $J_{ij}(x)$ is reported as incorrect. In the case of a large error bound, `CHJAC` uses a nearly optimal stepsize to recompute $g_{ij}(x)$ and reports that $J_{ij}(x)$ is correct if

$$|g_{ij}(x) - J_{ij}(x)| < 2\tau\, |J_{ij}(x)|$$

Otherwise, $J_{ij}(x)$ is considered incorrect unless the error bound for the optimal step is greater than $\tau\, |J_{ij}(x)|$. In this case, the numeric gradient may be impossible to compute correctly. For more details, see Schnabel (1985).

GGUES

Generates points in an `N`-dimensional space.

Required Arguments

A — Vector of length `N`. (Input)
See `B`.

B — Real vector of length `N`. (Input)
`A` and `B` define the rectangular region in which the points will be generated, i.e., `A(I)` < `S(I)` < `B(I)` for `I` = 1, 2, …, `N`. Note that if `B(I)` < `A(I)`, then `B(I)` < `S(I)` < `A(I)`.

K — The number of points to be generated. (Input)

IDO — Initialization parameter. (Input/Output)
`IDO` must be set to zero for the first call. `GGUES` resets `IDO` to 1 and returns the first generated point in `S`. Subsequent calls should be made with `IDO` = 1.

S — Vector of length `N` containing the generated point. (Output)
Each call results in the next generated point being stored in `S`.

Optional Arguments

N — Dimension of the space. (Input)
Default: `N` = size (`B`,1).

FORTRAN 90 Interface

Generic: `CALL GGUES (A, B, K, IDO, S [,…])`

Specific: The specific interface names are `S_GGUES` and `D_GGUES`.

FORTRAN 77 Interface

Single: `CALL GGUES (N, A, B, K, IDO, S)`

Double: The double precision name is `DGGUES`.

Example

We want to search the rectangle with vertices at coordinates (1, 1), (3, 1), (3, 2), and (1, 2) ten times for a global optimum of a nonlinear function. To do this, we need to generate starting points. The following example illustrates the use of `GGUES` in this process:

```
      USE GGUES_INT
      USE UMACH_INT
!                                 Variable Declarations
      INTEGER    N
      PARAMETER  (N=2)
!
      INTEGER    IDO, J, K, NOUT
      REAL       A(N), B(N), S(N)
!                                 Initializations
!
!                                 A   = ( 1.0, 1.0)
!                                 B   = ( 3.0, 2.0)
!
      DATA A/1.0, 1.0/
      DATA B/3.0, 2.0/
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99998)
99998 FORMAT ('  Point Number', 7X, 'Generated Point')
!
      K = 10
      IDO = 0
      DO 10  J=1, K
         CALL GGUES (A, B, K, IDO, S)
!
         WRITE (NOUT,99999) J, S(1), S(2)
99999    FORMAT (1X, I7, 14X, '(', F4.1, ',', F6.3, ')')
!
   10 CONTINUE
!
      END
```

Output

```
  Point Number       Generated Point
        1              ( 1.5, 1.125)
        2              ( 2.0, 1.500)
        3              ( 2.5, 1.750)
        4              ( 1.5, 1.375)
        5              ( 2.0, 1.750)
        6              ( 1.5, 1.625)
        7              ( 2.5, 1.250)
        8              ( 1.5, 1.875)
        9              ( 2.0, 1.250)
       10              ( 2.5, 1.500)
```

Comments

1. Workspace may be explicitly provided, if desired, by use of G2UES/DG2UES. The reference is:

   ```
   CALL G2UES (N, A, B, K, IDO, S, WK, IWK)
   ```

   The additional arguments are:

   WK — Work vector of length N. WK must be preserved between calls to G2UES.

   IWK — Work vector of length 10. IWK must be preserved between calls to G2UES.

2. Informational error

Type	Code	
4	1	Attempt to generate more than K points.

3. The routine GGUES may be used with any nonlinear optimization routine that requires starting points. The rectangle to be searched (defined by A, B, and N) must be determined; and the number of starting points, K, must be chosen. One possible use for

`GGUES` would be to call `GGUES` to generate a point in the chosen rectangle. Then, call the nonlinear optimization routine using this point as an initial guess for the solution. Repeat this process `K` times. The number of iterations that the optimization routine is allowed to perform should be quite small (5 to 10) during this search process. The best (or best several) point(s) found during the search may be used as an initial guess to allow the optimization routine to determine the optimum more accurately. In this manner, an `N` dimensional rectangle may be effectively searched for a global optimum of a nonlinear function. The choice of `K` depends upon the nonlinearity of the function being optimized. A function with many local optima requires a larger value than a function with only a few local optima.

Description

The routine `GGUES` generates starting points for algorithms that optimize functions of several variables–or, almost equivalently–algorithms that solve simultaneous nonlinear equations.

The routine `GGUES` is based on systematic placement of points to optimize the dispersion of the set. For more details, see Aird and Rice (1977).

Appendix B: Alphabetical Summary of Routines

IMSL MATH/LIBRARY

ACBCB	1441	Adds two complex band matrices, both in band storage mode.
ACHAR	1624	Returns a character given its ASCII value.
AMACH	1685	Retrieves single-precision machine constants.
ARBRB	1438	Adds two band matrices, both in band storage mode.
BCLSF	1274	Solves a nonlinear least squares problem subject to bounds on the variables using a modified Levenberg-Marquardt algorithm and a finite-difference Jacobian.
BCLSJ	1281	Solves a nonlinear least squares problem subject to bounds on the variables using a modified Levenberg-Marquardt algorithm and a user-supplied Jacobian.
BCNLS	1288	Solves a nonlinear least-squares problem subject to bounds on the variables and general linear constraints.
BCOAH	1263	Minimizes a function of `N` variables subject to bounds the variables using a modified Newton method and a user-supplied Hessian.
BCODH	1257	Minimizes a function of `N` variables subject to bounds the variables using a modified Newton method and a finite-difference Hessian.
BCONF	1243	Minimizes a function of `N` variables subject to bounds the variables using a quasi-Newton method and a finite-difference gradient.
BCONG	1249	Minimizes a function of `N` variables subject to bounds the variables using a quasi-Newton method and a user-supplied gradient.
BCPOL	1271	Minimizes a function of `N` variables subject to bounds the variables using a direct search complex algorithm.

BLINF	1427	Computes the bilinear form $x^T Ay$.
BS1GD	656	Evaluates the derivative of a spline on a grid, given its B-spline representation.
BS2DR	653	Evaluates the derivative of a two-dimensional tensor-product spline, given its tensor-product B-spline representation.
BS2GD	656	Evaluates the derivative of a two-dimensional tensor-product spline, given its tensor-product B-spline representation on a grid.
BS2IG	661	Evaluates the integral of a tensor-product spline on a rectangular domain, given its tensor-product B-spline representation.
BS2IN	631	Computes a two-dimensional tensor-product spline interpolant, returning the tensor-product B-spline coefficients.
BS2VL	651	Evaluates a two-dimensional tensor-product spline, given its tensor-product B-spline representation.
BS3DR	666	Evaluates the derivative of a three-dimensional tensor-product spline, given its tensor-product B-spline representation.
BS3GD	670	Evaluates the derivative of a three-dimensional tensor-product spline, given its tensor-product B-spline representation on a grid.
BS3IG	676	Evaluates the integral of a tensor-product spline in three dimensions over a three-dimensional rectangle, given its tensorproduct B-spline representation.
BS3IN	635	Computes a three-dimensional tensor-product spline interpolant, returning the tensor-product B-spline coefficients.
BS3VL	664	Evaluates a three-dimensional tensor-product spline, given its tensor-product B-spline representation.
BSCPP	680	Converts a spline in B-spline representation to piecewise polynomial representation.
BSDER	643	Evaluates the derivative of a spline, given its B-spline representation.
BSINT	622	Computes the spline interpolant, returning the B-spline coefficients.
BSITG	649	Evaluates the integral of a spline, given its B-spline representation.

BSLS2	743	Computes a two-dimensional tensor-product spline approximant using least squares, returning the tensor-product B-spline coefficients.
BSLS3	748	Computes a three-dimensional tensor-product spline approximant using least squares, returning the tensor-product B-spline coefficients.
BSLSQ	725	Computes the least-squares spline approximation, and return the B-spline coefficients.
BSNAK	625	Computes the ‘not-a-knot’ spline knot sequence.
BSOPK	628	Computes the ‘optimal’ spline knot sequence.
BSVAL	641	Evaluates a spline, given its B-spline representation.
BSVLS	729	Computes the variable knot B-spline least squares approximation to given data.
BVPFD	870	Solves a (parameterized) system of differential equations with boundary conditions at two points, using a variable order, variable step size finite-difference method with deferred corrections.
BVPMS	882	Solves a (parameterized) system of differential equations with boundary conditions at two points, using a multiple-shooting method.
CADD	1319	Adds a scalar to each component of a vector, $x \leftarrow x + a$, all complex.
CAXPY	1320	Computes the scalar times a vector plus a vector, $y \leftarrow ax + y$, all complex.
CCBCB	1393	Copies a complex band matrix stored in complex band storage mode.
CCBCG	1400	Converts a complex matrix in band storage mode to a complex matrix in full storage mode.
CCGCB	1398	Converts a complex general matrix to a matrix in complex band storage mode.
CCGCG	1390	Copies a complex general matrix.
CCONV	1064	Computes the convolution of two complex vectors.
CCOPY	1319	Copies a vector x to a vector y, both complex.
CCORL	1073	Computes the correlation of two complex vectors.
CDGRD	1336	Approximates the gradient using central differences.
CDOTC	1320	Computes the complex conjugate dot product, $\bar{x}^T y$.
CDOTU	1320	Computes the complex dot product $x^T y$.

CGBMV	1330	Computes one of the matrix-vector operations: $y \leftarrow \alpha Ax + \beta y,\ y \leftarrow \alpha A^T x + \beta y,$ or $y \leftarrow \alpha \overline{A}^T + \beta y$, where A is a matrix stored in band storage mode.
CGEMM	1333	Computes one of the matrix-matrix operations: $C \leftarrow \alpha AB + \beta C,\ C \leftarrow \alpha A^T B + \beta C,\ C \leftarrow \alpha AB^T$ $+\beta C,\ C \leftarrow \alpha A^T B^T + \beta C,\ C \leftarrow \alpha A\overline{B}^T + \beta C,$ or $C \leftarrow \alpha \overline{A}^T B + \beta C,\ C \leftarrow \alpha A^T \overline{B}^T + \beta C,$ $C \leftarrow \alpha \overline{A}^T B^T + \beta C,$ or $C \leftarrow \alpha \overline{A}^T \overline{B}^T + \beta C$
CGEMV	1329	Computes one of the matrix-vector operations: $y \leftarrow \alpha Ax + \beta y,\ y \leftarrow \alpha A^T x + \beta y,$ or $y \leftarrow \alpha \overline{A}^T + \beta y$,
CGERC	1384	Computes the rank-one update of a complex general matrix: $A \leftarrow A + \alpha x\overline{y}^T$.
CGERU	1384	Computes the rank-one update of a complex general matrix: $A \leftarrow A + \alpha xy^T$.
CHBCB	1411	Copies a complex Hermitian band matrix stored in band Hermitian storage mode to a complex band matrix stored in band storage mode.
CHBMV	1381	Computes the matrix-vector operation $y \leftarrow \alpha Ax + \beta y$, where A is an Hermitian band matrix in band Hermitian storage.
CHEMM	1385	Computes one of the matrix-matrix operations: $C \leftarrow \alpha AB + \beta C$ or $C \leftarrow \alpha BA + \beta C$, where A is an Hermitian matrix and B and C are m by n matrices.
CHEMV	1381	Computes the matrix-vector operation $y \leftarrow \alpha Ax + \beta y$, where A is an Hermitian matrix.
CHER	1384	Computes the rank-one update of an Hermitian matrix: $A \leftarrow A + \alpha x\overline{x}^T$ with x complex and α real.
CHER2	1384	Computes a rank-two update of an Hermitian matrix: $A \leftarrow A + \alpha x\overline{y}^T + \overline{\alpha} y\overline{x}^T$.
CHER2K	1387	Computes one of the Hermitian rank $2k$ operations: $C \leftarrow \alpha A\overline{B}^T + \overline{\alpha} B\overline{A}^T + \beta C$ or $C \leftarrow \alpha \overline{A}^T B + \overline{\alpha}\overline{B}^T A + \beta C$, where C is an n by n Hermitian matrix and A and B are n

		by k matrices in the first case and k by n matrices in the second case.
CHERK	1386	Computes one of the Hermitian rank k operations: $C \leftarrow \alpha A\overline{A}^T + \beta C$ or $C \leftarrow \alpha \overline{A}^T A + \beta C$, where C is an n by n Hermitian matrix and A is an n by k matrix in the first case and a k by n matrix in the second case.
CHFCG	1408	Extends a complex Hermitian matrix defined in its upper triangle to its lower triangle.
CHGRD	1349	Checks a user-supplied gradient of a function.
CHHES	1352	Checks a user-supplied Hessian of an analytic function.
CHJAC	1355	Checks a user-supplied Jacobian of a system of equations with `M` functions in `N` unknowns.
CHOL	1475	Computes the Cholesky factorization of a positive-definite, symmetric or self-adjoint matrix, A.
COND	1476	Computes the condition number of a rectangular matrix, A.
CONFT	734	Computes the least-squares constrained spline approximation, returning the B-spline coefficients.
CONST	1669	Returns the value of various mathematical and physical constants.
CPSEC	1631	Returns CPU time used in seconds.
CRBCB	1405	Converts a real matrix in band storage mode to a complex matrix in band storage mode.
CRBRB	1392	Copies a real band matrix stored in band storage mode.
CRBRG	1397	Converts a real matrix in band storage mode to a real general matrix.
CRGCG	1402	Copies a real general matrix to a complex general matrix.
CRGRB	1395	Converts a real general matrix to a matrix in band storage mode.
CRGRG	1389	Copies a real general matrix.
CRRCR	1403	Copies a real rectangular matrix to a complex rectangular matrix.
CS1GD	602	Evaluates the derivative of a cubic spline on a grid.
CSAKM	500	Computes the Akima cubic spline interpolant.
CSBRB	1409	Copies a real symmetric band matrix stored in band symmetric storage mode to a real band matrix stored in band storage mode.

CSCAL	1319	Multiplies a vector by a scalar, $y \leftarrow ay$, both complex.
CSCON	603	Computes a cubic spline interpolant that is consistent with the concavity of the data.
CSDEC	593	Computes the cubic spline interpolant with specified derivative endpoint conditions.
CSDER	610	Evaluates the derivative of a cubic spline.
CSET	1318	Sets the components of a vector to a scalar, all complex.
CSFRG	1406	Extends a real symmetric matrix defined in its upper triangle to its lower triangle.
CSHER	597	Computes the Hermite cubic spline interpolant.
CSIEZ	587	Computes the cubic spline interpolant with the 'not-a-knot' condition and return values of the interpolant at specified points.
CSINT	590	Computes the cubic spline interpolant with the 'not-a-knot' condition.
CSITG	616	Evaluates the integral of a cubic spline.
CSPER	506	Computes the cubic spline interpolant with periodic boundary conditions.
CSROT	1325	Applies a complex Givens plane rotation.
CSROTM	1326	Applies a complex modified Givens plane rotation.
CSSCAL	1319	Multiplies a complex vector by a single-precision scalar, $y \leftarrow ay$.
CSSCV	761	Computes a smooth cubic spline approximation to noisy data using cross-validation to estimate the smoothing parameter.
CSSED	754	Smooths one-dimensional data by error detection.
CSSMH	758	Computes a smooth cubic spline approximation to noisy data.
CSUB	1319	Subtracts each component of a vector from a scalar, $x \leftarrow a - x$, all complex.
CSVAL	609	Evaluates a cubic spline.
CSVCAL	1319	Multiplies a complex vector by a single-precision scalar and store the result in another complex vector, $y \leftarrow ax$.
CSWAP	1320	Interchanges vectors x and y, both complex.
CSYMM	1334	Computes one of the matrix-matrix operations: $C \leftarrow \alpha AB + \beta C$ or $C \leftarrow \alpha BA + \beta C$, where A is a symmetric matrix and B and C are m by n matrices.

CSYR2K	1335	Computes one of the symmetric rank 2*k* operations: $C \leftarrow \alpha AB^T + \alpha BA^T + \beta C$ or $C \leftarrow \alpha A^T B + \alpha B^T A + \beta C$, where *C* is an *n* by *n* symmetric matrix and *A* and *B* are *n* by *k* matrices in the first case and *k* by *n* matrices in the second case.
CSYRK	1334	Computes one of the symmetric rank *k* operations: $C \leftarrow \alpha AA^T + \beta C$ or $C \leftarrow \alpha A^T A + \beta C$, where *C* is an *n* by *n* symmetric matrix and *A* is an *n* by *k* matrix in the first case and a *k* by *n* matrix in the second case.
CTBMV	1331	Computes one of the matrix-vector operations: $x \leftarrow Ax, x \leftarrow A^T x$, or $x \leftarrow \overline{A}^T x$, where *A* is a triangular matrix in band storage mode.
CTBSV	1332	Solves one of the complex triangular systems: $x \leftarrow A^{-1}x$, $x \leftarrow \left(A^{-1}\right)^T x$, or $x \leftarrow \left(\overline{A}^T\right)^{-1} x$, where *A* is a triangular matrix in band storage mode.
CTRMM	1335	Computes one of the matrix-matrix operations: $B \leftarrow \alpha AB, B \leftarrow \alpha A^T B, B \leftarrow \alpha BA, B \leftarrow \alpha BA^T$, $B \leftarrow \alpha \overline{A}^T B$, or $B \leftarrow \alpha B\overline{A}^T$ where *B* is an *m* by *n* matrix and *A* is a triangular matrix.
CTRMV	1331	Computes one of the matrix-vector operations: $x \leftarrow Ax, x \leftarrow A^T x$, or $x \leftarrow \overline{A}^T x$, where *A* is a triangular matrix.
CTRSM	1336	Solves one of the complex matrix equations: $B \leftarrow \alpha A^{-1}B, B \leftarrow \alpha BA^{-1}, B \leftarrow \alpha\left(A^{-1}\right)^T B, B \leftarrow \alpha B\left(A^{-1}\right)^T$, $B \leftarrow \alpha\left(\overline{A}^T\right)^{-1} B$, or $B \leftarrow \alpha B\left(\overline{A}^T\right)^{-1}$ where *A* is a traiangular matrix.
CTRSV	1331	Solves one of the complex triangular systems: $x \leftarrow A^{-1}x, x \leftarrow \left(A^{-1}\right)^T x$, or $x \leftarrow \left(\overline{A}^T\right)^{-1} x$, where *A* is a triangular matrix.
CUNIT	1672	Converts `X` in units `XUNITS` to `Y` in units `YUNITS`.
CVCAL	1319	Multiplies a vector by a scalar and store the result in another vector, $y \leftarrow ax$, all complex.
CVTSI	1630	Converts a character string containing an integer number into the corresponding integer form.

CZCDOT	1321	Computes the sum of a complex scalar plus a complex conjugate dot product, $a+\bar{x}^T y$, using a double-precision accumulator.
CZDOTA	1321	Computes the sum of a complex scalar, a complex dot product and the double-complex accumulator, which is set to the result ACC ← ACC + $a + x^T y$.
CZDOTC	1320	Computes the complex conjugate dot product, $\bar{x}^T y$, using a double-precision accumulator.
CZDOTI	1321	Computes the sum of a complex scalar plus a complex dot product using a double-complex accumulator, which is set to the result ACC ← $a + x^T y$.
CZDOTU	1320	Computes the complex dot product $x^T y$ using a double-precision accumulator.
CZUDOT	1321	Computes the sum of a complex scalar plus a complex dot product, $a + x^T y$, using a double-precision accumulator.
DASPG	889	Solves a first order differential-algebraic system of equations, $g(t, y, y') = 0$, using Petzold–Gear BDF method.
DERIV	827	Computes the first, second or third derivative of a user-supplied function.
DET	1477	Computes the determinant of a rectangular matrix, A.
DIAG	1479	Constructs a square diagonal matrix from a rank-1 array or several diagonal matrices from a rank-2 array.
DIAGONALS	1479	Extracts a rank-1 array whose values are the diagonal terms of a rank-2 array argument.
DISL1	1452	Computes the 1-norm distance between two points.
DISL2	1450	Computes the Euclidean (2-norm) distance between two points.
DISLI	1454	Computes the infinity norm distance between two points.
DLPRS	1297	Solves a linear programming problem via the revised simplex algorithm.
DMACH	1686	See AMACH.
DQADD	1460	Adds a double-precision scalar to the accumulator in extended precision.
DQINI	1460	Initializes an extended-precision accumulator with a double-precision scalar.

DQMUL	1460	Multiplies double-precision scalars in extended precision.
DQSTO	1460	Stores a double-precision approximation to an extended-precision scalar.
DSDOT	1371	Computes the single-precision dot product $x^T y$ using a double precision accumulator.
DUMAG	1664	This routine handles MATH/LIBRARY and STAT/LIBRARY type `DOUBLE PRECISION` options.
EIG	1480	Computes the eigenvalue-eigenvector decomposition of an ordinary or generalized eigenvalue problem.
EPICG	467	Computes the performance index for a complex eigensystem.
EPIHF	518	Computes the performance index for a complex Hermitian eigensystem.
EPIRG	460	Computes the performance index for a real eigensystem.
EPISB	501	Computes the performance index for a real symmetric eigensystem in band symmetric storage mode.
EPISF	483	Computes the performance index for a real symmetric eigensystem.
ERROR_POST	1568	Prints error messages that are generated by IMSL routines using EPACK
ERSET	1679	Sets error handler default print and stop actions.
EVAHF	508	Computes the largest or smallest eigenvalues of a complex Hermitian matrix.
EVASB	490	Computes the largest or smallest eigenvalues of a real symmetric matrix in band symmetric storage mode.
EVASF	473	Computes the largest or smallest eigenvalues of a real symmetric matrix.
EVBHF	513	Computes the eigenvalues in a given range of a complex Hermitian matrix.
EVBSB	495	Computes the eigenvalues in a given interval of a real symmetric matrix stored in band symmetric storage mode.
EVBSF	478	Computes selected eigenvalues of a real symmetric matrix.
EVCCG	464	Computes all of the eigenvalues and eigenvectors of a complex matrix.
EVCCH	526	Computes all of the eigenvalues and eigenvectors of a complex upper Hessenberg matrix.

EVCHF	505	Computes all of the eigenvalues and eigenvectors of a complex Hermitian matrix.
EVCRG	457	Computes all of the eigenvalues and eigenvectors of a real matrix.
EVCRH	522	Computes all of the eigenvalues and eigenvectors of a real upper Hessenberg matrix.
EVCSB	487	Computes all of the eigenvalues and eigenvectors of a real symmetric matrix in band symmetric storage mode.
EVCSF	471	Computes all of the eigenvalues and eigenvectors of a real symmetric matrix.
EVEHF	510	Computes the largest or smallest eigenvalues and the corresponding eigenvectors of a complex Hermitian matrix.
EVESB	492	Computes the largest or smallest eigenvalues and the corresponding eigenvectors of a real symmetric matrix in band symmetric storage mode.
EVESF	475	Computes the largest or smallest eigenvalues and the corresponding eigenvectors of a real symmetric matrix.
EVFHF	515	Computes the eigenvalues in a given range and the corresponding eigenvectors of a complex Hermitian matrix.
EVFSB	498	Computes the eigenvalues in a given interval and the corresponding eigenvectors of a real symmetric matrix stored in band symmetric storage mode.
EVFSF	480	Computes selected eigenvalues and eigenvectors of a real symmetric matrix.
EVLCG	462	Computes all of the eigenvalues of a complex matrix.
EVLCH	525	Computes all of the eigenvalues of a complex upper Hessenberg matrix.
EVLHF	502	Computes all of the eigenvalues of a complex Hermitian matrix.
EVLRG	455	Computes all of the eigenvalues of a real matrix.
EVLRH	520	Computes all of the eigenvalues of a real upper Hessenberg matrix.
EVLSB	485	Computes all of the eigenvalues of a real symmetric matrix in band symmetric storage mode.
EVLSF	469	Computes all of the eigenvalues of a real symmetric matrix.
EYE	1481	Creates a rank-2 square array whose diagonals are all the value one.

FAURE_FREE	1655	Frees the structure containing information about the Faure sequence.
FAURE_INIT	1655	Shuffled Faure sequence initialization.
FAURE_NEXT	1656	Computes a shuffled Faure sequence.
FAST_DFT	992	Computes the Discrete Fourier Transform of a rank-1 complex array, *x*.
FAST_2DFT	1000	Computes the Discrete Fourier Transform (2DFT) of a rank-2 complex array, *x*.
FAST_3DFT	1006	Computes the Discrete Fourier Transform (2DFT) of a rank-3 complex array, *x*.
FCOSI	1030	Computes parameters needed by `FCOST`.
FCOST	1028	Computes the discrete Fourier cosine transformation of an even sequence.
FDGRD	1338	Approximates the gradient using forward differences.
FDHES	1340	Approximates the Hessian using forward differences and function values.
FDJAC	1346	Approximates the Jacobian of `M` functions in `N` unknowns using forward differences.
FFT	1482	The Discrete Fourier Transform of a complex sequence and its inverse transform.
FFT_BOX	1482	The Discrete Fourier Transform of several complex or real sequences.
FFT2B	1048	Computes the inverse Fourier transform of a complex periodic two-dimensional array.
FFT2D	1045	Computes Fourier coefficients of a complex periodic two-dimensional array.
FFT3B	1055	Computes the inverse Fourier transform of a complex periodic three-dimensional array.
FFT3F	1051	Computes Fourier coefficients of a complex periodic threedimensional array.
FFTCB	1019	Computes the complex periodic sequence from its Fourier coefficients.
FFTCF	1017	Computes the Fourier coefficients of a complex periodic sequence.
FFTCI	1022	Computes parameters needed by `FFTCF` and `FFTCB`.
FFTRB	1012	Computes the real periodic sequence from its Fourier coefficients.

FFTRF	1009	Computes the Fourier coefficients of a real periodic sequence.
FFTRI	1015	Computes parameters needed by `FFTRF` and `FFTRB`.
FNLSQ	720	Computes a least-squares approximation with user-supplied basis functions.
FPS2H	961	Solves Poisson's or Helmholtz's equation on a two-dimensional rectangle using a fast Poisson solver based on the `HODIE` finite-difference scheme on a uni mesh.
FPS3H	967	Solves Poisson's or Helmholtz's equation on a three-dimensional box using a fast Poisson solver based on the `HODIE` finite-difference scheme on a uniform mesh.
FQRUL	824	Computes a Fejér quadrature rule with various classical weight functions.
FSINI	1026	Computes parameters needed by `FSINT`.
FSINT	1024	Computes the discrete Fourier sine transformation of an odd sequence.
GDHES	1343	Approximates the Hessian using forward differences and a user-supplied gradient.
GGUES	1359	Generates points in an `N`-dimensional space.
GMRES	368	Uses restarted `GMRES` with reverse communication to generate an approximate solution of $Ax = b$.
GPICG	542	Computes the performance index for a generalized complex eigensystem $Az = \lambda Bz$.
GPIRG	535	Computes the performance index for a generalized real eigensystem $Az = \lambda Bz$.
GPISP	549	Computes the performance index for a generalized real symmetric eigensystem problem.
GQRCF	815	Computes a Gauss, Gauss-Radau or Gauss-Lobatto quadrature rule given the recurrence coefficients for the monic polynomials orthogonal with respect to the weight function.
GQRUL	811	Computes a Gauss, Gauss-Radau, or Gauss-Lobatto quadrature rule with various classical weight functions.
GVCCG	540	Computes all of the eigenvalues and eigenvectors of a generalized complex eigensystem $Az = \lambda Bz$.
GVCRG	531	Computes all of the eigenvalues and eigenvectors of a generalized real eigensystem $Az = \lambda Bz$.

GVCSP	547	Computes all of the eigenvalues and eigenvectors of the generalized real symmetric eigenvalue problem $Az = \lambda Bz$, with B symmetric positive definite.
GVLCG	537	Computes all of the eigenvalues of a generalized complex eigensystem $Az = \lambda Bz$.
GVLRG	529	Computes all of the eigenvalues of a generalized real eigensystem $Az = \lambda Bz$.
GVLSP	544	Computes all of the eigenvalues of the generalized real symmetric eigenvalue problem $Az = \lambda Bz$, with B symmetric positive definite.
HRRRR	1425	Computes the Hadamard product of two real rectangular matrices.
HYPOT	1675	Computes $\sqrt{a^2 + b^2}$ without underflow or overflow.
IACHAR	1625	Returns the integer ASCII value of a character argument.
IADD	1319	Adds a scalar to each component of a vector, $x \leftarrow x + a$, all integer.
ICAMAX	1324	Finds the smallest index of the component of a complex vector having maximum magnitude.
ICAMIN	1323	Finds the smallest index of the component of a complex vector having minimum magnitude.
ICASE	1626	Returns the ASCII value of a character converted to uppercase.
ICOPY	1319	Copies a vector x to a vector y, both integer.
IDYWK	1637	Computes the day of the week for a given date.
IERCD	1680	Retrieves the code for an informational error.
IFFT	1483	The inverse of the Discrete Fourier Transform of a complex sequence.
IFFT_BOX	1484	The inverse Discrete Fourier Transform of several complex or real sequences.
IFNAN(X)	1686	Checks if a value is NaN (not a number).
IICSR	1627	Compares two character strings using the ASCII collating sequence but without regard to case.
IIDEX	1629	Determines the position in a string at which a given character sequence begins without regard to case.
IIMAX	1323	Finds the smallest index of the maximum component of a integer vector.
IIMIN	1323	Finds the smallest index of the minimum of an integer vector.

IMACH	1683	Retrieves integer machine constants.
INLAP	1078	Computes the inverse Laplace transform of a complex function.
ISAMAX	1374	Finds the smallest index of the component of a single-precision vector having maximum absolute value.
ISAMIN	1374	Finds the smallest index of the component of a single-precision vector having minimum absolute value.
ISET	1318	Sets the components of a vector to a scalar, all integer.
ISMAX	1374	Finds the smallest index of the component of a single-precision vector having maximum value.
ISMIN	1374	Finds the smallest index of the component of a single-precision vector having minimum value.
ISNAN	1485	This is a generic logical function used to test scalars or arrays for occurrence of an IEEE 754 Standard format of floating point (ANSI/IEEE 1985) NaN, or not-a-number.
ISRCH	1620	Searches a sorted integer vector for a given integer and return its index.
ISUB	1319	Subtracts each component of a vector from a scalar, $x \leftarrow a - x$, all integer.
ISUM	1322	Sums the values of an integer vector.
ISWAP	1320	Interchanges vectors x and y, both integer.
IUMAG	1658	Sets or retrieves MATH/LIBRARY integer options.
IVMRK	844	Solves an initial-value problem $y' = f(t, y)$ for ordinary differential equations using Runge-Kutta pairs of various orders.
IVPAG	854	Solves an initial-value problem for ordinary differential equations using either Adams-Moulton's or Gear's BDF method.
IVPRK	837	Solves an initial-value problem for ordinary differential equations using the Runge-Kutta-Verner fifth-order and sixth-order method.
IWKCIN	1701	Initializes bookkeeping locations describing the character workspace stack.
IWKIN	1700	Initializes bookkeeping locations describing the workspace stack.
JCGRC	365	Solves a real symmetric definite linear system using the Jacobi preconditioned conjugate gradient method with reverse communication.

LCHRG	406	Computes the Cholesky decomposition of a symmetric positive semidefinite matrix with optional column pivoting.
LCLSQ	388	Solves a linear least-squares problem with linear constraints.
LCONF	1310	Minimizes a general objective function subject to linear equality/inequality constraints.
LCONG	1316	Minimizes a general objective function subject to linear equality/inequality constraints.
LDNCH	412	Downdates the R^TR Cholesky factorization of a real symmetric positive definite matrix after a rank-one matrix is removed.
LFCCB	262	Computes the LU factorization of a complex matrix in band storage mode and estimate its L_1 condition number.
LFCCG	108	Computes the LU factorization of a complex general matrix and estimate its L_1 condition number.
LFCCT	132	Estimates the condition number of a complex triangular matrix.
LFCDH	179	Computes the $R^H R$ factorization of a complex Hermitian positive definite matrix and estimate its L_1 condition number.
LFCDS	143	Computes the $R^T R$ Cholesky factorization of a real symmetric positive definite matrix and estimate its L_1condition number.
LFCHF	197	Computes the $U\,DU^H$ factorization of a complex Hermitian matrix and estimate its L_1 condition number.
LFCQH	284	Computes the $R^H R$ factorization of a complex Hermitian positive definite matrix in band Hermitian storage mode and estimate its L_1 condition number.
LFCQS	240	Computes the $R^T R$ Cholesky factorization of a real symmetric positive definite matrix in band symmetric storage mode and estimate its L_1 condition number.
LFCRB	219	Computes the LU factorization of a real matrix in band storage mode and estimate its L_1 condition number.
LFCRG	89	Computes the LU factorization of a real general matrix and estimate its L_1 condition number.
LFCRT	125	Estimates the condition number of a real triangular matrix.

LFCSF	162	Computes the $U\,DU^T$ factorization of a real symmetric matrix and estimate its L_1 condition number.
LFDCB	274	Computes the determinant of a complex matrix given the *LU* factorization of the matrix in band storage mode.
LFDCG	119	Computes the determinant of a complex general matrix given the *LU* factorization of the matrix.
LFDCT	134	Computes the determinant of a complex triangular matrix.
LFDDH	190	Computes the determinant of a complex Hermitian positive definite matrix given the $R^H\,R$ Cholesky factorization of the matrix.
LFDDS	153	Computes the determinant of a real symmetric positive definite matrix given the $R^H\,R$ Cholesky factorization of the matrix.
LFDHF	207	Computes the determinant of a complex Hermitian matrix given the $U\,DU^H$ factorization of the matrix.
LFDQH	295	Computes the determinant of a complex Hermitian positive definite matrix given the $R^H\,R$ Cholesky factorization in band Hermitian storage mode.
LFDQS	250	Computes the determinant of a real symmetric positive definite matrix given the $R^T\,R$ Cholesky factorization of the band symmetric storage mode.
LFDRB	230	Computes the determinant of a real matrix in band storage mode given the *LU* factorization of the matrix.
LFDRG	99	Computes the determinant of a real general matrix given the *LU* factorization of the matrix.
LFDRT	127	Computes the determinant of a real triangular matrix.
LFDSF	172	Computes the determinant of a real symmetric matrix given the $U\,DU^T$ factorization of the matrix.
LFICB	270	Uses iterative refinement to improve the solution of a complex system of linear equations in band storage mode.
LFICG	116	Uses iterative refinement to improve the solution of a complex general system of linear equations.
LFIDH	187	Uses iterative refinement to improve the solution of a complex Hermitian positive definite system of linear equations.
LFIDS	150	Uses iterative refinement to improve the solution of a real symmetric positive definite system of linear equations.

LFIHF	204	Uses iterative refinement to improve the solution of a complex Hermitian system of linear equations.
LFIQH	292	Uses iterative refinement to improve the solution of a complex Hermitian positive definite system of linear equations in band Hermitian storage mode.
LFIQS	247	Uses iterative refinement to improve the solution of a real symmetric positive definite system of linear equations in band symmetric storage mode.
LFIRB	227	Uses iterative refinement to improve the solution of a real system of linear equations in band storage mode.
LFIRG	96	Uses iterative refinement to improve the solution of a real general system of linear equations.
LFISF	169	Uses iterative refinement to improve the solution of a real symmetric system of linear equations.
LFSCB	268	Solves a complex system of linear equations given the LU factorization of the coefficient matrix in band storage mode.
LFSCG	114	Solves a complex general system of linear equations given the LU factorization of the coefficient matrix.
LFSDH	184	Solves a complex Hermitian positive definite system of linear equations given the $R^H R$ factorization of the coefficient matrix.
LFSDS	148	Solves a real symmetric positive definite system of linear equations given the $R^T R$ Choleksy factorization of the coefficient matrix.
LFSHF	202	Solves a complex Hermitian system of linear equations given the $U DU^H$ factorization of the coefficient matrix.
LFSQH	290	Solves a complex Hermitian positive definite system of linear equations given the factorization of the coefficient matrix in band Hermitian storage mode.
LFSQS	245	Solves a real symmetric positive definite system of linear equations given the factorization of the coefficient matrix in band symmetric storage mode.
LFSRB	225	Solves a real system of linear equations given the LU factorization of the coefficient matrix in band storage mode.
LFSRG	94	Solves a real general system of linear equations given the LU factorization of the coefficient matrix.
LFSSF	167	Solves a real symmetric system of linear equations given the $U DU^T$ factorization of the coefficient matrix.

LFSXD	336	Solves a real sparse symmetric positive definite system of linear equations, given the Cholesky factorization of the coefficient matrix.
LFSXG	306	Solves a sparse system of linear equations given the *LU* factorization of the coefficient matrix.
LFSZD	349	Solves a complex sparse Hermitian positive definite system of linear equations, given the Cholesky factorization of the coefficient matrix.
LFSZG	319	Solves a complex sparse system of linear equations given the *LU* factorization of the coefficient matrix.
LFTCB	265	Computes the *LU* factorization of a complex matrix in band storage mode.
LFTCG	111	Computes the *LU* factorization of a complex general matrix.
LFTDH	182	Computes the $R^H R$ factorization of a complex Hermitian positive definite matrix.
LFTDS	146	Computes the $R^T R$ Cholesky factorization of a real symmetric positive definite matrix.
LFTHF	200	Computes the $U DU^H$ factorization of a complex Hermitian matrix.
LFTQH	288	Computes the $R^H R$ factorization of a complex Hermitian positive definite matrix in band Hermitian storage mode.
LFTQS	243	Computes the $R^T R$ Cholesky factorization of a real symmetric positive definite matrix in band symmetric storage mode.
LFTRB	222	Computes the *LU* factorization of a real matrix in band storage mode.
LFTRG	92	Computes the *LU* factorization of a real general matrix.
LFTSF	164	Computes the $U DU^T$ factorization of a real symmetric matrix.
LFTXG	301	Computes the *LU* factorization of a real general sparse matrix.
LFTZG	314	Computes the *LU* factorization of a complex general sparse matrix.
LINCG	121	Computes the inverse of a complex general matrix.
LINCT	136	Computes the inverse of a complex triangular matrix.
LINDS	154	Computes the inverse of a real symmetric positive definite matrix.

LINRG	101	Computes the inverse of a real general matrix.
LINRT	128	Computes the inverse of a real triangular matrix.
LIN_EIG_GEN	439	Computes the eigenvalues of a self-adjoint matrix, A.
LIN_EIG_SELF	432	Computes the eigenvalues of a self-adjoint matrix, A.
LIN_GEIG_SELF	448	Computes the generalized eigenvalues of an $n \times n$ matrix pencil, $Av = \lambda Bv$.
LIN_SOL_GEN	9	Solves a general system of linear equations $Ax = b$.
LIN_SOL_LSQ	27	Solves a rectangular system of linear equations $Ax \cong b$, in a least-squares sense.
LIN_SOL_SELF	17	Solves a system of linear equations $Ax = b$, where A is a self-adjoint matrix.
LIN_SOL_SVD	36	Solves a rectangular least-squares system of linear equations $Ax \cong b$ using singular value decomposition.
LIN_SOL_TRI	44	Solves multiple systems of linear equations.
LIN_SVD	57	Computes the singular value decomposition (SVD) of a rectangular matrix, A.
LNFXD	331	Computes the numerical Cholesky factorization of a sparse symmetrical matrix A.
LNFZD	344	Computes the numerical Cholesky factorization of a sparse Hermitian matrix A.
LQERR	396	Accumulates the orthogonal matrix Q from its factored form given the QR factorization of a rectangular matrix A.
LQRRR	392	Computes the QR decomposition, $AP = QR$, using Householder transformations.
LQRRV	381	Computes the least-squares solution using Householder transformations applied in blocked form.
LQRSL	398	Computes the coordinate transformation, projection, and complete the solution of the least-squares problem $Ax = b$.
LSACB	257	Solves a complex system of linear equations in band storage mode with iterative refinement.
LSACG	103	Solves a complex general system of linear equations with iterative refinement.
LSADH	173	Solves a Hermitian positive definite system of linear equations with iterative refinement.
LSADS	138	Solves a real symmetric positive definite system of linear equations with iterative refinement.

LSAHF	191	Solves a complex Hermitian system of linear equations with iterative refinement.
LSAQH	276	Solves a complex Hermitian positive definite system of linear equations in band Hermitian storage mode with iterative refinement.
LSAQS	232	Solves a real symmetric positive definite system of linear equations in band symmetric storage mode with iterative refinement.
LSARB	213	Solves a real system of linear equations in band storage mode with iterative refinement.
LSARG	83	Solves a real general system of linear equations with iterative refinement.
LSASF	156	Solves a real symmetric system of linear equations with iterative refinement.
LSBRR	385	Solves a linear least-squares problem with iterative refinement.
LSCXD	327	Performs the symbolic Cholesky factorization for a sparse symmetric matrix using a minimum degree ordering or a userspecified ordering, and set up the data structure for the numerical Cholesky factorization.
LSGRR	424	Computes the generalized inverse of a real matrix.
LSLCB	259	Solves a complex system of linear equations in band storage mode without iterative refinement.
LSLCC	356	Solves a complex circulant linear system.
LSLCG	106	Solves a complex general system of linear equations without iterative refinement.
LSLCQ	253	Computes the *LDU* factorization of a complex tridiagonal matrix *A* using a cyclic reduction algorithm.
LSLCR	211	Computes the *LDU* factorization of a real tridiagonal matrix *A* using a cyclic reduction algorithm.
LSLCT	130	Solves a complex triangular system of linear equations.
LSLDH	176	Solves a complex Hermitian positive definite system of linear equations without iterative refinement.
LSLDS	140	Solves a real symmetric positive definite system of linear equations without iterative refinement.
LSLHF	194	Solves a complex Hermitian system of linear equations without iterative refinement.

LSLPB	237	Computes the $R^T DR$ Cholesky factorization of a real symmetric positive definite matrix A in codiagonal band symmetric storage mode. Solve a system $Ax = b$.
LSLQB	281	Computes the $R^H DR$ Cholesky factorization of a complex hermitian positive-definite matrix A in codiagonal band hermitian storage mode. Solve a system $Ax = b$.
LSLQH	279	Solves a complex Hermitian positive definite system of linearequations in band Hermitian storage mode without iterative refinement.
LSLQS	234	Solves a real symmetric positive definite system of linear equations in band symmetric storage mode without iterative refinement.
LSLRB	216	Solves a real system of linear equations in band storage mode without iterative refinement.
LSLRG	85	Solves a real general system of linear equations without iterative refinement.
LSLRT	123	Solves a real triangular system of linear equations.
LSLSF	159	Solves a real symmetric system of linear equations without iterative refinement.
LSLTC	354	Solves a complex Toeplitz linear system.
LSLTO	352	Solves a real Toeplitz linear system.
LSLTQ	252	Solves a complex tridiagonal system of linear equations.
LSLTR	209	Solves a real tridiagonal system of linear equations.
LSLXD	323	Solves a sparse system of symmetric positive definite linear algebraic equations by Gaussian elimination.
LSLXG	297	Solves a sparse system of linear algebraic equations by Gaussian elimination.
LSLZD	340	Solves a complex sparse Hermitian positive definite system of linear equations by Gaussian elimination.
LSLZG	309	Solves a complex sparse system of linear equations by Gaussian elimination.
LSQRR	378	Solves a linear least-squares problem without iterative refinement.
LSVCR	419	Computes the singular value decomposition of a complex matrix.
LSVRR	415	Computes the singular value decomposition of a real matrix.

LUPCH	409	Updates the R^TR Cholesky factorization of a real symmetric positive definite matrix after a rank-one matrix is added.
LUPQR	402	Computes an updated QR factorization after the rank-one matrix αxy^T is added.
MCRCR	1423	Multiplies two complex rectangular matrices, AB.
MOLCH	946	Solves a system of partial differential equations of the form $u_t = f(x, t, u, u_x, u_{xx})$ using the method of lines. The solution is represented with cubic Hermite polynomials.
MRRRR	1421	Multiplies two real rectangular matrices, AB.
MUCBV	1436	Multiplies a complex band matrix in band storage mode by a complex vector.
MUCRV	1435	Multiplies a complex rectangular matrix by a complex vector.
MURBV	1433	Multiplies a real band matrix in band storage mode by a real vector.
MURRV	1431	Multiplies a real rectangular matrix by a vector.
MXTXF	1415	Computes the transpose product of a matrix, A^TA.
MXTYF	1416	Multiplies the transpose of matrix A by matrix B, A^TB.
MXYTF	1418	Multiplies a matrx A by the transpose of a matrix B, AB^T.
NAN	1486	Returns, as a scalar function, a value corresponding to the IEEE 754 Standard format of floating point (ANSI/IEEE 1985) for NaN. .
N1RTY	1680	Retrieves an error type for the most recently called IMSL routine.
NDAYS	1634	Computes the number of days from January 1, 1900, to the given date.
NDYIN	1636	Gives the date corresponding to the number of days since January 1, 1900.
NEQBF	1169	Solves a system of nonlinear equations using factored secant update with a finite-difference approximation to the Jacobian.
NEQBJ	1174	Solves a system of nonlinear equations using factored secant update with a user-supplied Jacobian.
NEQNF	1162	Solves a system of nonlinear equations using a modified Powell hybrid algorithm and a finite-difference approximation to the Jacobian.

NEQNJ	1165	Solves a system of nonlinear equations using a modified Powell hybrid algorithm with a user-supplied Jacobian.
NNLPF	1323	Uses a sequential equality constrained QP method.
NNLPG	1329	Uses a sequential equality constrained QP method.
NORM	1487	Computes the norm of a rank-1 or rank-2 array. For rank-3 arrays, the norms of each rank-2 array, in dimension 3, are computed.
NR1CB	1449	Computes the 1-norm of a complex band matrix in band storage mode.
NR1RB	1447	Computes the 1-norm of a real band matrix in band storage mode.
NR1RR	1444	Computes the 1-norm of a real matrix.
NR2RR	1446	Computes the Frobenius norm of a real rectangular matrix.
NRIRR	1443	Computes the infinity norm of a real matrix.
OPERATOR: .h.	1472	Computes transpose and conjugate transpose of a matrix.
OPERATOR: .hx.	1471	Computes matrix-vector and matrix-matrix products.
OPERATOR:.i.	1473	Computes the inverse matrix, for square non-singular matrices.
OPERATOR:.ix.	1474	Computes the inverse matrix times a vector or matrix for square non-singular matrices.
OPERATOR:..t.	1472	Computes transpose and conjugate transpose of a matrix.
OPERATOR:.tx.	1471	Computes matrix-vector and matrix-matrix products.
OPERATOR:.x.	1471	Computes matrix-vector and matrix-matrix products..
OPERATOR:..xh.	1471	Computes matrix-vector and matrix-matrix products.
OPERATOR:..xi.	1474	Computes the inverse matrix times a vector or matrix for square non-singular matrices.
OPERATORS:.xt.	1471	Computes matrix-vector and matrix-matrix products.
ORTH	1488	Orthogonalizes the columns of a rank-2 or rank-3 array.
PCGRC	359	Solves a real symmetric definite linear system using a preconditioned conjugate gradient method with reverse communication.
PARALLEL_NONNEGATIVE_LSQ	67	Solves a linear, non-negative constrained least-squares system.
PARALLEL_BOUNDED_LSQ	75	Solves a linear least-squares system with bounds on the unknowns.
PDE_1D_MG	913	Method of lines with Variable Griddings.

PERMA	1602	Permutes the rows or columns of a matrix.
PERMU	1600	Rearranges the elements of an array as specified by a permutation.
PGOPT	1599	Sets or retrieves page width and length for printing.
PLOTP	1664	Prints a plot of up to 10 sets of points.
POLRG	1429	Evaluates a real general matrix polynomial.
PP1GD	687	Evaluates the derivative of a piecewise polynomial on a grid.
PPDER	684	Evaluates the derivative of a piecewise polynomial.
PPITG	690	Evaluates the integral of a piecewise polynomial.
PPVAL	681	Evaluates a piecewise polynomial.
PRIME	1668	Decomposes an integer into its prime factors.
QAND	806	Integrates a function on a hyper-rectangle.
QCOSB	1041	Computes a sequence from its cosine Fourier coefficients with only odd wave numbers.
QCOSF	1039	Computes the coefficients of the cosine Fourier transform with only odd wave numbers.
QCOSI	1043	Computes parameters needed by QCOSF and QCOSB.
QD2DR	699	Evaluates the derivative of a function defined on a rectangular grid using quadratic interpolation.
QD2VL	696	Evaluates a function defined on a rectangular grid using quadratic interpolation.
QD3DR	705	Evaluates the derivative of a function defined on a rectangular three-dimensional grid using quadratic interpolation.
QD3VL	702	Evaluates a function defined on a rectangular three-dimensional grid using quadratic interpolation.
QDAG	775	Integrates a function using a globally adaptive scheme based on Gauss-Kronrod rules.
QDAGI	782	Integrates a function over an infinite or semi-infinite interval.
QDAGP	779	Integrates a function with singularity points given.
QDAGS	772	Integrates a function (which may have endpoint singularities).
QDAWC	796	Integrates a function F(X)/(X – C) in the Cauchy principal value sense.
QDAWF	789	Computes a Fourier integral.

QDAWO	785	Integrates a function containing a sine or a cosine.
QDAWS	793	Integrates a function with algebraic-logarithmic singularities.
QDDER	694	Evaluates the derivative of a function defined on a set of points using quadratic interpolation.
QDNG	799	Integrates a smooth function using a nonadaptive rule.
QDVAL	692	Evaluates a function defined on a set of points using quadratic interpolation.
QMC	809	Integrates a function over a hyperrectangle using a quasi-Monte Carlo method.
QPROG	1307	Solves a quadratic programming problem subject to linear equality/inequality constraints.
QSINB	1034	Computes a sequence from its sine Fourier coefficients with only odd wave numbers.
QSINF	1032	Computes the coefficients of the sine Fourier transform with only odd wave numbers.
QSINI	1037	Computes parameters needed by `QSINF` and `QSINB`.
RAND	1489	Computes a scalar, rank-1, rank-2 or rank-3 array of random numbers.
RAND_GEN	1639	Generates a rank-1 array of random numbers.
RANK	1490	Computes the mathematical rank of a rank-2 or rank-3 array.
RATCH	764	Computes a rational weighted Chebyshev approximation to a continuous function on an interval.
RCONV	1059	Computes the convolution of two real vectors.
RCORL	1068	Computes the correlation of two real vectors.
RCURV	716	Fits a polynomial curve using least squares.
RECCF	818	Computes recurrence coefficients for various monic polynomials.
RECQR	821	Computes recurrence coefficients for monic polynomials given a quadrature rule.
RLINE	713	Fits a line to a set of data points using least squares.
RNGET	1648	Retrieves the current value of the seed used in the IMSL random number generators.
RNOPT	1650	Selects the uniform (0, 1) multiplicative congruential pseudorandom number generator.
RNSET	1649	Initializes a random seed for use in the IMSL random number generators.

RNUN	1653	Generates pseudorandom numbers from a uniform (0, 1) distribution.
RNUNF	1651	Generates a pseudorandom number from a uniform (0, 1) distribution.
SADD	1370	Adds a scalar to each component of a vector, $x \leftarrow x + a$, all single precision.
SASUM	1373	Sums the absolute values of the components of a single-precision vector.
SAXPY	1370	Computes the scalar times a vector plus a vector, $y \leftarrow ax + y$, all single precision.
ScaLaPACK_READ	1545	Reads matrix data from a file and transmits it into the two-dimensional block-cyclic form required by *ScaLAPACK* routines.
ScaLaPACK_WRITE	1547	Writes the matrix data to a file.
SCASUM	1322	Sums the absolute values of the real part together with the absolute values of the imaginary part of the components of a complex vector.
SCNRM2	1322	Computes the Euclidean norm of a complex vector.
SCOPY	1369	Copies a vector x to a vector y, both single precision.
SDDOTA	1321	Computes the sum of a single-precision scalar, a single-precision dot product and the double-precision accumulator, which is set to the result ACC $\leftarrow$ ACC $+ a + x^T y$.
SDDOTI	1372	Computes the sum of a single-precision scalar plus a singleprecision dot product using a double-precision accumulator, which is set to the result ACC $\leftarrow a + x^T y$.
SDOT	1370	Computes the single-precision dot product $x^T y$.
SDSDOT	1371	Computes the sum of a single-precision scalar and a single precision dot product, $a + x^T y$, using a double-precision accumulator.
SGBMV	1381	Computes one of the matrix-vector operations: $y \leftarrow \alpha Ax + \beta y,$ or $y \leftarrow \alpha A^T x + \beta y$, where A is a matrix stored in band storage mode.
SGEMM	1385	Computes one of the matrix-matrix operations: $C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha A^T B + \beta C, C \leftarrow \alpha AB^T + \beta C$, or $C \leftarrow \alpha A^T B^T + \beta C$

SGEMV	1381	Computes one of the matrix-vector operations: $y \leftarrow \alpha Ax + \beta y$, or $y \leftarrow \alpha A^T x + \beta y$,
SGER	1383	Computes the rank-one update of a real general matrix: $A \leftarrow A + \alpha xy^T$.
SHOW	1571	Prints rank-1 or rank-2 arrays of numbers in a readable format.
SHPROD	1372	Computes the Hadamard product of two single-precision vectors.
SINLP	1081	Computes the inverse Laplace transform of a complex function.
SLCNT	986	Calculates the indices of eigenvalues of a Sturm-Liouville problem with boundary conditions (at regular points) in a specified subinterval of the real line, [α, β].
SLEIG	973	Determines eigenvalues, eigenfunctions and/or spectral density functions for Sturm-Liouville problems in the form with boundary conditions (at regular points).
SLPRS	1301	Solves a sparse linear programming problem via the revised simplex algorithm.
SNRM2	1373	Computes the Euclidean length or L_2 norm of a single-precision vector.
SORT_REAL	1604	Sorts a rank-1 array of real numbers x so the y results are algebraically nondecreasing, $y_1 \le y_2 \le \dots y_n$.
SPLEZ	618	Computes the values of a spline that either interpolates or fits user-supplied data.
SPLINE_CONSTRAINTS	562	Returns the derived type array result.
SPLINE_FITTING	564	Weighted least-squares fitting by B-splines to discrete One-Dimensional data is performed.
SPLINE_VALUES	563	Returns an array result, given an array of input
SPRDCT	1373	Multiplies the components of a single-precision vector.
SRCH	1618	Searches a sorted vector for a given scalar and return its index.
SROT	1375	Applies a Givens plane rotation in single precision.
SROTG	1374	Constructs a Givens plane rotation in single precision.
SROTM	1377	Applies a modified Givens plane rotation in single precision.
SROTMG	1376	Constructs a modified Givens plane rotation in single precision.

SSBMV	1382	Computes the matrix-vector operation $y \leftarrow \alpha Ax + \beta y$, where A is a symmetric matrix in band symmetric storage mode.
SSCAL	1369	Multiplies a vector by a scalar, $y \leftarrow ay$, both single precision.
SSET	1369	Sets the components of a vector to a scalar, all single precision.
SSRCH	1622	Searches a character vector, sorted in ascending ASCII order, for a given string and return its index.
SSUB	1370	Subtracts each component of a vector from a scalar, $x \leftarrow a - x$, all single precision.
SSUM	1372	Sums the values of a single-precision vector.
SSWAP	1370	Interchanges vectors x and y, both single precision.
SSYMM	1385	Computes one of the matrix-matrix operations: $C \leftarrow \alpha AB + \beta C$ or $C \leftarrow \alpha BA + \beta C$, where A is a symmetric matrix and B and C are m by n matrices.
SSYMV	1382	Computes the matrix-vector operation $y \leftarrow \alpha Ax + \beta y$, where A is a symmetric matrix.
SSYR	1384	Computes the rank-one update of a real symmetric matrix: $A \leftarrow A + \alpha xx^T$.
SSYR2	1384	Computes the rank-two update of a real symmetric matrix: $A \leftarrow A + \alpha xy^T + \alpha yx^T$.
SSYR2K	1386	Computes one of the symmetric rank $2k$ operations: $C \leftarrow \alpha AB^T + \alpha BA^T + \beta C$ or $C \leftarrow \alpha A^T B + \alpha B^T A + \beta C$, where C is an n by n symmetric matrix and A and B are n by k matrices in the first case and k by n matrices in the second case.
SSYRK	1386	Computes one of the symmetric rank k operations: $C \leftarrow \alpha AA^T + \beta C$ or $C \leftarrow \alpha A^T A + \beta C$, where C is an n by n symmetric matrix and A is an n by k matrix in the first case and a k by n matrix in the second case.
STBMV	1382	Computes one of the matrix-vector operations: $x \leftarrow Ax$ or $x \leftarrow A^T x$, where A is a triangular matrix in band storage mode.

STBSV	1383	Solves one of the triangular systems: $x \leftarrow A^{-1}x$ or $x \leftarrow \left(A^{-1}\right)^T x$, where A is a triangular matrix in band storage mode.
STRMM	1387	Computes one of the matrix-matrix operations: $B \leftarrow \alpha AB, B \leftarrow \alpha A^T B$ or $B \leftarrow \alpha BA, B \leftarrow \alpha BA^T$, where B is an m by n matrix and A is a triangular matrix.
STRMV	1382	Computes one of the matrix-vector operations: $x \leftarrow Ax$ or $x \leftarrow A^T x$, where A is a triangular matrix.
STRSM	1387	Solves one of the matrix equations: $B \leftarrow \alpha A^{-1}B, B \leftarrow \alpha BA^{-1}, B \leftarrow \alpha\left(A^{-1}\right)^T B$, or $B \leftarrow \alpha B\left(A^{-1}\right)^T$ where B is an m by n matrix and A is a triangular matrix.
STRSV	1383	Solves one of the triangular linear systems: $x \leftarrow A^{-1}x$ or $x \leftarrow \left(A^{-1}\right)^T x$ where A is a triangular matrix.
SUMAG	1664	Sets or retrieves MATH/LIBRARY single-precision options.
SURF	710	Computes a smooth bivariate interpolant to scattered data that is locally a quintic polynomial in two variables.
SURFACE_CONSTRAINTS	574	Returns the derived type array result given optional input.
SURFACE_FITTING	577	Weighted least-squares fitting by tensor product B-splines to discrete two-dimensional data is performed.
SURFACE_VALUES	575	Returns a tensor product array result, given two arrays of independent variable values.
SVCAL	1369	Multiplies a vector by a scalar and store the result in another vector, $y \leftarrow ax$, all single precision.
SVD	1491	Computes the singular value decomposition of a rank-2 or rank-3 array, $A = USV^T$.
SVIBN	1615	Sorts an integer array by nondecreasing absolute value.
SVIBP	1617	Sorts an integer array by nondecreasing absolute value and returns the permutation that rearranges the array.
SVIGN	1610	Sorts an integer array by algebraically increasing value.

SVIGP	1611	Sorts an integer array by algebraically increasing value and returns the permutation that rearranges the array.
SVRBN	1612	Sorts a real array by nondecreasing absolute value.
SVRBP	1614	Sorts a real array by nondecreasing absolute value and returns the permutation that rearranges the array.
SVRGN	1607	Sorts a real array by algebraically increasing value.
SVRGP	1608	Sorts a real array by algebraically increasing value and returns the permutation that rearranges the array.
SXYZ	1372	Computes a single-precision *xyz* product.
TDATE	1633	Gets today's date.
TIMDY	1632	Gets time of day.
TRNRR	1413	Transposes a rectangular matrix.
TWODQ	801	Computes a two-dimensional iterated integral.
UMACH	1688	Sets or retrieves input or output device unit numbers.
UMAG	1661	Handles MATH/LIBRARY and STAT/LIBRARY type `REAL` and double precision options.
UMCGF	1219	Minimizes a function of `N` variables using a conjugate gradient algorithm and a finite-difference gradient.
UMCGG	1223	Minimizes a function of `N` variables using a conjugate gradient algorithm and a user-supplied gradient.
UMIAH	1213	Minimizes a function of `N` variables using a modified Newton method and a user-supplied Hessian.
UMIDH	1208	Minimizes a function of `N` variables using a modified Newton method and a finite-difference Hessian.
UMINF	1196	Minimizes a function of `N` variables using a quasi-New method and a finite-difference gradient.
UMING	1202	Minimizes a function of `N` variables using a quasi-New method and a user-supplied gradient.
UMPOL	1227	Minimizes a function of `N` variables using a direct search polytope algorithm.
UNIT	1492	Normalizes the columns of a rank-2 or rank-3 array so each has Euclidean length of value one.
UNLSF	1231	Solves a nonlinear least squares problem using a modified Levenberg-Marquardt algorithm and a finite-difference Jacobian.
UNLSJ	1237	Solves a nonlinear least squares problem using a modified Levenberg-Marquardt algorithm and a user-supplied Jacobian.

UVMGS	1193	Finds the minimum point of a nonsmooth function of a single variable.
UVMID	1189	Finds the minimum point of a smooth function of a single variable using both function evaluations and first derivative evaluations.
UVMIF	1186	Finds the minimum point of a smooth function of a single variable using only function evaluations.
VCONC	1457	Computes the convolution of two complex vectors.
VCONR	1455	Computes the convolution of two real vectors.
VERML	1638	Obtains IMSL MATH/LIBRARY-related version, system and license numbers.
WRCRL	1588	Prints a complex rectangular matrix with a given format and labels.
WRCRN	1586	Prints a complex rectangular matrix with integer row and column labels.
WRIRL	1583	Prints an integer rectangular matrix with a given format and labels.
WRIRN	1581	Prints an integer rectangular matrix with integer row and column labels.
WROPT	1591	Sets or retrieves an option for printing a matrix.
WRRRL	1577	Prints a real rectangular matrix with a given format and labels.
WRRRN	1575	Prints a real rectangular matrix with integer row and column labels.
ZANLY	1153	Finds the zeros of a univariate complex function using Müller’s method.
ZBREN	1156	Finds a zero of a real function that changes sign in a given interval.
ZPLRC	1148	Finds the zeros of a polynomial with real coefficients using Laguerre’s method.
ZPOCC	1152	Finds the zeros of a polynomial with complex coefficients using the Jenkins-Traub three-stage algorithm.
ZPORC	1150	Finds the zeros of a polynomial with real coefficients using the Jenkins-Traub three-stage algorithm.
ZQADD	1460	Adds a double complex scalar to the accumulator in extended precision.
ZQINI	1460	Initializes an extended-precision complex accumulator to a double complex scalar.

ZQMUL	1460	Multiplies double complex scalars using extended precision.
ZQSTO	1460	Stores a double complex approximation to an extended-precision complex scalar.
ZREAL	1159	Finds the real zeros of a real function using Müller’s method.

Appendix C: References

Aird and Howell

Aird, Thomas J., and Byron W. Howell (1991), IMSL Technical Report 9103, IMSL, Houston.

Aird and Rice

Aird, T.J., and J.R. Rice (1977), Systematic search in high dimensional sets, *SIAM Journal on Numerical Analysis*, **14**, 296–312.

Akima

Akima, H. (1970), A new method of interpolation and smooth curve fitting based on local procedures, *Journal of the ACM*, **17**, 589−602.

Akima, H. (1978), A method of bivariate interpolation and smooth surface fitting for irregularly distributed data points, *ACM Transactions on Mathematical Software*, **4**, 148−159.

Arushanian et al.

Arushanian, O.B., M.K. Samarin, V.V. Voevodin, E.E. Tyrtyshikov, B.S. Garbow, J.M. Boyle, W.R. Cowell, and K.W. Dritz (1983), *The TOEPLITZ Package Users' Guide*, Argonne National Laboratory, Argonne, Illinois.

Ashcraft

Ashcraft, C. (1987), *A vector implementation of the multifrontal method for large sparse, symmetric positive definite linear systems*, Technical Report ETA-TR-51, Engineering Technology Applications Division, Boeing Computer Services, Seattle, Washington.

Ashcraft et al.

Ashcraft, C., R.Grimes, J. Lewis, B. Peyton, and H. Simon (1987), Progress in sparse matrix methods for large linear systems on vector supercomputers. *Intern. J. Supercomputer Applic.*, **1**(**4**), 10–29.

Atkinson

Atkinson, Ken (1978), *An Introduction to Numerical Analysis*, John Wiley & Sons, New York.

Atchison and Hanson

Atchison, M.A., and R.J. Hanson (1991), *An Options Manager for the IMSL Fortran 77 Libraries*, Technical Report 9101, IMSL, Houston.

Bischof et al.

Bischof, C., J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, D. Sorensen (1988), LAPACK Working Note #5: Provisional Contents, Argonne National Laboratory Report ANL-88-38, Mathematics and Computer Science.

Bjorck

Bjorck, Ake (1967), Iterative refinement of linear least squares solutions I, BIT, **7**, 322–337.

Bjorck, Ake (1968), Iterative refinement of linear least squares solutions II, BIT, **8**, 8–30.

Boisvert (1984)

Boisvert, Ronald (1984), A fourth order accurate fast direct method for the Helmholtz equation, *Elliptic Problem Solvers II*, (edited by G. Birkhoff and A. Schoenstadt), Academic Press, Orlando, Florida, 35–44.

Boisvert, Howe, and Kahaner

Boisvert, Ronald F., Sally E. Howe, and David K. Kahaner (1985), GAMS: A framework for the management of scientific software, *ACM Transactions on Mathematical Software*, **11**, 313–355.

Boisvert, Howe, Kahaner, and Springmann

Boisvert, Ronald F., Sally E. Howe, David K. Kahaner, and Jeanne L. Springmann (1990), *Guide to Available Mathematical Software*, NISTIR 90-4237, National Institute of Standards and Technology, Gaithersburg, Maryland.

Brankin et al.

Brankin, R.W., I. Gladwell, and L.F. Shampine, RKSUITE: a Suite of Runge-Kutta Codes for the Initial Value Problem for ODEs, Softreport 91-1, Mathematics Department, Southern Methodist University, Dallas, Texas, 1991.

Brenan, Campbell, and Petzold

Brenan, K.E., S.L. Campbell, L.R. Petzold (1989), *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, Elseview Science Publ. Co.

Brenner

Brenner, N. (1973), Algorithm 467: Matrix transposition in place [F1], *Communication of ACM*, **16**, 692–694.

Brent

Brent, R.P. (1971), An algorithm with guaranteed convergence for finding a zero of a function, *The Computer Journal*, **14**, 422–425.

Brent, Richard P. (1973), *Algorithms for Minimization without Derivatives*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Brigham

Brigham, E. Oran (1974), *The Fast Fourier Transform*, Prentice-Hall, Englewood Cliffs, New Jersey.

Cheney

Cheney, E.W. (1966), *Introduction to Approximation Theory*, McGraw-Hill, New York.

Cline et al.

Cline, A.K., C.B. Moler, G.W. Stewart, and J.H. Wilkinson (1979), An estimate for the condition number of a matrix, *SIAM Journal of Numerical Analysis*, **16**, 368−375.

Cody, Fraser, and Hart

Cody, W.J., W. Fraser, and J.F. Hart (1968), Rational Chebyshev approximation using linear equations, *Numerische Mathematik*, **12**, 242–251.

Cohen and Taylor

Cohen, E. Richard, and Barry N. Taylor (1986), *The 1986 Adjustment of the Fundamental Physical Constants*, Codata Bulletin, Pergamon Press, New York.

Cooley and Tukey

Cooley, J.W., and J.W. Tukey (1965), An algorithm for the machine computation of complex Fourier series, *Mathematics of Computation*, **19**, 297–301.

Courant and Hilbert

Courant, R., and D. Hilbert (1962), *Methods of Mathematical Physics, Volume II*, John Wiley & Sons, New York, NY.

Craven and Wahba

Craven, Peter, and Grace Wahba (1979), Smoothing noisy data with spline functions, *Numerische Mathematik*, **31**, 377–403.

Crowe et al.

Crowe, Keith, Yuan-An Fan, Jing Li, Dale Neaderhouser, and Phil Smith (1990), *A direct sparse linear equation solver using linked list storage*, IMSL Technical Report 9006, IMSL, Houston.

Crump

Crump, Kenny S. (1976), Numerical inversion of Laplace transforms using a Fourier series approximation, *Journal of the Association for Computing Machinery*, **23**, 89–96.

Davis and Rabinowitz

Davis, Philip F., and Philip Rabinowitz (1984), *Methods of Numerical Integration*, Academic Press, Orlando, Florida.

de Boor

de Boor, Carl (1978), *A Practical Guide to Splines*, Springer-Verlag, New York.

de Hoog, Knight, and Stokes

de Hoog, F.R., J.H. Knight, and A.N. Stokes (1982), An improved method for numerical inversion of Laplace transforms. *SIAM Journal on Scientific and Statistical Computing*, **3**, 357–366.

Dennis and Schnabel

Dennis, J.E., Jr., and Robert B. Schnabel (1983), *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, New Jersey.

Dongarra et al.

Dongarra, J.J., and C.B. Moler, (1977) *EISPACK – A package for solving matrix eigenvalue problems*, Argonne National Laboratory, Argonne, Illinois.

Dongarra, J.J., J.R. Bunch, C.B. Moler, and G.W. Stewart (1979), *LINPACK Users' Guide*, SIAM, Philadelphia.

Dongarra, J.J., J. DuCroz, S. Hammarling, R. J. Hanson (1988), An Extended Set of Fortran basic linear algebra subprograms, *ACM Transactions on Mathematical Software*, **14** , 1–17.

Dongarra, J.J., J. DuCroz, S. Hammarling, I. Duff (1990), A set of level 3 basic linear algebra subprograms, *ACM Transactions on Mathematical Software*, **16** , 1–17.

Draper and Smith

Draper, N.R., and H. Smith (1981), *Applied Regression Analysis*, second edition, John Wiley & Sons, New York.

Du Croz et al.

Du Croz, Jeremy, P. Mayes, G. and Radicati (1990), Factorization of band matrices using Level-3 BLAS, *Proceedings of CONPAR 90 VAPP IV, Lecture Notes in Computer Science*, Springer, Berlin, 222.

Duff and Reid

Duff, I.S., and J.K. Reid (1983), The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Transactions on Mathematical Software*, **9**, 302–325.

Duff, I.S., and J.K. Reid (1984), The multifrontal solution of unsymmetric sets of linear equations. *SIAM Journal on Scientific and Statistical Computing*, **5**, 633−641.

Duff et al.

Duff, I.S., A.M. Erisman, and J.K. Reid (1986), *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford.

Enright and Pryce

Enright, W.H., and J.D. Pryce (1987), Two FORTRAN packages for assessing initial value methods, *ACM Transactions on Mathematical Software*, **13**, 1−22.

Forsythe

Forsythe, G.E. (1957), Generation and use of orthogonal polynomials for fitting data with a digital computer, *SIAM Journal on Applied Mathematics*, **5**, 74−88.

Fox, Hall, and Schryer

Fox, P.A., A.D. Hall, and N.L. Schryer (1978), The PORT mathematical subroutine library, *ACM Transactions on Mathematical Software*, **4**, 104−126.

Garbow

Garbow, B.S. (1978) CALGO Algorithm 535: The QZ algorithm to solve the generalized eigenvalue problem for complex matrices, *ACM Transactions on Mathematical Software*, **4**, 404−410.

Garbow et al.

Garbow, B.S., J.M. Boyle, J.J. Dongarra, and C.B. Moler (1972), *Matrix eigensystem Routines: EISPACK Guide Extension*, Springer-Verlag, New York.

Garbow, B.S., J.M. Boyle, J.J. Dongarra, and C.B. Moler (1977), *Matrix Eigensystem Routines–EISPACK Guide Extension*, Springer-Verlag, New York.

Garbow, B.S., G. Giunta, J.N. Lyness, and A. Murli (1988), Software for an implementation of Weeks' method for the inverse Laplace transform problem, *ACM Transactions of Mathematical Software*, **14**, 163−170.

Gautschi

Gautschi, Walter (1968), Construction of Gauss-Christoffel quadrature formulas, *Mathematics of Computation*, **22**, 251−270.

Gautschi and Milovanofic

Gautschi, Walter, and Gradimir V. Milovanofic (1985), Gaussian quadrature involving Einstein and Fermi functions with an application to summation of series, *Mathematics of Computation*, **44**, 177−190.

Gay

Gay, David M. (1981), Computing optimal locally constrained steps, *SIAM Journal on Scientific and Statistical Computing*, **2**, 186–197.

Gay, David M. (1983), Algorithm 611: Subroutine for unconstrained minimization using a model/trust-region approach, *ACM Transactions on Mathematical Software*, **9**, 503– 524.

Gear

Gear, C.W. (1971), *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, New Jersey.

Gear and Petzold

Gear, C.W., and Linda R. Petzold (1984), ODE methods for the solutions of differential/algebraic equations, *SIAM Journal Numerical Analysis*, **21**, #4, 716.

George and Liu

George, A., and J.W.H. Liu (1981), *Computer Solution of Large Sparse Positive-definite Systems*, Prentice-Hall, Englewood Cliffs, New Jersey.

Gill et al.

Gill, Philip E., and Walter Murray (1976), *Minimization subject to bounds on the variables*, NPL Report NAC 72, National Physical Laboratory, England.

Gill, Philip E., Walter Murray, and Margaret Wright (1981), *Practical Optimization*, Academic Press, New York.

Gill, P.E., W. Murray, M.A. Saunders, and M.H. Wright (1985), Model building and practical aspects of nonlinear programming, in *Computational Mathematical Programming*, (edited by K. Schittkowski), NATO ASI Series, **15**, Springer-Verlag, Berlin, Germany.

Goldfarb and Idnani

Goldfarb, D., and A. Idnani (1983), A numerically stable dual method for solving strictly convex quadratic programs, *Mathematical Programming*, **27**, 1–33.

Golub

Golub, G.H. (1973), Some modified matrix eigenvalue problems, *SIAM Review*, **15**, 318–334.

Golub and Van Loan

Golub, Gene H., and Charles F. Van Loan (1983), *Matrix Computations*, Johns Hopkins University Press, Baltimore, Maryland.

Golub, Gene H., and Charles F. Van Loan (1989), *Matrix Computations*, 2d ed., Johns Hopkins University Press, Baltimore, Maryland.

Golub and Welsch

Golub, G.H., and J.H. Welsch (1969), Calculation of Gaussian quadrature rules, *Mathematics of Computation*, **23**, 221–230.

Gregory and Karney

Gregory, Robert, and David Karney (1969), *A Collection of Matrices for Testing Computational Algorithms*, Wiley-Interscience, John Wiley & Sons, New York.

Griffin and Redish

Griffin, R., and K.A. Redish (1970), Remark on Algorithm 347: An efficient algorithm for sorting with minimal storage, *Communications of the ACM*, **13**, 54.

Grosse

Grosse, Eric (1980), Tensor spline approximation, *Linear Algebra and its Applications*, **34**, 29–41.

Guerra and Tapia

Guerra, V., and R. A. Tapia (1974), *A local procedure for error detection and data smoothing*, MRC Technical Summary Report 1452, Mathematics Research Center, University of Wisconsin, Madison.

Hageman and Young

Hageman, Louis A., and David M.Young (1981), *Applied Iterative Methods*, Academic Press, New York.

Hanson

Hanson, Richard J. (1986), Least squares with bounds and linear constraints, *SIAM Journal Sci. Stat. Computing*, **7**, #3.

Hanson, Richard.J. (1990), *A cyclic reduction solver for the IMSL Mathematics Library*, IMSL Technical Report 9002, IMSL, Houston.

Hanson et al.

Hanson, Richard J., R. Lehoucq, J. Stolle, and A. Belmonte (1990), *Improved performance of certain matrix eigenvalue computations for the IMSL/MATH Library*, IMSL Technical Report 9007, IMSL, Houston.

Hartman

Hartman, Philip (1964) *Ordinary Differential Equations*, John Wiley and Sons, New York, NY.

Hausman

Hausman, Jr., R.F. (1971), *Function Optimization on a Line Segment by Golden Section*, Lawrence Radiation Laboratory, University of California, Livermore.

Hindmarsh

Hindmarsh, A.C. (1974), *GEAR: Ordinary differential equation system solver*, Lawrence Livermore Laboratory Report UCID–30001, Revision 3.

Hull et al.

Hull, T.E., W.H. Enright, and K.R. Jackson (1976), *User's guide for DVERK – A subroutine for solving non-stiff ODEs*, Department of Computer Science Technical Report 100, University of Toronto.

IEEE

ANSI/IEEE Std 754-1985 (1985), *IEEE Standard for Binary Floating-Point Arithmetic*, The IEEE, Inc., New York.

IMSL (1991)

IMSL (1991), IMSL STAT/LIBRARY *User's Manual, Version 2.0*, IMSL, Houston.

Irvine et al.

Irvine, Larry D., Samuel P. Marin, and Philip W. Smith (1986), Constrained interpolation and smoothing, *Constructive Approximation*, **2**, 129–151.

Jenkins

Jenkins, M.A. (1975), Algorithm 493: Zeros of a real polynomial, *ACM Transactions on Mathematical Software*, **1**, 178–189.

Jenkins and Traub

Jenkins, M.A., and J.F. Traub (1970), A three-stage algorithm for real polynomials using quadratic iteration, *SIAM Journal on Numerical Analysis*, **7**, 545–566.

Jenkins, M.A., and J.F. Traub (1970), A three-stage variable-shift iteration for polynomial zeros and its relation to generalized Rayleigh iteration, *Numerische Mathematik*, **14**, 252–263.

Jenkins, M.A., and J.F. Traub (1972), Zeros of a complex polynomial, *Communications of the ACM*, **15**, 97–99.

Kennedy and Gentle

Kennedy, William J., Jr., and James E. Gentle (1980), *Statistical Computing*, Marcel Dekker, New York.

Kershaw

Kershaw, D. (1982), Solution of tridiagonal linear systems and vectorization of the ICCG algorithm on the Cray-1, *Parallel Computations*, Academic Press, Inc., 85-99.

Knuth

Knuth, Donald E. (1973), *The Art of Computer Programming*, Volume 3: *Sorting and Searching*, Addison-Wesley Publishing Company, Reading, Mass.

Lawson et al.

Lawson, C.L., R.J. Hanson, D.R. Kincaid, and F.T. Krogh (1979), Basic linear algebra subprograms for Fortran usage, *ACM Transactions on Mathematical Software*, **5**, 308− 323.

Leavenworth

Leavenworth, B. (1960), Algorithm 25: Real zeros of an arbitrary function, *Communications of the ACM*, **3**, 602.

Levenberg

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

Lewis et al.

Lewis, P.A. W., A.S. Goodman, and J.M. Miller (1969), A pseudo-random number generator for the System/360, *IBM Systems Journa*l, **8**, 136−146.

Liepman

Liepman, David S. (1964), Mathematical constants, in *Handbook of Mathematical Functions*, Dover Publications, New York.

Liu

Liu, J.W.H. (1986), On the storage requirement in the out-of-core multifrontal method for sparse factorization. *ACM Transactions on Mathematical Software*, **12**, 249–264.

Liu, J.W.H. (1987), *A collection of routines for an implementation of the multifrontal method*, Technical Report CS-87-10, Department of Computer Science, York University, North York, Ontario, Canada.

Liu, J.W.H. (1989), The multifrontal method and paging in sparse Cholesky factorization. *ACM Transactions on Mathematical Software*, **15**, 310–325.

Liu, J.W.H. (1990), The multifrontal method for sparse matrix solution: theory and practice, Technical Report CS-90-04, Department of Computer Science, York University, North York, Ontario, Canada.

Liu and Ashcraft

Liu, J., and C. Ashcraft (1987), *A vector implementation of the multifrontal method for large sparse, symmetric positive definite linear systems*, Technical Report ETA-TR-51, Engineering Technology Applications Division, Boeing Computer Services, Seattle, Washington.

Lyness and Giunta

Lyness, J.N. and G. Giunta (1986), A modification of the Weeks Method for numerical inversion of the Laplace transform, *Mathmetics of Computation*, **47**, 313–322.

Madsen and Sincovec

Madsen, N.K., and R.F. Sincovec (1979), Algorithm 540: PDECOL, General collocation software for partial differential equations, *ACM Transactions on Mathematical Software*, **5**, #3, 326-351.

Marquardt

Marquardt, D. (1963), An algorithm for least-squares estimation of nonlinear parameters, *SIAM Journal on Applied Mathematics*, **11**, 431–441.

Martin and Wilkinson

Martin, R.S., and J.W. Wilkinson (1968), Reduction of the symmetric eigenproblem $Ax = \lambda Bx$ and related problems to standard form, *Numerische Mathematik*, **11**, 99–119.

Micchelli et al.

Micchelli, C.A., T.J. Rivlin, and S. Winograd (1976), The optimal recovery of smooth functions, *Numerische Mathematik*, **26**, 279–285

Micchelli, C.A., Philip W. Smith, John Swetits, and Joseph D. Ward (1985), Constrained L_p approximation, *Constructive Approximation*, **1**, 93–102.

Moler and Stewart

Moler, C., and G.W. Stewart (1973), An algorithm for generalized matrix eigenvalue problems, *SIAM Journal on Numerical Analysis*, **10**, 241–256.

More et al.

More, Jorge, Burton Garbow, and Kenneth Hillstrom (1980), *User guide for MINPACK-1*, Argonne National Labs Report ANL-80-74, Argonne, Illinois.

Muller

Muller, D.E. (1956), A method for solving algebraic equations using an automatic computer, *Mathematical Tables and Aids to Computation*, **10**, 208–215.

Murtagh

Murtagh, Bruce A. (1981), *Advanced Linear Programming: Computation and Practice*, McGraw-Hill, New York.

Murty

Murty, Katta G. (1983), *Linear Programming*, John Wiley and Sons, New York.

Nelder and Mead

Nelder, J.A., and R. Mead (1965), A simplex method for function minimization, *Computer Journa*l **7**, 308–313.

Neter and Wasserman

Neter, John, and William Wasserman (1974), *Applied Linear Statistical Models*, Richard D. Irwin, Homewood, Ill.

Park and Miller

Park, Stephen K., and Keith W. Miller (1988), Random number generators: good ones are hard to find, *Communications of the ACM*, **31**, 1192–1201.

Parlett

Parlett, B.N. (1980), *The Symmetric Eigenvalue Problem*, Prentice–Hall, Inc., Englewood Cliffs, New Jersey.

Pereyra

Pereyra, Victor (1978), PASVA3: An adaptive finite-difference FORTRAN program for first order nonlinear boundary value problems, in *Lecture Notes in Computer Science*, **76**, Springer-Verlag, Berlin, 67–88.

Petro

Petro, R. (1970), Remark on Algorithm 347: An efficient algorithm for sorting with minimal storage, *Communications of the ACM*, **13**, 624.

Petzold

Petzold, L.R. (1982), A description of DASSL: A differential/ algebraic system solver, *Proceedings of the IMACS World Congress*, Montreal, Canada.

Piessens et al.

Piessens, R., E. deDoncker-Kapenga, C.W. Uberhuber, and D.K. Kahaner (1983), *QUADPACK*, Springer-Verlag, New York.

Powell

Powell, M.J.D. (1977), Restart procedures for the conjugate gradient method, *Mathematical Programming*, **12**, 241–254.

Powell, M.J.D. (1978), A fast algorithm for nonlinearly constrained optimization calculations, in *Numerical Analysis Proceedings, Dundee 1977, Lecture Notes in Mathematics*, (edited by G.A. Watson), **630**, Springer-Verlag, Berlin, Germany, 144–157.

Powell, M.J.D. (1983), ZQPCVX a FORTRAN *subroutine for convex quadratic programming*, DAMTP Report NA17, Cambridge, England.

Powell, M.J.D. (1985), On the quadratic programming algorithm of Goldfarb and Idnani, *Mathematical Programming Study*, **25**, 46-61.

Powell, M.J.D. (1988), *A tolerant algorithm for linearly constrained optimization calculations*, DAMTP Report NA17, University of Cambridge, England.

Powell, M.J.D. (1989), TOLMIN: *A fortran package for linearly constrained optimization calculations*, DAMTP Report NA2, University of Cambridge, England.

Pruess and Fulton

Pruess, S. and C.T. Fulton (1993), Mathematical Software for Sturm-Liouville Problems, *ACM Transactions on Mathematical Software*, **17**, *3*, 360–376.

Reinsch

Reinsch, Christian H. (1967), Smoothing by spline functions, *Numerische Mathematik*, **10**, 177–183.

Rice

Rice, J.R. (1983), *Numerical Methods, Software, and Analysis*, McGraw-Hill, New York.

Saad and Schultz

Saad, Y., and M.H. Schultz (1986), GMRES: a generalized minimal residual residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Stat. Comput*., **7**, 856–869.

Schittkowski

Schittkowski, K. (1987), *More test examples for nonlinear programming codes*, SpringerVerlag, Berlin, 74.

Schnabel

Schnabel, Robert B. (1985), Finite Difference Derivatives – Theory and Practice, Report, National Bureau of Standards, Boulder, Colorado.

Schreiber and Van Loan

Schreiber, R., and C. Van Loan (1989), A Storage–Efficient *WY* Representation for Products of Householder Transformations, *SIAM J. Sci. Stat. Comp*., Vol. 10, No. 1, pp. 53-57, January (1989).

Scott et al.

Scott, M.R., L.F. Shampine, and G.M. Wing (1969), Invariant Embedding and the Calculation of Eigenvalues for Sturm-Liouville Systems, *Computing*, **4**, 10–23.

Sewell

Sewell, Granville (1982), *IMSL software for differential equations in one space variable*, IMSL Technical Report 8202, IMSL, Houston.

Shampine

Shampine, L.F. (1975), Discrete least-squares polynomial fits, *Communications of the ACM*, **18**, 179–180.

Shampine and Gear

Shampine, L.F. and C.W. Gear (1979), A user's view of solving stiff ordinary differential equations, *SIAM Review*, **21**, 1–17.

Sincovec and Madsen

Sincovec, R.F., and N.K. Madsen (1975), Software for nonlinear partial differential equations, *ACM Transactions on Mathematical Software*, **1**, #3, 232-260.

Singleton

Singleton, R.C. (1969), Algorithm 347: An efficient algorithm for sorting with minimal storage, *Communications of the ACM*, **12**, 185–187.

Smith

Smith, B.T. (1967), *ZERPOL, A Zero Finding Algorithm for Polynomials Using Laguerre's Method*, Department of Computer Science, University of Toronto.

Smith et al.

Smith, B.T., J.M. Boyle, J.J. Dongarra, B.S. Garbow, Y. Ikebe, V.C. Klema, and C.B. Moler (1976), *Matrix Eigensystem Routines – EISPACK Guide*, Springer-Verlag, New York.

Spang

Spang, III, H.A. (1962), A review of minimization techniques for non-linear functions, *SIAM Review*, **4**, 357–359.

Stewart

Stewart, G.W. (1973), *Introduction to Matrix Computations*, Academic Press, New York.

Stewart, G.W. (1976), The economical storage of plane rotations, *Numerische Mathematik*, **25**, 137–139.

Stoer

Stoer, J. (1985), Principles of sequential quadratic programming methods for solving nonlinear programs, in *Computational Mathematical Programming*, (edited by K. Schittkowski), NATO ASI Series, **15**, Springer-Verlag, Berlin, Germany.

Stroud and Secrest

Stroud, A.H., and D.H. Secrest (1963), *Gaussian Quadrature Formulae*, Prentice-Hall, Englewood Cliffs, New Jersey.

Titchmarsh

Titchmarsh, E. *Eigenfunction Expansions Associated with Second Order Differential Equations, Part I*, 2d Ed., Oxford University Press, London, 1962.

Trench

Trench, W.F. (1964), An algorithm for the inversion of finite Toeplitz matrices, J*ournal of the Society for Industrial and Applied Mathematics*, **12**, 515–522.

Walker

Walker, H.F. (1988), Implementation of the GMRES method using Householder transformations, *SIAM J. Sci. Stat. Comput*., **9**, 152–163.

Washizu

Washizu, K. (1968), *Variational Methods in Elasticity and Plasticity*, Pergamon Press, New York.

Watkins and Elsner

Watkins, D.S., and L. Elsner (1990), Convergence of algorithms of decomposition type for the eigenvalue problem, *Linear Algebra and Applications* (to appear).

Weeks

Weeks, W.T. (1966), Numerical inversion of Laplace transforms using Laguerre functions, *J. ACM*, **13**, 419–429.

Wilkinson

Wilkinson, J.H. (1965),*The Algebraic Eigenvalue Problem*, Oxford University Press, London, 635.