nonlin

1

Generated by Doxygen 1.8.11

Contents

Chapter 1

Modules Index

1.1 Modules List

Here is a list of all documented modules with brief descriptions:

Chapter 2

Data Type Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Data Type Index

3.1 Data Types List

Here are the data types with brief descriptions:

Chapter 4

Module Documentation

4.1 nonlin_c_binding Module Reference

nonlin_c_binding

Data Types

- interface cfcn1var
  The C-friendly interface to fcn1var.
- type cfcn1var_helper
  A container allowing the use of cfcn1var in the solver codes.
- interface cjacobianfcn
  The C-friendly interface to jacobianfcn.
- interface cvecfcn
  The C-friendly interface to vecfcn.
- type cvecfcn_helper
  A container allowing the use of cvecfcn in the solver codes.
- type line_search_control
  Defines a set of line search controls.
- type solver_control
  Defines a set of solver control information.

Functions/Subroutines

- real(dp) function cf1h_fcn (this, x)
  Executes the routine containing the function to evaluate.
- pure logical function cf1h_is_fcn_defined (this)
  Tests if the pointer to the function containing the equation to solve has been assigned.
- subroutine cf1h_set_fcn (this, fcn)
  Establishes a pointer to the routine containing the equations to solve.
- subroutine cvfh_set_fcn (this, fcn, nfcn, nvar)
  Establishes a pointer to the routine containing the system of equations to solve.
- pure logical function cvfh_is_fcn_defined (this)
  Tests if the pointer to the subroutine containing the system of equations to solve has been assigned.

- subroutine cvfh_fcn (this, x, f)

  Executes the routine containing the system of equations to solve. No action is taken if the pointer to the subroutine has not been defined.
- subroutine cvfh_set_jac (this, jac)

  Establishes a pointer to the routine for computing the Jacobian matrix of the system of equations. If no routine is defined, the Jacobian matrix will be computed numerically (this is the default state).
- pure logical function cvfh_is_jac_defined (this)

  Tests if the pointer to the subroutine containing the system of equations to solve has been assigned.
- subroutine cvfh_jac_fcn (this, x, jac, fv, work, olwork, err)

  Executes the routine containing the Jacobian matrix if supplied. If not supplied, the Jacobian is computed via finite differences.
- subroutine brent_solver_c (fcn, lim, x, f, tol, ib, err)

  Solves an equation of one variable using Brent's method.
- subroutine quasi_newton_c (fcn, jac, n, x, fvec, tol, lsearch, ib, err)

  Applies the quasi-Newton's method developed by Broyden in conjunction with a backtracking type line search to solve N equations of N unknowns.
- subroutine newton_c (fcn, jac, n, x, fvec, tol, lsearch, ib, err)

  Applies Newton's method in conjunction with a backtracking type line search to solve N equations of N unknowns.
- subroutine levmarq_c (fcn, jac, neqn, nvar, x, fvec, tol, ib, err)

  Applies the Levenberg-Marquardt method to solve the nonlinear least-squares problem.

4.1.1 Detailed Description

nonlin_c_binding

Purpose

Provides C bindings to the nonlin library.

4.1.2 Function/Subroutine Documentation

4.1.2.1 subroutine nonlin_c_binding::brent_solver_c (type(c_funptr), intent(in), value *fcn,* type(value_pair), intent(in), value *lim,* real(dp), intent(out) *x,* real(dp), intent(out) *f,* type(solver_control), intent(in) *tol,* type(iteration_behavior), intent(out) *ib,* type(c_ptr), intent(in), value *err*)

Solves an equation of one variable using Brent's method.

Parameters

in	*fcn*	A pointer to the routine containing the function to solve.
in	*lim*	A value_pair object defining the search limits.
out	*x*	On output, the solution.
out	*f*	On output, the residual as computed at x.
in	*tol*	A solver_control object defining the solver control parameters.
out	*ib*	On output, an iteration_behavior object containing the iteration performance statistics.
in	*err*	A pointer to the C error handler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows. • NL_INVALID_OPERATION_ERROR: Occurs if no equations have been defined. • NL_INVALID_INPUT_ERROR: Occurs if the number of equations is different than the number of variables. • NL_CONVERGENCE_ERROR: Occurs if the algorithm cannot converge within the allowed number of iterations.

Definition at line 435 of file nonlin_c_binding.f90.

4.1.2.2 real(dp) function nonlin_c_binding::cf1h_fcn (class(cfcn1var_helper), intent(in) *this*, real(dp), intent(in) *x*)

Executes the routine containing the function to evaluate.

Parameters

in	*this*	The cfcn1var_helper object.
in	*x*	The value of the independent variable at which the function should be evaluated.

Returns

The value of the function at `x`.

Definition at line 162 of file nonlin_c_binding.f90.

4.1.2.3 pure logical function nonlin_c_binding::cf1h_is_fcn_defined (class(cfcn1var_helper), intent(in) *this*)

Tests if the pointer to the function containing the equation to solve has been assigned.

Parameters

in	*this*	The cfcn1var_helper object.

Returns

Returns true if the pointer has been assigned; else, false.

Definition at line 177 of file nonlin_c_binding.f90.

4.1.2.4 subroutine nonlin_c_binding::cf1h_set_fcn (class(cfcn1var_helper), intent(inout) *this*, procedure(cfcn1var), intent(in), pointer *fcn*)

Establishes a pointer to the routine containing the equations to solve.

Parameters

in,out	*this*	The cfcn1var_helper object.
in	*fcn*	The function pointer.

Definition at line 189 of file nonlin_c_binding.f90.

4.1.2.5 subroutine nonlin_c_binding::cvfh_fcn (class(cvecfcn_helper), intent(in) *this*, real(dp), dimension(:), intent(in) *x*, real(dp), dimension(:), intent(out) *f*)

Executes the routine containing the system of equations to solve. No action is taken if the pointer to the subroutine has not been defined.

Parameters

in	*this*	The cvecfcn_helper object.
in	*x*	An N-element array containing the independent variables.
out	*f*	An M-element array that, on output, contains the values of the M functions.

Definition at line 235 of file nonlin_c_binding.f90.

4.1.2.6 pure logical function nonlin_c_binding::cvfh_is_fcn_defined (class(cvecfcn_helper), intent(in) *this*)

Tests if the pointer to the subroutine containing the system of equations to solve has been assigned.

Parameters

in	*this*	The cvecfcn_helper object.

Returns

Returns true if the pointer has been assigned; else, false.

Definition at line 221 of file nonlin_c_binding.f90.

4.1.2.7 pure logical function nonlin_c_binding::cvfh_is_jac_defined (class(cvecfcn_helper), intent(in) *this*)

Tests if the pointer to the subroutine containing the system of equations to solve has been assigned.

Parameters

in	*this*	The vecfcn_helper object.

Returns

Returns true if the pointer has been assigned; else, false.

Definition at line 266 of file nonlin_c_binding.f90.

4.1.2.8 subroutine nonlin_c_binding::cvfh_jac_fcn (class(cvecfcn_helper), intent(in) *this*, real(dp), dimension(:), intent(inout) *x*, real(dp), dimension(:,:), intent(out) *jac*, real(dp), dimension(:), intent(in), optional, target *fv*, real(dp), dimension(:), intent(out), optional, target *work*, integer(i32), intent(out), optional *olwork*, integer(i32), intent(out), optional *err*)

Executes the routine containing the Jacobian matrix if supplied. If not supplied, the Jacobian is computed via finite differences.

Parameters

in	*this*	The vecfcn_helper object.
in	*x*	An N-element array containing the independent variabls defining the point about which the derivatives will be calculated.
out	*jac*	An M-by-N matrix where, on output, the Jacobian will be written.
in	*fv*	An optional M-element array containing the function values at `x`. If not supplied, the function values are computed at `x`.
out	*work*	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least `olwork`. Notice, a workspace array is only utilized if the user does not provide a routine for computing the Jacobian.
out	*olwork*	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for `work`, and returns without performing any actual calculations.
out	*err*	An optional integer output that can be used to determine error status. If not used, and an error is encountered, the routine simply returns silently. If used, the following error codes identify error status: • 0: No error has occurred. • n: A positive integer denoting the index of an invalid input. • -1: Indicates internal memory allocation failed.

Definition at line 298 of file nonlin_c_binding.f90.

4.1.2.9 subroutine nonlin_c_binding::cvfh_set_fcn (class(cvecfcn_helper), intent(inout) *this*, procedure(cvecfcn), intent(in), pointer *fcn*, integer(i32), intent(in) *nfcn*, integer(i32), intent(in) *nvar*)

Establishes a pointer to the routine containing the system of equations to solve.

Parameters

in,out	*this*	The cvecfcn_helper object.
in	*fcn*	The function pointer.
in	*nfcn*	The number of functions.
in	*nvar*	The number of variables.

Definition at line 205 of file nonlin_c_binding.f90.

4.1.2.10 subroutine nonlin_c_binding::cvfh_set_jac (class(cvecfcn_helper), intent(inout) *this*, procedure(cjacobianfcn), intent(in), pointer *jac*)

Establishes a pointer to the routine for computing the Jacobian matrix of the system of equations. If no routine is defined, the Jacobian matrix will be computed numerically (this is the default state).

Parameters

in,out	*this*	The cvecfcn_helper object.
in	*jac*	The function pointer.

Definition at line 254 of file nonlin_c_binding.f90.

4.1.2.11 subroutine nonlin_c_binding::levmarq_c (type(c_funptr), intent(in), value *fcn,* type(c_funptr), intent(in), value *jac,* integer(i32), intent(in), value *neqn,* integer(i32), intent(in), value *nvar,* real(dp), dimension(nvar), intent(inout) *x,* real(dp), dimension(neqn), intent(out) *fvec,* type(solver_control), intent(in) *tol,* type(iteration_behavior), intent(out) *ib,* type(c_ptr), intent(in), value *err*)

Applies the Levenberg-Marquardt method to solve the nonlinear least-squares problem.

Parameters

in	*fcn*	A pointer to the routine containing the system of equations to solve.
in	*jac*	A pointer to a routine used to compute the Jacobian of the system of equations. To let the program compute the Jacobian numerically, simply pass NULL.
in	*neqn*	The number of equations.
in	*nvar*	The number of unknowns. This must be less than or equal to `neqn`.
in,out	*x*	On input, an N-element array containing an initial estimate to the solution. On output, the updated solution estimate. N is the number of variables.
out	*fvec*	An N-element array that, on output, will contain the values of each equation as evaluated at the variable values given in `x`.
in	*tol*	A solver_control object defining the solver control parameters.
out	*ib*	On output, an iteration_behavior object containing the iteration performance statistics.
in	*err*	A pointer to the C error handler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows. • NL_INVALID_OPERATION_ERROR: Occurs if no equations have been defined. • NL_INVALID_INPUT_ERROR: Occurs if the number of equations is less than than the number of variables. • NL_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized correctly. • NL_CONVERGENCE_ERROR: Occurs if the line search cannot converge within the allowed number of iterations. • NL_OUT_OF_MEMORY_ERROR: Occurs if there is insufficient memory available. • NL_TOLERANCE_TOO_SMALL_ERROR: Occurs if the requested tolerance is to small to be practical for the problem at hand.

Definition at line 696 of file nonlin_c_binding.f90.

4.1.2.12 subroutine nonlin_c_binding::newton_c (type(c_funptr), intent(in), value *fcn,* type(c_funptr), intent(in), value *jac,* integer(i32), intent(in), value *n,* real(dp), dimension(n), intent(inout) *x,* real(dp), dimension(n), intent(out) *fvec,* type(solver_control), intent(in) *tol,* type(c_ptr), intent(in), value *lsearch,* type(iteration_behavior), intent(out) *ib,* type(c_ptr), intent(in), value *err*)

Applies Newton's method in conjunction with a backtracking type line search to solve N equations of N unknowns.

Parameters

in	*fcn*	A pointer to the routine containing the system of equations to solve.

Parameters

in	*jac*	A pointer to a routine used to compute the Jacobian of the system of equations. To let the program compute the Jacobian numerically, simply pass NULL.
in	*n*	The number of equations, and the number of unknowns.
in,out	*x*	On input, an N-element array containing an initial estimate to the solution. On output, the updated solution estimate. N is the number of variables.
out	*fvec*	An N-element array that, on output, will contain the values of each equation as evaluated at the variable values given in `x`.
in	*tol*	A solver_control object defining the solver control parameters.
in	*lsearch*	A pointer to a line_search_control object defining the line search control parameters. If no line search is desired, simply pass NULL.
out	*ib*	On output, an iteration_behavior object containing the iteration performance statistics.
in	*err*	A pointer to the C error handler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows. • NL_INVALID_OPERATION_ERROR: Occurs if no equations have been defined. • NL_INVALID_INPUT_ERROR: Occurs if the number of equations is different than the number of variables. • NL_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized correctly. • NL_DIVERGENT_BEHAVIOR_ERROR: Occurs if the direction vector is pointing in an apparent uphill direction. • NL_CONVERGENCE_ERROR: Occurs if the line search cannot converge within the allowed number of iterations. • NL_OUT_OF_MEMORY_ERROR: Occurs if there is insufficient memory available. • NL_SPURIOUS_CONVERGENCE_ERROR: Occurs as a warning if the slope of the gradient vector becomes sufficiently close to zero.

Definition at line 604 of file nonlin_c_binding.f90.

4.1.2.13 subroutine nonlin_c_binding::quasi_newton_c (type(c_funptr), intent(in), value *fcn*, type(c_funptr), intent(in), value *jac*, integer(i32), intent(in), value *n*, real(dp), dimension(n), intent(inout) *x*, real(dp), dimension(n), intent(out) *fvec*, type(**solver_control**), intent(in) *tol*, type(c_ptr), intent(in), value *lsearch*, type(**iteration_behavior**), intent(out) *ib*, type(c_ptr), intent(in), value *err*)

Applies the quasi-Newton's method developed by Broyden in conjunction with a backtracking type line search to solve N equations of N unknowns.

Parameters

in	*fcn*	A pointer to the routine containing the system of equations to solve.
in	*jac*	A pointer to a routine used to compute the Jacobian of the system of equations. To let the program compute the Jacobian numerically, simply pass NULL.
in	*n*	The number of equations, and the number of unknowns.
in,out	*x*	On input, an N-element array containing an initial estimate to the solution. On output, the updated solution estimate. N is the number of variables.

Parameters

out	*fvec*	An N-element array that, on output, will contain the values of each equation as evaluated at the variable values given in `x`.
in	*tol*	A solver_control object defining the solver control parameters.
in	*lsearch*	A pointer to a line_search_control object defining the line search control parameters. If no line search is desired, simply pass NULL.
out	*ib*	On output, an iteration_behavior object containing the iteration performance statistics.
in	*err*	A pointer to the C error handler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows. • NL_INVALID_OPERATION_ERROR: Occurs if no equations have been defined. • NL_INVALID_INPUT_ERROR: Occurs if the number of equations is different than the number of variables. • NL_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized correctly. • NL_DIVERGENT_BEHAVIOR_ERROR: Occurs if the direction vector is pointing in an apparent uphill direction. • NL_CONVERGENCE_ERROR: Occurs if the line search cannot converge within the allowed number of iterations. • NL_OUT_OF_MEMORY_ERROR: Occurs if there is insufficient memory available. • NL_SPURIOUS_CONVERGENCE_ERROR: Occurs as a warning if the slope of the gradient vector becomes sufficiently close to zero.

Definition at line 509 of file nonlin_c_binding.f90.

4.2 nonlin_least_squares Module Reference

nonlin_least_squares

Data Types

- type least_squares_solver

  Defines a Levenberg-Marquardt based solver for unconstrained least-squares problems.

Functions/Subroutines

- pure real(dp) function lss_get_factor (this)

  Gets a factor used to scale the bounds on the initial step.
- subroutine lss_set_factor (this, x)

  Sets a factor used to scale the bounds on the initial step.
- subroutine lss_solve (this, fcn, x, fvec, ib, err)

  Applies the Levenberg-Marquardt method to solve the nonlinear least-squares problem.
- subroutine lmpar (r, ipvt, diag, qtb, delta, par, x, sdiag, wa1, wa2)

Completes the solution of the Levenberg-Marquardt problem when provided with a QR factored form of the system Jacobian matrix. The form of the problem at this stage is J∗X = B (J = Jacobian), and D∗X = 0, where D is a diagonal matrix.

- subroutine lmfactor (a, pivot, ipvt, rdiag, acnorm, wa)

  Computes the QR factorization of an M-by-N matrix.

- subroutine lmsolve (r, ipvt, diag, qtb, x, sdiag, wa)

  Solves the QR factored system A∗X = B, coupled with the diagonal system D∗X = 0 in the least-squares sense.

4.2.1 Detailed Description

nonlin_least_squares

Purpose

To provide routines capable of solving the nonlinear least squares problem.

4.2.2 Function/Subroutine Documentation

4.2.2.1 subroutine nonlin_least_squares::lmfactor (real(dp), dimension(:,:), intent(inout) *a*, logical, intent(in) *pivot*, integer(i32), dimension(:), intent(out) *ipvt*, real(dp), dimension(:), intent(out) *rdiag*, real(dp), dimension(:), intent(out) *acnorm*, real(dp), dimension(:), intent(out) *wa*) [private]

Computes the QR factorization of an M-by-N matrix.

Parameters

in,out	*a*	On input, the M-by-N matrix to factor. On output, the strict upper triangular portion contains matrix R1 of the factorization, the lower trapezoidal portion contains the factored form of Q1, and the diagonal contains the corresponding elementary reflector.
in	*pivot*	Set to true to utilize column pivoting; else, set to false for no pivoting.
out	*ipvt*	An N-element array that is used to contain the pivot indices unless `pivot` is set to false. In such event, this array is unused.
out	*rdiag*	An N-element array used to store the diagonal elements of the R1 matrix.
out	*acnorm*	An N-element array used to contain the norms of each column in the event column pivoting is used. If pivoting is not used, this array is unused.
out	*wa*	An N-element workspace array.

Remarks

This routines is based upon the MINPACK routine QRFAC.

Definition at line 661 of file nonlin_least_squares.f90.

4.2.2.2 subroutine nonlin_least_squares::lmpar (real(dp), dimension(:,:), intent(inout) *r*, integer(i32), dimension(:), intent(in) *ipvt*, real(dp), dimension(:), intent(in) *diag*, real(dp), dimension(:), intent(in) *qtb*, real(dp), intent(in) *delta*, real(dp), intent(inout) *par*, real(dp), dimension(:), intent(out) *x*, real(dp), dimension(:), intent(out) *sdiag*, real(dp), dimension(:), intent(out) *wa1*, real(dp), dimension(:), intent(out) *wa2*) [private]

Completes the solution of the Levenberg-Marquardt problem when provided with a QR factored form of the system Jacobian matrix. The form of the problem at this stage is J∗X = B (J = Jacobian), and D∗X = 0, where D is a diagonal matrix.

Parameters

in,out	*r*	On input, the N-by-N upper triangular matrix R1 of the QR factorization. On output, the upper triangular portion is unaltered, but the strict lower triangle contains the strict upper triangle (transposed) of the matrix S.
in	*ipvt*	An N-element array tracking the pivoting operations from the original QR factorization.
in	*diag*	An N-element array containing the diagonal components of the matrix D.
in	*qtb*	An N-element array containing the first N elements of Q1∗∗T ∗ B.
in	*delta*	A positive input variable that specifies an upper bounds on the Euclidean norm of D∗X.
in,out	*par*	On input, the initial estimate of the Levenberg-Marquardt parameter. On output, the final estimate.
out	*x*	The N-element array that is the solution of A∗X = B, and of D∗X = 0.
out	*sdiag*	An N-element array containing the diagonal elements of the matrix S.
out	*wa1*	An N-element workspace array.
out	*wa2*	An N-element workspace array.

Remarks

This routines is based upon the MINPACK routine LMPAR.

Definition at line 494 of file nonlin_least_squares.f90.

4.2.2.3 subroutine nonlin_least_squares::lmsolve (real(dp), dimension(:,:), intent(inout) *r*, integer(i32), dimension(:), intent(in) *ipvt*, real(dp), dimension(:), intent(in) *diag*, real(dp), dimension(:), intent(in) *qtb*, real(dp), dimension(:), intent(out) *x*, real(dp), dimension(:), intent(out) *sdiag*, real(dp), dimension(:), intent(out) *wa*) [private]

Solves the QR factored system A∗X = B, coupled with the diagonal system D∗X = 0 in the least-squares sense.

Parameters

in,out	*r*	On input, the N-by-N upper triangular matrix R1 of the QR factorization. On output, the upper triangular portion is unaltered, but the strict lower triangle contains the strict upper triangle (transposed) of the matrix S.
in	*ipvt*	An N-element array tracking the pivoting operations from the original QR factorization.
in	*diag*	An N-element array containing the diagonal components of the matrix D.
in	*qtb*	An N-element array containing the first N elements of Q1∗∗T ∗ B.
out	*x*	The N-element array that is the solution of A∗X = B, and of D∗X = 0.
out	*sdiag*	An N-element array containing the diagonal elements of the matrix S.
out	*wa*	An N-element workspace array.

Remarks

This routines is based upon the MINPACK routine QRSOLV.

Definition at line 764 of file nonlin_least_squares.f90.

4.2.2.4 pure real(dp) function nonlin_least_squares::lss_get_factor (class(least_squares_solver), intent(in) *this*) [private]

Gets a factor used to scale the bounds on the initial step.

Parameters

in	*this*	The least_squares_solver object.

Returns

The factor.

Remarks

This factor is used to set the bounds on the initial step such that the initial step is bounded as the product of the factor with the Euclidean norm of the vector resulting from multiplication of the diagonal scaling matrix and the solution estimate. If zero, the factor itself is used.

Definition at line 49 of file nonlin_least_squares.f90.

4.2.2.5 subroutine nonlin_least_squares::lss_set_factor (class(least_squares_solver), intent(inout) *this*, real(dp), intent(in) *x*) [private]

Sets a factor used to scale the bounds on the initial step.

Parameters

in	*this*	The least_squares_solver object.
in	*x*	The factor. Notice, the factor is limited to the interval [0.1, 100].

Remarks

This factor is used to set the bounds on the initial step such that the initial step is bounded as the product of the factor with the Euclidean norm of the vector resulting from multiplication of the diagonal scaling matrix and the solution estimate. If zero, the factor itself is used.

Definition at line 68 of file nonlin_least_squares.f90.

4.2.2.6 subroutine nonlin_least_squares::lss_solve (class(least_squares_solver), intent(inout) *this*, class(vecfcn_helper), intent(in) *fcn*, real(dp), dimension(:), intent(inout) *x*, real(dp), dimension(:), intent(out) *fvec*, type(iteration_behavior), optional *ib*, class(errors), intent(in), optional, target *err*) [private]

Applies the Levenberg-Marquardt method to solve the nonlinear least-squares problem.

Parameters

in,out	*this*	The least_squares_solver object.
in	*fcn*	The vecfcn_helper object containing the equations to solve.
in,out	*x*	On input, an M-element array containing an initial estimate to the solution. On output, the updated solution estimate. M is the number of variables.
out	*fvec*	An N-element array that, on output, will contain the values of each equation as evaluated at the variable values given in `x`. N is the number of equations.
out	*ib*	An optional output, that if provided, allows the caller to obtain iteration performance statistics.

Parameters

out	*err*	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. • NL_INVALID_OPERATION_ERROR: Occurs if no equations have been defined. • NL_INVALID_INPUT_ERROR: Occurs if the number of equations is less than than the number of variables. • NL_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized correctly. • NL_CONVERGENCE_ERROR: Occurs if the line search cannot converge within the allowed number of iterations. • NL_OUT_OF_MEMORY_ERROR: Occurs if there is insufficient memory available. • NL_TOLERANCE_TOO_SMALL_ERROR: Occurs if the requested tolerance is to small to be practical for the problem at hand.

Remarks

This routines is based upon the MINPACK routine LMDIF.

Usage

The following code provides an example of how to solve a system of N equations of N unknonwns using the Levenberg-Marquardt method.

```
! System of Equations #1:
!
! x**2 + y**2 = 34
! x**2 - 2 * y**2 = 7
!
! Solution:
! x = +/-5
! y = +/-3
subroutine fcn1(x, f)
    real(dp), intent(in), dimension(:) :: x
    real(dp), intent(out), dimension(:) :: f
    f(1) = x(1)**2 + x(2)**2 - 34.0d0
    f(2) = x(1)**2 - 2.0d0 * x(2)**2 - 7.0d0
end subroutine

program main
    use linalg_constants, only : dp
    use nonlin_types, only : vecfcn, vecfcn_helper
    use nonlin_least_squares, only : least_squares_solver

    type(vecfcn_helper) :: obj
    procedure(vecfcn), pointer :: fcn
    type(least_squares_solver) :: solver
    real(dp) :: x(2), f(2)

    ! Set the initial conditions to [1, 1]
    x = 1.0d0

    ! Solve the system of equations.  The solution overwrites X
    call solver%solve(obj, x, f)

    ! Print the output and the residual:
    print '(AF5.3AF5.3A)', "The solution: (", x(1), ", ", x(2), ")"
    print '(AE8.3AE8.3A)', "The residual: (", f(1), ", ", f(2), ")"
end program
```

The above program returns the following results.

```
The solution: (5.000, 3.000)
The residual: (.000E+00, .000E+00)
```

See Also

- Wikipedia
- MINPACK (Wikipedia)

Definition at line 164 of file nonlin_least_squares.f90.

4.3 nonlin_linesearch Module Reference

nonlin_linesearch

Data Types

- type line_search

  Defines a type capable of performing an inexact, backtracking line search to find a point as far along the specified direction vector that is usable for unconstrained minimization problems.

Functions/Subroutines

- pure integer(i32) function ls_get_max_eval (this)

  Gets the maximum number of function evaluations allowed during a single line search.
- subroutine ls_set_max_eval (this, x)

  Sets the maximum number of function evaluations allowed during a single line search.
- pure real(dp) function ls_get_scale (this)

  *Gets the scaling of the product of the gradient and direction vectors (ALPHA) such that F(X + LAMBDA * P) <= F(X) + LAMBDA * ALPHA * P**T * G, where P is the search direction vector, G is the gradient vector, and LAMBDA is the scaling factor.*
- subroutine ls_set_scale (this, x)

  *sets the scaling of the product of the gradient and direction vectors (ALPHA) such that F(X + LAMBDA * P) <= F(X) + LAMBDA * ALPHA * P**T * G, where P is the search direction vector, G is the gradient vector, and LAMBDA is the scaling factor.*
- pure real(dp) function ls_get_dist (this)

  Gets a distance factor defining the minimum distance along the search direction vector is practical.
- subroutine ls_set_dist (this, x)

  Sets a distance factor defining the minimum distance along the search direction vector is practical.
- subroutine ls_search (this, fcn, xold, grad, dir, x, fvec, fold, fx, ib, err)

  Utilizes an inexact, backtracking line search to find a point as far along the specified direction vector that is usable for unconstrained minimization problems.

4.3.1 Detailed Description

nonlin_linesearch

Purpose

To provide line search routines capable of minimizing nondesireable influences of the nonlinear equation solver model on the convergence of the iteration process.

4.3.2 Function/Subroutine Documentation

4.3.2.1 pure real(dp) function nonlin_linesearch::ls_get_dist (class(line_search), intent(in) *this*) [private]

Gets a distance factor defining the minimum distance along the search direction vector is practical.

Parameters

in	*this*	The line_search object.

Returns

The distance factor. A value of 1 indicates the full length of the vector.

Definition at line 143 of file nonlin_linesearch.f90.

4.3.2.2 pure integer(i32) function nonlin_linesearch::ls_get_max_eval (class(line_search), intent(in) *this*) [private]

Gets the maximum number of function evaluations allowed during a single line search.

Parameters

in	*this*	The line_search object.

Returns

The maximum number of function evaluations.

Definition at line 90 of file nonlin_linesearch.f90.

4.3.2.3 pure real(dp) function nonlin_linesearch::ls_get_scale (class(line_search), intent(in) *this*) [private]

Gets the scaling of the product of the gradient and direction vectors (ALPHA) such that F(X + LAMBDA $*$ P) <= F(X) + LAMBDA $*$ ALPHA $*$ P$**$T $*$ G, where P is the search direction vector, G is the gradient vector, and LAMBDA is the scaling factor.

Parameters

in	*this*	The line_search object.

Returns

The scaling factor.

Definition at line 116 of file nonlin_linesearch.f90.

4.3.2.4 subroutine nonlin_linesearch::ls_search (class(line_search), intent(in) *this,* class(vecfcn_helper), intent(in) *fcn,* real(dp), dimension(:), intent(in) *xold,* real(dp), dimension(:), intent(in) *grad,* real(dp), dimension(:), intent(in) *dir,* real(dp), dimension(:), intent(out) *x,* real(dp), dimension(:), intent(out) *fvec,* real(dp), intent(in), optional *fold,* real(dp), intent(out), optional *fx,* type(iteration_behavior), optional *ib,* class(errors), intent(in), optional, target *err*) [private]

Utilizes an inexact, backtracking line search to find a point as far along the specified direction vector that is usable for unconstrained minimization problems.

Parameters

in	*this*	The line_search object.
in	*fcn*	A vecfcn_helper object containing the system of equations.
in	*xold*	An N-element array defining the initial point, where N is the number of variables.
in	*grad*	An N-element array defining the gradient of `fcn` evaluated at `xold`.
in	*dir*	An N-element array defining the search direction.
out	*x*	An N-element array where the updated solution point will be written.
out	*fvec*	An M-element array containing the M equation values evaluated at `x`, where M is the number of equations.
in	*fold*	An optional input that provides the value resulting from: 1/2 $*$ dot_product(fcn(xold), fcn(xold)). If not provided, `fcn` is evalauted at `xold`, and the aforementioned relationship is computed.
out	*fx*	The result of the operation: (1/2) $*$ dot_product(`fvec`, `fvec`). Remember `fvec` is evaluated at `x`.
out	*ib*	An optional output, that if provided, allows the caller to obtain iteration performance statistics.
out	*err*	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. • NL_INVALID_OPERATION_ERROR: Occurs if no equations have been defined. • NL_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized correctly. • NL_DIVERGENT_BEHAVIOR_ERROR: Occurs if the direction vector is pointing in an apparent uphill direction. • NL_CONVERGENCE_ERROR: Occurs if the line search cannot converge within the allowed number of iterations.

Definition at line 205 of file nonlin_linesearch.f90.

4.3.2.5 subroutine nonlin_linesearch::ls_set_dist (class(line_search), intent(inout) *this,* real(dp), intent(in) *x*) [private]

Sets a distance factor defining the minimum distance along the search direction vector is practical.

Parameters

in,out	*this*	The line_search object.
in	*x*	The distance factor. A value of 1 indicates the full length of the vector. Notice, this value is restricted to lie in the set [0.1, 1.0)

Definition at line 157 of file nonlin_linesearch.f90.

4.3.2.6 subroutine nonlin_linesearch::ls_set_max_eval (class(line_search), intent(inout) *this,* integer(i32), intent(in) *x*) [private]

Sets the maximum number of function evaluations allowed during a single line search.

Parameters

in,out	*this*	The line_search object.
in	*x*	The maximum number of function evaluations.

Definition at line 102 of file nonlin_linesearch.f90.

4.3.2.7 subroutine nonlin_linesearch::ls_set_scale (class(line_search), intent(inout) *this*, real(dp), intent(in) *x*) [private]

sets the scaling of the product of the gradient and direction vectors (ALPHA) such that F(X + LAMBDA ∗ P) <= F(X) + LAMBDA ∗ ALPHA ∗ P∗∗T ∗ G, where P is the search direction vector, G is the gradient vector, and LAMBDA is the scaling factor.

Parameters

in,out	*this*	The line_search object.
in	*x*	The scaling factor.

Definition at line 130 of file nonlin_linesearch.f90.

4.4 nonlin_solve Module Reference

nonlin_solve

Data Types

- type brent_solver

  Defines a solver based upon Brent's method for solving an equation of one variable without using derivatives.
- type line_search_solver

  A class describing nonlinear solvers that use a line search algorithm to improve convergence behavior.
- type newton_solver

  Defines a Newton solver.
- type quasi_newton_solver

  Defines a quasi-Newton type solver based upon Broyden's method.

Functions/Subroutines

- subroutine lss_get_line_search (this, ls)

  Gets the line search module.
- subroutine lss_set_line_search (this, ls)

  Sets the line search module.
- subroutine lss_set_default (this)

  Establishes a default line_search object for the line search module.
- pure logical function lss_is_line_search_defined (this)

  Tests to see if a line search module is defined.

- pure logical function lss_get_use_search (this)

  Gets a value determining if a line-search should be employed.
- subroutine lss_set_use_search (this, x)

  Sets a value determining if a line-search should be employed.
- subroutine qns_solve (this, fcn, x, fvec, ib, err)

  Applies the quasi-Newton's method developed by Broyden in conjunction with a backtracking type line search to solve N equations of N unknowns.
- pure integer(i32) function qns_get_jac_interval (this)

  Gets the number of iterations that may pass before forcing a recalculation of the Jacobian matrix.
- subroutine qns_set_jac_interval (this, n)

  Sets the number of iterations that may pass before forcing a recalculation of the Jacobian matrix.
- subroutine ns_solve (this, fcn, x, fvec, ib, err)

  Applies Newton's method in conjunction with a backtracking type line search to solve N equations of N unknowns.
- subroutine brent_solve (this, fcn, x, lim, f, ib, err)

  Solves the equation.
- subroutine test_convergence (x, xo, f, g, lg, xtol, ftol, gtol, c, cx, cf, cg, xnorm, fnorm)

  Tests for convergence.

4.4.1 Detailed Description

nonlin_solve

Purpose

To provide various routines capapble of solving systems of nonlinear equations.

4.4.2 Function/Subroutine Documentation

4.4.2.1 subroutine nonlin_solve::brent_solve (class(brent_solver), intent(inout) *this,* class(fcn1var_helper), intent(in) *fcn,* real(dp), intent(inout) *x,* type(value_pair), intent(in) *lim,* real(dp), intent(out), optional *f,* type(iteration_behavior), optional *ib,* class(errors), intent(in), optional, target *err*) [private]

Solves the equation.

Parameters

in,out	*this*	The brent_solver object.
in	*fcn*	The fcn1var_helper object containing the equation to solve.
in,out	*x*	A parameter used to return the solution. Notice, any input value will be ignored as this routine relies upon the search limits in `lim` to provide a starting point.
in	*lim*	A value_pair object defining the search limits.
out	*f*	An optional parameter used to return the function residual as computed at `x`.
out	*ib*	An optional output, that if provided, allows the caller to obtain iteration performance statistics.
out	*err*	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. • NL_INVALID_OPERATION_ERROR: Occurs if no equations have been defined. • NL_INVALID_INPUT_ERROR: Occurs if the number of equations is different than the number of variables. • NL_CONVERGENCE_ERROR: Occurs if the algorithm cannot converge within the allowed number of iterations.

Usage

The following code provides an example of how to solve an equation of one variable using Brent's method.

```
! f(x) = sin(x) / x, SOLUTION: x = n * pi for n = 1, 2, 3, ...
function fcn1(x) result(f)
    real(dp), intent(in) :: x
    real(dp) :: f
    f = sin(x) / x
end function

program main
    use linalg_constants, only : dp
    use nonlin_types, only : fcn1var, fcn1var_helper,
      value_pair
    use nonlin_solve, only : brent_solver

    type(fcn1var_helper) :: obj
    procedure(fcn1var), pointer :: fcn
    type(brent_solver) :: solver
    real(dp) :: x, f
    type(value_pair) :: limits

    ! Define the solution limits
    lmiits%x1 = 1.5d0
    limits%x2 = 5.0d0

    ! Solve the equation
    call solver%solve(obj, x, limits, f)

    ! Print the output and the residual:
    print '(AF5.3)', "The solution: ", x
    print '(AE9.3)', "The residual: ", f
end program
```

The above program returns the following results.

```
The solution: 3.142
The residual: -.751E-11
```

See Also

- Wikipedia
- Numerical Recipes
- R.P. Brent, "Algorithms for Minimization without Derivatives," Dover Publications, January 2002. ISBN 0-486-41998-3. Further information available here.

Definition at line 969 of file nonlin_solve.f90.

4.4.2.2 subroutine nonlin_solve::lss_get_line_search (class(line_search_solver), intent(in) *this*, class(line_search), intent(out), allocatable *ls*) [private]

Gets the line search module.

Parameters

in	*this*	The line_search_solver object.
out	*ls*	The line_search object.

Definition at line 96 of file nonlin_solve.f90.

4.4.2.3 pure logical function nonlin_solve::lss_get_use_search (class(line_search_solver), intent(in) *this*) [private]

Gets a value determining if a line-search should be employed.

Parameters

in	*this*	The line_search_solver object.

Returns

Returns true if a line search should be used; else, false.

Definition at line 142 of file nonlin_solve.f90.

4.4.2.4 pure logical function nonlin_solve::lss_is_line_search_defined (class(line_search_solver), intent(in) *this*) [private]

Tests to see if a line search module is defined.

Parameters

in	*this*	The line_search_solver object.

Returns

Returns true if a module is defined; else, false.

Definition at line 131 of file nonlin_solve.f90.

4.4.2.5 subroutine nonlin_solve::lss_set_default (class(line_search_solver), intent(inout) *this*) [private]

Establishes a default line_search object for the line search module.

Parameters

in,out	*this*	The line_search_solver object.

Definition at line 120 of file nonlin_solve.f90.

4.4.2.6 subroutine nonlin_solve::lss_set_line_search (class(line_search_solver), intent(inout) *this*, class(line_search), intent(in) *ls*) [private]

Sets the line search module.

Parameters

in,out	*this*	The line_search_solver object.
in	*ls*	The line_search object.

Definition at line 108 of file nonlin_solve.f90.

4.4.2.7 subroutine nonlin_solve::lss_set_use_search (class(line_search_solver), intent(inout) *this*, logical, intent(in) *x*) [private]

Sets a value determining if a line-search should be employed.

Parameters

in,out	*this*	The line_search_solver object.
in	*x*	Set to true if a line search should be used; else, false.

Definition at line 153 of file nonlin_solve.f90.

4.4.2.8 subroutine nonlin_solve::ns_solve (class(newton_solver), intent(inout) *this*, class(vecfcn_helper), intent(in) *fcn*, real(dp), dimension(:), intent(inout) *x*, real(dp), dimension(:), intent(out) *fvec*, type(iteration_behavior), optional *ib*, class(errors), intent(in), optional, target *err*) [private]

Applies Newton's method in conjunction with a backtracking type line search to solve N equations of N unknowns.

Parameters

in,out	*this*	The equation_solver-based object.
in	*fcn*	The vecfcn_helper object containing the equations to solve.
in,out	*x*	On input, an N-element array containing an initial estimate to the solution. On output, the updated solution estimate. N is the number of variables.
out	*fvec*	An N-element array that, on output, will contain the values of each equation as evaluated at the variable values given in `x`.
out	*ib*	An optional output, that if provided, allows the caller to obtain iteration performance statistics.
out	*err*	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. • NL_INVALID_OPERATION_ERROR: Occurs if no equations have been defined. • NL_INVALID_INPUT_ERROR: Occurs if the number of equations is different than the number of variables. • NL_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized correctly. • NL_DIVERGENT_BEHAVIOR_ERROR: Occurs if the direction vector is pointing in an apparent uphill direction. • NL_CONVERGENCE_ERROR: Occurs if the line search cannot converge within the allowed number of iterations. • NL_OUT_OF_MEMORY_ERROR: Occurs if there is insufficient memory available. • NL_SPURIOUS_CONVERGENCE_ERROR: Occurs as a warning if the slope of the gradient vector becomes sufficiently close to zero.

Usage

The following code provides an example of how to solve a system of N equations of N unknonwns using Newton's method.

```
! System of Equations #1:
!
! x**2 + y**2 = 34
! x**2 - 2 * y**2 = 7
!
! Solution:
! x = +/-5
! y = +/-3
subroutine fcn1(x, f)
    real(dp), intent(in), dimension(:) :: x
    real(dp), intent(out), dimension(:) :: f
    f(1) = x(1)**2 + x(2)**2 - 34.0d0
    f(2) = x(1)**2 - 2.0d0 * x(2)**2 - 7.0d0
end subroutine

program main
    use linalg_constants, only : dp
    use nonlin_types, only : vecfcn, vecfcn_helper
    use nonlin_solve, only : newton_solver

    type(vecfcn_helper) :: obj
    procedure(vecfcn), pointer :: fcn
    type(newton_solver) :: solver
    real(dp) :: x(2), f(2)

    ! Set the initial conditions to [1, 1]
    x = 1.0d0

    ! Solve the system of equations.  The solution overwrites X
    call solver%solve(obj, x, f)

    ! Print the output and the residual:
    print '(AF5.3AF5.3A)', "The solution: (", x(1), ", ", x(2), ")"
    print '(AE8.3AE8.3A)', "The residual: (", f(1), ", ", f(2), ")"
end program
```

The above program returns the following results.

```
The solution: (5.000, 3.000)
The residual: (.000E+00, .000E+00)
```

See Also

- Wikipedia

Definition at line 661 of file nonlin_solve.f90.

4.4.2.9 pure integer(i32) function nonlin_solve::qns_get_jac_interval (class(quasi_newton_solver), intent(in) *this*) [private]

Gets the number of iterations that may pass before forcing a recalculation of the Jacobian matrix.

Parameters

in	*this*	The quasi_newton_solver object.

Returns

The number of iterations.

Definition at line 559 of file nonlin_solve.f90.

4.4.2.10 subroutine nonlin_solve::qns_set_jac_interval (class(quasi_newton_solver), intent(inout) *this*, integer(i32), intent(in) *n*) [private]

Sets the number of iterations that may pass before forcing a recalculation of the Jacobian matrix.

Parameters

in,out	*this*	The quasi_newton_solver object.
in	*n*	The number of iterations.

Definition at line 571 of file nonlin_solve.f90.

4.4.2.11 subroutine nonlin_solve::qns_solve (class(quasi_newton_solver), intent(inout) *this*, class(vecfcn_helper), intent(in) *fcn*, real(dp), dimension(:), intent(inout) *x*, real(dp), dimension(:), intent(out) *fvec*, type(iteration_behavior), optional *ib*, class(errors), intent(in), optional, target *err*) [private]

Applies the quasi-Newton's method developed by Broyden in conjunction with a backtracking type line search to solve N equations of N unknowns.

Parameters

in,out	*this*	The equation_solver-based object.
in	*fcn*	The vecfcn_helper object containing the equations to solve.
in,out	*x*	On input, an N-element array containing an initial estimate to the solution. On output, the updated solution estimate. N is the number of variables.
out	*fvec*	An N-element array that, on output, will contain the values of each equation as evaluated at the variable values given in `x`.
out	*ib*	An optional output, that if provided, allows the caller to obtain iteration performance statistics.
out	*err*	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. • NL_INVALID_OPERATION_ERROR: Occurs if no equations have been defined. • NL_INVALID_INPUT_ERROR: Occurs if the number of equations is different than the number of variables. • NL_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized correctly. • NL_DIVERGENT_BEHAVIOR_ERROR: Occurs if the direction vector is pointing in an apparent uphill direction. • NL_CONVERGENCE_ERROR: Occurs if the line search cannot converge within the allowed number of iterations. • NL_OUT_OF_MEMORY_ERROR: Occurs if there is insufficient memory available. • NL_SPURIOUS_CONVERGENCE_ERROR: Occurs as a warning if the slope of the gradient vector becomes sufficiently close to zero.

Usage

The following code provides an example of how to solve a system of N equations of N unknonwns using this Quasi-Newton method.

```
! System of Equations #1:
!
! x**2 + y**2 = 34
! x**2 - 2 * y**2 = 7
```

```
!
! Solution:
! x = +/-5
! y = +/-3
subroutine fcn1(x, f)
    real(dp), intent(in), dimension(:) :: x
    real(dp), intent(out), dimension(:) :: f
    f(1) = x(1)**2 + x(2)**2 - 34.0d0
    f(2) = x(1)**2 - 2.0d0 * x(2)**2 - 7.0d0
end subroutine

program main
    use linalg_constants, only : dp
    use nonlin_types, only : vecfcn, vecfcn_helper
    use nonlin_solve, only : quasi_newton_solver

    type(vecfcn_helper) :: obj
    procedure(vecfcn), pointer :: fcn
    type(quasi_newton_solver) :: solver
    real(dp) :: x(2), f(2)

    ! Set the initial conditions to [1, 1]
    x = 1.0d0

    ! Solve the system of equations.  The solution overwrites X
    call solver%solve(obj, x, f)

    ! Print the output and the residual:
    print '(AF5.3AF5.3A)', "The solution: (", x(1), ", ", x(2), ")"
    print '(AE8.3AE8.3A)', "The residual: (", f(1), ", ", f(2), ")"
end program
```

The above program returns the following results.

```
The solution: (5.000, 3.000)
The residual: (.604E-10, .121E-09)
```

See Also

- Broyden's Paper
- Wikipedia
- Numerical Recipes

Definition at line 246 of file nonlin_solve.f90.

4.4.2.12 subroutine nonlin_solve::test_convergence (real(dp), dimension(:), intent(in) *x*, real(dp), dimension(:), intent(in) *xo*, real(dp), dimension(:), intent(in) *f*, real(dp), dimension(:), intent(in) *g*, logical, intent(in) *lg*, real(dp), intent(in) *xtol*, real(dp), intent(in) *ftol*, real(dp), intent(in) *gtol*, logical, intent(out) *c*, logical, intent(out) *cx*, logical, intent(out) *cf*, logical, intent(out) *cg*, real(dp), intent(out) *xnorm*, real(dp), intent(out) *fnorm*) [private]

Tests for convergence.

Parameters

in	*x*	The current solution estimate.
in	*xo*	The previous solution estimate.
in	*f*	The current residual based upon `x`.
in	*g*	The current estimate of the gradient vector at `x`.
in	*lg*	Set to true if the gradient slope check should be performed; else, false.
in	*xtol*	The tolerance on the change in variable.
in	*ftol*	The tolerance on the residual.
in	*gtol*	The tolerance on the slope of the gradient.
out	*c*	True if the solution converged on either the residual or change in variable.
out	*cx*	True if convergence occurred due to change in variable.
out	*cf*	True if convergence occurred due to residual.
out	*cg*	True if convergence occured due to slope of the gradient.
out	*xnorm*	The largest magnitude component of the scaled change in variable vector.
out	*fnorm*	The largest magnitude residual component

Definition at line 1190 of file nonlin_solve.f90.

4.5 nonlin_types Module Reference

nonlin_types

Data Types

- type equation_solver

  A base class for various solvers of nonlinear systems of equations.
- type equation_solver_1var

  A base class for various solvers of equations of one variable.
- interface fcn1var

  Describes a function of one variable.
- type fcn1var_helper

  Defines a type capable of encapsulating an equation of one variable of the form: f(x) = 0.
- type iteration_behavior

  Defines a set of parameters that describe the behavior of the iteration process.
- interface jacobianfcn

  Describes a routine capable of computing the Jacobian matrix of M functions of N unknowns.
- interface nonlin_solver

  Describes the interface of a nonlinear equation solver.
- interface nonlin_solver_1var

  Describes the interface of a solver for an equation of one variable.
- type value_pair

  Defines a pair of numeric values.
- interface vecfcn

  Describes an M-element vector-valued function of N-variables.
- type vecfcn_helper

  Defines a type capable of encapsulating a system of nonlinear equations of the form: F(X) = 0.

Functions/Subroutines

- subroutine vfh_set_fcn (this, fcn, nfcn, nvar)

  Establishes a pointer to the routine containing the system of equations to solve.
- subroutine vfh_set_jac (this, jac)

  Establishes a pointer to the routine for computing the Jacobian matrix of the system of equations. If no routine is defined, the Jacobian matrix will be computed numerically (this is the default state).
- pure logical function vfh_is_fcn_defined (this)

  Tests if the pointer to the subroutine containing the system of equations to solve has been assigned.
- pure logical function vfh_is_jac_defined (this)

  Tests if the pointer to the subroutine containing the system of equations to solve has been assigned.
- subroutine vfh_fcn (this, x, f)

  Executes the routine containing the system of equations to solve. No action is taken if the pointer to the subroutine has not been defined.
- subroutine vfh_jac_fcn (this, x, jac, fv, work, olwork, err)

  Executes the routine containing the Jacobian matrix if supplied. If not supplied, the Jacobian is computed via finite differences.

- pure integer(i32) function vfh_get_nfcn (this)

  Gets the number of equations in this system.
- pure integer(i32) function vfh_get_nvar (this)

  Gets the number of variables in this system.
- real(dp) function f1h_fcn (this, x)

  Executes the routine containing the function to evaluate.
- pure logical function f1h_is_fcn_defined (this)

  Tests if the pointer to the function containing the equation to solve has been assigned.
- subroutine f1h_set_fcn (this, fcn)

  Establishes a pointer to the routine containing the equations to solve.
- pure integer(i32) function es_get_max_eval (this)

  Gets the maximum number of function evaluations allowed during a single solve.
- subroutine es_set_max_eval (this, n)

  Sets the maximum number of function evaluations allowed during a single solve.
- pure real(dp) function es_get_fcn_tol (this)

  Gets the convergence on function value tolerance.
- subroutine es_set_fcn_tol (this, x)

  Sets the convergence on function value tolerance.
- pure real(dp) function es_get_var_tol (this)

  Gets the convergence on change in variable tolerance.
- subroutine es_set_var_tol (this, x)

  Sets the convergence on change in variable tolerance.
- pure real(dp) function es_get_grad_tol (this)

  Gets the convergence on slope of the gradient vector tolerance.
- subroutine es_set_grad_tol (this, x)

  Sets the convergence on slope of the gradient vector tolerance.
- pure logical function es_get_print_status (this)

  Gets a logical value determining if iteration status should be printed.
- subroutine es_set_print_status (this, x)

  Sets a logical value determining if iteration status should be printed.
- pure integer(i32) function es1_get_max_eval (this)

  Gets the maximum number of function evaluations allowed during a single solve.
- subroutine es1_set_max_eval (this, n)

  Sets the maximum number of function evaluations allowed during a single solve.
- pure real(dp) function es1_get_fcn_tol (this)

  Gets the convergence on function value tolerance.
- subroutine es1_set_fcn_tol (this, x)

  Sets the convergence on function value tolerance.
- pure real(dp) function es1_get_var_tol (this)

  Gets the convergence on change in variable tolerance.
- subroutine es1_set_var_tol (this, x)

  Sets the convergence on change in variable tolerance.
- pure logical function es1_get_print_status (this)

  Gets a logical value determining if iteration status should be printed.
- subroutine es1_set_print_status (this, x)

  Sets a logical value determining if iteration status should be printed.
- subroutine, public print_status (iter, nfeval, njaceval, xnorm, fnorm)

  Prints the iteration status.

Variables

- integer, parameter, public nl_invalid_input_error = 201
  An error flag denoting an invalid input.
- integer, parameter, public nl_array_size_error = 202
  An error flag denoting an improperly sized array.
- integer, parameter, public nl_out_of_memory_error = 203
  An error denoting that there is insufficient memory available.
- integer, parameter, public nl_invalid_operation_error = 204
  An error resulting from an invalid operation.
- integer, parameter, public nl_convergence_error = 205
  An error resulting from a lack of convergence.
- integer, parameter, public nl_divergent_behavior_error = 206
  An error resulting from a divergent condition.
- integer, parameter, public nl_spurious_convergence_error = 207
  An error indicating a possible spurious convergence condition.
- integer, parameter, public nl_tolerance_too_small_error = 208
  An error indicating the user-requested tolerance is too small to be practical for the problem at hand.

4.5.1 Detailed Description

nonlin_types

Purpose

To provide various types and constants useful in the solution of systems of nonlinear equations.

4.5.2 Function/Subroutine Documentation

4.5.2.1 pure real(dp) function nonlin_types::es1_get_fcn_tol (class(equation_solver_1var), intent(in) *this*) [private]

Gets the convergence on function value tolerance.

Parameters

in	*this*	The equation_solver_1var object.

Returns

The tolerance value.

Definition at line 768 of file nonlin_types.f90.

4.5.2.2 pure integer(i32) function nonlin_types::es1_get_max_eval (class(equation_solver_1var), intent(in) *this*) [private]

Gets the maximum number of function evaluations allowed during a single solve.

Parameters

in	*this*	The equation_solver_1var object.

Returns

The maximum number of function evaluations.

Definition at line 745 of file nonlin_types.f90.

4.5.2.3 pure logical function nonlin_types::es1_get_print_status (class(equation_solver_1var), intent(in) *this*) [private]

Gets a logical value determining if iteration status should be printed.

Parameters

in	*this*	The equation_solver_1var object.

Returns

True if the iteration status should be printed; else, false.

Definition at line 813 of file nonlin_types.f90.

4.5.2.4 pure real(dp) function nonlin_types::es1_get_var_tol (class(equation_solver_1var), intent(in) *this*) [private]

Gets the convergence on change in variable tolerance.

Parameters

in	*this*	The equation_solver_1var object.

Returns

The tolerance value.

Definition at line 790 of file nonlin_types.f90.

4.5.2.5 subroutine nonlin_types::es1_set_fcn_tol (class(equation_solver_1var), intent(inout) *this*, real(dp), intent(in) *x*) [private]

Sets the convergence on function value tolerance.

Parameters

`in,out`	*this*	The equation_solver_1var object.
`in`	*x*	The tolerance value.

Definition at line 779 of file nonlin_types.f90.

4.5.2.6 subroutine nonlin_types::es1_set_max_eval (class(equation_solver_1var), intent(inout) *this*, integer(i32), intent(in) *n*) [private]

Sets the maximum number of function evaluations allowed during a single solve.

Parameters

`in,out`	*this*	The equation_solver_1var object.
`in`	*n*	The maximum number of function evaluations.

Definition at line 757 of file nonlin_types.f90.

4.5.2.7 subroutine nonlin_types::es1_set_print_status (class(equation_solver_1var), intent(inout) *this*, logical, intent(in) *x*) [private]

Sets a logical value determining if iteration status should be printed.

Parameters

`in,out`	*this*	The equation_solver_1var object.
`in`	*x*	True if the iteration status should be printed; else, false.

Definition at line 825 of file nonlin_types.f90.

4.5.2.8 subroutine nonlin_types::es1_set_var_tol (class(equation_solver_1var), intent(inout) *this*, real(dp), intent(in) *x*) [private]

Sets the convergence on change in variable tolerance.

Parameters

`in,out`	*this*	The equation_solver_1var object.
`in`	*x*	The tolerance value.

Definition at line 801 of file nonlin_types.f90.

4.5.2.9 pure real(dp) function nonlin_types::es_get_fcn_tol (class(equation_solver), intent(in) *this*) [private]

Gets the convergence on function value tolerance.

Parameters

in	*this*	The equation_solver object.

Returns

The tolerance value.

Definition at line 652 of file nonlin_types.f90.

4.5.2.10 pure real(dp) function nonlin_types::es_get_grad_tol (class(equation_solver), intent(in) *this*) [private]

Gets the convergence on slope of the gradient vector tolerance.

Parameters

in	*this*	The equation_solver object.

Returns

The tolerance value.

Definition at line 696 of file nonlin_types.f90.

4.5.2.11 pure integer(i32) function nonlin_types::es_get_max_eval (class(equation_solver), intent(in) *this*) [private]

Gets the maximum number of function evaluations allowed during a single solve.

Parameters

in	*this*	The equation_solver object.

Returns

The maximum number of function evaluations.

Definition at line 629 of file nonlin_types.f90.

4.5.2.12 pure logical function nonlin_types::es_get_print_status (class(equation_solver), intent(in) *this*) [private]

Gets a logical value determining if iteration status should be printed.

Parameters

in	*this*	The equation_solver object.

Returns

True if the iteration status should be printed; else, false.

Definition at line 719 of file nonlin_types.f90.

4.5.2.13 pure real(dp) function nonlin_types::es_get_var_tol (class(equation_solver), intent(in) *this*) [private]

Gets the convergence on change in variable tolerance.

Parameters

in	*this*	The equation_solver object.

Returns

The tolerance value.

Definition at line 674 of file nonlin_types.f90.

4.5.2.14 subroutine nonlin_types::es_set_fcn_tol (class(equation_solver), intent(inout) *this,* real(dp), intent(in) *x*) [private]

Sets the convergence on function value tolerance.

Parameters

in,out	*this*	The equation_solver object.
in	*x*	The tolerance value.

Definition at line 663 of file nonlin_types.f90.

4.5.2.15 subroutine nonlin_types::es_set_grad_tol (class(equation_solver), intent(inout) *this,* real(dp), intent(in) *x*) [private]

Sets the convergence on slope of the gradient vector tolerance.

Parameters

in	*this*	The equation_solver object.

Returns

The tolerance value.

Definition at line 707 of file nonlin_types.f90.

4.5.2.16 subroutine nonlin_types::es_set_max_eval (class(equation_solver), intent(inout) *this*, integer(i32), intent(in) *n*) [private]

Sets the maximum number of function evaluations allowed during a single solve.

Parameters

in,out	*this*	The equation_solver object.
in	*n*	The maximum number of function evaluations.

Definition at line 641 of file nonlin_types.f90.

4.5.2.17 subroutine nonlin_types::es_set_print_status (class(equation_solver), intent(inout) *this*, logical, intent(in) *x*) [private]

Sets a logical value determining if iteration status should be printed.

Parameters

in,out	*this*	The equation_solver object.
in	*x*	True if the iteration status should be printed; else, false.

Definition at line 731 of file nonlin_types.f90.

4.5.2.18 subroutine nonlin_types::es_set_var_tol (class(equation_solver), intent(inout) *this*, real(dp), intent(in) *x*) [private]

Sets the convergence on change in variable tolerance.

Parameters

in,out	*this*	The equation_solver object.
in	*x*	The tolerance value.

Definition at line 685 of file nonlin_types.f90.

4.5.2.19 real(dp) function nonlin_types::f1h_fcn (class(fcn1var_helper), intent(in) *this*, real(dp), intent(in) *x*) [private]

Executes the routine containing the function to evaluate.

Parameters

in	*this*	The fcn1var_helper object.
in	*x*	The value of the independent variable at which the function should be evaluated.

Returns

The value of the function at `x`.

Definition at line 588 of file nonlin_types.f90.

4.5.2.20 pure logical function nonlin_types::f1h_is_fcn_defined (class(fcn1var_helper), intent(in) *this*) [private]

Tests if the pointer to the function containing the equation to solve has been assigned.

Parameters

in	*this*	The fcn1var_helper object.

Returns

Returns true if the pointer has been assigned; else, false.

Definition at line 603 of file nonlin_types.f90.

4.5.2.21 subroutine nonlin_types::f1h_set_fcn (class(fcn1var_helper), intent(inout) *this*, procedure(fcn1var), intent(in), pointer *fcn*) [private]

Establishes a pointer to the routine containing the equations to solve.

Parameters

in,out	*this*	The fcn1var_helper object.
in	*fcn*	The function pointer.

Definition at line 615 of file nonlin_types.f90.

4.5.2.22 subroutine, public nonlin_types::print_status (integer(i32), intent(in) *iter*, integer(i32), intent(in) *nfeval*, integer(i32), intent(in) *njaceval*, real(dp), intent(in) *xnorm*, real(dp), intent(in) *fnorm*)

Prints the iteration status.

Parameters

in	*iter*	The iteration number.
in	*nfeval*	The number of function evaluations.
in	*njaceval*	The number of Jacobian evaluations.
in	*xnorm*	The change in variable value.
in	*fnorm*	The residual.

Definition at line 842 of file nonlin_types.f90.

4.5.2.23 subroutine nonlin_types::vfh_fcn (class(vecfcn_helper), intent(in) *this*, real(dp), dimension(:), intent(in) *x*, real(dp), dimension(:), intent(out) *f*) [private]

Executes the routine containing the system of equations to solve. No action is taken if the pointer to the subroutine has not been defined.

Parameters

in	*this*	The vecfcn_helper object.
in	*x*	An N-element array containing the independent variables.
out	*f*	An M-element array that, on output, contains the values of the M functions.

Definition at line 408 of file nonlin_types.f90.

4.5.2.24 pure integer(i32) function nonlin_types::vfh_get_nfcn (class(vecfcn_helper), intent(in) *this*) [private]

Gets the number of equations in this system.

Parameters

in	*this*	The vecfcn_helper object.

Returns

The function count.

Definition at line 562 of file nonlin_types.f90.

4.5.2.25 pure integer(i32) function nonlin_types::vfh_get_nvar (class(vecfcn_helper), intent(in) *this*) [private]

Gets the number of variables in this system.

Parameters

in	*this*	The vecfcn_helper object.

Returns

The number of variables.

Definition at line 573 of file nonlin_types.f90.

4.5.2.26 pure logical function nonlin_types::vfh_is_fcn_defined (class(vecfcn_helper), intent(in) *this*) [private]

Tests if the pointer to the subroutine containing the system of equations to solve has been assigned.

Parameters

in	*this*	The vecfcn_helper object.

Returns

Returns true if the pointer has been assigned; else, false.

Definition at line 382 of file nonlin_types.f90.

4.5.2.27 pure logical function nonlin_types::vfh_is_jac_defined (class(vecfcn_helper), intent(in) *this*) [private]

Tests if the pointer to the subroutine containing the system of equations to solve has been assigned.

Parameters

in	*this*	The vecfcn_helper object.

Returns

Returns true if the pointer has been assigned; else, false.

Definition at line 394 of file nonlin_types.f90.

4.5.2.28 subroutine nonlin_types::vfh_jac_fcn (class(vecfcn_helper), intent(in) *this,* real(dp), dimension(:), intent(inout) *x,* real(dp), dimension(:,:), intent(out) *jac,* real(dp), dimension(:), intent(in), optional, target *fv,* real(dp), dimension(:), intent(out), optional, target *work,* integer(i32), intent(out), optional *olwork,* integer(i32), intent(out), optional *err*) [private]

Executes the routine containing the Jacobian matrix if supplied. If not supplied, the Jacobian is computed via finite differences.

Parameters

in	*this*	The vecfcn_helper object.
in	*x*	An N-element array containing the independent variabls defining the point about which the derivatives will be calculated.
out	*jac*	An M-by-N matrix where, on output, the Jacobian will be written.
in	*fv*	An optional M-element array containing the function values at `x`. If not supplied, the function values are computed at `x`.
out	*work*	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least `olwork`. Notice, a workspace array is only utilized if the user does not provide a routine for computing the Jacobian.
out	*olwork*	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for `work`, and returns without performing any actual calculations.

Parameters

out	*err*	An optional integer output that can be used to determine error status. If not used, and an error is encountered, the routine simply returns silently. If used, the following error codes identify error status: • 0: No error has occurred. • n: A positive integer denoting the index of an invalid input. • -1: Indicates internal memory allocation failed.

Definition at line 443 of file nonlin_types.f90.

4.5.2.29 subroutine nonlin_types::vfh_set_fcn (class(vecfcn_helper), intent(inout) *this*, procedure(vecfcn), intent(in), pointer *fcn*, integer(i32), intent(in) *nfcn*, integer(i32), intent(in) *nvar*)

Establishes a pointer to the routine containing the system of equations to solve.

Parameters

in,out	*this*	The vecfcn_helper object.
in	*fcn*	The function pointer.
in	*nfcn*	The number of functions.
in	*nvar*	The number of variables.

Definition at line 354 of file nonlin_types.f90.

4.5.2.30 subroutine nonlin_types::vfh_set_jac (class(vecfcn_helper), intent(inout) *this*, procedure(jacobianfcn), intent(in), pointer *jac*) [private]

Establishes a pointer to the routine for computing the Jacobian matrix of the system of equations. If no routine is defined, the Jacobian matrix will be computed numerically (this is the default state).

Parameters

in,out	*this*	The vecfcn_helper object.
in	*jac*	The function pointer.

Definition at line 370 of file nonlin_types.f90.

Chapter 5

Data Type Documentation

5.1 nonlin_solve::brent_solver Type Reference

Defines a solver based upon Brent's method for solving an equation of one variable without using derivatives.

Inheritance diagram for nonlin_solve::brent_solver:

Collaboration diagram for nonlin_solve::brent_solver:

Public Member Functions

- procedure, public solve => brent_solve
    Solves the equation.

5.1.1 Detailed Description

Defines a solver based upon Brent's method for solving an equation of one variable without using derivatives.

Definition at line 80 of file nonlin_solve.f90.

The documentation for this type was generated from the following file:

- /home/jason/Documents/Code/nonlin/src/nonlin_solve.f90

5.2 nonlin_c_binding::cfcn1var Interface Reference

The C-friendly interface to fcn1var.

Public Member Functions

- real(dp) function **cfcn1var** (x)

5.2.1 Detailed Description

The C-friendly interface to fcn1var.

Parameters

in	*x*	The independent variable.

Returns

The value of the function `x`.

Definition at line 27 of file nonlin_c_binding.f90.

The documentation for this interface was generated from the following file:

- /home/jason/Documents/Code/nonlin/src/nonlin_c_binding.f90

5.3 nonlin_c_binding::cfcn1var_helper Type Reference

A container allowing the use of cfcn1var in the solver codes.

Inheritance diagram for nonlin_c_binding::cfcn1var_helper:

5.4 nonlin_c_binding::cjacobianfcn Interface Reference

The C-friendly interface to jacobianfcn.

Public Member Functions

- subroutine **cjacobianfcn** (neqn, nvar, x, jac)

5.4.1 Detailed Description

The C-friendly interface to jacobianfcn.

Parameters

in	*neqn*	The number of equations.
in	*nvar*	The number of variables.
in	*x*	The NVAR-element array containing the independent variables.
out	*jac*	An NEQN-byNVAR matrix where the Jacobian will be written.

Definition at line 54 of file nonlin_c_binding.f90.

The documentation for this interface was generated from the following file:

- /home/jason/Documents/Code/nonlin/src/nonlin_c_binding.f90

5.5 nonlin_c_binding::cvecfcn Interface Reference

The C-friendly interface to vecfcn.

Public Member Functions

- subroutine **cvecfcn** (neqn, nvar, x, f)

5.5.1 Detailed Description

The C-friendly interface to vecfcn.

Parameters

in	*neqn*	The number of equations.
in	*nvar*	The number of variables.
in	*x*	The NVAR-element array containing the independent variables.
out	*f*	The NEQN-element array containing the function values.

Definition at line 41 of file nonlin_c_binding.f90.

The documentation for this interface was generated from the following file:

- /home/jason/Documents/Code/nonlin/src/nonlin_c_binding.f90

5.6 nonlin_c_binding::cvecfcn_helper Type Reference

A container allowing the use of cvecfcn in the solver codes.

Inheritance diagram for nonlin_c_binding::cvecfcn_helper:

Collaboration diagram for nonlin_c_binding::cvecfcn_helper:

Public Member Functions

- procedure, public set_cfcn => cvfh_set_fcn

  Establishes a pointer to the routine containing the system of equations to solve.
- procedure, public set_cjacobian => cvfh_set_jac

  Establishes a pointer to the routine for computing the Jacobian matrix of the system of equations. If no routine is defined, the Jacobian matrix will be computed numerically (this is the default state).
- procedure, public is_fcn_defined => cvfh_is_fcn_defined

  Tests if the pointer to the subroutine containing the system of equations to solve has been assigned.
- procedure, public is_jacobian_defined => cvfh_is_jac_defined

  Tests if the pointer to the subroutine containing the system of equations to solve has been assigned.
- procedure, public fcn => cvfh_fcn

  Executes the routine containing the system of equations to solve. No action is taken if the pointer to the subroutine has not been defined.
- procedure, public jacobian => cvfh_jac_fcn

  Executes the routine containing the Jacobian matrix if supplied. If not supplied, the Jacobian is computed via finite differences.

Public Attributes

- procedure(cvecfcn), pointer, nopass m_cfcn => null()

    A pointer to the target cvecfcn routine.
- procedure(cjacobianfcn), pointer, nopass m_cjac => null()

    A pointer to the Jacobian routine.

5.6.1 Detailed Description

A container allowing the use of cvecfcn in the solver codes.

Definition at line 120 of file nonlin_c_binding.f90.

The documentation for this type was generated from the following file:

- /home/jason/Documents/Code/nonlin/src/nonlin_c_binding.f90

5.7 nonlin_types::equation_solver Type Reference

A base class for various solvers of nonlinear systems of equations.

Inheritance diagram for nonlin_types::equation_solver:

Public Member Functions

- procedure, public get_max_fcn_evals => es_get_max_eval

    Gets the maximum number of function evaluations allowed during a single solve.
- procedure, public set_max_fcn_evals => es_set_max_eval

    Sets the maximum number of function evaluations allowed during a single solve.
- procedure, public get_fcn_tolerance => es_get_fcn_tol

    Gets the convergence on function value tolerance.
- procedure, public set_fcn_tolerance => es_set_fcn_tol

    Sets the convergence on function value tolerance.
- procedure, public get_var_tolerance => es_get_var_tol

    Gets the convergence on change in variable tolerance.
- procedure, public set_var_tolerance => es_set_var_tol

    Sets the convergence on change in variable tolerance.
- procedure, public get_gradient_tolerance => es_get_grad_tol

    Gets the convergence on slope of the gradient vector tolerance.
- procedure, public set_gradient_tolerance => es_set_grad_tol

    Sets the convergence on slope of the gradient vector tolerance.
- procedure, public get_print_status => es_get_print_status

    Gets a logical value determining if iteration status should be printed.
- procedure, public set_print_status => es_set_print_status

    Sets a logical value determining if iteration status should be printed.
- procedure(nonlin_solver), deferred, pass, public solve

    Solves the system of equations.

Private Attributes

- integer(i32) m_maxeval = 100

    The maximum number of function evaluations allowed per solve.
- real(dp) m_fcntol = 1.0d-8

    The convergence criteria on function values.
- real(dp) m_xtol = 1.0d-12

    The convergence criteria on change in variable values.
- real(dp) m_gtol = 1.0d-12

    The convergence criteria for the slope of the gradient vector.
- logical m_printstatus = .false.

    Set to true to print iteration status; else, false.

5.7.1 Detailed Description

A base class for various solvers of nonlinear systems of equations.

Definition at line 179 of file nonlin_types.f90.

The documentation for this type was generated from the following file:

- /home/jason/Documents/Code/nonlin/src/nonlin_types.f90

5.8 nonlin_types::equation_solver_1var Type Reference

A base class for various solvers of equations of one variable.

Inheritance diagram for nonlin_types::equation_solver_1var:

Public Member Functions

- procedure, public get_max_fcn_evals => es1_get_max_eval

    Gets the maximum number of function evaluations allowed during a single solve.
- procedure, public set_max_fcn_evals => es1_set_max_eval

    Sets the maximum number of function evaluations allowed during a single solve.
- procedure, public get_fcn_tolerance => es1_get_fcn_tol

    Gets the convergence on function value tolerance.
- procedure, public set_fcn_tolerance => es1_set_fcn_tol

    Sets the convergence on function value tolerance.
- procedure, public get_var_tolerance => es1_get_var_tol

    Gets the convergence on change in variable tolerance.
- procedure, public set_var_tolerance => es1_set_var_tol

    Sets the convergence on change in variable tolerance.
- procedure, public get_print_status => es1_get_print_status

    Gets a logical value determining if iteration status should be printed.
- procedure, public set_print_status => es1_set_print_status

    Sets a logical value determining if iteration status should be printed.
- procedure(nonlin_solver_1var), deferred, pass, public solve

    Solves the equation.

Private Attributes

- integer(i32) m_maxeval = 100

  The maximum number of function evaluations allowed per solve.
- real(dp) m_fcntol = 1.0d-8

  The convergence criteria on function value.
- real(dp) m_xtol = 1.0d-12

  The convergence criteria on change in variable value.
- logical m_printstatus = .false.

  Set to true to print iteration status; else, false.

5.8.1 Detailed Description

A base class for various solvers of equations of one variable.

Definition at line 224 of file nonlin_types.f90.

The documentation for this type was generated from the following file:

- /home/jason/Documents/Code/nonlin/src/nonlin_types.f90

5.9 nonlin_types::fcn1var Interface Reference

Describes a function of one variable.

Private Member Functions

- real(dp) function **fcn1var** (x)

5.9.1 Detailed Description

Describes a function of one variable.

Parameters

in	*x*	The independent variable.

Returns

The value of the function at `x`.

Definition at line 63 of file nonlin_types.f90.

The documentation for this interface was generated from the following file:

- /home/jason/Documents/Code/nonlin/src/nonlin_types.f90

5.10 nonlin_types::fcn1var_helper Type Reference

Defines a type capable of encapsulating an equation of one variable of the form: f(x) = 0.

Inheritance diagram for nonlin_types::fcn1var_helper:

Public Member Functions

- procedure, public fcn => f1h_fcn

  Executes the routine containing the function to evaluate.
- procedure, public is_fcn_defined => f1h_is_fcn_defined

  Tests if the pointer to the function containing the equation to solve has been assigned.
- procedure, public set_fcn => f1h_set_fcn

  Establishes a pointer to the routine containing the equations to solve.

Private Attributes

- procedure(fcn1var), pointer, nopass m_fcn => null()

  A pointer to the target fcn1var routine.

5.10.1 Detailed Description

Defines a type capable of encapsulating an equation of one variable of the form: f(x) = 0.

Definition at line 139 of file nonlin_types.f90.

The documentation for this type was generated from the following file:

- /home/jason/Documents/Code/nonlin/src/nonlin_types.f90

5.11 nonlin_types::iteration_behavior Type Reference

Defines a set of parameters that describe the behavior of the iteration process.

Private Attributes

- integer(i32) iter_count

  Specifies the number of iterations performed.
- integer(i32) fcn_count

  Specifies the number of function evaluations performed.
- integer(i32) jacobian_count

  Specifies the number of Jacobian evaluations performed.
- logical(c_bool) converge_on_fcn

  True if the solution converged as a result of a zero-valued function; else, false.
- logical(c_bool) converge_on_chng

  True if the solution converged as a result of no appreciable change in solution points between iterations; else, false.
- logical(c_bool) converge_on_zero_diff

  True if the solution appears to have settled on a stationary point such that the gradient of the function is zero-valued; else, false.

5.11.1 Detailed Description

Defines a set of parameters that describe the behavior of the iteration process.

Definition at line 157 of file nonlin_types.f90.

The documentation for this type was generated from the following file:

- /home/jason/Documents/Code/nonlin/src/nonlin_types.f90

5.12 nonlin_types::jacobianfcn Interface Reference

Describes a routine capable of computing the Jacobian matrix of M functions of N unknowns.

Private Member Functions

- subroutine **jacobianfcn** (x, jac)

5.12.1 Detailed Description

Describes a routine capable of computing the Jacobian matrix of M functions of N unknowns.

Parameters

`in`	*x*	An N-element array containing the independent variables.
`out`	*jac*	An M-by-N matrix where the Jacobian will be written.

Definition at line 85 of file nonlin_types.f90.

The documentation for this interface was generated from the following file:

- /home/jason/Documents/Code/nonlin/src/nonlin_types.f90

5.13 nonlin_least_squares::least_squares_solver Type Reference

Defines a Levenberg-Marquardt based solver for unconstrained least-squares problems.

Inheritance diagram for nonlin_least_squares::least_squares_solver:

Collaboration diagram for nonlin_least_squares::least_squares_solver:

Public Member Functions

- procedure, public get_step_scaling_factor => lss_get_factor

  Gets a factor used to scale the bounds on the initial step.
- procedure, public set_step_scaling_factor => lss_set_factor

  Sets a factor used to scale the bounds on the initial step.
- procedure, public solve => lss_solve

  Solves the system of equations.

Private Attributes

- real(dp) m_factor = 100.0d0

  Initial step bounding factor.

5.13.1 Detailed Description

Defines a Levenberg-Marquardt based solver for unconstrained least-squares problems.

Definition at line 20 of file nonlin_least_squares.f90.

The documentation for this type was generated from the following file:

- /home/jason/Documents/Code/nonlin/src/nonlin_least_squares.f90

5.14 nonlin_linesearch::line_search Type Reference

Defines a type capable of performing an inexact, backtracking line search to find a point as far along the specified direction vector that is usable for unconstrained minimization problems.

Public Member Functions

- procedure, public get_max_fcn_evals => ls_get_max_eval

  Gets the maximum number of function evaluations allowed during a single line search.
- procedure, public set_max_fcn_evals => ls_set_max_eval

  Sets the maximum number of function evaluations allowed during a single line search.
- procedure, public get_scaling_factor => ls_get_scale

  Gets the scaling of the product of the gradient and direction vectors.
- procedure, public set_scaling_factor => ls_set_scale

  Sets the scaling of the product of the gradient and direction vectors.
- procedure, public get_distance_factor => ls_get_dist

  Gets a distance factor defining the minimum distance along the search direction vector is practical.
- procedure, public set_distance_factor => ls_set_dist

  Sets a distance factor defining the minimum distance along the search direction vector is practical.
- procedure, public search => ls_search

  Utilizes an inexact, backtracking line search based on the Armijo-Goldstein condition to find a point as far along the specified direction vector that is usable for unconstrained minimization problems.

Private Attributes

- integer(i32) m_maxeval = 100

  The maximum number of function evaluations allowed during a single line search.
- real(dp) m_alpha = 1.0d-4

  Defines the scaling of the product of the gradient and direction vectors such that F(X + LAMBDA ∗ P) <= F(X) + LAMBDA ∗ ALPHA ∗ P∗∗T ∗ G, where P is the search direction vector, G is the gradient vector, and LAMBDA is the scaling factor. The parameter must exist on the set (0, 1). A value of 1e-4 is typically a good starting point.
- real(dp) m_factor = 0.1d0

  Defines a minimum factor X used to determine a minimum value LAMBDA such that MIN(LAMBDA) = X ∗ LAMBDA, where LAMBDA defines the distance along the line search direction assuming a value of one means the full length of the direction vector is traversed. As such, the value must exist on the set (0, 1); however, for practical considerations, the minimum value should be limited to 0.1 such that the value must exist on the set [0.1, 1).

5.14.1 Detailed Description

Defines a type capable of performing an inexact, backtracking line search to find a point as far along the specified direction vector that is usable for unconstrained minimization problems.

See Also

- Wikipedia
- Oxfford Lecture Notes
- Northwestern University - Line Search
- Northwestern University - Trust Region Methods
- Wolfram
- Numerical Recipes

Definition at line 34 of file nonlin_linesearch.f90.

The documentation for this type was generated from the following file:

- /home/jason/Documents/Code/nonlin/src/nonlin_linesearch.f90

5.15 nonlin_c_binding::line_search_control Type Reference

Defines a set of line search controls.

Public Attributes

- integer(i32) max_evals

  The maximum number of function evaluations allowed per search.
- real(dp) alpha

  Defines the scaling of the product of the gradient and direction vectors such that F(X + LAMBDA ∗ P) <= F(X) + LAMBDA ∗ ALPHA ∗ P∗∗T ∗ G, where P is the search direction vector, G is the gradient vector, and LAMBDA is the scaling factor. The parameter must exist on the set (0, 1). A value of 1e-4 is typically a good starting point.
- real(dp) factor

  Defines a minimum factor X used to determine a minimum value LAMBDA such that MIN(LAMBDA) = X ∗ LAMBDA, where LAMBDA defines the distance along the line search direction assuming a value of one means the full length of the direction vector is traversed. As such, the value must exist on the set (0, 1); however, for practical considerations, the minimum value should be limited to 0.1 such that the value must exist on the set [0.1, 1).

5.15.1 Detailed Description

Defines a set of line search controls.

Definition at line 81 of file nonlin_c_binding.f90.

The documentation for this type was generated from the following file:

- /home/jason/Documents/Code/nonlin/src/nonlin_c_binding.f90

5.16 nonlin_solve::line_search_solver Type Reference

A class describing nonlinear solvers that use a line search algorithm to improve convergence behavior.

Inheritance diagram for nonlin_solve::line_search_solver:

Collaboration diagram for nonlin_solve::line_search_solver:

Public Member Functions

- procedure, public get_line_search => lss_get_line_search
  Gets the line search module.
- procedure, public set_line_search => lss_set_line_search
  Sets the line search module.
- procedure, public set_default_line_search => lss_set_default
  Establishes a default line_search object for the line search module.
- procedure, public is_line_search_defined =>lss_is_line_search_defined
  Tests to see if a line search module is defined.
- procedure, public get_use_line_search => lss_get_use_search
  Gets a value determining if a line-search should be employed.
- procedure, public set_use_line_search => lss_set_use_search
  Sets a value determining if a line-search should be employed.

Private Attributes

- class(line_search), allocatable m_linesearch
  The line search module.
- logical m_uselinesearch = .true.
  Set to true if a line search should be used regardless of the status of m_lineSearch.

5.16.1 Detailed Description

A class describing nonlinear solvers that use a line search algorithm to improve convergence behavior.

Definition at line 28 of file nonlin_solve.f90.

The documentation for this type was generated from the following file:

- /home/jason/Documents/Code/nonlin/src/nonlin_solve.f90

5.17 nonlin_solve::newton_solver Type Reference

Defines a Newton solver.

Inheritance diagram for nonlin_solve::newton_solver:

Collaboration diagram for nonlin_solve::newton_solver:

Public Member Functions

- procedure, public solve => ns_solve

  Solves the system of equations.

5.17.1 Detailed Description

Defines a Newton solver.

Definition at line 71 of file nonlin_solve.f90.

The documentation for this type was generated from the following file:

- /home/jason/Documents/Code/nonlin/src/nonlin_solve.f90

5.18 nonlin_types::nonlin_solver Interface Reference

Describes the interface of a nonlinear equation solver.

Private Member Functions

- subroutine **nonlin_solver** (this, fcn, x, fvec, ib, err)

5.18.1 Detailed Description

Describes the interface of a nonlinear equation solver.

Parameters

`in,out`	*this*	The equation_solver-based object.
`in`	*fcn*	The vecfcn_helper object containing the equations to solve.
`in,out`	*x*	On input, an N-element array containing an initial estimate to the solution. On output, the updated solution estimate. N is the number of variables.
`out`	*fvec*	An M-element array that, on output, will contain the values of each equation as evaluated at the variable values given in `x`.
`out`	*ib*	An optional output, that if provided, allows the caller to obtain iteration performance statistics.
`out`	*err*	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. The possible error codes returned will likely vary from solver to solver.

Definition at line 291 of file nonlin_types.f90.

The documentation for this interface was generated from the following file:

- /home/jason/Documents/Code/nonlin/src/nonlin_types.f90

5.19 nonlin_types::nonlin_solver_1var Interface Reference

Describes the interface of a solver for an equation of one variable.

Private Member Functions

- subroutine **nonlin_solver_1var** (this, fcn, x, lim, f, ib, err)

5.19.1 Detailed Description

Describes the interface of a solver for an equation of one variable.

Parameters

`in,out`	*this*	The equation_solver_1var-based object.
`in`	*fcn*	The fcn1var_helper object containing the equation to solve.
`in,out`	*x*	On input the initial guess at the solution. On output the updated solution estimate.
`in`	*lim*	A value_pair object defining the search limits.
`out`	*f*	An optional parameter used to return the function residual as computed at `x`.
`out`	*ib*	An optional output, that if provided, allows the caller to obtain iteration performance statistics.
`out`	*err*	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. The possible error codes returned will likely vary from solver to solver.

Definition at line 324 of file nonlin_types.f90.

The documentation for this interface was generated from the following file:

- /home/jason/Documents/Code/nonlin/src/nonlin_types.f90

5.20 nonlin_solve::quasi_newton_solver Type Reference

Defines a quasi-Newton type solver based upon Broyden's method.

Inheritance diagram for nonlin_solve::quasi_newton_solver:

Collaboration diagram for nonlin_solve::quasi_newton_solver:

Public Member Functions

- procedure, public solve => qns_solve

    Solves the system of equations.
- procedure, public get_jacobian_interval => qns_get_jac_interval

    Gets the number of iterations that may pass before forcing a recalculation of the Jacobian matrix.
- procedure, public set_jacobian_interval => qns_set_jac_interval

    Sets the number of iterations that may pass before forcing a recalculation of the Jacobian matrix.

Private Attributes

- integer(i32) m_jdelta = 5

    The number of iterations that may pass between Jacobian calculation.

5.20.1 Detailed Description

Defines a quasi-Newton type solver based upon Broyden's method.

Definition at line 54 of file nonlin_solve.f90.

The documentation for this type was generated from the following file:

- /home/jason/Documents/Code/nonlin/src/nonlin_solve.f90

5.21 nonlin_c_binding::solver_control Type Reference

Defines a set of solver control information.

Public Attributes

- integer(i32) max_evals

    The maximum number of function evaluations allowed.
- real(dp) fcn_tolerance

    The convergence criteria on function values.
- real(dp) var_tolerance

    The convergence criteria on change in variable values.
- real(dp) grad_tolerance

    The convergence criteria for the slope of the gradient vector.
- logical(c_bool) print_status

    Controls whether iteration status is printed.

5.21.1 Detailed Description

Defines a set of solver control information.

Definition at line 66 of file nonlin_c_binding.f90.

The documentation for this type was generated from the following file:

- /home/jason/Documents/Code/nonlin/src/nonlin_c_binding.f90

5.22 nonlin_types::value_pair Type Reference

Defines a pair of numeric values.

Private Attributes

- real(dp) x1
  Value 1.
- real(dp) x2
  Value 2.

5.22.1 Detailed Description

Defines a pair of numeric values.

Definition at line 261 of file nonlin_types.f90.

The documentation for this type was generated from the following file:

- /home/jason/Documents/Code/nonlin/src/nonlin_types.f90

5.23 nonlin_types::vecfcn Interface Reference

Describes an M-element vector-valued function of N-variables.

Private Member Functions

- subroutine **vecfcn** (x, f)

5.23.1 Detailed Description

Describes an M-element vector-valued function of N-variables.

Parameters

in	*x*	An N-element array containing the independent variables.
out	*f*	An M-element array that, on output, contains the values of the M functions.

Definition at line 74 of file nonlin_types.f90.

The documentation for this interface was generated from the following file:

- /home/jason/Documents/Code/nonlin/src/nonlin_types.f90

5.24 nonlin_types::vecfcn_helper Type Reference

Defines a type capable of encapsulating a system of nonlinear equations of the form: F(X) = 0.

Inheritance diagram for nonlin_types::vecfcn_helper:

Public Member Functions

- procedure, public set_fcn => vfh_set_fcn

  Establishes a pointer to the routine containing the system of equations to solve.
- procedure, public set_jacobian => vfh_set_jac

  Establishes a pointer to the routine for computing the Jacobian matrix of the system of equations. If no routine is defined, the Jacobian matrix will be computed numerically (this is the default state).
- procedure, public is_fcn_defined => vfh_is_fcn_defined

  Tests if the pointer to the subroutine containing the system of equations to solve has been assigned.
- procedure, public is_jacobian_defined => vfh_is_jac_defined

  Tests if the pointer to the subroutine containing the system of equations to solve has been assigned.
- procedure, public fcn => vfh_fcn

  Executes the routine containing the system of equations to solve. No action is taken if the pointer to the subroutine has not been defined.
- procedure, public jacobian => vfh_jac_fcn

  Executes the routine containing the Jacobian matrix if supplied. If not supplied, the Jacobian is computed via finite differences.
- procedure, public get_equation_count => vfh_get_nfcn

  Gets the number of equations in this system.
- procedure, public get_variable_count => vfh_get_nvar

  Gets the number of variables in this system.

Private Attributes

- procedure(vecfcn), pointer, nopass m_fcn => null()

  A pointer to the target vecfcn routine.
- procedure(jacobianfcn), pointer, nopass m_jac => null()

  A pointer to the jacobian routine - null if no routine is supplied.
- integer(i32) m_nfcn = 0

  The number of functions in m_fcn.
- integer(i32) m_nvar = 0

  The number of variables in m_fcn.

5.24.1 Detailed Description

Defines a type capable of encapsulating a system of nonlinear equations of the form: F(X) = 0.

Definition at line 97 of file nonlin_types.f90.

The documentation for this type was generated from the following file:

- /home/jason/Documents/Code/nonlin/src/nonlin_types.f90

Index