nonlin

1.1.2

Generated by Doxygen 1.8.11

Contents

1 Main Page

1.1 Introduction

NONLIN is a library that provides routines to compute the solutions to systems of nonlinear equations.

Author

Jason Christopherson

Version

1.1.2

2 Modules Index

2.1 Modules List

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

3 Data Type Index

3.1 Class Hierarchy

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

4 Data Type Index

4.1 Data Types List

Here are the data types with brief descriptions:

5 Module Documentation

5.1 nonlin_c_binding Module Reference

nonlin_c_binding

Data Types

- type c_polynomial

  A C compatible type encapsulating a polynomial object.
- interface cfcn1var

  The C-friendly interface to fcn1var.
- type cfcn1var_helper

  A type allowing the use of cfcn1var in the solver codes.
- interface cfcnnvar

  The C-friendly interface to fcnnvar.
- type cfcnnvar_helper

  A type allowing the use of cfcnnvar in the solver codes.
- interface cgradientfcn

  A C-friendly interface to gradientfcn.
- interface cjacobianfcn

  The C-friendly interface to jacobianfcn.
- interface cvecfcn

The C-friendly interface to vecfcn.
- type cvecfcn_helper

  A type 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 procedure 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 cfnh_set_fcn (this, fcn, nvar)

  Establishes a poitner to the routine containing the equation to solve.
- pure logical function cfnh_is_fcn_defined (this)

  Tests if the pointer to the procedure containing the system of equations to solve has been assigned.
- real(dp) function cfnh_fcn (this, x)

  Executes the routine containing the function to evaluate.
- subroutine cfnh_set_grad (this, fcn)

  Establishes a pointer to the routine containing the gradient vector of the function.
- pure logical function cfnh_is_grad_defined (this)

  Tests if the pointer to the routine containing the gradient has been assigned.
- 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.
- subroutine set_nonlin_defaults (tol)

  Sets defaults for the solver_control type.
- subroutine set_nonlin_ls_defaults (ls)

  Sets defaults for the line_search_control type.

- subroutine nelder_mead_c (fcn, nvar, x, f, smplx, tol, ib, err)

  Utilizes the Nelder-Mead simplex method for finding a minimum value of the specified function.
- subroutine bfgs_c (fcn, grad, nvar, x, f, tol, lsearch, ib, err)

  Utilizes the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm for finding a minimum value of the specified function.
- subroutine alloc_polynomial (obj, order)

  Initializes a new polynomial object.
- subroutine free_polynomial (obj)

  Frees resources held by a c_polynomial object.
- subroutine get_polynomial (obj, poly)

  Retrieves the polynomial object from the C compatible c_polynomial data structure.
- integer(i32) function get_polynomial_order_c (poly)

  Gets the order of the polynomial.
- subroutine fit_polynomial (poly, n, x, y, order, err)

  Fits a polynomial of the specified order to a data set.
- subroutine fit_polynomial_thru_zero (poly, n, x, y, order, err)

  Fits a polynomial of the specified order that passes through zero to a data set.
- subroutine evaluate_polynomial (poly, n, x, y)

  Evaluates a polynomial at the specified points.
- subroutine evaluate_polynomial_cmplx (poly, n, x, y)

  Evaluates a polynomial at the specified points.
- subroutine polynomial_roots_c (poly, n, rts, err)

  Computes all the roots of a polynomial by computing the eigenvalues of the polynomial companion matrix.
- real(dp) function get_polynomial_coefficient (poly, ind, err)

  Gets the requested polynomial coefficient by index. The coefficient index is established as follows: c(1) + c(2) ∗ x + c(3) ∗ x∗∗2 + ... c(n) ∗ x∗∗n-1.
- subroutine set_polynomial_set_coefficient (poly, ind, x, err)

  Sets the requested polynomial coefficient by index. The coefficient index is established as follows: c(1) + c(2) ∗ x + c(3) ∗ x∗∗2 + ... c(n) ∗ x∗∗n-1.
- subroutine polynomial_add (p1, p2, rst)

  Adds two polynomials.
- subroutine polynomial_subtract (p1, p2, rst)

  Subtracts two polynomials.
- subroutine polynomial_multiply (p1, p2, rst)

  Multiplies two polynomials.
- subroutine polynomial_copy (src, dst)

  Copies the contents of one polynomial object to another.

5.1.1 Detailed Description

nonlin_c_binding

Purpose

Provides C bindings to the nonlin library.

5.1.2 Function/Subroutine Documentation

5.1.2.1 subroutine nonlin_c_binding::alloc_polynomial (type(c_polynomial), intent(out) *obj*, integer(i32), intent(in), value *order*)

Initializes a new polynomial object.

Parameters

out	*obj*	The c_polynomial object to initialize.
in	*order*	The order of the polynomial. This value must be > 0.

Definition at line 922 of file nonlin_c_binding.f90.

5.1.2.2 subroutine nonlin_c_binding::bfgs_c (type(c_funptr), intent(in), value *fcn*, type(c_funptr), intent(in), value *grad*, integer(i32), intent(in), value *nvar*, real(dp), dimension(nvar), intent(inout) *x*, real(dp), intent(out) *f*, type(solver_control), intent(in) *tol*, type(c_ptr), intent(in), value *lsearch*, type(iteration_behavior), intent(out) *ib*, type(errorhandler), intent(inout) *err*)

Utilizes the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm for finding a minimum value of the specified function.

Parameters

in	*fcn*	A pointer to the routine containing the function on which to operate.
in	*grad*	An optional pointer to a routine capable of computing the gradient of the function contained within `fcn`. If no routine is supplied (NULL), the solver will numerically estimate the gradient.
in	*nvar*	The dimension of the problem (number of variables).
in,out	*x*	On input, the initial guess at the optimal point. On output, the updated optimal point estimate.
out	*f*	An optional output, that if provided, returns the value of the function 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*	The errorhandler 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 `x` is not appropriately sized for the problem as defined in `fcn`. • NL_OUT_OF_MEMORY_ERROR: Occurs if there is insufficient memory available. • NL_CONVERGENCE_ERROR: Occurs if the algorithm cannot converge within the allowed number of iterations.

Definition at line 856 of file nonlin_c_binding.f90.

5.1.2.3 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(errorhandler), intent(inout) *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.

Parameters

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*	The errorhandler 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 424 of file nonlin_c_binding.f90.

5.1.2.4 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 218 of file nonlin_c_binding.f90.

5.1.2.5 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 233 of file nonlin_c_binding.f90.

5.1.2.6 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 245 of file nonlin_c_binding.f90.

5.1.2.7 real(dp) function nonlin_c_binding::cfnh_fcn (class(cfcnnvar_helper), intent(in) *this*, real(dp), dimension(:), intent(in) *x*)

Executes the routine containing the function to evaluate.

Parameters

in	*this*	The cfcnnvar_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 366 of file nonlin_c_binding.f90.

5.1.2.8 pure logical function nonlin_c_binding::cfnh_is_fcn_defined (class(cfcnnvar_helper), intent(in) *this*)

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

Parameters

in	*this*	The cfcnnvar_helper object.

Returns

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

Definition at line 353 of file nonlin_c_binding.f90.

5.1.2.9 pure logical function nonlin_c_binding::cfnh_is_grad_defined (class(cfcnnvar_helper), intent(in) *this*)

Tests if the pointer to the routine containing the gradient has been assigned.

Parameters

in	*this*	The cfcnnvar_helper object.

Returns

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

Definition at line 395 of file nonlin_c_binding.f90.

5.1.2.10 subroutine nonlin_c_binding::cfnh_set_fcn (class(cfcnnvar_helper), intent(inout) *this*, procedure(cfcnnvar), intent(in), pointer *fcn*, integer(i32), intent(in) *nvar*)

Establishes a poitner to the routine containing the equation to solve.

Parameters

in,out	*this*	The cfcnnvar_helper object.
in	*fcn*	The function pointer.
in	*nvar*	The number of variables.

Definition at line 337 of file nonlin_c_binding.f90.

5.1.2.11 subroutine nonlin_c_binding::cfnh_set_grad (class(cfcnnvar_helper), intent(inout) *this*, procedure(cgradientfcn), intent(in), pointer *fcn*)

Establishes a pointer to the routine containing the gradient vector of the function.

Parameters

in,out	*this*	The cfcnnvar_helper object.
in	*fcn*	The pointer to the gradient routine.

Definition at line 383 of file nonlin_c_binding.f90.

5.1.2.12 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 291 of file nonlin_c_binding.f90.

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

Tests if the pointer to the procedure 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 277 of file nonlin_c_binding.f90.

5.1.2.14 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 322 of file nonlin_c_binding.f90.

5.1.2.15 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 261 of file nonlin_c_binding.f90.

5.1.2.16 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 310 of file nonlin_c_binding.f90.

5.1.2.17 subroutine nonlin_c_binding::evaluate_polynomial (type(c_polynomial), intent(in) *poly,* integer(i32), intent(in), value *n,* real(dp), dimension(n), intent(in) *x,* real(dp), dimension(n), intent(out) *y*)

Evaluates a polynomial at the specified points.

Parameters

in	*poly*	The c_polynomial object.
in	*n*	The number of points to evaluate.
in	*x*	An N-element array containing the points at which to evaluate the polynomial.
out	*y*	An N-element array where the resulting polynomial outputs will be written.

Definition at line 1100 of file nonlin_c_binding.f90.

5.1.2.18 subroutine nonlin_c_binding::evaluate_polynomial_cmplx (type(c_polynomial), intent(in) *poly*, integer(i32), intent(in), value *n*, complex(dp), dimension(n), intent(in) *x*, complex(dp), dimension(n), intent(out) *y*)

Evaluates a polynomial at the specified points.

Parameters

in	*poly*	The c_polynomial object.
in	*n*	The number of points to evaluate.
in	*x*	An N-element array containing the points at which to evaluate the polynomial.
out	*y*	An N-element array where the resulting polynomial outputs will be written.

Definition at line 1126 of file nonlin_c_binding.f90.

5.1.2.19 subroutine nonlin_c_binding::fit_polynomial (type(c_polynomial), intent(out) *poly*, integer(i32), intent(in), value *n*, real(dp), dimension(n), intent(in) *x*, real(dp), dimension(n), intent(inout) *y*, integer(i32), intent(in), value *order*, type(errorhandler), intent(inout) *err*)

Fits a polynomial of the specified order to a data set.

Parameters

out	*poly*	The c_polynomial object to initialize.
in	*n*	The size of the arrays.
in	*x*	An N-element array containing the independent variable data points. Notice, must be N > `order`.
in,out	*y*	On input, an N-element array containing the dependent variable data points. On output, the contents are overwritten.
in	*order*	The order of the polynomial (must be >= 1).
in,out	*err*	The errorhandler 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_INPUT_ERROR: Occurs if a zero or negative polynomial order was specified, or if order is too large for the data set. • NL_OUT_OF_MEMORY_ERROR: Occurs if insufficient memory is available. • NL_ARRAY_SIZE_ERROR: Occurs if `x` and `y` are different sizes.

Definition at line 1022 of file nonlin_c_binding.f90.

5.1.2.20 subroutine nonlin_c_binding::fit_polynomial_thru_zero (type(c_polynomial), intent(out) *poly*, integer(i32), intent(in), value *n*, real(dp), dimension(n), intent(in) *x*, real(dp), dimension(n), intent(inout) *y*, integer(i32), intent(in), value *order*, type(errorhandler), intent(inout) *err*)

Fits a polynomial of the specified order that passes through zero to a data set.

Parameters

out	*poly*	The c_polynomial object to initialize.
in	*n*	The size of the arrays.
in	*x*	An N-element array containing the independent variable data points. Notice, must be N > `order`.
in,out	*y*	On input, an N-element array containing the dependent variable data points. On output, the contents are overwritten.
in	*order*	The order of the polynomial (must be >= 1).
in,out	*err*	The errorhandler 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_INPUT_ERROR: Occurs if a zero or negative polynomial order was specified, or if order is too large for the data set. • NL_OUT_OF_MEMORY_ERROR: Occurs if insufficient memory is available. • NL_ARRAY_SIZE_ERROR: Occurs if `x` and `y` are different sizes.

Definition at line 1066 of file nonlin_c_binding.f90.

5.1.2.21 subroutine nonlin_c_binding::free_polynomial (type(c_polynomial), intent(inout), target *obj*)

Frees resources held by a c_polynomial object.

Parameters

in,out	*obj*	The c_polynomial object.

Definition at line 941 of file nonlin_c_binding.f90.

5.1.2.22 subroutine nonlin_c_binding::get_polynomial (type(c_polynomial), intent(in), target *obj*, type(polynomial), intent(out), pointer *poly*)

Retrieves the polynomial object from the C compatible c_polynomial data structure.

Parameters

in	*obj*	The C compatible c_polynomial data structure.
out	*poly*	The resulting polynomials object.

Definition at line 965 of file nonlin_c_binding.f90.

5.1.2.23 real(dp) function nonlin_c_binding::get_polynomial_coefficient (type(c_polynomial), intent(in) *poly*, integer(i32), intent(in), value *ind*, type(errorhandler), intent(inout) *err*)

Gets the requested polynomial coefficient by index. The coefficient index is established as follows: c(1) + c(2) ∗ x + c(3) ∗ x∗∗2 + ... c(n) ∗ x∗∗n-1.

Parameters

in	*poly*	The c_polynomial object.
in	*ind*	The polynomial coefficient index (0 < ind <= order + 1).
in,out	*err*	The errorhandler 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_INPUT_ERROR: Occurs if the requested index is less than or equal to zero, or if the requested index exceeds the number of polynomial coefficients.

Definition at line 1200 of file nonlin_c_binding.f90.

5.1.2.24 integer(i32) function nonlin_c_binding::get_polynomial_order_c (type(c_polynomial), intent(in) *poly*)

Gets the order of the polynomial.

Parameters

in	*poly*	The c_polynomial object.

Returns

The order of the polynomial object.

Definition at line 988 of file nonlin_c_binding.f90.

5.1.2.25 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(errorhandler), intent(inout) *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`.

Parameters

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*	The errorhandler 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 685 of file nonlin_c_binding.f90.

5.1.2.26 subroutine nonlin_c_binding::nelder_mead_c (type(c_funptr), intent(in), value *fcn*, integer(i32), intent(in), value *nvar*, real(dp), dimension(nvar), intent(inout) *x*, real(dp), intent(out) *f*, type(c_ptr), intent(in), value *smplx*, type(solver_control), intent(in) *tol*, type(iteration_behavior), intent(out) *ib*, type(errorhandler), intent(inout) *err*)

Utilizes the Nelder-Mead simplex method for finding a minimum value of the specified function.

Parameters

in	*fcn*	A pointer to the routine containing the function on which to operate.
in	*nvar*	The dimension of the problem (number of variables).
in,out	*x*	On input, the initial guess at the optimal point. On output, the updated optimal point estimate.
out	*f*	An optional output, that if provided, returns the value of the function at `x`.
in	*smplx*	An optional NVAR-by-(NVAR + 1) matrix, that if supplied provides an initial simplex geometry (each column is a vertex location). If not provided (NULL), the solver generates its own estimate of a starting simplex geometry.
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*	The errorhandler 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 `x` is not appropriately sized for the problem as defined in `fcn`. • NL_OUT_OF_MEMORY_ERROR: Occurs if there is insufficient memory available. • NL_CONVERGENCE_ERROR: Occurs if the algorithm cannot converge within the allowed number of iterations.

Definition at line 788 of file nonlin_c_binding.f90.

5.1.2.27 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(errorhandler), intent(inout) *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.
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*	The errorhandler 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 593 of file nonlin_c_binding.f90.

5.1.2.28 subroutine nonlin_c_binding::polynomial_add (type(c_polynomial), intent(in) *p1,* type(c_polynomial), intent(in) *p2,* type(c_polynomial), intent(out) *rst*)

Adds two polynomials.

Parameters

in	*p1*	The left-hand-side argument.

Parameters

in	*p2*	The right-hand-side argument.
out	*rst*	The resulting polynomial.

Definition at line 1267 of file nonlin_c_binding.f90.

5.1.2.29 subroutine nonlin_c_binding::polynomial_copy (type(c_polynomial), intent(in) *src*, type(c_polynomial), intent(out) *dst*)

Copies the contents of one polynomial object to another.

Parameters

in	*src*	The source polynomial object.
out	*dst*	The destination polynomial.

Definition at line 1337 of file nonlin_c_binding.f90.

5.1.2.30 subroutine nonlin_c_binding::polynomial_multiply (type(c_polynomial), intent(in) *p1*, type(c_polynomial), intent(in) *p2*, type(c_polynomial), intent(out) *rst*)

Multiplies two polynomials.

Parameters

in	*p1*	The left-hand-side argument.
in	*p2*	The right-hand-side argument.
out	*rst*	The resulting polynomial.

Definition at line 1315 of file nonlin_c_binding.f90.

5.1.2.31 subroutine nonlin_c_binding::polynomial_roots_c (type(c_polynomial), intent(in) *poly*, integer(i32), intent(in), value *n*, complex(dp), dimension(n), intent(out) *rts*, type(errorhandler), intent(inout) *err*)

Computes all the roots of a polynomial by computing the eigenvalues of the polynomial companion matrix.

Parameters

in	*poly*	The c_polynomial object.
in	*n*	The size of `rts`. This value should be the same as the order of the polynomial.
out	*rts*	An N-element array where the roots of the polynomial will be written.
in,out	*err*	The errorhandler 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. • LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available. • LA_CONVERGENCE_ERROR: Occurs if the algorithm failed to converge.

Definition at line 1159 of file nonlin_c_binding.f90.

5.1.2.32 subroutine nonlin_c_binding::polynomial_subtract (type(c_polynomial), intent(in) *p1*, type(c_polynomial), intent(in) *p2*, type(c_polynomial), intent(out) *rst*)

Subtracts two polynomials.

Parameters

in	*p1*	The left-hand-side argument.
in	*p2*	The right-hand-side argument.
out	*rst*	The resulting polynomial.

Definition at line 1291 of file nonlin_c_binding.f90.

5.1.2.33 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(errorhandler), intent(inout) *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.
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.

Parameters

in	*err*	The errorhandler 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 498 of file nonlin_c_binding.f90.

5.1.2.34 subroutine nonlin_c_binding::set_nonlin_defaults (type(solver_control), intent(out) *tol*)

Sets defaults for the solver_control type.

Parameters

out	*tol*	The solver_control object.

Definition at line 728 of file nonlin_c_binding.f90.

5.1.2.35 subroutine nonlin_c_binding::set_nonlin_ls_defaults (type(line_search_control), intent(out) *ls*)

Sets defaults for the line_search_control type.

Parameters

out	*ls*	The line_search_control object.

Definition at line 745 of file nonlin_c_binding.f90.

5.1.2.36 subroutine nonlin_c_binding::set_polynomial_set_coefficient (type(c_polynomial), intent(inout) *poly*, integer(i32), intent(in), value *ind*, real(dp), intent(in), value *x*, type(errorhandler), intent(inout) *err*)

Sets the requested polynomial coefficient by index. The coefficient index is established as follows: c(1) + c(2) ∗ x + c(3) ∗ x∗∗2 + ... c(n) ∗ x∗∗n-1.

Parameters

`in,out`	*poly*	The c_polynomial object.
`in`	*ind*	The polynomial coefficient index (0 < ind <= order + 1).
`in`	*x*	The polynomial coefficient.
`in,out`	*err*	The errorhandler 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_INPUT_ERROR: Occurs if the requested index is less than or equal to zero, or if the requested index exceeds the number of polynomial coefficients.

Definition at line 1239 of file nonlin_c_binding.f90.

5.2 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.

5.2.1 Detailed Description

nonlin_least_squares

Purpose

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

5.2.2 Function/Subroutine Documentation

5.2.2.1 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 734 of file nonlin_least_squares.f90.

5.2.2.2 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 567 of file nonlin_least_squares.f90.

5.2.2.3 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 837 of file nonlin_least_squares.f90.

5.2.2.4 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 48 of file nonlin_least_squares.f90.

5.2.2.5 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 67 of file nonlin_least_squares.f90.

5.2.2.6 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(inout), 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.
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.

Example 1

The following code provides an example of how to solve a system of N equations of N unknonwns using the

Levenberg-Marquardt method.

```
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

    ! Define the function
    fcn => fcn1
    call obj%set_fcn(fcn, 2, 2)

    ! 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), ")"
contains
    ! System of Equations:
    !
    ! 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
end program
```

The above program returns the following results.

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

Example 2

```
program example
    use linalg_constants, only : dp, i32
    use nonlin_types, only : vecfcn_helper, vecfcn
    use nonlin_least_squares, only : least_squares_solver
    implicit none

    ! Local Variables
    type(vecfcn_helper) :: obj
    procedure(vecfcn), pointer :: fcn
    type(least_squares_solver) :: solver
    real(dp) :: x(4), f(21) ! There are 4 coefficients and 21 data points

    ! Locate the routine containing the equations to solve
    fcn => fcns
    call obj%set_fcn(fcn, 21, 4)

    ! Define an initial guess
    x = 1.0d0 ! Equivalent to x = [1.0d0, 1.0d0, 1.0d0, 1.0d0]

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

    ! Display the output
    print "(AF12.8)", "c1: ", x(1)
    print "(AF12.8)", "c2: ", x(2)
    print "(AF12.8)", "c3: ", x(3)
    print "(AF12.8)", "c4: ", x(4)
    print "(AF9.5)", "Max Residual: ", maxval(abs(f))

contains
    ! The function containing the data to fit
    subroutine fcns(x, f)
        ! Arguments
        real(dp), intent(in), dimension(:) :: x  ! Contains the coefficients
        real(dp), intent(out), dimension(:) :: f
```

```
        ! Local Variables
        real(dp), dimension(21) :: xp, yp

        ! Data to fit (21 data points)
        xp = [0.0d0, 0.1d0, 0.2d0, 0.3d0, 0.4d0, 0.5d0, 0.6d0, 0.7d0, 0.8d0, &
            0.9d0, 1.0d0, 1.1d0, 1.2d0, 1.3d0, 1.4d0, 1.5d0, 1.6d0, 1.7d0, &
            1.8d0, 1.9d0, 2.0d0]
        yp = [1.216737514d0, 1.250032542d0, 1.305579195d0, 1.040182335d0, &
            1.751867738d0, 1.109716707d0, 2.018141531d0, 1.992418729d0, &
            1.807916923d0, 2.078806005d0, 2.698801324d0, 2.644662712d0, &
            3.412756702d0, 4.406137221d0, 4.567156645d0, 4.999550779d0, &
            5.652854194d0, 6.784320119d0, 8.307936836d0, 8.395126494d0, &
            10.30252404d0]

        ! We'll apply a cubic polynomial model to this data:
        ! y = c1 * x**3 + c2 * x**2 + c3 * x + c4
        f = x(1) * xp**3 + x(2) * xp**2 + x(3) * xp + x(4) - yp

        ! For reference, the data was generated by adding random errors to
        ! the following polynomial: y = x**3 - 0.3 * x**2 + 1.2 * x + 0.3
    end subroutine
end program
```

The above program returns the following results.

```
c1:   1.06476276
c2:  -0.12232029
c3:   0.44661345
c4:   1.18661422
Max Residual:   0.50636
```

See Also

- Wikipedia
- MINPACK (Wikipedia)

Definition at line 237 of file nonlin_least_squares.f90.

5.3 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_mimo (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.
- subroutine ls_search_miso (this, fcn, xold, grad, dir, x, 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.
- pure real(dp) function min_backtrack_search (mode, f0, f, f1, alam, alam1, slope)

  Minimizes either the quadratic or cubic representation for a backtracking-type line search.
- subroutine, public limit_search_vector (x, lim)

  Provides a means of scaling the length of the search direction vector.

5.3.1 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.

5.3.2 Function/Subroutine Documentation

5.3.2.1 subroutine, public nonlin_linesearch::limit_search_vector (real(dp), dimension(:), intent(inout) *x*, real(dp), intent(in) *lim*)

Provides a means of scaling the length of the search direction vector.

Parameters

in,out	*x*	On input, the search direction vector. On output, the search direction vector limited in length to that specified by `lim`. If the vector is originally shorter than the limit length, no change is made.
in	*lim*	The length limit value.

Definition at line 643 of file nonlin_linesearch.f90.

5.3.2.2 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 146 of file nonlin_linesearch.f90.

5.3.2.3 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 93 of file nonlin_linesearch.f90.

5.3.2.4 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 119 of file nonlin_linesearch.f90.

5.3.2.5 subroutine nonlin_linesearch::ls_search_mimo (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(inout), 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`.

Parameters

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 209 of file nonlin_linesearch.f90.

5.3.2.6 subroutine nonlin_linesearch::ls_search_miso (class(line_search), intent(in) *this,* class(fcnnvar_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), intent(in), optional *fold,* real(dp), intent(out), optional *fx,* type(iteration_behavior), optional *ib,* class(errors), intent(inout), 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.
in	*fold*	An optional input that provides the function value at `xold`. If not provided, `fcn` is evalauted at `xold`.
out	*fx*	The value of the function as evaluated at `x`.
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_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 412 of file nonlin_linesearch.f90.

5.3.2.7 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 160 of file nonlin_linesearch.f90.

5.3.2.8 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 105 of file nonlin_linesearch.f90.

5.3.2.9 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 133 of file nonlin_linesearch.f90.

5.3.2.10 pure real(dp) function nonlin_linesearch::min_backtrack_search (integer(i32), intent(in) *mode*, real(dp), intent(in) *f0*, real(dp), intent(in) *f*, real(dp), intent(in) *f1*, real(dp), intent(in) *alam*, real(dp), intent(in) *alam1*, real(dp), intent(in) *slope*) [private]

Minimizes either the quadratic or cubic representation for a backtracking-type line search.

Parameters

in	*mode*	Set to 1 to apply the quadratic model; else, any other value will apply the cubic model.
in	*f0*	The previous function value.
in	*f*	The current function value.
in	*f1*	The predicted function value.
in	*alam*	The step length scaling factor at `f`.
in	*alam1*	The step length scaling factor at `f1`.
in	*slope*	The slope of the direction vector.

Returns

The new step length scaling factor.

Definition at line 592 of file nonlin_linesearch.f90.

5.4 nonlin_optimize Module Reference

nonlin_optimize

Data Types

- type bfgs

  Defines a Broyden–Fletcher–Goldfarb–Shanno (BFGS) solver for minimization of functions of multiple variables.
- type line_search_optimizer

  A class describing equation optimizers that use a line search algorithm to improve convergence behavior.
- type nelder_mead

  Defines a solver based upon Nelder and Mead's simplex algorithm for minimization of functions of multiple variables.

Functions/Subroutines

- subroutine nm_solve (this, fcn, x, fout, ib, err)

  Utilizes the Nelder-Mead simplex method for finding a minimum value of the specified function.
- real(dp) function nm_extrapolate (this, fcn, y, pcent, ihi, fac, neval, work)

  Extrapolates by the specified factor through the simplex across from the largest point. If the extrapolation results in a better estimate, the current high point is replaced with the new estimate.
- pure real(dp) function, dimension(:,:), allocatable nm_get_simplex (this)

  Gets an N-by-(N+1) matrix containing the current simplex.
- subroutine nm_set_simplex (this, x)

  Sets an N-by-(N+1) matrix as the current simplex. Notice, if this matrix is different in size from the problem dimensionallity, the Nelder-Mead routine will replace it with an appropriately sized matrix.
- pure real(dp) function nm_get_size (this)

  Gets the size of the initial simplex that will be utilized by the Nelder-Mead algorithm in the event that the user does not supply a simplex geometry, or if the user supplies an invalid simplex geometry.
- subroutine nm_set_size (this, x)

  Sets the size of the initial simplex that will be utilized by the Nelder-Mead algorithm in the event that the user does not supply a simplex geometry, or if the user supplies an invalid simplex geometry.
- subroutine lso_get_line_search (this, ls)

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

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

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

  Tests to see if a line search module is defined.
- pure logical function lso_get_use_search (this)

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

  Sets a value determining if a line-search should be employed.
- pure real(dp) function lso_get_var_tol (this)

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

  Sets the convergence on change in variable tolerance.
- subroutine bfgs_solve (this, fcn, x, fout, ib, err)

  Utilizes the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm for finding a minimum value of the specified function.

5.4.1 Detailed Description

nonlin_optimize

Purpose

To provide various optimization routines.

5.4.2 Function/Subroutine Documentation

5.4.2.1 subroutine nonlin_optimize::bfgs_solve (class(bfgs), intent(inout) *this,* class(fcnnvar_helper), intent(in) *fcn,* real(dp), dimension(:), intent(inout) *x,* real(dp), intent(out), optional *fout,* type(iteration_behavior), optional *ib,* class(errors), intent(inout), optional, target *err*) [private]

Utilizes the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm for finding a minimum value of the specified function.

Parameters

`in,out`	*this*	The bfgs_mead object.
`in`	*fcn*	The fcnnvar_helper object containing the equation to optimize.
`in,out`	*x*	On input, the initial guess at the optimal point. On output, the updated optimal point estimate.
`out`	*fout*	An optional output, that if provided, returns the value of the function 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 `x` is not appropriately sized for the problem as defined in `fcn`. • NL_OUT_OF_MEMORY_ERROR: Occurs if there is insufficient memory available. • NL_CONVERGENCE_ERROR: Occurs if the algorithm cannot converge within the allowed number of iterations.

Usage

The following example illustrates how to find the minimum of Rosenbrock's function using this BFGS solver.

```
program example
    use linalg_constants, only : dp, i32
    use nonlin_optimize, only : bfgs
    use nonlin_types, only : fcnnvar, fcnnvar_helper,
      iteration_behavior
    implicit none

    ! Local Variables
    type(bfgs) :: solver
    type(fcnnvar_helper) :: obj
    procedure(fcnnvar), pointer :: fcn
    real(dp) :: x(2), fout
    type(iteration_behavior) :: ib

    ! Initialization
    fcn => rosenbrock
    call obj%set_fcn(fcn, 2)

    ! Define an initial guess - the solution is (1, 1)
    call random_number(x)

    ! Call the solver
    call solver%solve(obj, x, fout, ib)

    ! Display the output
    print '(AF8.5AF8.5A)', "Rosenbrock Minimum: (", x(1), ", ", x(2), ")"
    print '(AE9.3)', "Function Value: ", fout
    print '(AI0)', "Iterations: ", ib%iter_count
    print '(AI0)', "Function Evaluations: ", ib%fcn_count
contains
    ! Rosenbrock's Function
    function rosenbrock(x) result(f)
        real(dp), intent(in), dimension(:) :: x
        real(dp) :: f
        f = 1.0d2 * (x(2) - x(1)**2)**2 + (x(1) - 1.0d0)**2
    end function
end
```

The above program yields the following output:

```
Rosenbrock Minimum: ( 1.00000,  0.99999)
Function Value: 0.200E-10
Iterations: 47
Function Evaluations: 70
```

See Also

- Wikipedia - BFGS Methods
- Wikipedia - Quasi-Newton Methods
- minFunc

Definition at line 731 of file nonlin_optimize.f90.

5.4.2.2 subroutine nonlin_optimize::lso_get_line_search (class(line_search_optimizer), intent(in) *this*, class(line_search), intent(out), allocatable *ls*) [private]

Gets the line search module.

Parameters

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

Definition at line 565 of file nonlin_optimize.f90.

5.4.2.3 pure logical function nonlin_optimize::lso_get_use_search (class(line_search_optimizer), intent(in) *this*) [private]

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

Parameters

in	*this*	The line_search_optimizer object.

Returns

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

Definition at line 611 of file nonlin_optimize.f90.

5.4.2.4 pure real(dp) function nonlin_optimize::lso_get_var_tol (class(line_search_optimizer), intent(in) *this*) [private]

Gets the convergence on change in variable tolerance.

Parameters

in	*this*	The line_search_optimizer object.

Returns

The tolerance value.

Definition at line 633 of file nonlin_optimize.f90.

5.4.2.5 pure logical function nonlin_optimize::lso_is_line_search_defined (class(line_search_optimizer), intent(in) *this*) [private]

Tests to see if a line search module is defined.

Parameters

in	*this*	The line_search_optimizer object.

Returns

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

Definition at line 600 of file nonlin_optimize.f90.

5.4.2.6 subroutine nonlin_optimize::lso_set_default (class(line_search_optimizer), intent(inout) *this*) [private]

Establishes a default line_search object for the line search module.

Parameters

in,out	*this*	The line_search_optimizer object.

Definition at line 589 of file nonlin_optimize.f90.

5.4.2.7 subroutine nonlin_optimize::lso_set_line_search (class(line_search_optimizer), intent(inout) *this*, class(line_search), intent(in) *ls*) [private]

Sets the line search module.

Parameters

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

Definition at line 577 of file nonlin_optimize.f90.

5.4.2.8 subroutine nonlin_optimize::lso_set_use_search (class(line_search_optimizer), 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_optimizer object.
in	*x*	Set to true if a line search should be used; else, false.

Definition at line 622 of file nonlin_optimize.f90.

5.4.2.9 subroutine nonlin_optimize::lso_set_var_tol (class(line_search_optimizer), intent(inout) *this,* real(dp), intent(in) *x*) [private]

Sets the convergence on change in variable tolerance.

Parameters

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

Definition at line 644 of file nonlin_optimize.f90.

5.4.2.10 real(dp) function nonlin_optimize::nm_extrapolate (class(nelder_mead), intent(inout) *this,* class(fcnnvar_helper), intent(in) *fcn,* real(dp), dimension(:), intent(inout) *y,* real(dp), dimension(:), intent(inout) *pcent,* integer(i32), intent(in) *ihi,* real(dp), intent(in) *fac,* integer(i32), intent(inout) *neval,* real(dp), dimension(:), intent(out) *work*) [private]

Extrapolates by the specified factor through the simplex across from the largest point. If the extrapolation results in a better estimate, the current high point is replaced with the new estimate.

Parameters

in,out	*this*	The nelder_mead object.
in	*fcn*	The function to evaluate.
in,out	*y*	An array containing the function values at each vertex.
in,out	*pcent*	An array containing the centroid of vertex position information.
in	*ihi*	The index of the largest magnitude vertex.
in,out	*neval*	The number of function evaluations.
out	*work*	An N-element workspace array where N is the number of dimensions of the problem.

Returns

The new function estimate.

Definition at line 447 of file nonlin_optimize.f90.

5.4.2.11 pure real(dp) function, dimension(:,:), allocatable nonlin_optimize::nm_get_simplex (class(nelder_mead), intent(in) *this*) [private]

Gets an N-by-(N+1) matrix containing the current simplex.

Parameters

in	*this*	The nelder_mead object.

Returns

The N-by-(N+1) matrix containing the simplex. Each vertex of the simplex is stored as its own column of this matrix.

Definition at line 494 of file nonlin_optimize.f90.

5.4.2.12 pure real(dp) function nonlin_optimize::nm_get_size (class(nelder_mead), intent(in) *this*) [private]

Gets the size of the initial simplex that will be utilized by the Nelder-Mead algorithm in the event that the user does not supply a simplex geometry, or if the user supplies an invalid simplex geometry.

Parameters

in	*this*	The nelder_mead object.

Returns

The size of the simplex (length of an edge).

Definition at line 539 of file nonlin_optimize.f90.

5.4.2.13 subroutine nonlin_optimize::nm_set_simplex (class(nelder_mead), intent(inout) *this*, real(dp), dimension(:,:) *x*) [private]

Sets an N-by-(N+1) matrix as the current simplex. Notice, if this matrix is different in size from the problem dimensionallity, the Nelder-Mead routine will replace it with an appropriately sized matrix.

Parameters

in,out	*this*	The nelder_mead object.
in	*x*	The simplex matrix. Each column of the matrix must contain the coordinates of each vertex of the simplex.

Definition at line 514 of file nonlin_optimize.f90.

5.4.2.14 subroutine nonlin_optimize::nm_set_size (class(nelder_mead), intent(inout) *this*, real(dp), intent(in) *x*) [private]

Sets the size of the initial simplex that will be utilized by the Nelder-Mead algorithm in the event that the user does not supply a simplex geometry, or if the user supplies an invalid simplex geometry.

Parameters

in,out	*this*	The nelder_mead object.
in	*x*	The size of the simplex (length of an edge).

Definition at line 552 of file nonlin_optimize.f90.

5.4.2.15 subroutine nonlin_optimize::nm_solve (class(nelder_mead), intent(inout) *this*, class(fcnnvar_helper), intent(in) *fcn*, real(dp), dimension(:), intent(inout) *x*, real(dp), intent(out), optional *fout*, type(iteration_behavior), optional *ib*, class(errors), intent(inout), optional, target *err*) [private]

Utilizes the Nelder-Mead simplex method for finding a minimum value of the specified function.

Parameters

in,out	*this*	The nelder_mead object.
in	*fcn*	The fcnnvar_helper object containing the equation to optimize.
in,out	*x*	On input, the initial guess at the optimal point. On output, the updated optimal point estimate.
out	*fout*	An optional output, that if provided, returns the value of the function 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 x is not appropriately sized for the problem as defined in fcn. • NL_OUT_OF_MEMORY_ERROR: Occurs if there is insufficient memory available. • NL_CONVERGENCE_ERROR: Occurs if the algorithm cannot converge within the allowed number of iterations.

Usage

The following example illustrates how to find the minimum of Rosenbrock's function using this Nelder-Mead solver.

```
program example
    use linalg_constants, only : dp, i32
    use nonlin_optimize, only : nelder_mead
    use nonlin_types, only : fcnnvar, fcnnvar_helper,
      iteration_behavior
    implicit none

    ! Local Variables
    type(nelder_mead) :: solver
    type(fcnnvar_helper) :: obj
    procedure(fcnnvar), pointer :: fcn
    real(dp) :: x(2), fout
    type(iteration_behavior) :: ib

    ! Initialization
    fcn => rosenbrock
    call obj%set_fcn(fcn, 2)

    ! Define an initial guess - the solution is (1, 1)
    call random_number(x)

    ! Call the solver
    call solver%solve(obj, x, fout, ib)

    ! Display the output
    print '(AF8.5AF8.5A)', "Rosenbrock Minimum: (", x(1), ", ", x(2), ")"
    print '(AE9.3)', "Function Value: ", fout
    print '(AI0)', "Iterations: ", ib%iter_count
    print '(AI0)', "Function Evaluations: ", ib%fcn_count
contains
    ! Rosenbrock's Function
    function rosenbrock(x) result(f)
        real(dp), intent(in), dimension(:) :: x
        real(dp) :: f
        f = 1.0d2 * (x(2) - x(1)**2)**2 + (x(1) - 1.0d0)**2
    end function
end
```

The above program yields the following output:

```
Rosenbrock Minimum: ( 1.00000,  1.00000)
Function Value: 0.264E-12
Iterations: 59
Function Evaluations: 112
```

Remarks

The implementation of the Nelder-Mead algorithm presented here is a slight modification of the original work of Nelder and Mead. The Numerical Recipes implementation is also quite similar. In fact, the Numerical Recipes section relating to reflection, contraction, etc. is leveraged for this implemetation.

See Also

- Nelder, John A.; R. Mead (1965). "A simplex method for function minimization". Computer Journal. 7: 308–313.
- `Gao, Fuchang, Han, Lixing (2010). "Implementing the Nelder-Mead simplex algorithm with adaptive parameters."`
- `Wikipedia`
- `Numerical Recipes`

Definition at line 196 of file nonlin_optimize.f90.

5.5 nonlin_polynomials Module Reference

polynomials

Data Types

- interface assignment(=)

  Defines polynomial assignment.
- interface operator(∗)

  Defines polynomial multiplication.
- interface operator(+)

  Defines polynomial addition.
- interface operator(-)

  Defines polynomial subtraction.
- type polynomial

  Defines a polynomial, and associated routines for performing polynomial operations.

Functions/Subroutines

- subroutine init_poly (this, order, err)

  Initializes the polynomial instance, and sets all coefficients to zero.
- pure integer(i32) function get_poly_order (this)

  Returns the order of the polynomial object.
- subroutine poly_fit (this, x, y, order, err)

  Fits a polynomial of the specified order to a data set.
- subroutine poly_fit_thru_zero (this, x, y, order, err)

  Fits a polynomial of the specified order that passes through zero to a data set.
- elemental real(dp) function poly_eval_double (this, x)

  Evaluates a polynomial at the specified points.
- elemental complex(dp) function poly_eval_complex (this, x)

  Evaluates a polynomial at the specified points.
- pure real(dp) function, dimension(this%order(), this%order()) poly_companion_mtx (this)

  Returns the companion matrix for the polynomial.

- complex(dp) function, dimension(this%order()) poly_roots (this, err)

  Computes all the roots of a polynomial by computing the eigenvalues of the polynomial companion matrix.
- real(dp) function get_poly_coefficient (this, ind, err)

  Gets the requested polynomial coefficient by index. The coefficient index is established as follows: c(1) + c(2) ∗ x + c(3) ∗ x∗∗2 + ... c(n) ∗ x∗∗n-1.
- pure real(dp) function, dimension(this%order()+1) get_poly_coefficients (this)

  Gets an array containing all the coefficients of the polynomial. The coefficient index is established as follows: c(1) + c(2) ∗ x + c(3) ∗ x∗∗2 + ... c(n) ∗ x∗∗n-1.
- subroutine set_poly_coefficient (this, ind, c, err)

  Sets the requested polynomial coefficient by index. The coefficient index is established as follows: c(1) + c(2) ∗ x + c(3) ∗ x∗∗2 + ... c(n) ∗ x∗∗n-1.
- subroutine poly_equals (x, y)

  Assigns the contents of one polynomial to another.
- subroutine poly_dbl_equals (x, y)

  Assigns a number to each coefficient of the polynomial.
- type(polynomial) function poly_poly_add (x, y)

  Adds two polynomials.
- type(polynomial) function poly_poly_subtract (x, y)

  Subtracts two polynomials.
- type(polynomial) function poly_poly_mult (x, y)

  Multiplies two polynomials.
- type(polynomial) function poly_dbl_mult (x, y)

  Multiplies a polynomial by a scalar value.
- type(polynomial) function dbl_poly_mult (x, y)

  Multiplies a polynomial by a scalar value.

5.5.1 Detailed Description

polynomials

Purpose

Provides a means of defining and operating on polynomials.

5.5.2 Function/Subroutine Documentation

5.5.2.1 type(polynomial) function nonlin_polynomials::dbl_poly_mult (real(dp), intent(in) *x*, class(polynomial), intent(in) *y*) [private]

Multiplies a polynomial by a scalar value.

Parameters

in	*x*	The scalar value.
in	*y*	The polynomial.

Returns

The resulting polynomial.

Definition at line 931 of file nonlin_polynomials.f90.

5.5.2.2 real(dp) function nonlin_polynomials::get_poly_coefficient (class(polynomial), intent(in) *this*, integer(i32), intent(in) *ind*, class(errors), intent(inout), optional, target *err*) [private]

Gets the requested polynomial coefficient by index. The coefficient index is established as follows: c(1) + c(2) ∗ x + c(3) ∗ x∗∗2 + ... c(n) ∗ x∗∗n-1.

Parameters

in	*this*	The polynomial.
in	*ind*	The polynomial coefficient index (0 < ind <= order + 1).
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_INPUT_ERROR: Occurs if the requested index is less than or equal to zero, or if the requested index exceeds the number of polynomial coefficients.

Returns

The requested coefficient.

Definition at line 610 of file nonlin_polynomials.f90.

5.5.2.3 pure real(dp) function, dimension(this%order() + 1) nonlin_polynomials::get_poly_coefficients (class(polynomial), intent(in) *this*) [private]

Gets an array containing all the coefficients of the polynomial. The coefficient index is established as follows: c(1) + c(2) ∗ x + c(3) ∗ x∗∗2 + ... c(n) ∗ x∗∗n-1.

Parameters

in	*this*	The polynomial object.

Returns

The array of coefficients.

Definition at line 653 of file nonlin_polynomials.f90.

5.5.2.4 pure integer(i32) function nonlin_polynomials::get_poly_order (class(polynomial), intent(in) *this*) [private]

Returns the order of the polynomial object.

Parameters

in	*this*	The polynomial object.

Returns

The order of the polynomial. Returns -1 in the event no polynomial coefficients have been defined.

Definition at line 158 of file nonlin_polynomials.f90.

5.5.2.5 subroutine nonlin_polynomials::init_poly (class(polynomial), intent(inout) *this*, integer(i32), intent(in) *order*, class(errors), intent(inout), optional, target *err*) [private]

Initializes the polynomial instance, and sets all coefficients to zero.

Parameters

in,out	*this*	The polynomial object.
in	*order*	The order of the polynomial (must be >= 0).
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_INPUT_ERROR: Occurs if a zero or negative polynomial order was specified. • NL_OUT_OF_MEMORY_ERROR: Occurs if insufficient memory is available.

Definition at line 108 of file nonlin_polynomials.f90.

5.5.2.6 pure real(dp) function, dimension(this%order(), this%order()) nonlin_polynomials::poly_companion_mtx (class(polynomial), intent(in) *this*) [private]

Returns the companion matrix for the polynomial.

Parameters

in	*this*	The polynomial object.

Returns

The companion matrix.

See Also

- Wikipedia
- Wolfram MathWorld

Definition at line 484 of file nonlin_polynomials.f90.

5.5.2.7 subroutine nonlin_polynomials::poly_dbl_equals (class(polynomial), intent(inout) *x*, real(dp), intent(in) *y*) [private]

Assigns a number to each coefficient of the polynomial.

Parameters

in,out	*x*	The assignee.
in	*y*	The value to assign.

Definition at line 741 of file nonlin_polynomials.f90.

5.5.2.8 type(polynomial) function nonlin_polynomials::poly_dbl_mult (class(polynomial), intent(in) *x*, real(dp), intent(in) *y*) [private]

Multiplies a polynomial by a scalar value.

Parameters

in	*x*	The polynomial.
in	*y*	The scalar value.

Returns

The resulting polynomial.

Definition at line 907 of file nonlin_polynomials.f90.

5.5.2.9 subroutine nonlin_polynomials::poly_equals (class(polynomial), intent(inout) *x*, class(polynomial), intent(in) *y*) [private]

Assigns the contents of one polynomial to another.

Parameters

out	*x*	The assignee.

Definition at line 720 of file nonlin_polynomials.f90.

5.5.2.10 elemental complex(dp) function nonlin_polynomials::poly_eval_complex (class(polynomial), intent(in) *this*, complex(dp), intent(in) *x*) [private]

Evaluates a polynomial at the specified points.

Parameters

in	*this*	The polynomial object.
in	*x*	The value(s) at which to evaluate the polynomial.

Returns

The value(s) of the polynomial at `x`.

Definition at line 444 of file nonlin_polynomials.f90.

5.5.2.11 elemental real(dp) function nonlin_polynomials::poly_eval_double (class(polynomial), intent(in) *this*, real(dp), intent(in) *x*) [private]

Evaluates a polynomial at the specified points.

Parameters

in	*this*	The polynomial object.
in	*x*	The value(s) at which to evaluate the polynomial.

Returns

The value(s) of the polynomial at `x`.

Definition at line 407 of file nonlin_polynomials.f90.

5.5.2.12 subroutine nonlin_polynomials::poly_fit (class(polynomial), intent(inout) *this*, real(dp), dimension(:), intent(in) *x*, real(dp), dimension(:), intent(inout) *y*, integer(i32), intent(in) *order*, class(errors), intent(inout), optional, target *err*) [private]

Fits a polynomial of the specified order to a data set.

Parameters

in,out	*this*	The polynomial object.
in	*x*	An N-element array containing the independent variable data points. Notice, must be N > `order`.
in,out	*y*	On input, an N-element array containing the dependent variable data points. On output, the contents are overwritten.
in	*order*	The order of the polynomial (must be >= 1).
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_INPUT_ERROR: Occurs if a zero or negative polynomial order was specified, or if order is too large for the data set. • NL_OUT_OF_MEMORY_ERROR: Occurs if insufficient memory is available. • NL_ARRAY_SIZE_ERROR: Occurs if `x` and `y` are different sizes.

Usage

The following code provides an example of how to fit a polynomial to a set of data.

```
program example
    use linalg_constants, only : dp, i32
    use nonlin_polynomials

    ! Local Variables
    real(dp), dimension(21) :: xp, yp, yf, yc, err
    real(dp) :: res
    type(polynomial) :: p

    ! Data to fit
    xp = [0.0d0, 0.1d0, 0.2d0, 0.3d0, 0.4d0, 0.5d0, 0.6d0, 0.7d0, 0.8d0, &
```

```
        0.9d0, 1.0d0, 1.1d0, 1.2d0, 1.3d0, 1.4d0, 1.5d0, 1.6d0, 1.7d0, &
        1.8d0, 1.9d0, 2.0d0]
    yp = [1.216737514d0, 1.250032542d0, 1.305579195d0, 1.040182335d0, &
        1.751867738d0, 1.109716707d0, 2.018141531d0, 1.992418729d0, &
        1.807916923d0, 2.078806005d0, 2.698801324d0, 2.644662712d0, &
        3.412756702d0, 4.406137221d0, 4.567156645d0, 4.999550779d0, &
        5.652854194d0, 6.784320119d0, 8.307936836d0, 8.395126494d0, &
        10.30252404d0]

    ! Create a copy of yp as it will be overwritten in the fit command
    yc = yp

    ! Fit the polynomial
    call p%fit(xp, yp, 3)

    ! Evaluate the polynomial at xp, and then determine the residual
    yf = p%evaluate(xp)
    err = abs(yf - yc)
    res = maxval(err)

    ! Print out the coefficients
     print '(A)', "Polynomial Coefficients (c0 + c1*x + c2*x**2 + c3*x**3):"
     do i = 1, 4
         print '(AI0AF12.9)', "c", i - 1, " = ", p%get(i)
     end do
     print '(AE9.4)', "Residual: ", res
end program
```

The above program returns the following results.

```
Polynomial Coefficients (c0 + c1*x + c2*x**2 + c3*x**3):
c0 =  1.186614186
c1 =  0.446613631
c2 = -0.122320499
c3 =  1.064762822
Residual: .5064E+00
```

Definition at line 239 of file nonlin_polynomials.f90.

5.5.2.13 subroutine nonlin_polynomials::poly_fit_thru_zero (class(polynomial), intent(inout) *this*, real(dp), dimension(:), intent(in) *x*, real(dp), dimension(:), intent(inout) *y*, integer(i32), intent(in) *order*, class(errors), intent(inout), optional, target *err*) [private]

Fits a polynomial of the specified order that passes through zero to a data set.

Parameters

in,out	*this*	The polynomial object.
in	*x*	An N-element array containing the independent variable data points. Notice, must be N > order.
in,out	*y*	On input, an N-element array containing the dependent variable data points. On output, the contents are overwritten.
in	*order*	The order of the polynomial (must be >= 1).
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_INPUT_ERROR: Occurs if a zero or negative polynomial order was specified, or if order is too large for the data set. • NL_OUT_OF_MEMORY_ERROR: Occurs if insufficient memory is available. • NL_ARRAY_SIZE_ERROR: Occurs if x and y are different sizes.

Definition at line 330 of file nonlin_polynomials.f90.

5.5.2.14 type(polynomial) function nonlin_polynomials::poly_poly_add (class(polynomial), intent(in) *x*, class(polynomial), intent(in) *y*) [private]

Adds two polynomials.

Parameters

in	*x*	The left-hand-side argument.
in	*y*	The right-hand-side argument.

Returns

The resulting polynomial.

Definition at line 763 of file nonlin_polynomials.f90.

5.5.2.15 type(polynomial) function nonlin_polynomials::poly_poly_mult (class(polynomial), intent(in) *x*, class(polynomial), intent(in) *y*) [private]

Multiplies two polynomials.

Parameters

in	*x*	The left-hand-side argument.
in	*y*	The right-hand-side argument.

Returns

The resulting polynomial.

Definition at line 877 of file nonlin_polynomials.f90.

5.5.2.16 type(polynomial) function nonlin_polynomials::poly_poly_subtract (class(polynomial), intent(in) *x*, class(polynomial), intent(in) *y*) [private]

Subtracts two polynomials.

Parameters

in	*x*	The left-hand-side argument.
in	*y*	The right-hand-side argument.

Returns

The resulting polynomial.

Definition at line 820 of file nonlin_polynomials.f90.

5.5.2.17 complex(dp) function, dimension(this%order()) nonlin_polynomials::poly_roots (class(polynomial), intent(in) *this*, class(errors), intent(inout), optional, target *err*) [private]

Computes all the roots of a polynomial by computing the eigenvalues of the polynomial companion matrix.

Parameters

in	*this*	The polynomial object.
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_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available. • NL_CONVERGENCE_ERROR: Occurs if the algorithm failed to converge.

Usage

The following code provides an example of how to compute the roots of a polynomial. This examples uses a tenth order polynomial; however, this process is applicable to any order.

```
program example
    use linalg_constants, only : dp, i32
    use nonlin_polynomials

    ! Parameters
    integer(i32), parameter :: order = 10

    ! Local Variables
    integer(i32) :: i
    type(polynomial) :: p
    real(dp), dimension(order+1) :: coeff
    complex(dp), allocatable, dimension(:) :: rts, sol

    ! Define the polynomial
    call random_number(coeff)
    call p%initialize(order)
    do i = 1, size(coeff)
        call p%set(i, coeff(i))
    end do

    ! Compute the roots via the polynomial routine
    rts = p%roots()

    ! Compute the value of the polynomial at each root and ensure it
    ! is sufficiently close to zero.
    sol = p%evaluate(rts)
    do i = 1, size(sol)
        print '(AE9.3AE9.3A)', "(", real(sol(i)), ", ", aimag(sol(i)), ")"
    end do
end program
```

The above program returns the following results.

```
(-.466E-14, -.161E-14)
(-.466E-14, 0.161E-14)
(-.999E-15, 0.211E-14)
(-.999E-15, -.211E-14)
(0.444E-15, 0.108E-14)
(0.444E-15, -.108E-14)
(-.144E-14, -.433E-14)
(-.144E-14, 0.433E-14)
(0.644E-14, -.100E-13)
(0.644E-14, 0.100E-13)
```

Definition at line 569 of file nonlin_polynomials.f90.

5.5.2.18 subroutine nonlin_polynomials::set_poly_coefficient (class(polynomial), intent(inout) *this*, integer(i32), intent(in) *ind*, real(dp), intent(in) *c*, class(errors), intent(inout), optional, target *err*) [private]

Sets the requested polynomial coefficient by index. The coefficient index is established as follows: c(1) + c(2) ∗ x + c(3) ∗ x∗∗2 + ... c(n) ∗ x∗∗n-1.

Parameters

in,out	*this*	The polynomial.
in	*ind*	The polynomial coefficient index (0 < ind <= order + 1).
in	*c*	The polynomial coefficient.
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_INPUT_ERROR: Occurs if the requested index is less than or equal to zero, or if the requested index exceeds the number of polynomial coefficients.

Definition at line 679 of file nonlin_polynomials.f90.

5.6 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.

5.6.1 Detailed Description

nonlin_solve

Purpose

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

5.6.2 Function/Subroutine Documentation

5.6.2.1 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(inout), 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.

```
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

    ! Define the function
    fcn => fcn1
    call obj%set_fcn(fcn)

    ! 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
contains
    ! 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
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 986 of file nonlin_solve.f90.

5.6.2.2 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 95 of file nonlin_solve.f90.

5.6.2.3 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 141 of file nonlin_solve.f90.

5.6.2.4 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 130 of file nonlin_solve.f90.

5.6.2.5 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 119 of file nonlin_solve.f90.

5.6.2.6 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 107 of file nonlin_solve.f90.

5.6.2.7 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 152 of file nonlin_solve.f90.

5.6.2.8 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(inout), 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.

```
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

    ! Define the function
    fcn => fcn1
    call obj%set_fcn(fcn, 2, 2)

    ! 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), ")"
contains
    ! System of Equations:
    !
    ! 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
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 671 of file nonlin_solve.f90.

5.6.2.9 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 565 of file nonlin_solve.f90.

5.6.2.10 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 577 of file nonlin_solve.f90.

5.6.2.11 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(inout), 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.

```
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

        ! Define the function
        fcn => fcn1
        call obj%set_fcn(fcn, 2, 2)

        ! 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), ")"
    contains
        ! System of Equations:
        !
        ! 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
    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 249 of file nonlin_solve.f90.

5.6.2.12 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 1209 of file nonlin_solve.f90.

5.7 nonlin_types Module Reference

nonlin_types

Data Types

- type equation_optimizer

    A base class for optimization of an equation of multiple variables.
- 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.
- interface fcnnvar

    Describes a function of N variables.
- type fcnnvar_helper

    Defines a type capable of encapsulating an equation of N variables.
- interface gradientfcn

    Describes a routine capable of computing the gradient vector of an equation of N variables.
- 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_optimize

    Describes the interface of a routine for optimizing an equation of N variables.
- 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.
- real(dp) function fnh_fcn (this, x)

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

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

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

  Gets the number of variables in this system.
- subroutine fnh_set_grad (this, fcn)

  Establishes a pointer to the routine containing the gradient vector of the function.
- pure logical function fnh_is_grad_defined (this)

  Tests if the pointer to the routine containing the gradient has been assigned.
- subroutine fnh_grad_fcn (this, x, g, fv, err)

  Executes the routine containing the gradient, if supplied. If not supplied, the gradient is computed via finite differences.
- 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.
- pure integer(i32) function oe_get_max_eval (this)

  Gets the maximum number of function evaluations allowed.
- subroutine oe_set_max_eval (this, n)

  Sets the maximum number of function evaluations allowed.
- pure real(dp) function oe_get_tol (this)

  Gets the tolerance on convergence.
- subroutine oe_set_tol (this, x)

  Sets the tolerance on convergence.
- pure logical function oe_get_print_status (this)

  Gets a logical value determining if iteration status should be printed.
- subroutine oe_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 dp = real64

  Defines a double-precision (64-bit) floating-point type.
- integer, parameter, public i32 = int32

  Defines a 32-bit signed integer type.
- 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 = LA_OUT_OF_MEMORY_ERROR

  An error denoting that there is insufficient memory available.
- integer, parameter, public nl_invalid_operation_error = LA_INVALID_OPERATION_ERROR

  An error resulting from an invalid operation.
- integer, parameter, public nl_convergence_error = LA_CONVERGENCE_ERROR

  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.

5.7.1 Detailed Description

nonlin_types

Purpose

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

5.7.2 Function/Subroutine Documentation

5.7.2.1 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 1074 of file nonlin_types.f90.

5.7.2.2 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 1051 of file nonlin_types.f90.

5.7.2.3 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 1119 of file nonlin_types.f90.

5.7.2.4 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 1096 of file nonlin_types.f90.

5.7.2.5 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 1085 of file nonlin_types.f90.

5.7.2.6 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 1063 of file nonlin_types.f90.

5.7.2.7 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 1131 of file nonlin_types.f90.

5.7.2.8 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 1107 of file nonlin_types.f90.

5.7.2.9 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 958 of file nonlin_types.f90.

5.7.2.10 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 1002 of file nonlin_types.f90.

5.7.2.11 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 935 of file nonlin_types.f90.

5.7.2.12 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 1025 of file nonlin_types.f90.

5.7.2.13 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 980 of file nonlin_types.f90.

5.7.2.14 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 969 of file nonlin_types.f90.

5.7.2.15 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 1013 of file nonlin_types.f90.

5.7.2.16 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 947 of file nonlin_types.f90.

5.7.2.17 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 1037 of file nonlin_types.f90.

5.7.2.18 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 991 of file nonlin_types.f90.

5.7.2.19 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 734 of file nonlin_types.f90.

5.7.2.20 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 749 of file nonlin_types.f90.

5.7.2.21 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 761 of file nonlin_types.f90.

5.7.2.22 real(dp) function nonlin_types::fnh_fcn (class(fcnnvar_helper), intent(in) *this*, real(dp), dimension(:), intent(in) *x*) [private]

Executes the routine containing the function to evaluate.

Parameters

in	*this*	The fcnnvar_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 776 of file nonlin_types.f90.

5.7.2.23 pure integer(i32) function nonlin_types::fnh_get_nvar (class(fcnnvar_helper), intent(in) *this*) [private]

Gets the number of variables in this system.

Parameters

in	*this*	The fcnnvar_helper object.

Returns

The number of variables.

Definition at line 817 of file nonlin_types.f90.

5.7.2.24 subroutine nonlin_types::fnh_grad_fcn (class(fcnnvar_helper), intent(in) *this*, real(dp), dimension(:), intent(inout) *x*, real(dp), dimension(:), intent(out) *g*, real(dp), intent(in), optional *fv*, integer(i32), intent(out), optional *err*) [private]

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

Parameters

in	*this*	The fcnnvar_helper object.
in,out	*x*	An N-element array containing the independent variables defining the point about which the derivatives will be calculated. This array is restored upon output.
out	*g*	An N-element array where the gradient will be written upon output.
in	*fv*	An optional input providing the function value at x.
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.

Definition at line 864 of file nonlin_types.f90.

5.7.2.25 pure logical function nonlin_types::fnh_is_fcn_defined (class(fcnnvar_helper), intent(in) *this*) [private]

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

Parameters

in	*this*	The fcnnvar_helper object.

Returns

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

Definition at line 791 of file nonlin_types.f90.

5.7.2.26 pure logical function nonlin_types::fnh_is_grad_defined (class(fcnnvar_helper), intent(in) *this*) [private]

Tests if the pointer to the routine containing the gradient has been assigned.

Parameters

in	*this*	The fcnnvar_helper object.

Returns

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

Definition at line 841 of file nonlin_types.f90.

5.7.2.27 subroutine nonlin_types::fnh_set_fcn (class(fcnnvar_helper), intent(inout) *this*, procedure(fcnnvar), intent(in), pointer *fcn*, integer(i32), intent(in) *nvar*) [private]

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

Parameters

in,out	*this*	The fcnnvar_helper object.
in	*fcn*	The function pointer.
in	*nvar*	The number of variables in the function.

Definition at line 804 of file nonlin_types.f90.

5.7.2.28 subroutine nonlin_types::fnh_set_grad (class(fcnnvar_helper), intent(inout) *this*, procedure(gradientfcn), intent(in), pointer *fcn*) [private]

Establishes a pointer to the routine containing the gradient vector of the function.

Parameters

in,out	*this*	The fcnnvar_helper object.
in	*fcn*	The pointer to the gradient routine.

Definition at line 829 of file nonlin_types.f90.

5.7.2.29 pure integer(i32) function nonlin_types::oe_get_max_eval (class(equation_optimizer), intent(in) *this*) [private]

Gets the maximum number of function evaluations allowed.

Parameters

in	*this*	The equation_optimizer object.

Returns

The maximum number of function evaluations.

Definition at line 1144 of file nonlin_types.f90.

5.7.2.30 pure logical function nonlin_types::oe_get_print_status (class(equation_optimizer), intent(in) *this*) [private]

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

Parameters

in	*this*	The equation_optimizer object.

Returns

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

Definition at line 1189 of file nonlin_types.f90.

5.7.2.31 pure real(dp) function nonlin_types::oe_get_tol (class(equation_optimizer), intent(in) *this*) [private]

Gets the tolerance on convergence.

Parameters

in	*this*	The equation_optimizer object.

Returns

The convergence tolerance.

Definition at line 1166 of file nonlin_types.f90.

5.7.2.32 subroutine nonlin_types::oe_set_max_eval (class(equation_optimizer), intent(inout) *this*, integer(i32), intent(in) *n*) [private]

Sets the maximum number of function evaluations allowed.

Parameters

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

Definition at line 1155 of file nonlin_types.f90.

5.7.2.33 subroutine nonlin_types::oe_set_print_status (class(equation_optimizer), 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_optimizer object.
in	*x*	True if the iteration status should be printed; else, false.

Definition at line 1201 of file nonlin_types.f90.

5.7.2.34 subroutine nonlin_types::oe_set_tol (class(equation_optimizer), intent(inout) *this*, real(dp), intent(in) *x*) [private]

Sets the tolerance on convergence.

Parameters

in,out	*this*	The equation_optimizer object.
in	*x*	The convergence tolerance.

Definition at line 1177 of file nonlin_types.f90.

5.7.2.35 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 1217 of file nonlin_types.f90.

5.7.2.36 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 552 of file nonlin_types.f90.

5.7.2.37 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 708 of file nonlin_types.f90.

5.7.2.38 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 719 of file nonlin_types.f90.

5.7.2.39 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 526 of file nonlin_types.f90.

5.7.2.40 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 538 of file nonlin_types.f90.

5.7.2.41 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.
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 587 of file nonlin_types.f90.

5.7.2.42 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 498 of file nonlin_types.f90.

5.7.2.43 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 514 of file nonlin_types.f90.

6 Data Type Documentation

6.1 nonlin_polynomials::assignment(=) Interface Reference

Defines polynomial assignment.

Private Member Functions

- subroutine poly_equals (x, y)

  Assigns the contents of one polynomial to another.
- subroutine poly_dbl_equals (x, y)

  Assigns a number to each coefficient of the polynomial.

6.1.1 Detailed Description

Defines polynomial assignment.

Definition at line 32 of file nonlin_polynomials.f90.

6.1.2 Member Function/Subroutine Documentation

6.1.2.1 subroutine nonlin_polynomials::assignment(=)::poly_dbl_equals (class(polynomial), intent(inout) *x*, real(dp), intent(in) *y*) [private]

Assigns a number to each coefficient of the polynomial.

Parameters

in,out	*x*	The assignee.
in	*y*	The value to assign.

Definition at line 741 of file nonlin_polynomials.f90.

6.1.2.2 subroutine nonlin_polynomials::assignment(=)::poly_equals (class(polynomial), intent(inout) *x*, class(polynomial), intent(in) *y*) [private]

Assigns the contents of one polynomial to another.

Parameters

out	*x*	The assignee.

Definition at line 720 of file nonlin_polynomials.f90.

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

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

6.2 nonlin_optimize::bfgs Type Reference

Defines a Broyden–Fletcher–Goldfarb–Shanno (BFGS) solver for minimization of functions of multiple variables.

Inheritance diagram for nonlin_optimize::bfgs:

Collaboration diagram for nonlin_optimize::bfgs:

Public Member Functions

- procedure, public solve => bfgs_solve

  Optimizes the equation.

6.2.1 Detailed Description

Defines a Broyden–Fletcher–Goldfarb–Shanno (BFGS) solver for minimization of functions of multiple variables.

Definition at line 96 of file nonlin_optimize.f90.

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

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

6.3 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.

6.3.1 Detailed Description

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

Definition at line 79 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

6.4 nonlin_c_binding::c_polynomial Type Reference

A C compatible type encapsulating a polynomial object.

Public Attributes

- type(c_ptr) ptr

  A pointer to the polynomial object.
- integer(i32) n

  The size of the polynomial object, in bytes.

6.4.1 Detailed Description

A C compatible type encapsulating a polynomial object.

Definition at line 130 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

6.5 nonlin_c_binding::cfcn1var Interface Reference

The C-friendly interface to fcn1var.

Public Member Functions

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

6.5.1 Detailed Description

The C-friendly interface to fcn1var.

Parameters

in	*x*	The independent variable.

Returns

The value of the function `x`.

Definition at line 29 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

6.6 nonlin_c_binding::cfcn1var_helper Type Reference

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

Inheritance diagram for nonlin_c_binding::cfcn1var_helper:

6.7 nonlin_c_binding::cfcnnvar Interface Reference

The C-friendly interface to fcnnvar.

Public Member Functions

- real(dp) function **cfcnnvar** (nvar, x)

6.7.1 Detailed Description

The C-friendly interface to fcnnvar.

Parameters

in	*nvar*	The number of variables.
in	*x*	An NVAR-element array containing the independent variables.

Returns

The value of the function at `x`.

Definition at line 68 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

6.8 nonlin_c_binding::cfcnnvar_helper Type Reference

A type allowing the use of cfcnnvar in the solver codes.

Inheritance diagram for nonlin_c_binding::cfcnnvar_helper:

Collaboration diagram for nonlin_c_binding::cfcnnvar_helper:

Public Member Functions

- procedure, public fcn => cfnh_fcn

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

  Tests if the pointer to the function has been assigned.
- procedure, public set_cfcn => cfnh_set_fcn

  Establishes a pointer to the routine containing the function.
- procedure, public set_cgradient_fcn => cfnh_set_grad

  Establishes a pointer to the routine containing the gradient vector of the function.
- procedure, public is_gradient_defined => cfnh_is_grad_defined

  Tests if the pointer to the routine containing the gradient has been assigned.

Public Attributes

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

  A pointer to the target cfcnnvar routine.
- procedure(cgradientfcn), pointer, nopass m_cgrad => null()

  A pointer to the gradient routine.

6.8.1 Detailed Description

A type allowing the use of cfcnnvar in the solver codes.

Definition at line 185 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

6.9 nonlin_c_binding::cgradientfcn Interface Reference

A C-friendly interface to gradientfcn.

Public Member Functions

- subroutine **cgradientfcn** (nvar, x, g)

6.9.1 Detailed Description

A C-friendly interface to gradientfcn.

Parameters

in	*nvar*	The number of variables.
in	*x*	An NVAR-element array containing the independent variables.
out	*g*	An NVAR-element array where the gradient vector will be written as output.

Definition at line 81 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

6.10 nonlin_c_binding::cjacobianfcn Interface Reference

The C-friendly interface to jacobianfcn.

Public Member Functions

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

6.10.1 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 56 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

6.11 nonlin_c_binding::cvecfcn Interface Reference

The C-friendly interface to vecfcn.

Public Member Functions

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

6.11.1 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 43 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

6.12 nonlin_c_binding::cvecfcn_helper Type Reference

A type 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.

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.

6.12.1 Detailed Description

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

Definition at line 156 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

6.13 nonlin_types::equation_optimizer Type Reference

A base class for optimization of an equation of multiple variables.

Inheritance diagram for nonlin_types::equation_optimizer:

Public Member Functions

- procedure, public get_max_fcn_evals => oe_get_max_eval

  Gets the maximum number of function evaluations allowed.
- procedure, public set_max_fcn_evals => oe_set_max_eval

  Sets the maximum number of function evaluations allowed.
- procedure, public get_tolerance => oe_get_tol

  Gets the tolerance on convergence.
- procedure, public set_tolerance => oe_set_tol

  Sets the tolerance on convergence.
- procedure, public get_print_status => oe_get_print_status

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

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

  Optimizes the equation.

Private Attributes

- integer(i32) m_maxeval = 500

  The maximum number of function evaluations allowed.
- real(dp) m_tol = 1.0d-12

  The error tolerance used to determine convergence.
- logical m_printstatus = .false.

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

6.13.1 Detailed Description

A base class for optimization of an equation of multiple variables.

Definition at line 352 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

6.14 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.

6.14.1 Detailed Description

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

Definition at line 260 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

6.15 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.

6.15.1 Detailed Description

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

Definition at line 305 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

6.16 nonlin_types::fcn1var Interface Reference

Describes a function of one variable.

Private Member Functions

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

6.16.1 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 90 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

6.17 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.

6.17.1 Detailed Description

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

Definition at line 188 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

6.18 nonlin_types::fcnnvar Interface Reference

Describes a function of N variables.

Private Member Functions

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

6.18.1 Detailed Description

Describes a function of N variables.

Parameters

in	*x*	An N-element array containing the independent variables.

Returns

The value of the function at `x`.

Definition at line 122 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

6.19 nonlin_types::fcnnvar_helper Type Reference

Defines a type capable of encapsulating an equation of N variables.

Inheritance diagram for nonlin_types::fcnnvar_helper:

Public Member Functions

- procedure, public fcn => fnh_fcn

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

  Tests if the pointer to the function has been assigned.
- procedure, public set_fcn => fnh_set_fcn

  Establishes a pointer to the routine containing the function.
- procedure, public get_variable_count => fnh_get_nvar

  Gets the number of variables in this system.
- procedure, public set_gradient_fcn => fnh_set_grad

  Establishes a pointer to the routine containing the gradient vector of the function.
- procedure, public is_gradient_defined => fnh_is_grad_defined

  Tests if the pointer to the routine containing the gradient has been assigned.
- procedure, public gradient => fnh_grad_fcn

  Computes the gradient of the function.

Private Attributes

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

  A pointer to the target fcnnvar routine.
- procedure(gradientfcn), pointer, nopass m_grad => null()

  A pointer to the gradient routine.
- integer(i32) m_nvar = 0

  The number of variables in m_fcn.

6.19.1 Detailed Description

Defines a type capable of encapsulating an equation of N variables.

Definition at line 206 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

6.20 nonlin_types::gradientfcn Interface Reference

Describes a routine capable of computing the gradient vector of an equation of N variables.

Private Member Functions

- subroutine **gradientfcn** (x, g)

6.20.1 Detailed Description

Describes a routine capable of computing the gradient vector of an equation of N variables.

Parameters

in	*x*	An N-element array containing the independent variables.
out	*g*	An N-element array where the gradient vector will be written as output.

Definition at line 134 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

6.21 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.
- integer(i32) gradient_count

    Specifies the number of gradient vector 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.

6.21.1 Detailed Description

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

Definition at line 236 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

6.22 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)

6.22.1 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 112 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

6.23 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.

6.23.1 Detailed Description

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

Definition at line 19 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

6.24 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.
- generic, public search => ls_search_mimo, ls_search_miso

  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 Member Functions

- procedure **ls_search_mimo**
- procedure **ls_search_miso**

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).

6.24.1 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

6.25 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).

6.25.1 Detailed Description

Defines a set of line search controls.

Definition at line 108 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

6.26 nonlin_optimize::line_search_optimizer Type Reference

A class describing equation optimizers that use a line search algorithm to improve convergence behavior.

Inheritance diagram for nonlin_optimize::line_search_optimizer:

Collaboration diagram for nonlin_optimize::line_search_optimizer:

Public Member Functions

- procedure, public get_line_search => lso_get_line_search

  Gets the line search module.
- procedure, public set_line_search => lso_set_line_search

  Sets the line search module.
- procedure, public set_default_line_search => lso_set_default

  Establishes a default line_search object for the line search module.
- procedure, public is_line_search_defined =>lso_is_line_search_defined

  Tests to see if a line search module is defined.
- procedure, public get_use_line_search => lso_get_use_search

  Gets a value determining if a line-search should be employed.
- procedure, public set_use_line_search => lso_set_use_search

  Sets a value determining if a line-search should be employed.
- procedure, public get_var_tolerance => lso_get_var_tol

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

  Sets the convergence on change in variable tolerance.

Private Attributes

- class(line_search), allocatable m_linesearch

  The line search object.
- logical m_uselinesearch = .true.

  Set to true if a line search should be used regardless of the status of m_lineSearch.
- real(dp) m_xtol = 1.0d-12

  The convergence criteria on change in variable.

6.26.1 Detailed Description

A class describing equation optimizers that use a line search algorithm to improve convergence behavior.

Definition at line 63 of file nonlin_optimize.f90.

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

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

6.27 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.

6.27.1 Detailed Description

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

Definition at line 27 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

6.28 nonlin_optimize::nelder_mead Type Reference

Defines a solver based upon Nelder and Mead's simplex algorithm for minimization of functions of multiple variables.

Inheritance diagram for nonlin_optimize::nelder_mead:

Collaboration diagram for nonlin_optimize::nelder_mead:

Public Member Functions

- procedure, public solve => nm_solve

  Optimizes the equation.
- procedure, public get_simplex => nm_get_simplex

  Gets an N-by-(N+1) matrix containing the current simplex.
- procedure, public set_simplex => nm_set_simplex

  Sets an N-by-(N+1) matrix containing the current simplex.
- procedure, public get_initial_size => nm_get_size

  Gets the size of the initial simplex.
- procedure, public set_initial_size => nm_set_size

  Sets the size of the initial simplex.

Private Member Functions

- procedure, private extrapolate => nm_extrapolate

  Extrapolates by the specified factor through the simplex across from the largest point. If the extrapolation results in a better estimate, the current high point is replaced with the new estimate.

Private Attributes

- real(dp), dimension(:,:), allocatable m_simplex

  The simplex vertices.
- real(dp) m_initsize = 1.0d0

  A scaling parameter used to define the size of the simplex in each coordinate direction.

6.28.1 Detailed Description

Defines a solver based upon Nelder and Mead's simplex algorithm for minimization of functions of multiple variables.

Definition at line 34 of file nonlin_optimize.f90.

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

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

6.29 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.

6.29.1 Detailed Description

Defines a Newton solver.

Definition at line 70 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

6.30 nonlin_types::nonlin_optimize Interface Reference

Describes the interface of a routine for optimizing an equation of N variables.

Private Member Functions

- subroutine **nonlin_optimize** (this, fcn, x, fout, ib, err)

6.30.1 Detailed Description

Describes the interface of a routine for optimizing an equation of N variables.

Parameters

in,out	*this*	The equation_optimizer-based object.
in	*fcn*	The fcnnvar_helper object containing the equation to optimize.
in,out	*x*	On input, the initial guess at the optimal point. On output, the updated optimal point estimate.
out	*fout*	An optional output, that if provided, returns the value of the function 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 469 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

6.31 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)

6.31.1 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 402 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

6.32 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)

6.32.1 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 435 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

6.33 nonlin_polynomials::operator(*) Interface Reference

Defines polynomial multiplication.

Private Member Functions

- type(polynomial) function poly_poly_mult (x, y)

  Multiplies two polynomials.
- type(polynomial) function poly_dbl_mult (x, y)

  Multiplies a polynomial by a scalar value.
- type(polynomial) function dbl_poly_mult (x, y)

  Multiplies a polynomial by a scalar value.

6.33.1 Detailed Description

Defines polynomial multiplication.

Definition at line 48 of file nonlin_polynomials.f90.

6.33.2 Member Function/Subroutine Documentation

6.33.2.1 type(polynomial) function nonlin_polynomials::operator(∗)::dbl_poly_mult (real(dp), intent(in) *x*, class(polynomial), intent(in) *y*) [private]

Multiplies a polynomial by a scalar value.

Parameters

in	*x*	The scalar value.
in	*y*	The polynomial.

Returns

The resulting polynomial.

Definition at line 931 of file nonlin_polynomials.f90.

6.33.2.2 type(polynomial) function nonlin_polynomials::operator(∗)::poly_dbl_mult (class(polynomial), intent(in) *x*, real(dp), intent(in) *y*) [private]

Multiplies a polynomial by a scalar value.

Parameters

in	*x*	The polynomial.
in	*y*	The scalar value.

Returns

The resulting polynomial.

Definition at line 907 of file nonlin_polynomials.f90.

6.33.2.3 type(polynomial) function nonlin_polynomials::operator(∗)::poly_poly_mult (class(polynomial), intent(in) *x*, class(polynomial), intent(in) *y*) [private]

Multiplies two polynomials.

Parameters

in	*x*	The left-hand-side argument.
in	*y*	The right-hand-side argument.

Returns

The resulting polynomial.

Definition at line 877 of file nonlin_polynomials.f90.

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

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

6.34 nonlin_polynomials::operator(+) Interface Reference

Defines polynomial addition.

Private Member Functions

- type(polynomial) function poly_poly_add (x, y)
  Adds two polynomials.

6.34.1 Detailed Description

Defines polynomial addition.

Definition at line 38 of file nonlin_polynomials.f90.

6.34.2 Member Function/Subroutine Documentation

6.34.2.1 type(polynomial) function nonlin_polynomials::operator(+)::poly_poly_add (class(polynomial), intent(in) *x*, class(polynomial), intent(in) *y*) [private]

Adds two polynomials.

Parameters

in	*x*	The left-hand-side argument.
in	*y*	The right-hand-side argument.

Returns

The resulting polynomial.

Definition at line 763 of file nonlin_polynomials.f90.

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

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

6.35 nonlin_polynomials::operator(-) Interface Reference

Defines polynomial subtraction.

Private Member Functions

- type(polynomial) function poly_poly_subtract (x, y)

    Subtracts two polynomials.

6.35.1 Detailed Description

Defines polynomial subtraction.

Definition at line 43 of file nonlin_polynomials.f90.

6.35.2 Member Function/Subroutine Documentation

6.35.2.1 type(polynomial) function nonlin_polynomials::operator(-)::poly_poly_subtract (class(polynomial), intent(in) *x*, class(polynomial), intent(in) *y*) [private]

Subtracts two polynomials.

Parameters

in	*x*	The left-hand-side argument.
in	*y*	The right-hand-side argument.

Returns

The resulting polynomial.

Definition at line 820 of file nonlin_polynomials.f90.

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

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

6.36 nonlin_polynomials::polynomial Type Reference

Defines a polynomial, and associated routines for performing polynomial operations.

Public Member Functions

- procedure, public initialize => init_poly

    Initializes the polynomial instance.
- procedure, public order => get_poly_order

    Returns the order of the polynomial object.
- procedure, public fit => poly_fit

    Fits a polynomial of the specified order to a data set.
- procedure, public fit_thru_zero => poly_fit_thru_zero

    Fits a polynomial of the specified order that passes through zero to a data set.
- generic, public evaluate => evaluate_real, evaluate_complex

    Evaluates a polynomial at the specified points.
- procedure, public companion_mtx => poly_companion_mtx

    Returns the companion matrix for the polynomial.
- procedure, public roots => poly_roots

    Computes all the roots of a polynomial.
- procedure, public get => get_poly_coefficient

    Gets the requested polynomial coefficient.
- procedure, public get_all => get_poly_coefficients

    Gets an array containing all the coefficients of the polynomial.
- procedure, public set => set_poly_coefficient

    Sets the requested polynomial coefficient by index.

Private Member Functions

- procedure **evaluate_real** => poly_eval_double
- procedure **evaluate_complex** => poly_eval_complex

Private Attributes

- real(dp), dimension(:), allocatable m_coeffs

    An array that contains the polynomial coefficients in ascending order.

6.36.1 Detailed Description

Defines a polynomial, and associated routines for performing polynomial operations.

Definition at line 59 of file nonlin_polynomials.f90.

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

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

6.37 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.

6.37.1 Detailed Description

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

Definition at line 53 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

6.38 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.

6.38.1 Detailed Description

Defines a set of solver control information.

Definition at line 93 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

6.39 nonlin_types::value_pair Type Reference

Defines a pair of numeric values.

Private Attributes

- real(dp) x1

  Value 1.
- real(dp) x2

  Value 2.

6.39.1 Detailed Description

Defines a pair of numeric values.

Definition at line 342 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

6.40 nonlin_types::vecfcn Interface Reference

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

Private Member Functions

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

6.40.1 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 101 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

6.41 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.

6.41.1 Detailed Description

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

Definition at line 146 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