Bachelor’s degree final project

v1.0

Generated by Doxygen 1.7.6.1

Mon Jun 23 2014 05:27:27

Contents

1 Approximations of invariant curves of diffeomorphisms

Author

Joan Gimeno

Date

Spring 2014

1.1 What is the utility of the current library ?

Assumes that we have a discret system like that:

$$\begin{cases} \bar{x} = f(x,\theta) + \varepsilon g(x,\theta) \\ \bar{\theta} = \theta + \omega, \end{cases}$$

where f, g are differential maps with the second variable on the torus.

A continuous map $x\colon \mathbb{T} \to \mathbb{R}$ is an invariant curve of the discret system if, and only if,

$$x(\theta+\omega) = f(x(\theta),\theta) + \varepsilon g(x(\theta),\theta) \quad \text{for all } \theta.$$

The value ω is known as the rotation number of the curve and it is common to suppose that is a irrational real number.

The library compute an approximation of some invariant curve of the discret system.

2 Module Index

2.1 Modules

Here is a list of all modules:

3 Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

5 Module Documentation

5.1 Householder

Implements the householder matrix, the householder vector and some operations with them like as premultiply a householder matrix with other matrix.

Defines

- #define _OPENMP_HOUSEHOLDER_DIMENSION 1e2

  Minimum value for using openmp in Householder.

Functions

- void ∗ householder_vector (size_t n, double ∗const x, double ∗const v)

  Derive the householder vector.
- void ∗ householder_matrix (size_t m, double ∗∗const A, double ∗const v, size_t idx)

  Derive the householder vector using a matrix.
- void ∗ premult_matrix (size_t m, size_t n, double ∗∗const A, double ∗∗const QR, double ∗const v, double ∗const w, size_t idx)

  Compute $P_v A$ where A and P_v are respectively m-by-n and n-by-n matrixs.
- void premult_vector (size_t n, double ∗const v, double ∗const b)

  Compute $P_v b$ where b and P_v are respectively n-by-1 and n-by-n matrixs.

- void postmult (size_t m, size_t n, double **const A, double *const v, double *const w)

  Compute AP_v where A and P_v are respectively m-by-n and n-by-n matrixs.

5.1.1 Detailed Description

Implements the householder matrix, the householder vector and some operations with them like as premultiply a householder matrix with other matrix. The householder matrix with householder vector v is

$$P_v = I - \frac{2}{v^t v} v v^t$$

And the householder vector of a vector $x \neq 0$ is $v = x + sign(x_1)||x||_2$.

5.1.2 Define Documentation

5.1.2.1 #define _OPENMP_HOUSEHOLDER_DIMENSION 1e2

Minimum value for using openmp in Householder.

5.1.3 Function Documentation

5.1.3.1 void * householder_matrix (size_t *m*, double **const *A*, double *const *v*, size_t *idx*)

Derive the householder vector using a matrix.

Parameters

in	*m*	Dimension of vector
in	*A*	Matrix m-by-n
in	*idx*	Index of matrix where is started to compute the vector householder
out	*v*	Householder vector

Return values

NULL	If the vector's elements of v[idx:] are all 0
1	Otherwise

Precondition

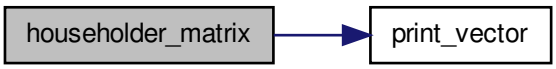
The dimension is $>$ 0 and idx is $<$ than number of columns of A

Postcondition

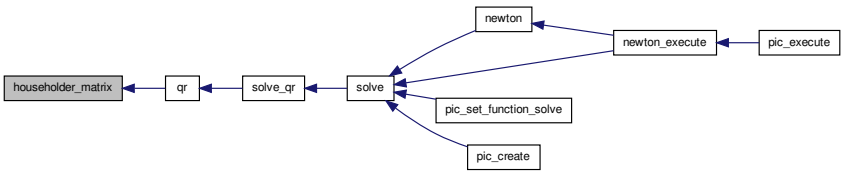
v[idx:] has been modified

References print_vector().

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.3.2 void * householder_vector (size_t *n*, double *const *x*, double *const *v*)

Derive the householder vector.

Parameters

in	*n*	Dimension of vector
in	*x*	Input vector
out	*v*	Householder vector

Return values

NULL	If the vector's elements of x are all 0
1	Otherwise

Precondition

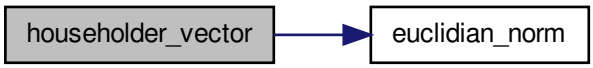
The dimension is not zero

Note

The vector pointers x and v can be the same

References euclidian_norm().

Here is the call graph for this function:



5.1.3.3 void postmult (size_t *m*, size_t *n*, double **const *A*, double *const *v*, double *const *w*)

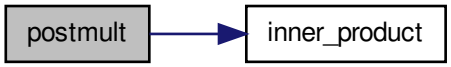
Compute AP_v where A and P_v are respectively m-by-n and n-by-n matrixs.

Parameters

in	*m*	Rows
in	*n*	Columns
in,out	*A*	Matrix m-by-n where will be saved the result
in	*v*	Householder vector of dimension n
in	*w*	Auxiliar vector of dimension m

References inner_product().

Here is the call graph for this function:



5.1.3.4 void premult_matrix (size_t *m*, size_t *n*, double **const *A*, double **const *QR*, double *const *v*, double *const *w*, size_t *idx*)

Compute P_vA where A and P_v are respectively m-by-n and n-by-n matrixs.

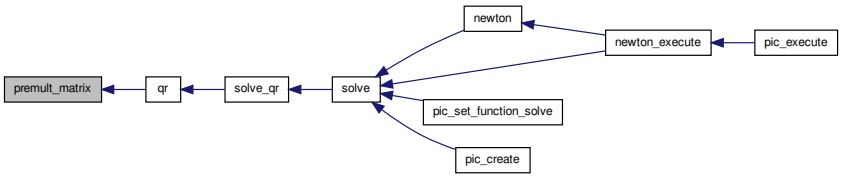
Parameters

in	*m*	Rows
in	*n*	Columns
in	*A*	Matrix m-by-n
in	*v*	Householder vector of dimension m
in	*w*	Auxiliar vector of dimension n
in	*idx*	Index of vectors where is started to compute
out	*QR*	Matrix m-by-n where will be saved the result

Precondition

n <= m and idx < m

Here is the caller graph for this function:



5.1.3.5 void premult_vector (size_t *n*, double *const *v*, double *const *b*)

Compute $P_v b$ where b and P_v are respectively n-by-1 and n-by-n matrixs.

Parameters

in	*m*	Rows
in	*n*	Columns
in	*v*	Householder vector of dimension n
in,out	*b*	Vector n-dimensional where will be saved the result

5.2 Solving a linear of equations system

Solving the system is find a vector *x* satisfying the linear equation Ax = b.

Functions

- void forward_substitution_ones (size_t n, double ∗∗const L, double ∗const x, double ∗const b)

  Forward substitution for lower triangular matrices (Lx = b) where the diagonal of L are 1's.
- void ∗ forward_substitution (size_t n, double ∗∗const L, double ∗const x, double ∗const b, double tol)

  Forward substitution for lower triangular matrices (Lx = b)
- void ∗ back_substitution (size_t n, double ∗∗const U, double ∗const x, double ∗const b, double tol)

  Back substitution for upper triangular matrices (Ux = b)
- void ∗ solve_qr (size_t m, size_t n, double ∗∗const A, double ∗const x, double ∗const b, double tol)

  QR solve linear system (QRx = b)
- void ∗ solve_lu (size_t m, size_t n, double ∗∗const A, double ∗const x, double ∗const b, double tol)

  LU solve linear system (LUx = b)
- void ∗ solve (size_t m, size_t n, double ∗∗const A, double ∗const x, double ∗const b, double tol, void ∗(∗s)(size_t, size_t, double ∗∗const, double ∗const, double ∗const, double))

  solve a linear system (Ax = b) with the pointer to a solve function
- solve_ptr get_solve_with_lu ()
- solve_ptr get_solve_with_qr ()

5.2.1 Detailed Description

Solving the system is find a vector *x* satisfying the linear equation Ax = b. The standard algorithm for solving a system of linear equations is based on Gaussian elimination with some modifications. Firstly, it is essential to avoid division by small numbers, which may lead to inaccurate results. This can be done by reordering the equations if necessary, a process known as pivoting. Secondly, the algorithm does not exactly do Gaussian elimination, but it computes the LU factorization of the matrix A. Other useful algorithm for solving a linear system is to compute QR factorization of the matrix A. This leads to the class of direct methods.

If the matrix A has some special structure, this can be exploited to obtain faster or more accurate algorithms. For instance, systems with a symmetric matrix or a symmetric positive definite matrix or a strictly diagonaly dominant matrix, ... Special methods exist also for matrices with many zero elements (so-called sparse matrices).

A completely different approach is often taken for very large systems, which would otherwise take too much time or memory. The idea is to start with an initial approximation

to the solution (which does not have to be accurate at all), and to change this approximation in several steps to bring it closer to the true solution. Once the approximation is sufficiently accurate, this is taken to be the solution to the system. For instance, - Jacobi method, Gauss-Seidel method, Successive over-relaxation (SOR) method, ... This leads to the class of iterative methods.

In this module, we implement some of these methods

5.2.2 Function Documentation

5.2.2.1 void* back_substitution (size_t *n*, double **const *U*, double *const *x*, double *const *b*, double *tol*)

Back substitution for upper triangular matrices (Ux = b)

Parameters

in	*n*	Dimension of matrix and vectors
in	*U*	Upper triangular matrix
in	*b*	Constant terms
out	*x*	Unknowns
in	*tol*	Tolerance

Return values

NULL	If some diagonal's element is less than or equal to tolerance
1	Otherwise

Precondition

- The pointers are not NULL
- The tolerance is positive

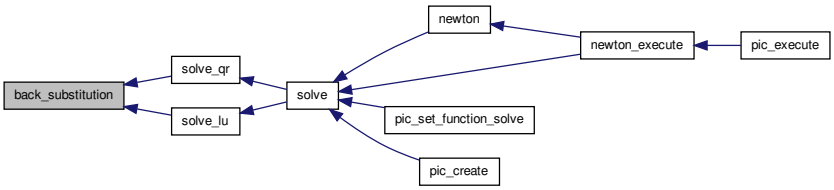
Postcondition

The vector x has the solution

Note

The complexity is $O(n^2)$

Here is the caller graph for this function:



5.2.2.2 void* forward_substitution (size_t *n*, double **const *L*, double *const *x*, double *const *b*, double *tol*)

Forward substitution for lower triangular matrices (Lx = b)

Parameters

in	*n*	Dimension of matrix and vectors
in	*L*	Lower triangular matrix
in	*b*	Constant terms
out	*x*	Unknowns
in	*tol*	Tolerance

Return values

NULL	If some diagonal's element is less than or equal to tolerance
1	Otherwise

Precondition

- The pointers are not NULL
- The tolerance is positive

Postcondition

The vector x has the solution

Remarks

The complexity is $O(n^2)$

5.2.2.3 void forward_substitution_ones (size_t *n*, double **const *L*, double *const *x*, double *const *b*)

Forward substitution for lower triangular matrices (Lx = b) where the diagonal of L are 1's.

Parameters

in	*n*	Dimension of matrix and vectors
in	*L*	Lower triangular matrix with 1's in its diagonal
in	*b*	Constant terms
out	*x*	Unknowns

Precondition

The pointers are not NULL

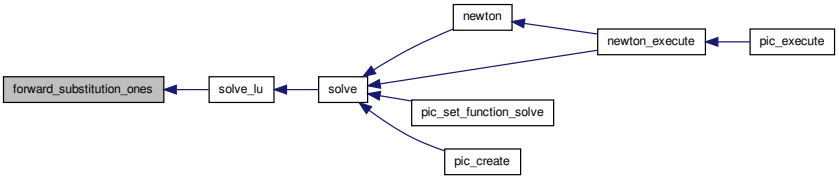
Postcondition

The vector x has the solution

Remarks

The complexity is $O(n^2)$

Here is the caller graph for this function:



5.2.2.4 solve_ptr get_solve_with_lu ( )

References SOLVE_WITH_LU.

5.2.2.5 solve_ptr get_solve_with_qr ( )

References SOLVE_WITH_QR.

5.2.2.6 void * solve (size_t *m*, size_t *n*, double **const *A*, double *const *x*, double *const *b*, double *tol*, void *(*)(size_t, size_t, double **const, double *const, double *const, double) *s*)

solve a linear system (Ax = b) with the pointer to a solve function

Parameters

in	*n*	Dimension of matrix and vectors
in,out	*A*	Matrix's system
in	*b*	Constant terms
out	*x*	Unknowns
in	*tol*	Tolerance
in	*s*	Pointer to a function that solve the linear system

Returns

The value that is returned by the pointer to the solve function

Precondition

The dimension is greater than one

Postcondition

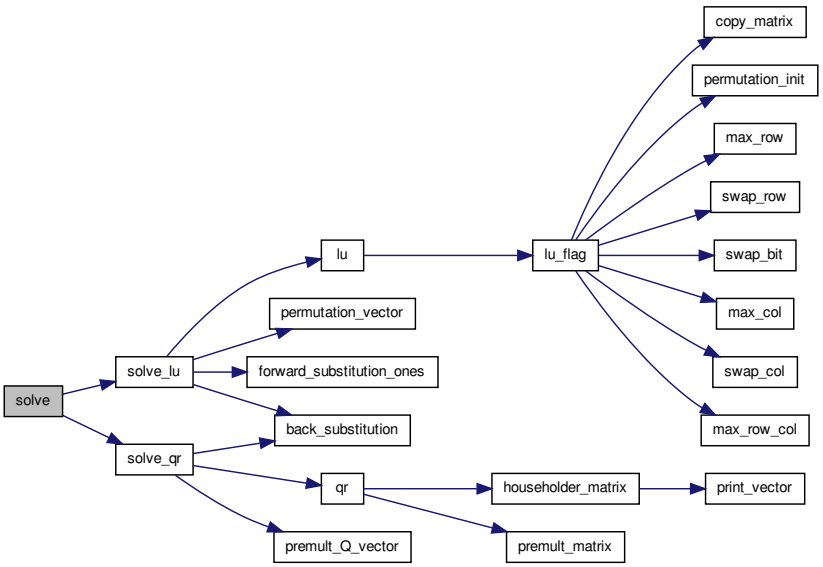
The x vector has the solution

See also

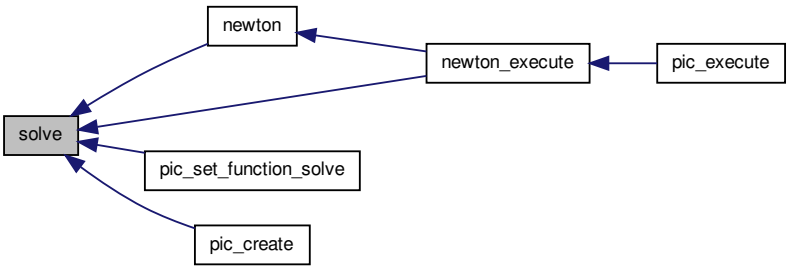
SOLVE_WITH_LU and SOLVE_WITH_QR

References solve_lu(), solve_qr(), SOLVE_WITH_LU, and SOLVE_WITH_QR.

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.2.7 void* solve_lu (size_t *m*, size_t *n*, double **const *A*, double *const *x*, double *const *b*, double *tol*)

LU solve linear system (LUx = b)

Parameters

in	*n*	Dimension of matrix and vectors
in,out	*A*	Matrix's system
in	*b*	Constant terms
out	*x*	Unknowns
in	*tol*	Tolerance

Return values

NULL	When LU factorization was unsuccessful or could not be allocated memory
1	Otherwise

Precondition

The dimension is greater than one

Postcondition

The x vector has the solution

Note

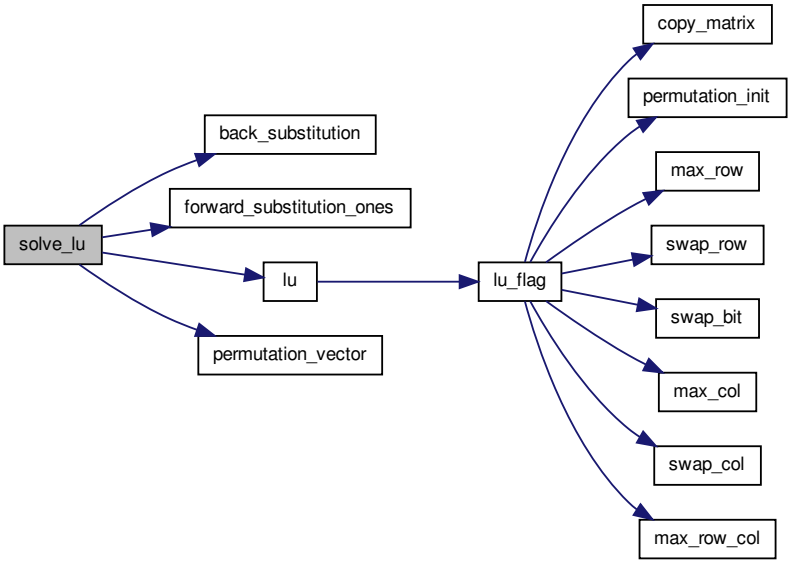
- The vector of constant terms will be modified
- The unknowns pointer *x* and constant terms pointer *b* can be the same

See also

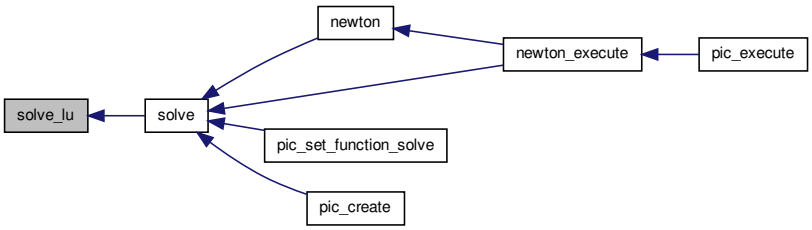
lu

References back_substitution(), forward_substitution_ones(), FULL_PIVOTING, lu(), PARTIAL_PIVOTING_COL, PARTIAL_PIVOTING_ROW, permutation_vector(), and WHITOUT_PIVOTING.

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.2.8 void* solve_qr (size_t *m*, size_t *n*, double **const *A*, double *const *x*, double *const *b*, double *tol*)

QR solve linear system (QRx = b)

Parameters

in	*n*	Dimension of matrix and vectors
in,out	*A*	Matrix's system
in,out	*b*	Constant terms
out	*x*	Unknowns

Return values

NULL	When QR factorization was unsuccessful or could not be allocated memory
1	Otherwise

Precondition

The dimension is greater than one

Postcondition

The x vector has the solution

Note

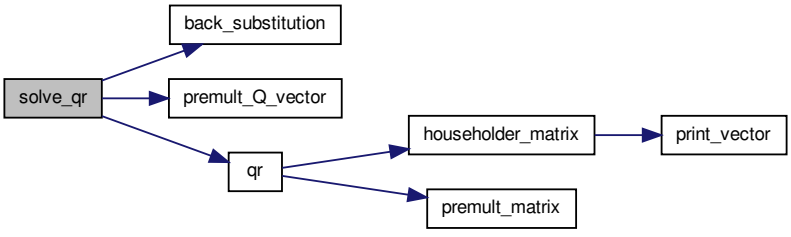
- The vector of constant terms will be modified
- The unknowns pointer *x* and constant terms pointer *b* can be the same

See also

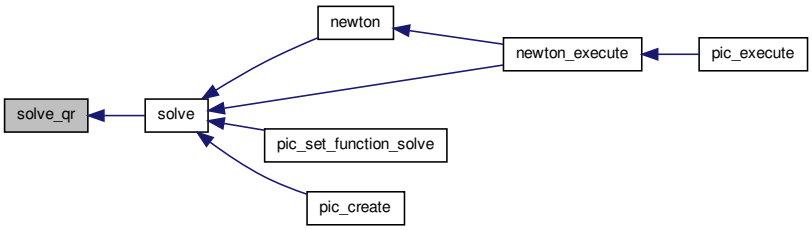
qr

References back_substitution(), premult_Q_vector(), and qr().

Here is the call graph for this function:



Here is the caller graph for this function:



5.3 LU factorization

Implement the LU decomposition of a square matrix with partial pivoting (rows or columns) or full pivoting are admitted.

Defines

- #define WHITOUT_PIVOTING

  LU decomposition without any pivoting.
- #define PARTIAL_PIVOTING_ROW

  Partial pivoting by rows in LU decomposition.
- #define PARTIAL_PIVOTING_COL

  Partial pivoting by columns in LU decomposition.
- #define FULL_PIVOTING

  Complete pivoting by rows and columns in LU decomposition.
- #define _OPENMP_LU_DIMENSION 1e2

  Minimum value for using the openmp library in LU decomposition.

Functions

- double max_row (size_t m, double **const A, size_t idx, size_t *const row)

  Find the maximum value of a column based on the same row index.
- double max_col (size_t n, double **const A, size_t idx, size_t *const col)

  Find the first maximum value of a column based on the same column index.
- double max_row_col (size_t m, size_t n, double **const A, size_t idx, size_t *const row, size_t *const col)

  Find the maximum value of a column and row based on the same column and row index.
- void swap_bit (size_t i, size_t j, size_t *const x, size_t *const y)

  Swap two values from vector x to y.
- void swap_row (size_t row1, size_t row2, size_t n, double **const A, double **const B)

  Swap two rows from matrix A to B based on the same minimum row index.
- void swap_col (size_t col1, size_t col2, size_t m, double **const A, double **const B)

  Swap two columns from matrix A to B based on the same minimum column index.
- void * lu_flag (size_t m, size_t n, double **const A, double **const LU, size_t *const pr, size_t *const pc, double tol, unsigned int flag)

  Compute the LU factorization of a matrix m-by-n with a specific pivoting.
- void * lu (size_t m, size_t n, double **const A, double **const LU, size_t *const pr, size_t *const pc, double tol)

  Compute the LU factorization of a matrix n-by-n.

5.3.1 Detailed Description

Implement the LU decomposition of a square matrix with partial pivoting (rows or columns) or full pivoting are admitted. Let A be a square matrix. An LU factorization refers to the factorization of A, with proper row and/or column orderings or permutations, into two factors, a lower triangular matrix L with 1 in its diagonal and an upper triangular matrix U, that is A = LU.

It turns out that a proper permutation in rows (or columns) is sufficient for the LU factorization. The LU factorization with Partial Pivoting by rows refers often to the LU factorization with row permutations only, that can be denoted by PA = LU, where P is a permutation matrix which, when left-multiplied to A, reorders the rows of A.

The LU factorization with Partial Pivoting by columns refers to the LU factorization with column permutations, that can be denoted by AQ = LU where Q is a permutation matrix that reorders the columns of A.

It turns out that all square matrices can be factorized in this form, and the factorization is numerically stable in practice.

An LU factorization with full pivoting involves both row and column permutations, that can be denoted by PAQ = LU.

5.3.2 The LU factorization of a rectangular matrix

The LU factorization of a rectangular matrix $A \in \mathbb{R}(m,n)$ can also be performed and it is guaranteed to exists if $A[1:i][1:i]$ is not singular for $i = 1,\ldots,\min\{m,n\}$.

The calculation of a LU factorization of a rectangular matrix requires $mn^2 - n^3/3$ flops and *A* can be overwritten by the strictly lower triangular portion of $L \in \mathbb{R}(m,\min\{m,n\})$ and the upper triangular portio of $U \in \mathbb{R}(\min\{m,n\},n)$.

5.3.3 Define Documentation

5.3.3.1 #define _OPENMP_LU_DIMENSION 1e2

Minimum value for using the openmp library in LU decomposition.

5.3.3.2 #define FULL_PIVOTING

Complete pivoting by rows and columns in LU decomposition.

5.3.3.3 #define PARTIAL_PIVOTING_COL

Partial pivoting by columns in LU decomposition.

5.3.3.4 #define PARTIAL_PIVOTING_ROW

Partial pivoting by rows in LU decomposition.

5.3.3.5 #define WHITOUT_PIVOTING

LU decomposition without any pivoting.

5.3.4 Function Documentation

5.3.4.1 void* lu (size_t *m*, size_t *n*, double **const *A*, double **const *LU*, size_t *const *pr*, size_t *const *pc*, double *tol*)

Compute the LU factorization of a matrix n-by-n.

Parameters

in	*m*	Rows
in	*n*	Columns
in	*A*	Matrix n-by-n where will be computed LU factorization
out	*LU*	Matrix n-by-n where has been computed LU factorization
out	*pr*	Vector n-dimensional where will be saved the pivoting by rows
out	*pc*	Vector n-dimensional where will be saved the pivoting by columns
in	*tol*	Tolerance that indicates a diagonal's element is valid for applying the Gaussian elimination

Return values

NULL	If tolerance is greater than the pivot in absolute value
1	Otherwise

Precondition

- The dimension is greater than 1
- The pointers *A* and *LU* are not NULL
- The tolerance is positive

Postcondition

- The LU factorization has been saved in *LU*
- The vector permutations (*pr* and *pc*) have been initialized, so the free should be done

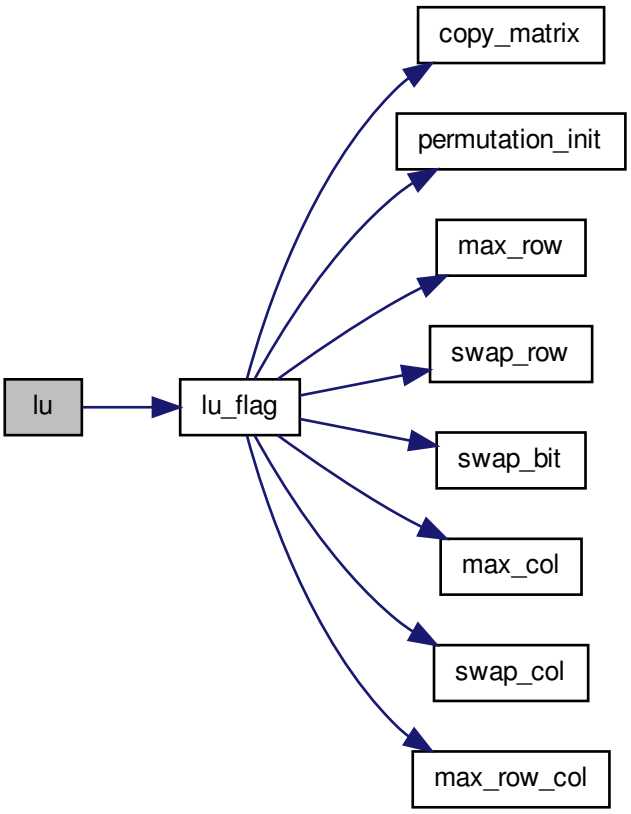
Note

- LU and A can be the same
- If *pr* is NULL and *pc* is NULL, pivoting will not be done
- If *pr* is NULL and *pc* is not NULL, partial pivoting by columns will be done
- If *pr* is not NULL and *pc* is NULL, partial pivoting by rows will be done

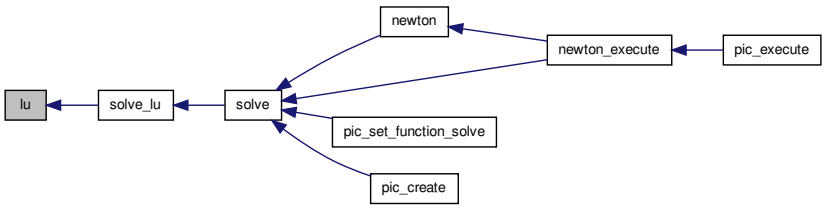
- If *pr* is not NULL and *pc* is not NULL, full pivoting will be done

References FULL_PIVOTING, lu_flag(), PARTIAL_PIVOTING_COL, PARTIAL_PIVOTING_ROW, and WHITOUT_PIVOTING.

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.4.2 void * lu_flag (size_t *m*, size_t *n*, double **const *A*, double **const *LU*, size_t *const *pr*, size_t *const *pc*, double *tol*, unsigned int *flag*)

Compute the LU factorization of a matrix m-by-n with a specific pivoting.

Parameters

in	*m*	Rows
in	*n*	Columns
in	*A*	Matrix m-by-n where will be computed LU factorization
out	*LU*	Matrix m-by-n where has been computed LU factorization
out	*pr*	Vector m-dimensional where will be saved the pivoting by rows
out	*pc*	Vector n-dimensional where will be saved the pivoting by columns
in	*tol*	Tolerance that indicates a diagonal's element is valid for applying the Gaussian elimination
in	*flag*	Type of pivoting

Return values

NULL	If tolerance is greater than the pivot in absolute value
1	Otherwise

Precondition

- The dimension are greater than 1
- The pointers *A* and *LU* are not NULL
- The tolerance is positive
- Row vector permutation pointer and Column vector permutation are either NULL or else different

Postcondition

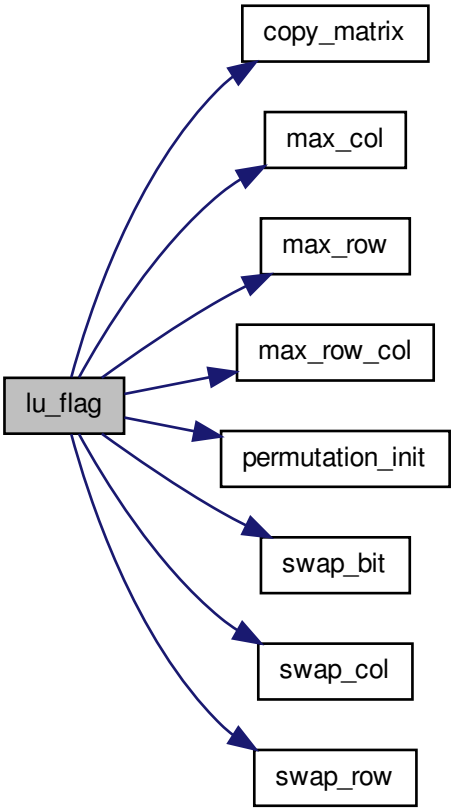
- The LU factorization has been saved in *LU*
- The vector permutations (*pr* and *pc*) can have been initialized, so the free should be done

Note

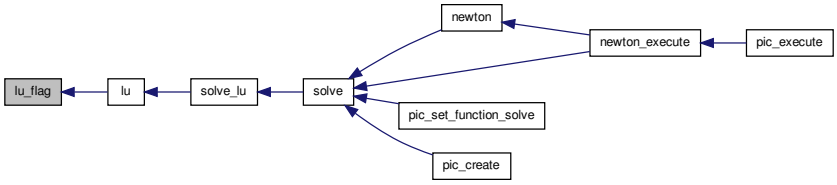
- LU and A can be the same
- The *flags* can be WHITOUT_PIVOTING , PARTIAL_PIVOTING_ROW , PARTIAL_PIVOTING_COL or FULL_PIVOTING

References copy_matrix(), FULL_PIVOTING, max_col(), max_row(), max_row_col(), PARTIAL_PIVOTING_COL, PARTIAL_PIVOTING_ROW, permutation_init(), swap_bit(), swap_col(), swap_row(), and WHITOUT_PIVOTING.

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.4.3 double max_col (size_t *n*, double **const *A*, size_t *idx*, size_t *const *col*)

Find the first maximum value of a column based on the same column index.

Parameters

in	*n*	Dimension of matrixs
in	*A*	Matrix m-by-n where will be read the column
in	*idx*	Row index
out	*col*	Column index with the first maximum value

Returns

The first maximum value in absolute value in the row index

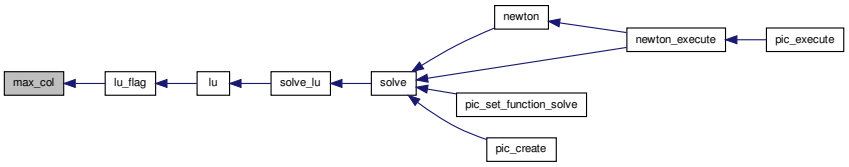
Precondition

The indexs is less than the dimension and the pointer is not NULL

Note

The column indexes can be the same

Here is the caller graph for this function:



5.3.4.4 double max_row (size_t *m*, double ∗∗const *A*, size_t *idx*, size_t ∗const *row*)

Find the maximum value of a column based on the same row index.

Parameters

in	*m*	Dimension of matrixs
in	*A*	Matrix m-by-n where will be read the column
in	*idx*	Column index
out	*row*	Row index with the first maximum value

Returns

The first maximum value in absolute value in the column index

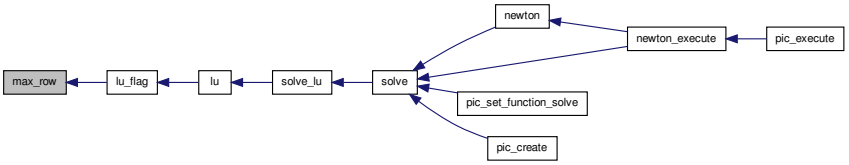
Precondition

Dimension is not 0, the indexs is less than the dimension and the pointer is not NULL

Note

The row indexes can be the same

Here is the caller graph for this function:



5.3.4.5 double max_row_col (size_t *m*, size_t *n*, double ∗∗const *A*, size_t *idx*, size_t ∗const *row*, size_t ∗const *col*)

Find the maximum value of a column and row based on the same column and row index.

Parameters

in	*m*	Rows
in	*n*	Columns
in	*A*	Matrix m-by-n where will be read the column
in	*idx*	Index
out	*row*	Row index
out	*col*	Column index

Returns

The maximum value in absolute value

Precondition

- The indexs is less than the dimension
- The pointers are not NULL

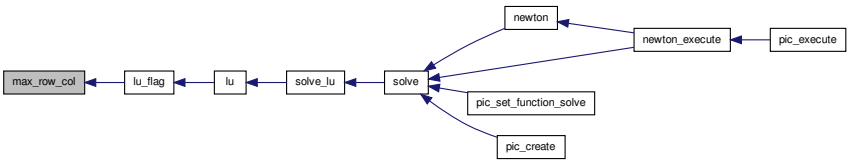
Postcondition

Either row index or else column index is different than *idx* index

Note

The column indexes can be the same

Here is the caller graph for this function:



5.3.4.6 void swap_bit (size_t *i*, size_t *j*, size_t *const *x*, size_t *const *y*)

Swap two values from vector x to y.

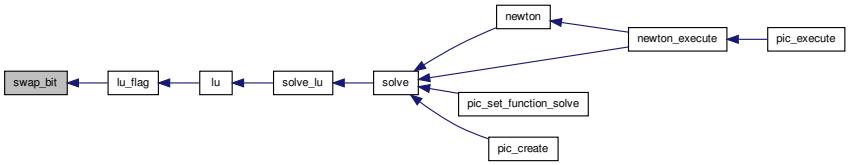
Parameters

in	*i*	Index
in	*j*	Index
in	*x*	Source vector
out	*y*	Destination vector

Note

The vectors can be the same. If one of them is NULL, anything is done

Here is the caller graph for this function:



5.3.4.7 void swap_col (size_t *col1*, size_t *col2*, size_t *m*, double **const *A*, double **const *B*)

Swap two columns from matrix A to B based on the same minimum column index.

Parameters

in	*col1*	Col index
in	*col2*	Col index
in	*m*	Rows
in	*A*	Matrix m-by-n where will be read the rows
out	*B*	Matrix m-by-n where will be saved the rows

Precondition

- The indexs are less than the number of rows
- The pointers are not NULL

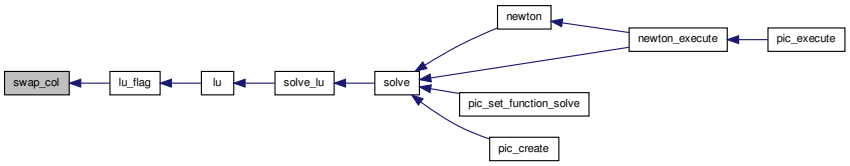
Postcondition

The swapping is in B

Note

- The column indexes can be the same
- The matrix can be the same

Here is the caller graph for this function:



5.3.4.8 void swap_row (size_t *row1*, size_t *row2*, size_t *n*, double **const *A*, double **const *B*)

Swap two rows from matrix A to B based on the same minimum row index.

Parameters

in	*row1*	Row index
in	*row2*	Row index
in	*n*	Columns
in	*A*	Matrix m-by-n where will be read the rows
out	*B*	Matrix m-by-n where will be saved the rows

Precondition

- The indexs are less than the number of rows
- The pointers are not NULL

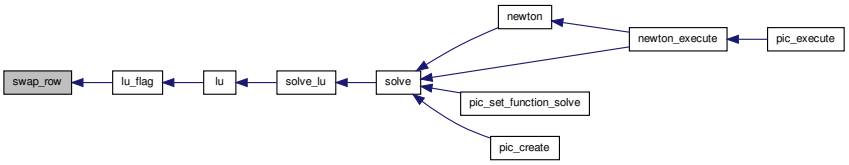
Postcondition

The swapping is in B

Note

- The row indexes can be the same
- The matrix can be the same

Here is the caller graph for this function:



5.4 Vector permutation

The act of permuting objects or values is called permutation and we implement a permutation to a vector.

Defines

- #define _OPENMP_PERMUTATION_DIMENSION 1e2

  Minimum value for using the openmp library in permutation.

Functions

- void permutation_init (size_t n, size_t *const p)

  Initialize a identity vector permutation.
- void permutation_vector (size_t n, size_t *const p, double *const x, double *const px)

  Permute the values of the vector x into px based on the vector permutation p.

5.4.1 Detailed Description

The act of permuting objects or values is called permutation and we implement a permutation to a vector. A permutation τ of $n+1$ elements is a bijection of $\{0,\ldots,n\}$ from that set onto itself and with the composition is a non-commutative group. The Cauchy's two-line notation is

$$\tau = \begin{pmatrix} 0 & \cdots & n \\ \tau(0) & \cdots & \tau(n) \end{pmatrix}.$$

In one-line notation, one gives only the second row of this array. In this functions, one-line notation will be used.

5.4.2 Define Documentation

5.4.2.1 #define _OPENMP_PERMUTATION_DIMENSION 1e2

Minimum value for using the openmp library in permutation.

5.4.3 Function Documentation

5.4.3.1 void permutation_init (size_t *n*, size_t *const *p*)

Initialize a identity vector permutation.

Parameters

in	*n*	Dimension of vector
out	*p*	Vector permutation

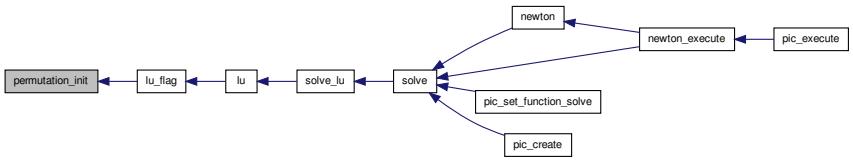
Precondition

Dimension is not zero

Note

If vector permutation is NULL, anything is done

Here is the caller graph for this function:



5.4.3.2 void permutation_vector (size_t *n*, size_t *const *p*, double *const *x*, double *const *px*)

Permute the values of the vector x into px based on the vector permutation p.

Parameters

in	*n*	Dimension of vectors
in	*p*	Vector permutation
in	*x*	Vector n-dimensional where will be read the data
out	*px*	Vector n-dimensional where will be saved the permutations by the vector permutation

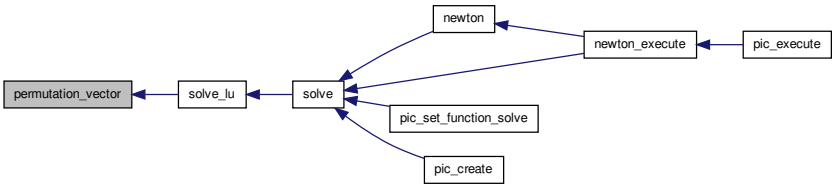
Precondition

Dimension is not 0 and the pointers x and px are not NULL

Note

The vectors can be the same If the vector permutation is NULL, anything is done

Here is the caller graph for this function:



5.5 QR factorization

Defines

- #define _OPENMP_QR_DIMENSION 1e2

  Minimum value for using the openmp library in QR decomposition.

Functions

- void premult_Q_vector (size_t m, size_t n, double **const QR, double *const b)

  Compute $P_1 \cdots P_n b$ where b and P_i are respectively m-by-1 and m-by-m matrixs.
- void * qr (size_t m, size_t n, double **const A, double **const QR)

  Compute the QR factorization of a matrix m-by-n.

5.5.1 Detailed Description

5.5.2 Define Documentation

5.5.2.1 #define _OPENMP_QR_DIMENSION 1e2

Minimum value for using the openmp library in QR decomposition.

5.5.3 Function Documentation

5.5.3.1 void premult_Q_vector (size_t *m*, size_t *n*, double **const *QR*, double *const *b*)

Compute $P_1 \cdots P_n b$ where b and P_i are respectively m-by-1 and m-by-m matrixs.

Parameters

in	*m*	Rows of linear system's matrix
in	*n*	Columns of linear system's matrix
in	*QR*	Householder vector of dimension n
in,out	*b*	Vector m-dimensional where will be saved the result

Precondition

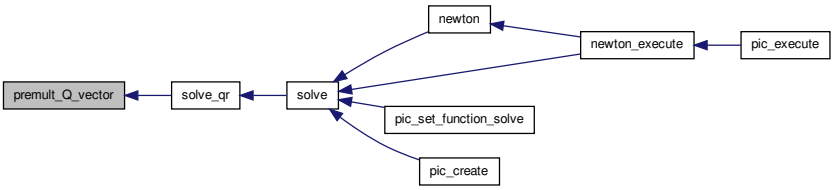
It has been computed the QR descomposition over the matrix input

Postcondition

The calculation was carried out

References pic::n.

Here is the caller graph for this function:



5.5.3.2 void* qr (size_t *m*, size_t *n*, double **const *A*, double **const *QR*)

Compute the QR factorization of a matrix m-by-n.

Parameters

in	*m*	Rows
in	*n*	Columns
in	*A*	Matrix m-by-n where will be computed QR factorization
out	*QR*	Matrix m-by-n where has been computed QR factorization

Return values

NULL	If the memory can not be allocated
0	If some error ocurred and number of columns where has been applied the decomposition in QR

Returns

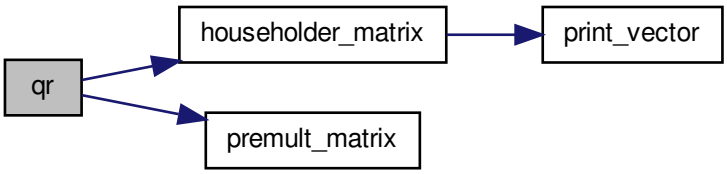
Otherwise the last index for the column changed

Note

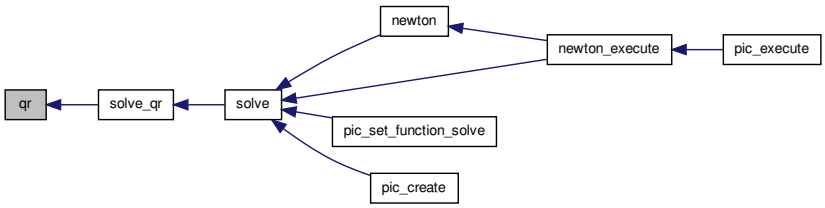
QR and A can be the same

References householder_matrix(), premult_matrix(), and pic::w.

Here is the call graph for this function:



Here is the caller graph for this function:



6 Data Structure Documentation

6.1 pic Struct Reference

Periodic Invariant Curve.

`#include <pic.h>`

Data Fields

- size_t n

  Dimension of the curve's points.
- size_t N

  Mesh size of one-dimensional torus.
- double * x

  (2N+1)n points that mesh the periodic invariant curve
- double * X

  n(2N + 1) Fourier coefficients
- double ** DF

  Pointer to (2N+1)n-by-(2N+1)n matrix for Newton methods.
- double w

  Irrational number.
- short unsigned int autonomous

  Flag that indicates if the dynamic system is autonomous or not PIC_AUTONOMOUS and PIC_NAUTONOMOUS.
- void(* f)(size_t, size_t, double *const, double *const, double **const, double, double)

  Pointer to function that is given by user.
- void *(* solve)(size_t, size_t, double **const, double *const, double *const, double)

  Pointer to function that is given by user for solving a linear of equation system.

6.1.1 Detailed Description

Periodic Invariant Curve.

Fields for a periodic invariant curve

6.1.2 Field Documentation

6.1.2.1 pic::autonomous

Flag that indicates if the dynamic system is autonomous or not PIC_AUTONOMOUS and PIC_NAUTONOMOUS.

6.1.2.2 pic::DF

Pointer to (2N+1)n-by-(2N+1)n matrix for Newton methods.

6.1.2.3 pic::f

Pointer to function that is given by user.

6.1.2.4 pic::n

Dimension of the curve's points.

Precondition

Is not zero

6.1.2.5 pic::N

Mesh size of one-dimensional torus.

Precondition

Is not zero

6.1.2.6 pic::solve

Pointer to function that is given by user for solving a linear of equation system.

6.1.2.7 pic::w

Irrational number.

6.1.2.8 pic::x

(2N+1)n points that mesh the periodic invariant curve

6.1.2.9 pic::X

n(2N + 1) Fourier coefficients

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

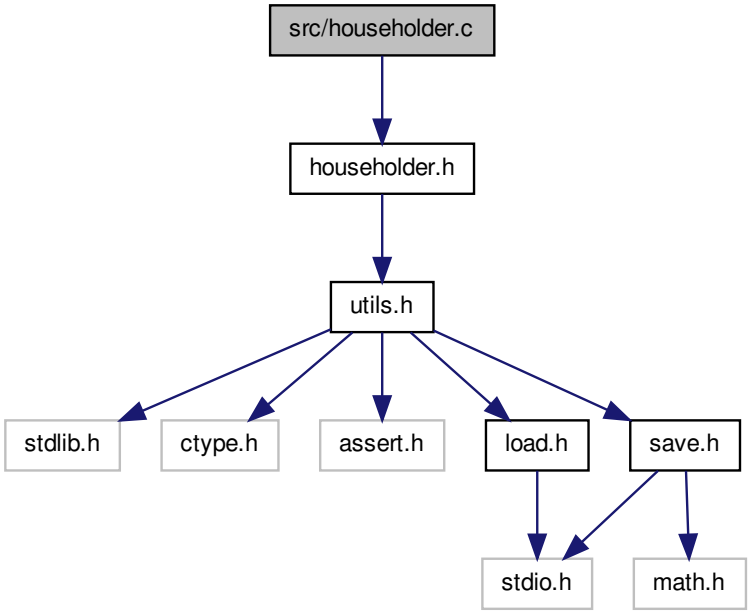
- src/include/pic.h

7 File Documentation

7.1 src/householder.c File Reference

Functions for using householder vectors and matrix.

`#include "householder.h"` Include dependency graph for householder.c:



Defines

- #define _OPENMP_HOUSEHOLDER_DIMENSION 1e2

  Minimum value for using openmp in Householder.

Functions

- void ∗ householder_vector (size_t n, double ∗const x, double ∗const v)

  Derive the householder vector.
- void ∗ householder_matrix (size_t m, double ∗∗const A, double ∗const v, size_t idx)

  Derive the householder vector using a matrix.
- void ∗ premult_matrix (size_t m, size_t n, double ∗∗const A, double ∗∗const QR, double ∗const v, double ∗const w, size_t idx)

  Compute $P_v A$ where A and P_v are respectively m-by-n and n-by-n matrixs.
- void premult_vector (size_t n, double ∗const v, double ∗const b)

  Compute $P_v b$ where b and P_v are respectively n-by-1 and n-by-n matrixs.

- void postmult (size_t m, size_t n, double **const A, double *const v, double *const w)

  Compute AP_v where A and P_v are respectively m-by-n and n-by-n matrixs.

7.1.1 Detailed Description

Functions for using householder vectors and matrix.

Author

Joan

Date

21/01/2014 (start)

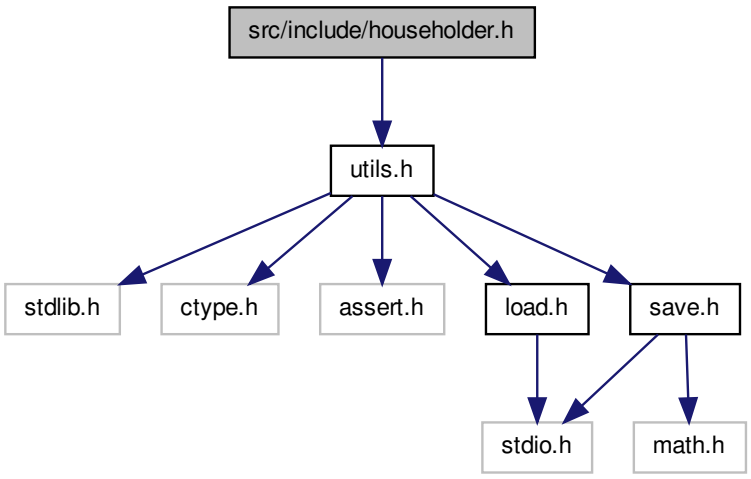
Version

1.0

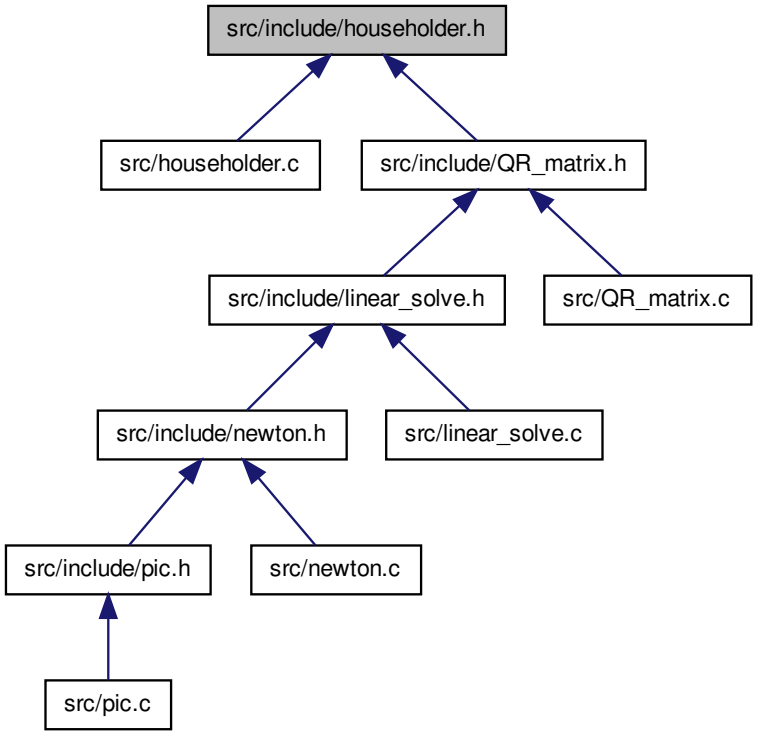
7.2 src/include/householder.h File Reference

QR method header file.

`#include "utils.h"` Include dependency graph for householder.h:



This graph shows which files directly or indirectly include this file:



Functions

- void ∗ householder_matrix (size_t m, double ∗∗const A, double ∗const v, size_t idx)

  Derive the householder vector using a matrix.
- void ∗ premult_matrix (size_t m, size_t n, double ∗∗const A, double ∗∗const QR, double ∗const v, double ∗const w, size_t idx)

  Compute $P_v A$ where A and P_v are respectively m-by-n and n-by-n matrixs.

7.2.1 Detailed Description

QR method header file.

Author

Joan Gimeno

Date

21/01/2014 (start)

Version

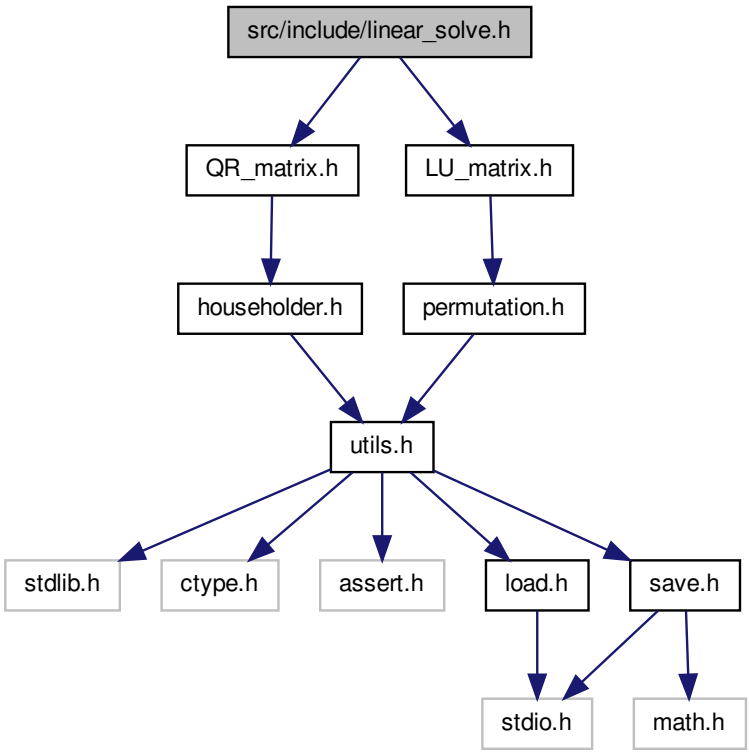
1.0

This file contains the header for ::QR_decomposition

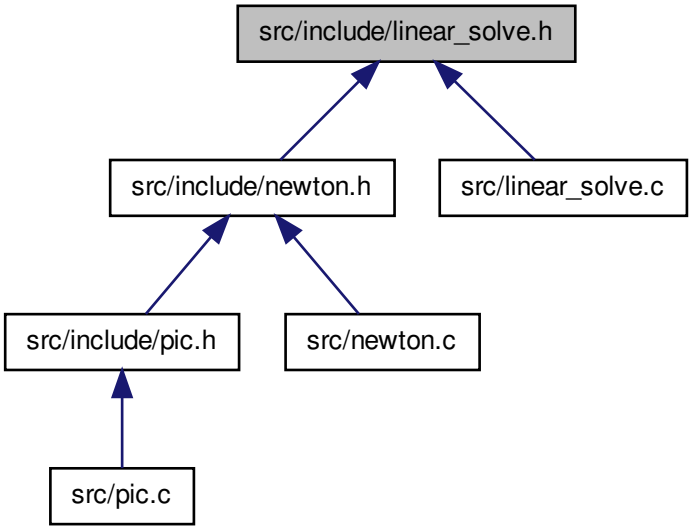
7.3 src/include/linear_solve.h File Reference

Linear solve header file.

`#include "QR_matrix.h"` `#include "LU_matrix.h"` Include dependency graph for linear_solve.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define SOLVE_WITH_LU

  Pointer to a solve a linear system with LU decomposition.
- #define SOLVE_WITH_QR

  Pointer to a solve a linear system with QR decomposition.
- #define _SOLVE_PROTOTYPE

  Prototype of a function pointer that solves a linear system.

Typedefs

- typedef void ∗(∗ solve_ptr)(size_t, size_t, double ∗∗const, double ∗const, double ∗const, double)

Functions

- void forward_substitution_ones (size_t n, double ∗∗const L, double ∗const x, double ∗const b)

  Forward substitution for lower triangular matrices (Lx = b) where the diagonal of L are 1's.

- void * forward_substitution (size_t n, double **const L, double *const x, double *const b, double tol)

  Forward substitution for lower triangular matrices (Lx = b)
- void * back_substitution (size_t n, double **const U, double *const x, double *const b, double tol)

  Back substitution for upper triangular matrices (Ux = b)
- void * solve_qr (size_t m, size_t n, double **const A, double *const x, double *const b, double tol)

  QR solve linear system (QRx = b)
- void * solve_lu (size_t m, size_t n, double **const A, double *const x, double *const b, double tol)

  LU solve linear system (LUx = b)
- solve_ptr get_solve_with_lu ()
- solve_ptr get_solve_with_qr ()

7.3.1 Detailed Description

Linear solve header file.

Author

Joan Gimeno

Date

22/11/2013 (start)

Version

1.0

This file contains the header for ::linear_solve

7.3.2 Define Documentation

7.3.2.1 #define _SOLVE_PROTOTYPE

Prototype of a function pointer that solves a linear system.

7.3.2.2 #define SOLVE_WITH_LU

Pointer to a solve a linear system with LU decomposition.

7.3.2.3 #define SOLVE_WITH_QR

Pointer to a solve a linear system with QR decomposition.

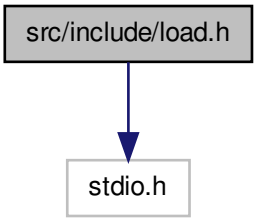
7.3.3 Typedef Documentation

7.3.3.1 typedef void*(* solve_ptr)(size_t, size_t, double **const, double *const, double *const, double)

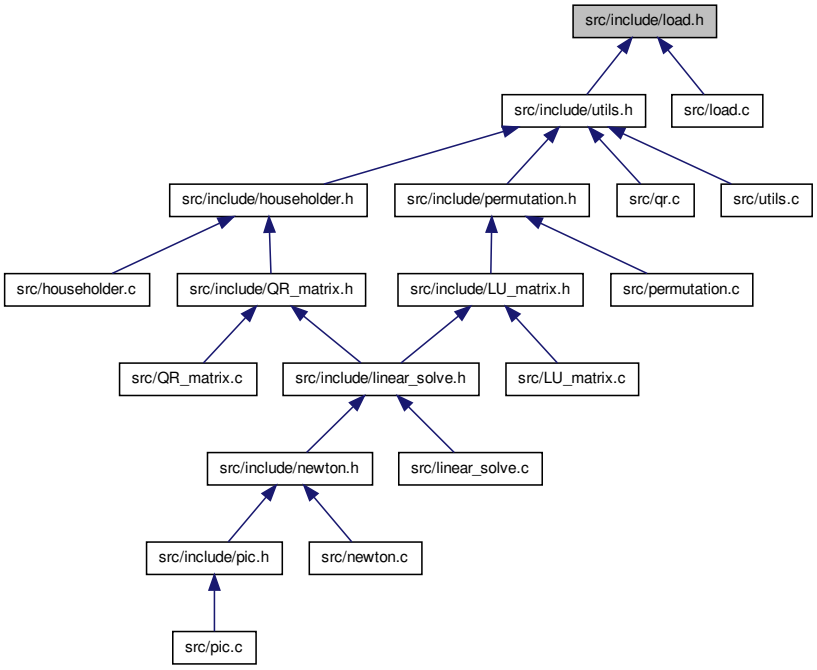
7.4 src/include/load.h File Reference

Header of load.

`#include <stdio.h>` Include dependency graph for load.h:



This graph shows which files directly or indirectly include this file:



Functions

- int load_vector (double ∗const, FILE ∗const)

  Function that loads the values of a vector in a given file.
- int loadas_vector (double ∗const, char ∗const)

  Function that loads the values of a vector in a given filename.
- int load_matrix (double ∗∗const, FILE ∗const)

  Function that loads the values of a matrix in a given file.
- int loadas_matrix (double ∗∗const, char ∗const)

  Function that loads the values of a matrix in a given filename.

7.4.1 Detailed Description

Header of load.

Author

Joan Gimeno

Date

22/05/2014 (start)

Version

1.0

7.4.2 Function Documentation

7.4.2.1 int load_matrix (double ** const A, FILE * const f)

Function that loads the values of a matrix in a given file.

Parameters

in	*A*	Pointer to a matrix
in	*f*	Output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

Here is the caller graph for this function:

7.4.2.2 int load_vector (double * const b, FILE * const f)

Function that loads the values of a vector in a given file.

Parameters

in	*b*	Pointer to a vector
in	*f*	Output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

Here is the caller graph for this function:

7.4.2.3 int loadas_matrix (double ** const *A*, char * const *name*)

Function that loads the values of a matrix in a given filename.

Parameters

in	*A*	Pointer to a matrix
in	*name*	Filename

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

References load_matrix().

Here is the call graph for this function:

7.4.2.4 int loadas_vector (double * const *b*, char * const *name*)

Function that loads the values of a vector in a given filename.

Parameters

in	*b*	Pointer to a vector
in	*name*	Filename

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

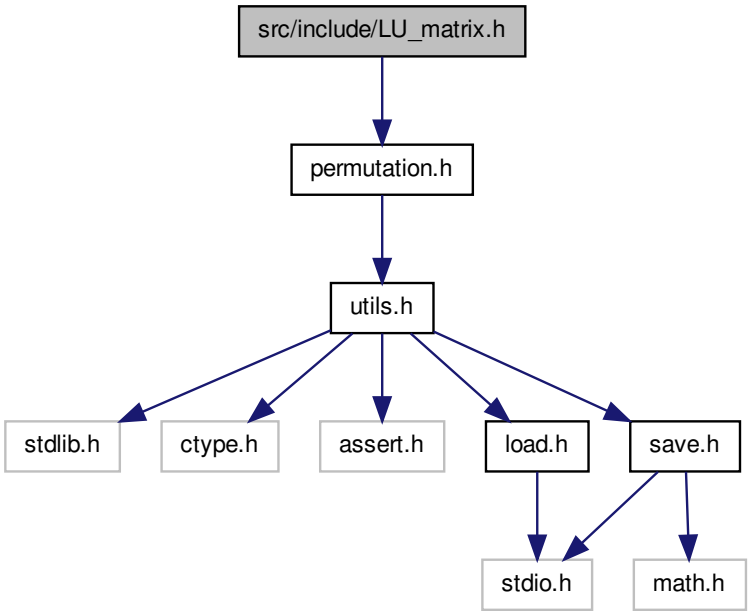
References load_vector().

Here is the call graph for this function:

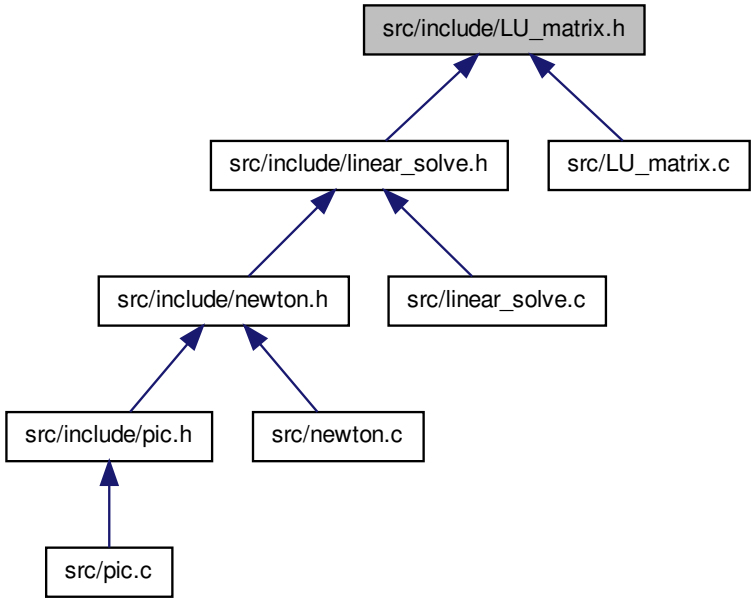
7.5 src/include/LU_matrix.h File Reference

LU method header file.

`#include "permutation.h"` Include dependency graph for LU_matrix.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define WHITOUT_PIVOTING

  LU decomposition without any pivoting.
- #define PARTIAL_PIVOTING_ROW

  Partial pivoting by rows in LU decomposition.
- #define PARTIAL_PIVOTING_COL

  Partial pivoting by columns in LU decomposition.
- #define FULL_PIVOTING

  Complete pivoting by rows and columns in LU decomposition.

Functions

- void ∗ lu (size_t m, size_t n, double ∗∗const A, double ∗∗const LU, size_t ∗const pr, size_t ∗const pc, double tol)

  Compute the LU factorization of a matrix n-by-n.

7.5.1 Detailed Description

LU method header file.

Author

Joan Gimeno

Date

19/01/2014 (start)

Version

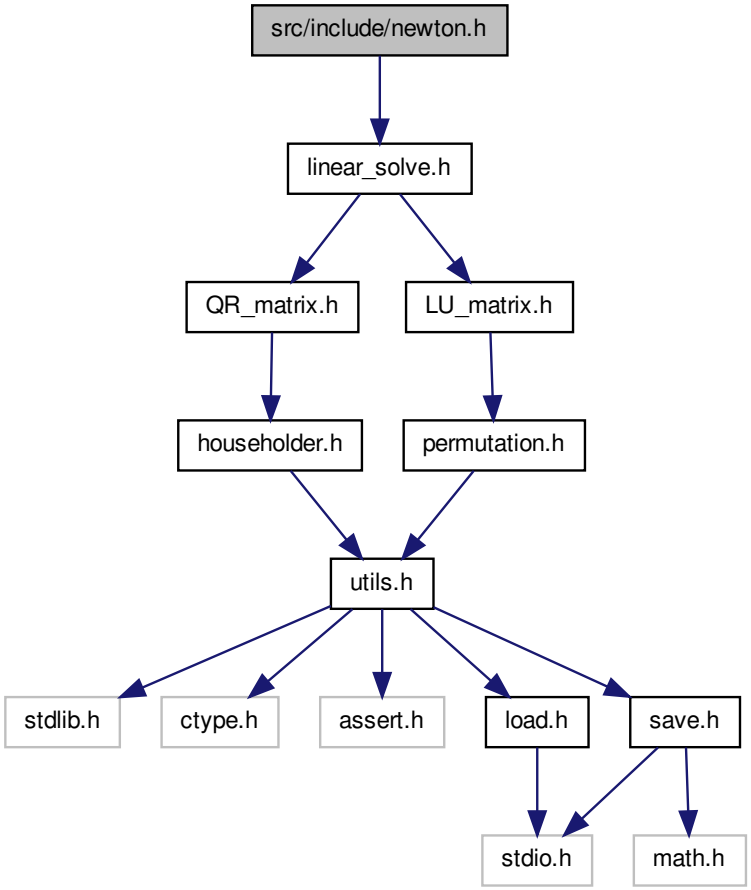
1.0

This file contains the header for ::LU

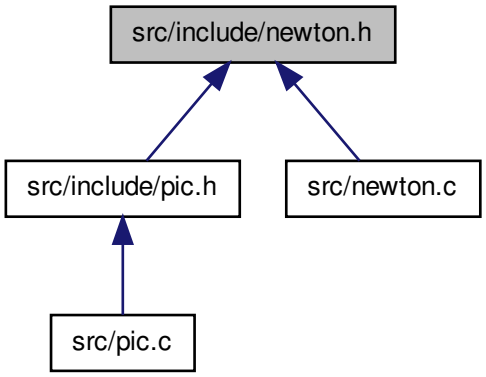
7.6 src/include/newton.h File Reference

Header of compute a solutions of an equation with Newton method.

`#include "linear_solve.h"` Include dependency graph for newton.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define _OPENMP_NEWTON_METHOD 1e2
- #define _FUNCTION_PROTOTYPE

  Function prototype of a pic struct.

Functions

- unsigned int get_newton_max_iterations ()

  Get the maximum number of newton iterations.
- unsigned int set_newton_max_iterations (unsigned int newton_max_iter)

  Set the maximum number of newton iterations.
- void * newton_execute (size_t n, size_t N, double *const x, double *const X, double **const DF, double w, double tol, void(*f)(size_t, size_t, double *const, double *const, double **const, double, double), void *(*solve)(size_t, size_t, double **const, double *const, double *const, double), short unsigned int autonomous)

  Implement the execution of a Newton's method.
- void newton_differential (size_t n, size_t N, double *const x, double *const X, double **const DF, void(*f)(size_t, size_t, double *const, double *const, double **const, double, double), double w, double *h)

  Compute the differential for the Newton's method.
- void dft_r2r_1d (size_t n, size_t N, double *const dft, double *const x, double theta)

Compute the Discrete Fourier Transform.

- int newton (size_t m, size_t n, double ∗const x, double ∗const f, double ∗∗const df, void ∗(∗solve)(size_t, size_t, double ∗∗const, double ∗const, double ∗const, double), double tol)

  Implements the compute of a newton method's iteration.

7.6.1 Detailed Description

Header of compute a solutions of an equation with Newton method.

Author

Joan Gimeno

Date

22/02/2014 (start)

Version

1.0

7.6.2 Define Documentation

7.6.2.1 #define _FUNCTION_PROTOTYPE

Function prototype of a pic struct.

7.6.2.2 #define _OPENMP_NEWTON_METHOD 1e2

7.6.3 Function Documentation

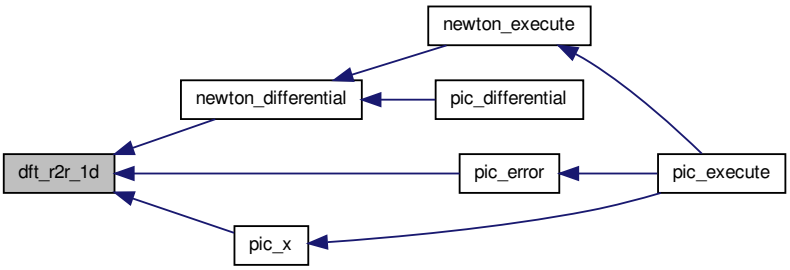
7.6.3.1 void dft_r2r_1d (size_t *n*, size_t *N*, double ∗const *dft*, double ∗const *x*, double *theta*)

Compute the Discrete Fourier Transform.

Parameters

in	*n*	Space's dimension
in	*N*	Mesh size
out	*x*	Discrete Fourier Transform
in	*X*	Fourier coefficients
in	*theta*	Angle evaluation

Here is the caller graph for this function:



7.6.3.2 unsigned int get_newton_max_iterations ( )

Get the maximum number of newton iterations.

Returns

Maximum number iterations for Newton's method

7.6.3.3 int newton (size_t *m*, size_t *n*, double ∗const *x*, double ∗const *f*, double ∗∗const *df*, void ∗(∗)(size_t, size_t, double ∗∗const, double ∗const, double ∗const, double) *solve*, double *tol*)

Implements the compute of a newton method's iteration.

Parameters

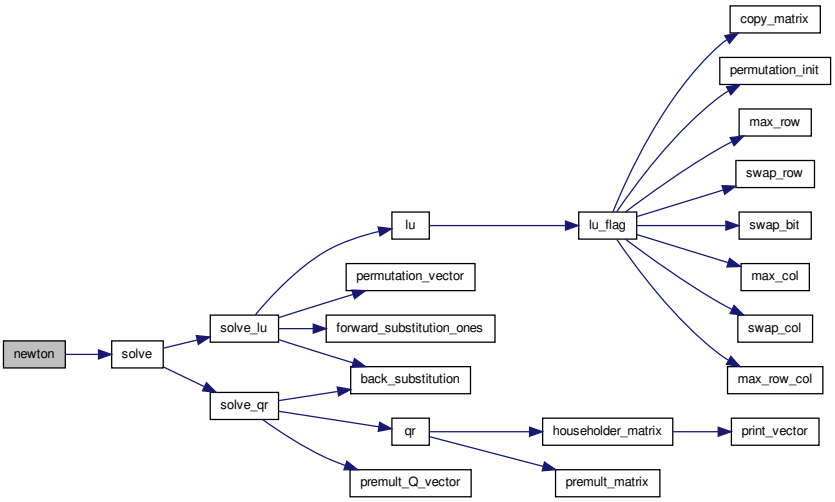
in	*m*	Rows of the matrix
in	*n*	Columns of the matrix
in	*x*	Current iteration value
out	*x*	Next iteration value
in	*f*	Constant terms
in	*df*	Matrix of the linear system
in	∗*solve*	Pointer to solver linear systems of equations
in	*tol*	Tolerance

Return values

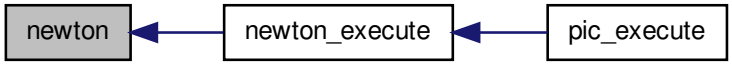
0	Some problem has ocurred
1	When the solution x is accepted by the tolerance given

References solve().

Here is the call graph for this function:



Here is the caller graph for this function:



7.6.3.4 void newton_differential (size_t *n*, size_t *N*, double ∗const *x*, double ∗const *X*, double ∗∗const *DF*, void(∗)(size_t, size_t, double ∗const, double ∗const, double ∗∗const, double, double) *f*, double *w*, double ∗ *h*)

Compute the differential for the Newton's method.

Parameters

in	*n*	Space's dimension
in	*N*	Mesh size
in,out	*x*	Discrete Fourier Transform for each mesh's element, that is, n(2N+1) components
in	*X*	Fourier coefficients
in	*DF*	Matrix of the linear system

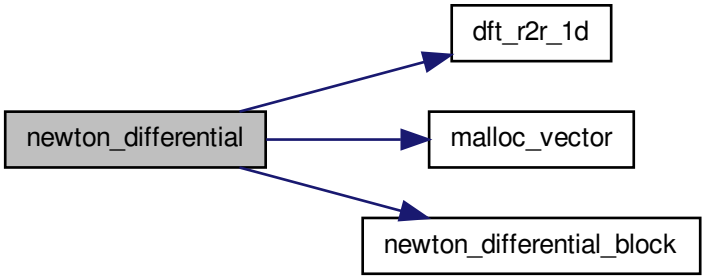
in	*f*	Pointer to function given by the user
in	*w*	Parameter
in	*h*	Auxiliar vector

Returns

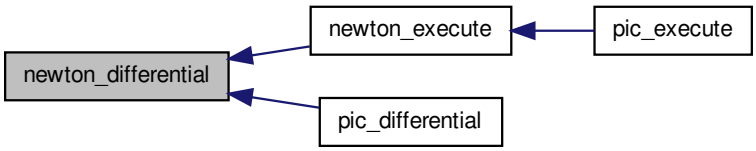
Number of iterations have been performed or NULL if some problem has ocurred

References dft_r2r_1d(), malloc_vector(), and newton_differential_block().

Here is the call graph for this function:



Here is the caller graph for this function:



7.6.3.5 void * newton_execute (size_t *n*, size_t *N*, double *const *x*, double *const *X*, double **const *DF*, double *w*, double *tol*, void(*)(size_t, size_t, double *const, double *const, double **const, double, double) *f*, void *(*)(size_t, size_t, double **const, double *const, double *const, double) *solve*, short unsigned int *autonomous*)

Implement the execution of a Newton's method.

Parameters

in	*n*	Space's dimension
in	*N*	Mesh size
in,out	*x*	Discrete Fourier Transform for each mesh's element, that is, n(2N+1) components
in	*X*	Fourier coefficients
in	*DF*	Matrix of the linear system
in	*w*	Parameter
in	*tol*	Tolerance
in	*f*	Pointer to function given by the user
in	*solve*	Pointer to solver linear systems of equations

Returns

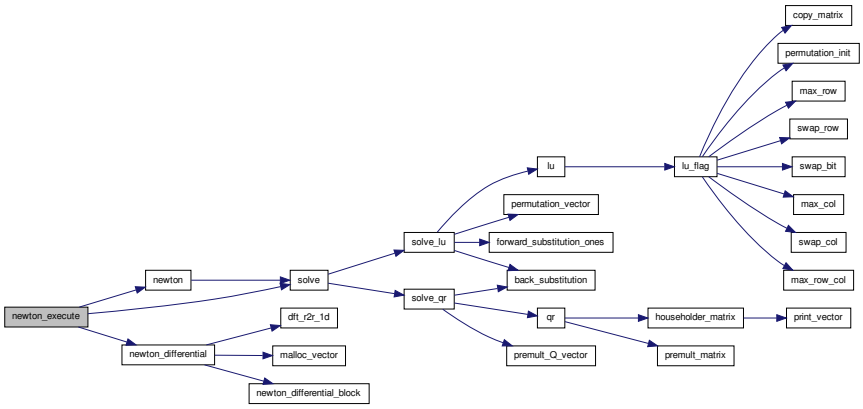
Number of iterations have been performed or NULL if some problem has ocurred

See also

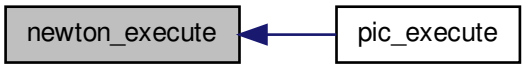
newton_max_iterations

References newton(), newton_differential(), and solve().

Here is the call graph for this function:



Here is the caller graph for this function:



7.6.3.6 unsigned int set_newton_max_iterations (unsigned int *newton_max_iter*)

Set the maximum number of newton iterations.

Parameters

in	*newton_max_iter*	New value

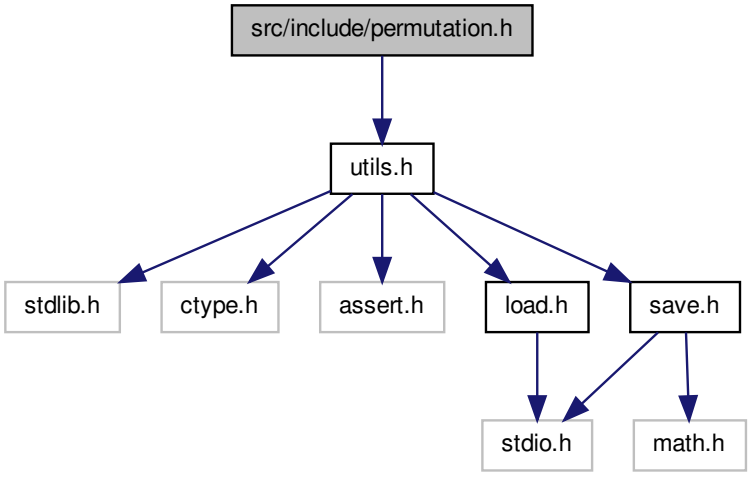
Returns

The old maximum number iterations for Newton's method

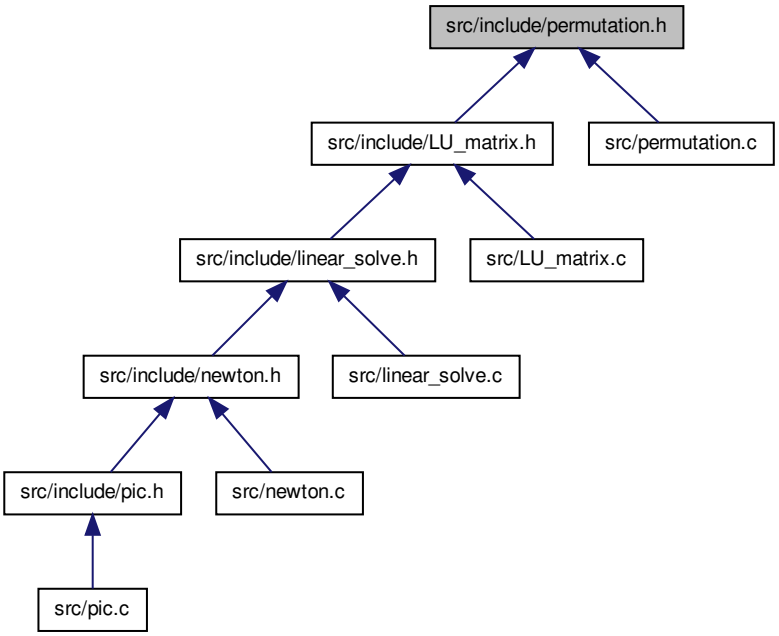
7.7 src/include/permutation.h File Reference

Permutation basic function header file.

`#include "utils.h"` Include dependency graph for permutation.h:



This graph shows which files directly or indirectly include this file:



Functions

- void permutation_init (size_t n, size_t ∗const p)

  Initialize a identity vector permutation.
- void permutation_vector (size_t n, size_t ∗const p, double ∗const x, double ∗const px)

  Permute the values of the vector x into px based on the vector permutation p.
- void swap_bit (size_t i, size_t j, size_t ∗const x, size_t ∗const y)

  Swap two values from vector x to y.

7.7.1 Detailed Description

Permutation basic function header file.

Author

Joan Gimeno

Date

20/01/2014 (start)

Version

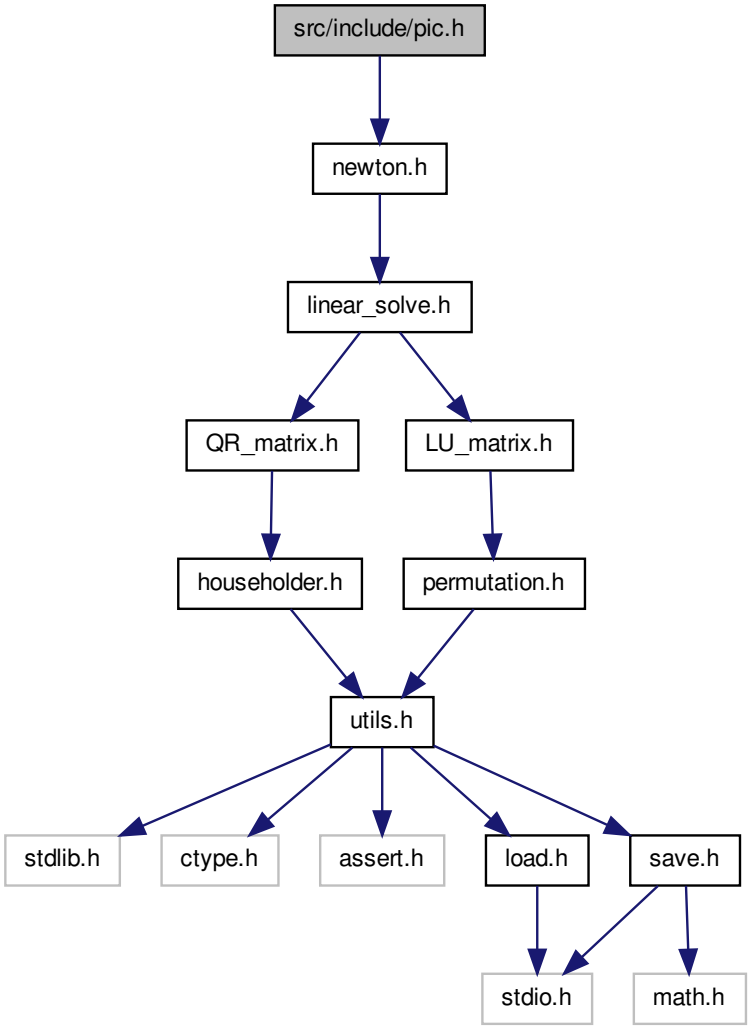
1.0

This file contains the header for basic operation and function of permutations

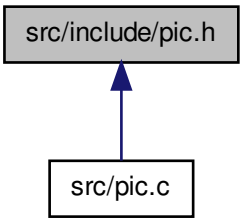
7.8 src/include/pic.h File Reference

Header of periodic invariant curve.

`#include "newton.h"` Include dependency graph for pic.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct pic

  Periodic Invariant Curve.

Defines

- #define PIC_AUTONOMOUS

  The dynamic system is autonomous.
- #define PIC_NAUTONOMOUS

  The dynamic system is not autonomous.

Typedefs

- typedef struct pic pic

Functions

- unsigned int pic_get_max_iterations ()

  Get maximum number of iterations.
- unsigned int pic_set_max_iterations (unsigned int max_iter)

  Set maximum number of iterations.
- size_t pic_get_n (struct pic *const)
- size_t pic_get_N (struct pic *const)
- double * pic_get_x (struct pic *const)
- double * pic_get_X (struct pic *const)
- double ** pic_get_DF (struct pic *const)
- double pic_get_w (struct pic *const)

- size_t pic_get_mesh_size (struct pic *const)
- size_t pic_get_dimension_size (struct pic *const)
- int pic_set_mesh_size (struct pic *const p, size_t N)

  Set the mesh size.
- int pic_set_dimension_size (struct pic *const p, size_t n)

  Set the dimension.
- int pic_set_function_pointer (struct pic *const p, _FUNCTION_PROTOTYPE)
- int pic_set_function_solve (struct pic *const p, _SOLVE_PROTOTYPE)
- struct pic * pic_create (size_t n, size_t N, double w, double *const X, _FUNCTION_PROTOTYPE, _SOLVE_PROTOTYPE)

  Function that initializes a struct pic.
- void * pic_execute (struct pic *const p, double tol)

  Function that compute the Discrete Fourier Transform.
- double ** pic_differential (struct pic *const p, size_t *const m, size_t *const n)

  Return a differential matrix for the current pic struct's values.
- double * pic_load_fourier_coef (size_t n, size_t N, FILE *const f)

  Function that load the Fourier coefficients from a given file.
- double * pic_loadas_fourier_coef (size_t n, size_t N, char *const name)

  Function that load the Fourier coefficients from a given file by name.
- int pic_save_fourier_coef (struct pic *const p, FILE *const f)

  Function that saves the Fourier coefficients in a given file.
- int pic_saveas_fourier_coef (struct pic *const p, char *const name)

  Function that saves the Fourier coefficients in a given file by name.
- int pic_save_curve (struct pic *const p, FILE *const f)

  Function that saves the values of the solution curve in a given file.
- int pic_saveas_curve (struct pic *const p, char *const name)

  Function that saves the values of the solution curve in a given file by name.
- int pic_saveas_differential (struct pic *const p, char *const name)

  Function that saves the values of the differential of Newton method in a given file by name.
- int pic_save_differential (struct pic *const p, FILE *const f)

  Function that saves the values of the differential of Newton method in a given file.
- int pic_saveas_differential_indexs (struct pic *const p, char *const name)

  Function that saves the values of the differential of Newton method in a given file by name from its indexs.
- int pic_save_differential_indexs (struct pic *const p, FILE *const f)

  Function that saves the values of the differential of Newton method in a given file from its indexs.
- void pic_destroy (struct pic *const p)

  Function that frees the memory space pointed to by the differents pointers of struct pic.

7.8.1 Detailed Description

Header of periodic invariant curve.

Author

Joan Gimeno

Date

22/01/2014 (start)

Version

1.0

7.8.2 Define Documentation

7.8.2.1 #define PIC_AUTONOMOUS

The dynamic system is autonomous.

7.8.2.2 #define PIC_NAUTONOMOUS

The dynamic system is not autonomous.

7.8.3 Typedef Documentation

7.8.3.1 pic

7.8.4 Function Documentation

7.8.4.1 struct pic * pic_create (size_t *n*, size_t *N*, double *w*, double *const *X*, _FUNCTION_PROTOTYPE , _SOLVE_PROTOTYPE) [read]

Function that initializes a struct pic.

Parameters

in	*n*	Space's dimension of the function $f: \mathbb{R}^n \to \mathbb{R}^n$
in	*N*	Mesh size of the $\mathbb{T}^1$, 2N+1
in	*w*	Real number densely filling the circumference
in	*X*	Seed of the newton method. The size has to be n(2N+1)
in	*f*	Function pointer
in	*solve*	Function pointer for solves a linear system equations

Return values

NULL	If memory space has not been allocated
p	Pointer to struct pic

Precondition

The space's dimension is not zero

Postcondition

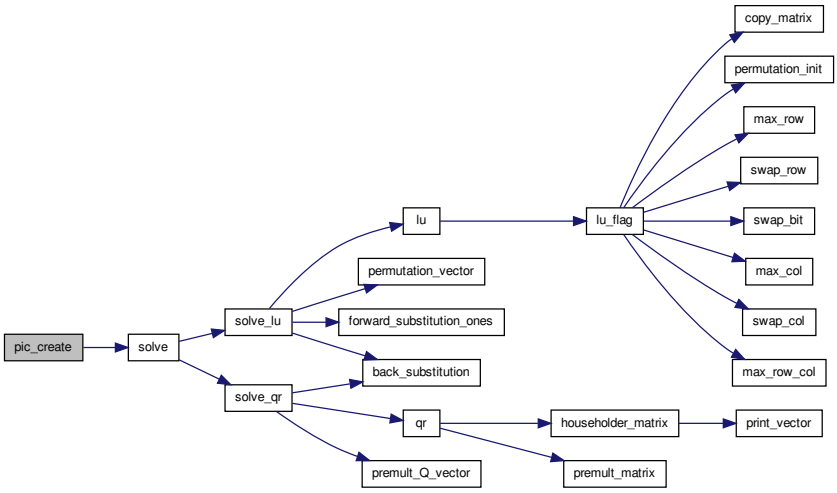
The struct pic has been initialized

Note

If the seed of newton method is NULL, the default seed will be the vector zero

References pic::autonomous, pic::DF, pic::f, pic::n, pic::N, PIC_NAUTONOMOUS, pic::solve, solve(), SOLVE_WITH_LU, pic::w, pic::x, and pic::X.

Here is the call graph for this function:



7.8.4.2 void pic_destroy (struct pic *const *p*)

Function that frees the memory space pointed to by the differents pointers of struct pic.

Parameters

in	*p*	Pointer to periodic invariant curve

Postcondition

The memory space has been freed

References pic::autonomous, pic::DF, pic::n, pic::N, pic::x, and pic::X.

7.8.4.3 double ** pic_differential (struct pic *const p, size_t *const m, size_t *const n)

Return a differential matrix for the current pic struct's values.

Parameters

in	*p*	Pointer to a pic struct
out	*m*	Number of differential matrix's rows
out	*n*	Number of differential matrix's columns

Returns

Pointer to initial position of the matrix differential or NULL if some problem occurred

Precondition

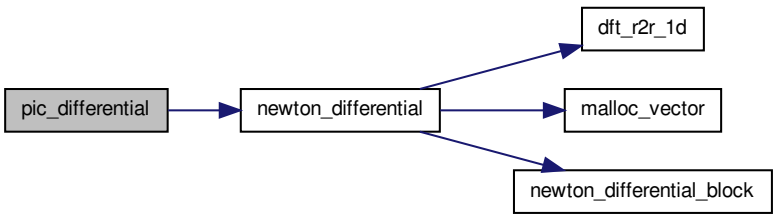
The pointer to a pic struct is not NULL

Note

The pointers for rows and columns can be NULL

References pic::autonomous, pic::DF, pic::f, pic::n, pic::N, newton_differential(), pic::w, pic::x, and pic::X.

Here is the call graph for this function:



7.8.4.4 void * pic_execute (struct pic *const p, double tol)

Function that compute the Discrete Fourier Transform.

Parameters

in	*p*	Pointer to the pic struct
in	*tol*	Tolerance for the result

Returns

NULL if some problem ocurrer. Otherwise the number of iterations performed until the result (minimum 1)

Note

The mesh size of pic struct can have been changed. Use

See also

pic_get_mesh_size

Precondition

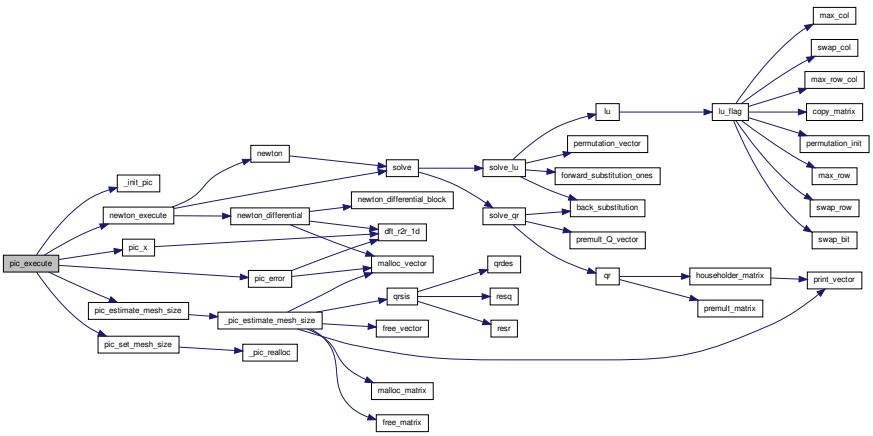
The pointers are not NULL

Postcondition

The vector p->x has the solution

References _init_pic(), pic::autonomous, pic::DF, pic::f, pic::n, pic::N, newton_execute(), pic_error(), pic_estimate_mesh_size(), pic_set_mesh_size(), pic_x(), pic::solve, pic::w, pic::x, and pic::X.

Here is the call graph for this function:



7.8.4.5 double ** pic_get_DF (struct pic * const p)

Parameters

in	*p*	Pointer to a pic struct

Returns

The pointer to the differential of the pic struct

Precondition

The pointer is not NULL

References pic::DF.

7.8.4.6 size_t pic_get_dimension_size (struct pic * const p)

Parameters

in	*p*	Pointer to a pic struct

Returns

Space's dimension of the pic struct

Precondition

The pointer is not NULL

References pic::n.

7.8.4.7 unsigned int pic_get_max_iterations ( )

Get maximum number of iterations.

Returns

Maximum iterations that increase the mesh size

7.8.4.8 size_t pic_get_mesh_size (struct pic * const p)

Parameters

in	*p*	Pointer to a pic struct

Returns

Mesh size of the pic struct

Precondition

The pointer is not NULL

References pic::N.

7.8.4.9 size_t pic_get_n (struct pic * const p)

Parameters

in	*p*	Pointer to a pic struct

Returns

Space's dimension of the pic struct

Precondition

The pointer is not NULL

References pic::n.

7.8.4.10 size_t pic_get_N (struct pic * const p)

Parameters

in	*p*	Pointer to a pic struct

Returns

The mesh size of the pic struct

Precondition

The pointer is not NULL

References pic::N.

7.8.4.11 double pic_get_w (struct pic * const p)

Parameters

in	*p*	Pointer to a pic struct

Returns

The real number of the pic struct

Precondition

The pointer is not NULL

References pic::w.

7.8.4.12 double * pic_get_x (struct pic * const p)

Parameters

in	*p*	Pointer to a pic struct

Returns

The value table of the curve solution of the pic struct

Precondition

The pointer is not NULL

References pic::x.

7.8.4.13 double * pic_get_X (struct pic * const p)

Parameters

in	*p*	Pointer to a pic struct

Returns

The Fourier's coefficients of the pic struct

Precondition

The pointer is not NULL

References pic::X.

7.8.4.14 double * pic_load_fourier_coef (size_t *n*, size_t *N*, FILE *const *f*)

Function that load the Fourier coefficients from a given file.

Parameters

in	*n*	Dimension of each coefficient
in	*N*	Number of coefficient
in	*f*	Pointer to a file

Returns

n∗(2N+1)-dimensional vector with the Fourier coefficients saved

Precondition

The pointer is not NULL

Postcondition

The user has to free the memory for the pointer returned

Note

The structure of the vector is a_0, a_1, b_1, ..., a_N, b_N where each element is a n-dimensional vector

References _MAX_LENGTH_STRING_INPUT, and pic::n.

Here is the caller graph for this function:

7.8.4.15 double ∗ pic_loadas_fourier_coef (size_t *n*, size_t *N*, char ∗const *name*)

Function that load the Fourier coefficients from a given file by name.

Parameters

in	*n*	Dimension of each coefficient
in	*N*	Number of coefficient
in	*f*	Pointer to a file

Returns

n∗(2N+1)-dimensional vector with the Fourier coefficients saved

Postcondition

The user has to free the memory for the pointer returned

Precondition

The pointer is not NULL

Note

The structure of the vector is a_0, a_1, b_1, ..., a_N, b_N where each element is a n-dimensional vector

References pic::f, and pic_load_fourier_coef().

Here is the call graph for this function:

7.8.4.16 int pic_save_curve (struct pic *const p, FILE *const f)

Function that saves the values of the solution curve in a given file.

Parameters

in	*p*	Pointer to struct pic
in	*f*	Output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

References pic::n, pic::N, and pic::x.

Here is the caller graph for this function:

7.8.4.17 int pic_save_differential (struct pic *const p, FILE *const f)

Function that saves the values of the differential of Newton method in a given file.

Parameters

in	*p*	Pointer to struct pic
in	*f*	Output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

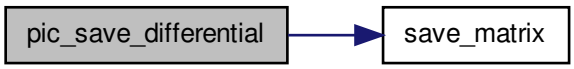
The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

References pic::DF, pic::n, pic::N, and save_matrix().

Here is the call graph for this function:



7.8.4.18 int pic_save_differential_indexs (struct pic *const p, FILE *const f)

Function that saves the values of the differential of Newton method in a given file from its indexs.

Parameters

in	*p*	Pointer to struct pic
in	*f*	Output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

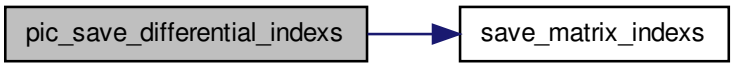
The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

References pic::DF, pic::n, pic::N, and save_matrix_indexs().

Here is the call graph for this function:



7.8.4.19 int pic_save_fourier_coef (struct pic *const p, FILE *const f)

Function that saves the Fourier coefficients in a given file.

Parameters

in	*p*	Pointer to struct pic
in	*f*	Output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The pointers are not NULL

Postcondition

The file has been created or modified with the result

References pic::n, pic::N, and pic::X.

Here is the caller graph for this function:

7.8.4.20 int pic_saveas_curve (struct pic *const p, char *const name)

Function that saves the values of the solution curve in a given file by name.

Parameters

in	*p*	Pointer to struct pic
in	*name*	Name of the output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The struct pointer is not NULL

Postcondition

The file with name has been created or modified with the result

References pic::f, and pic_save_curve().

Here is the call graph for this function:

7.8.4.21 int pic_saveas_differential (struct pic *const p, char *const name)

Function that saves the values of the differential of Newton method in a given file by name.

Parameters

in	*p*	Pointer to struct pic
in	*name*	Name of the output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

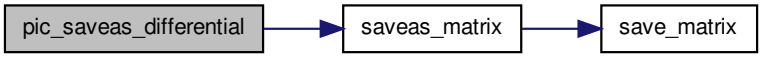
The struct pointer is not NULL

Postcondition

The file with name has been created or modified with the result

References pic::DF, pic::n, pic::N, and saveas_matrix().

Here is the call graph for this function:



7.8.4.22 int pic_saveas_differential_indexs (struct pic ∗const *p*, char ∗const *name*)

Function that saves the values of the differential of Newton method in a given file by name from its indexs.

Parameters

in	*p*	Pointer to struct pic
in	*name*	Name of the output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

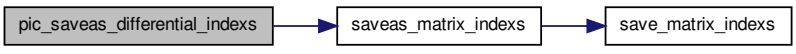
The struct pointer is not NULL

Postcondition

The file with name has been created or modified with the result

References pic::DF, pic::n, pic::N, and saveas_matrix_indexs().

Here is the call graph for this function:



7.8.4.23 int pic_saveas_fourier_coef (struct pic ∗const *p*, char ∗const *name*)

Function that saves the Fourier coefficients in a given file by name.

Parameters

in	*p*	Pointer to struct pic
in	*f*	Output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The pointers are not NULL

Postcondition

The file has been created or modified with the result

References pic::f, and pic_save_fourier_coef().

Here is the call graph for this function:

7.8.4.24 int pic_set_dimension_size (struct pic ∗const *p*, size_t *n*)

Set the dimension.

Parameters

in	*p*	Pointer to a pic struct
in	*N*	New dimension

Returns

0 if some error occurred or the old dimension

Precondition

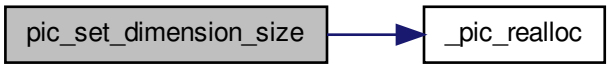
The new mesh size is not 0 and the pointer is not NULL

Postcondition

The pointer memory has been reallocated with the new dimension

References _pic_realloc(), pic::autonomous, pic::n, and pic::N.

Here is the call graph for this function:



7.8.4.25 int pic_set_function_pointer (struct pic *const p, _FUNCTION_PROTOTYPE)

Parameters

in	*p*	Pointer to a pic struct
in	*f*	Pointer to a function pointer

Return values

0	if some argument is NULL
1	Otherwise

7.8.4.26 int pic_set_function_solve (struct pic *const p, _SOLVE_PROTOTYPE)

Parameters

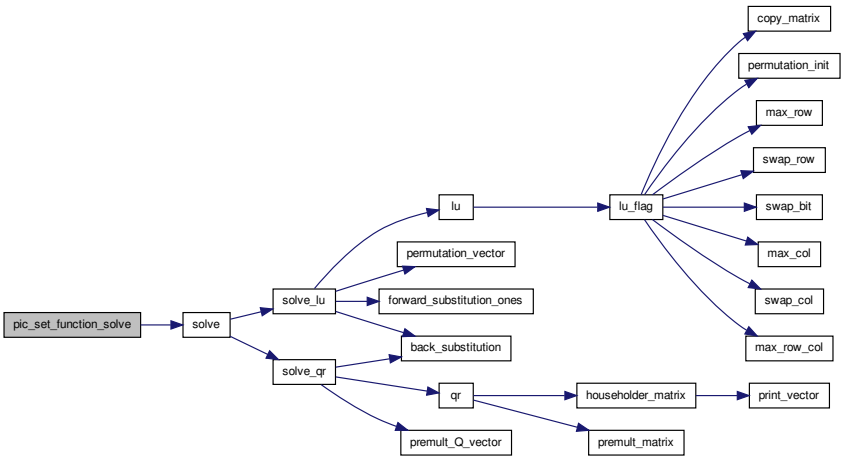
in	*p*	Pointer to a pic struct
in	*f*	Pointer to a function pointer

Return values

0	if some argument is NULL
1	Otherwise

References pic::solve, and solve().

Here is the call graph for this function:



7.8.4.27 unsigned int pic_set_max_iterations (unsigned int *max_iter*)

Set maximum number of iterations.

Parameters

in	*max_iter*	New value

Returns

The old maximum iterations that increase the mesh size

7.8.4.28 int pic_set_mesh_size (struct pic *const *p*, size_t *N*)

Set the mesh size.

Parameters

in	*p*	Pointer to a pic struct
in	*N*	New mesh size

Returns

0 if some error occurred or the old mesh size

Precondition

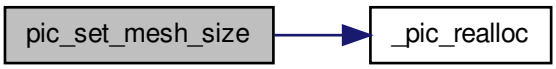
The new mesh size is not 0 and the pointer is not NULL

Postcondition

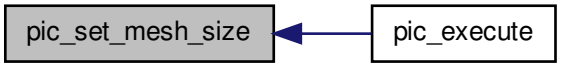
The pointer memory has been reallocated with the new mesh size

References _pic_realloc(), pic::autonomous, pic::n, and pic::N.

Here is the call graph for this function:



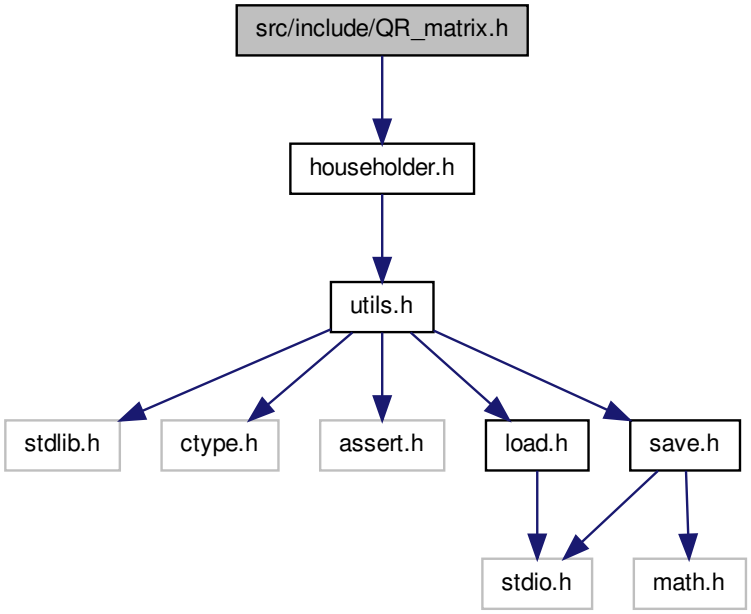
Here is the caller graph for this function:



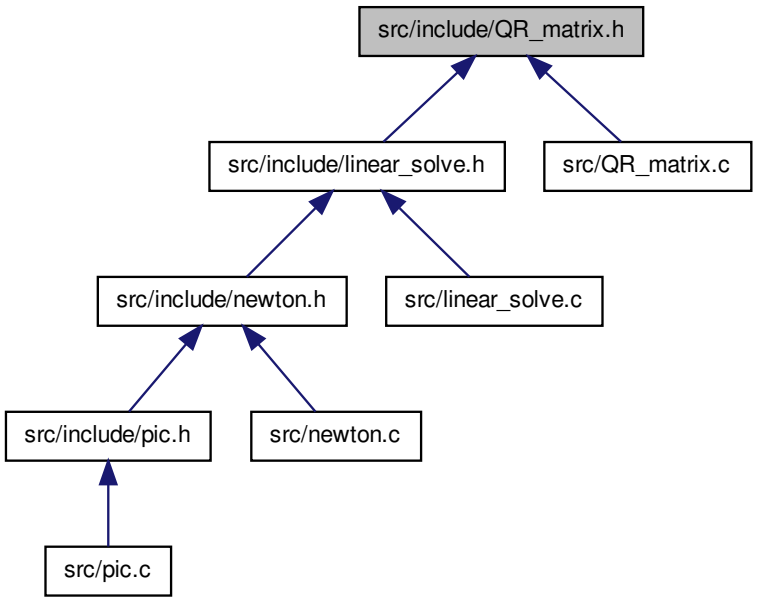
7.9 src/include/QR_matrix.h File Reference

QR method header file.

`#include "householder.h"` Include dependency graph for QR_matrix.h:



This graph shows which files directly or indirectly include this file:



Functions

- void ∗ qr (size_t m, size_t n, double ∗∗const A, double ∗∗const QR)

  Compute the QR factorization of a matrix m-by-n.
- void premult_Q_vector (size_t m, size_t n, double ∗∗const QR, double ∗const b)

  Compute $P_1 \cdots P_n b$ where b and P_i are respectively m-by-1 and m-by-m matrixs.

7.9.1 Detailed Description

QR method header file.

Author

Joan Gimeno

Date

22/11/2013 (start)

Version

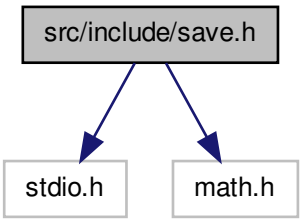
1.0

This file contains the header for ::QR_method

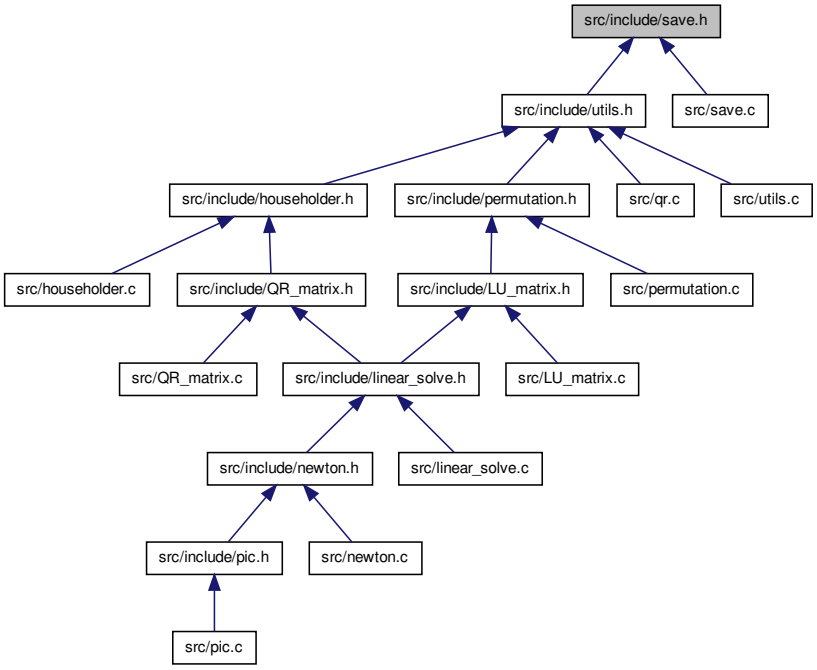
7.10 src/include/save.h File Reference

Header of save.

`#include <stdio.h>` `#include <math.h>` Include dependency graph for save.h:



This graph shows which files directly or indirectly include this file:



Functions

- int save_vector (size_t, size_t, double ∗const, FILE ∗const)

  Function that saves the values of a vector in a given file.
- int saveas_vector (size_t, size_t, double ∗const, char ∗const)

  Function that saves the values of a vector in a given filename.
- int save_vector_indexs (size_t, size_t, double ∗const, FILE ∗const)

  Function that saves the values not zero of a vector with its indexs in a given file.
- int saveas_vector_indexs (size_t, size_t, double ∗const, char ∗const)

  Function that saves the values not zero of a vector with its indexs in a given filename.
- int save_matrix (size_t, size_t, size_t, size_t, double ∗∗const, FILE ∗const)

  Function that saves the values of a matrix in a given file.
- int saveas_matrix (size_t, size_t, size_t, size_t, double ∗∗const, char ∗const)

  Function that saves the values of a matrix in a given filename.
- int save_matrix_indexs (size_t, size_t, size_t, size_t, double ∗∗const, FILE ∗const)

  Function that saves the values not zero of a matrix with its indexs in a given file.

- int saveas_matrix_indexs (size_t, size_t, size_t, size_t, double **const, char *const)

  Function that saves the values not zero of a matrix with its indexs in a given filename.

7.10.1 Detailed Description

Header of save.

Author

Joan Gimeno

Date

22/05/2014 (start)

Version

1.0

7.10.2 Function Documentation

7.10.2.1 int save_matrix (size_t *idx_row*, size_t *m*, size_t *idx_col*, size_t *n*, double ** const *A*, FILE * const *f*)

Function that saves the values of a matrix in a given file.

Parameters

in	*idx_row*	Start index row
in	*m*	Rows
in	*idx_col*	Start index column
in	*n*	Columns
in	*A*	Pointer to a matrix
in	*f*	Output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

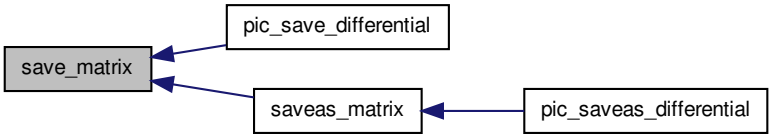
The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

References pic::n.

Here is the caller graph for this function:



7.10.2.2 int save_matrix_indexs (size_t *idx_row*, size_t *m*, size_t *idx_col*, size_t *n*, double ** const *A*, FILE * const *f*)

Function that saves the values not zero of a matrix with its indexs in a given file.

Parameters

in	*idx_row*	Start index row
in	*m*	Rows
in	*idx_col*	Start index column
in	*n*	Columns
in	*A*	Pointer to a matrix
in	*f*	Output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

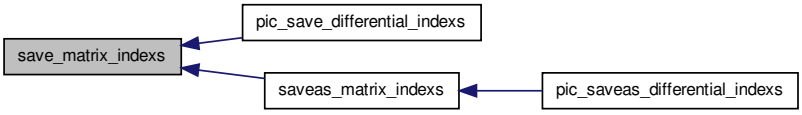
The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

References EPS, and pic::n.

Here is the caller graph for this function:



7.10.2.3 int save_vector (size_t *idx*, size_t *n*, double * const *b*, FILE * const *f*)

Function that saves the values of a vector in a given file.

Parameters

in	*idx*	Start index
in	*n*	Dimension
in	*b*	Pointer to a vector
in	*f*	Output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

References pic::n.

Here is the caller graph for this function:

7.10.2.4 int save_vector_indexs (size_t *idx*, size_t *n*, double * const *b*, FILE * const *f*)

Function that saves the values not zero of a vector with its indexs in a given file.

Parameters

in	*idx*	Start index
in	*n*	Dimension
in	*b*	Pointer to a vector
in	*f*	Output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

References EPS, and pic::n.

Here is the caller graph for this function:

7.10.2.5 int saveas_matrix (size_t *idx_row*, size_t *m*, size_t *idx_col*, size_t *n*, double ** const *A*, char * const *name*)

Function that saves the values of a matrix in a given filename.

Parameters

in	*idx_row*	Start index row
in	*m*	Rows
in	*idx_col*	Start index column
in	*n*	Columns
in	*A*	Pointer to a matrix
in	*name*	Filename

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

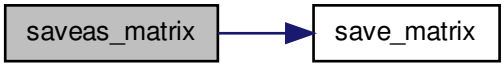
The pointers are not NULL

Postcondition

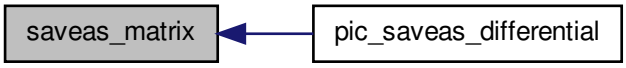
The file with name has been created or modified with the result

References pic::f, and save_matrix().

Here is the call graph for this function:



Here is the caller graph for this function:



7.10.2.6 int saveas_matrix_indexs (size_t *idx_row*, size_t *m*, size_t *idx_col*, size_t *n*, double ** const *A*, char * const *name*)

Function that saves the values not zero of a matrix with its indexs in a given filename.

Parameters

in	*idx_row*	Start index row
in	*m*	Rows

in	*idx_col*	Start index column
in	*n*	Columns
in	*A*	Pointer to a matrix
in	*name*	Filename

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The pointers are not NULL

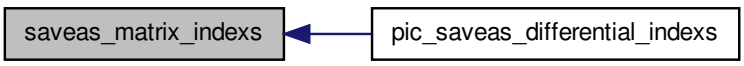
Postcondition

The file with name has been created or modified with the result

References pic::f, and save_matrix_indexs().

Here is the call graph for this function:

Here is the caller graph for this function:



7.10.2.7 int saveas_vector (size_t *idx*, size_t *n*, double * const *b*, char * const *name*)

Function that saves the values of a vector in a given filename.

Parameters

in	*idx*	Start index
in	*n*	Dimension
in	*b*	Pointer to a vector
in	*name*	Filename

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

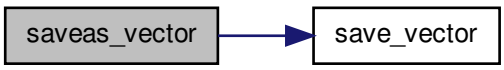
The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

References pic::f, and save_vector().

Here is the call graph for this function:



7.10.2.8 int saveas_vector_indexs (size_t *idx*, size_t *n*, double * const *b*, char * const *name*)

Function that saves the values not zero of a vector with its indexs in a given filename.

Parameters

in	*idx*	Start index
in	*n*	Dimension
in	*b*	Pointer to a vector
in	*name*	Filename

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

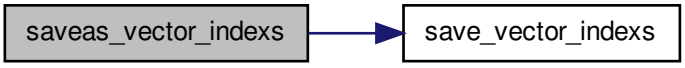
The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

References pic::f, and save_vector_indexs().

Here is the call graph for this function:

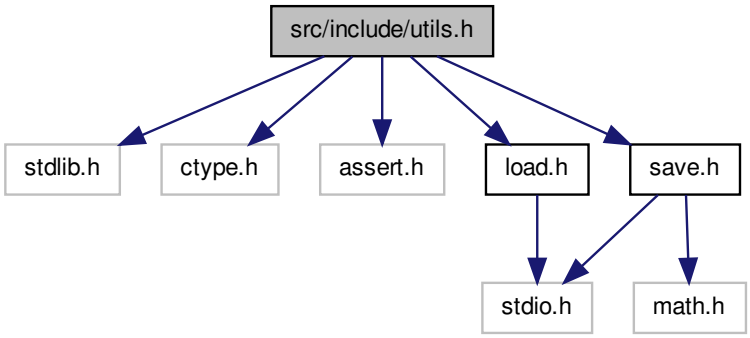


7.11 src/include/utils.h File Reference

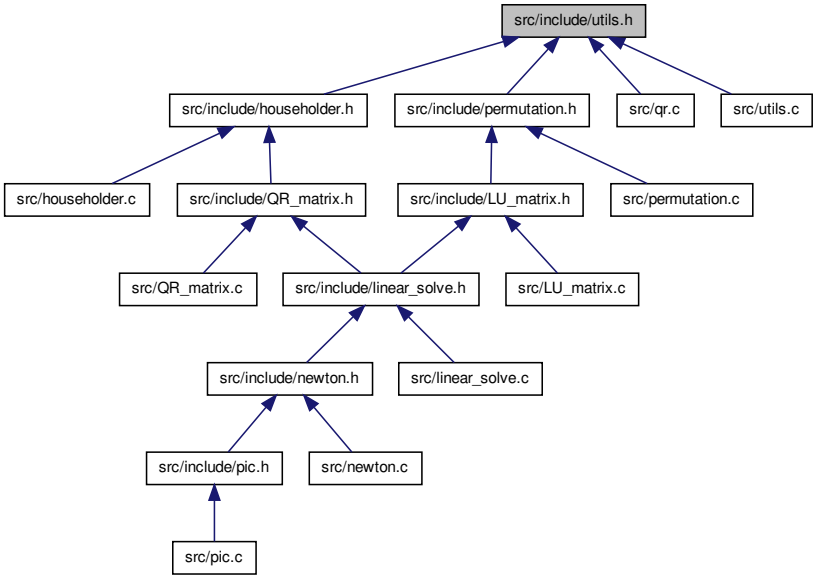
Utilities.

`#include <stdlib.h>` `#include <ctype.h>` `#include <assert.h>` `#include "load.h"` `#include "save.h"` Include dependency graph for

utils.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define inline static

- #define SWAP(a, b) typeof(a) _SWAPPING_VAR; _SWAPPING_VAR = a; a = b; b = _SWAPPING_VAR

  A macro that swaps two generic values.
- #define free_mem(a)

  Used to help ensure everything malloc'ed gets freed.

Functions

- double euclidian_norm (size_t n, double *const x)

  Derive the euclidian norm of a vector, that is, $||x||_2 = \left(\sum_{i=0}^{n-1} x_i^2\right)^{\frac{1}{2}}$.
- double inner_product (size_t n, double *const x)

  Derive the ordinary inner product of a vector, that is, $\langle x,x\rangle = \sum_{i=0}^{n-1} x_i^2$.
- void copy_vector (size_t n, double *const x, double *const y)

  Copy the vector x in y.
- void copy_matrix (size_t m, size_t n, double **const A, double **const B)

  Copy the matrix A in B.
- void print_matrix (size_t m, size_t n, double **const A, char *const name)

  Display in stdout a matrix m-by-n.
- void print_vector (size_t idx, size_t n, double *const v, char *const name)

  Display in stdout a n-dimensional vector starting by an index.
- double * malloc_vector (size_t n)

  Malloc a n-dimensional vector.
- void free_vector (double *p)

  Free a vector.
- double ** malloc_matrix (size_t m, size_t n)

  Malloc a m-by-n matrix.
- void free_matrix (size_t m, double **p)

  Free a matrix with m rows.
- int qrsis (int n, int m, double **a, double *b, double tol)

7.11.1 Detailed Description

Utilities.

Author

Joan Gimeno

Date

12/11/2013 (start)

7.11.2 Define Documentation

7.11.2.1 #define free_mem(a)

Used to help ensure everything malloc'ed gets freed.

7.11.2.2 #define inline static

7.11.2.3 #define SWAP(a, b) typeof(a) _SWAPPING_VAR; _SWAPPING_VAR = a; a = b; b = _SWAPPING_VAR

A macro that swaps two generic values.

7.11.3 Function Documentation

7.11.3.1 void copy_matrix (size_t m, size_t n, double **const A, double **const B)

Copy the matrix A in B.

Parameters

in	*m*	Rows
in	*m*	Columns
in	*A*	Source matrix
out	*B*	Destination matrix

Precondition

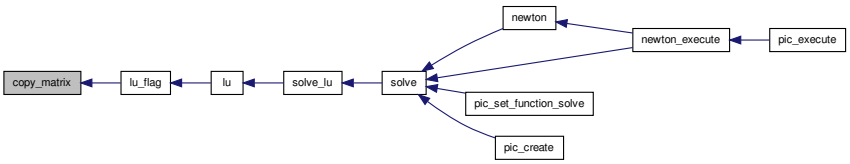
The pointers are not NULL

Warning

The pointers should not be the same

References pic::n.

Here is the caller graph for this function:



7.11.3.2 void copy_vector (size_t *n*, double *const *x*, double *const *y*)

Copy the vector x in y.

Parameters

in	*n*	Dimension
in	*x*	Source vector
out	*y*	Destination vector

Precondition

The pointers are not NULL

Warning

The pointers should not be the same

References pic::n.

7.11.3.3 double euclidian_norm (size_t *n*, double *const *x*)

Derive the euclidian norm of a vector, that is, $||x||_2 = \left(\sum_{i=0}^{n-1} x_i^2\right)^{\frac{1}{2}}$.

Parameters

in	*n*	Dimension of vector
in	*x*	Input vector

Returns

Euclidian norm

Postcondition

Thre result is greater than or equal to zero

Note

If the pointer is NULL, the result is zero

References pic::n.

Here is the caller graph for this function:

7.11.3.4 void free_matrix (size_t *m*, double ** *p*)

Free a matrix with m rows.

Parameters

in	*m*	Rows

Here is the caller graph for this function:

7.11.3.5 void free_vector (double * *p*)

Free a vector.

Parameters

in	*p*	Pointer

Here is the caller graph for this function:

7.11.3.6 double inner_product (size_t *n*, double *const *x*)

Derive the ordinary inner product of a vector, that is, $\langle x,x\rangle = \sum_{i=0}^{n-1} x_i^2$.

Parameters

in	*n*	Dimension of vector
in	*x*	Input vector

Returns

Inner product

Postcondition

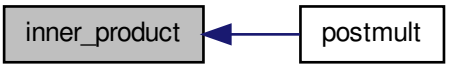
The result is greater than or equal to zero

Note

If the pointer is NULL, the result is zero

References pic::n.

Here is the caller graph for this function:



7.11.3.7 double ** malloc_matrix (size_t *m*, size_t *n*)

Malloc a m-by-n matrix.

Parameters

in	*m*	Rows
in	*n*	Columns

Returns

Pointer to a matrix

Note

The return value could be NULL

Here is the caller graph for this function:

7.11.3.8 double * malloc_vector (size_t *n*)

Malloc a n-dimensional vector.

Parameters

in	*n*	End index

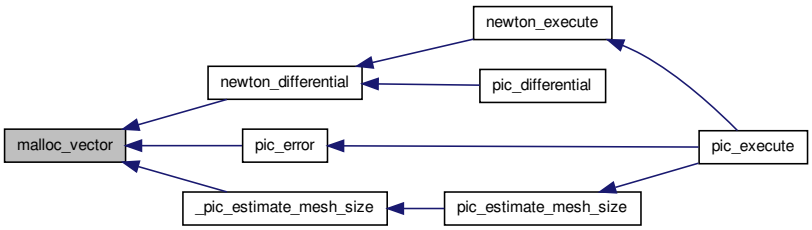
Returns

Pointer to a malloc vector

Note

The return value could be NULL

Here is the caller graph for this function:



7.11.3.9 void print_matrix (size_t *m*, size_t *n*, double **const *A*, char *const *name*)

Display in stdout a matrix m-by-n.

Parameters

in	*m*	Rows
in	*n*	Columns
in	*A*	Matrix m-by-n
in	*name*	Name of the matrix in stdout

Precondition

The dimension are not zeros and the pointers are not NULL

References pic::n.

7.11.3.10 void print_vector (size_t *idx*, size_t *n*, double *const *v*, char *const *name*)

Display in stdout a n-dimensional vector starting by an index.

Parameters

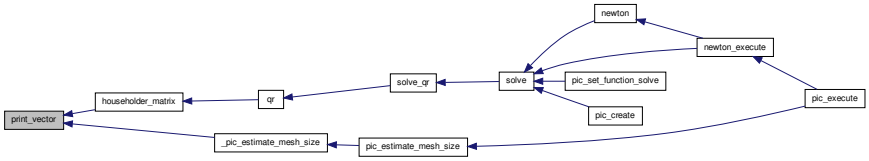
in	*idx*	Start index
in	*n*	End index
in	*v*	Vector
in	*name*	Name of the vector in stdout

Precondition

The idx is less than n and the pointers are not NULL

References pic::n.

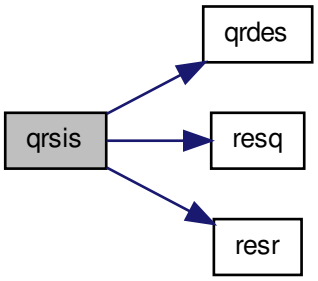
Here is the caller graph for this function:



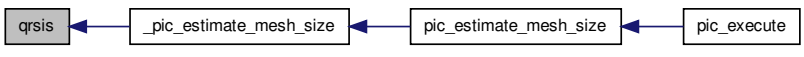
7.11.3.11 int qrsis (int *n*, int *m*, double ** *a*, double * *b*, double *tol*)

References pic::n, qrdes(), resq(), and resr().

Here is the call graph for this function:



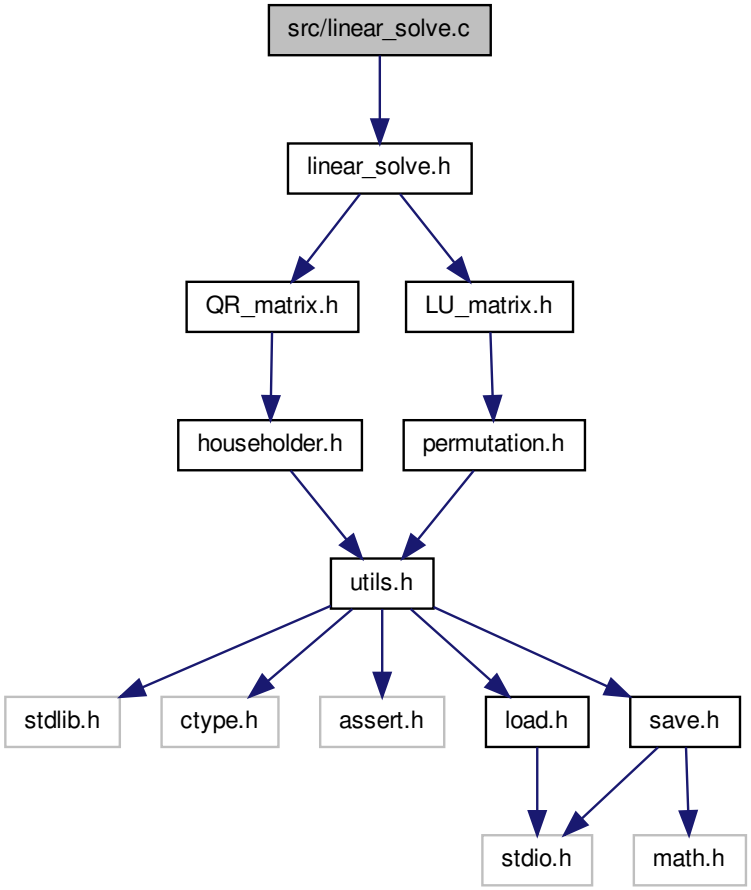
Here is the caller graph for this function:



7.12 src/linear_solve.c File Reference

Set of different function for solving linear systems.

`#include "linear_solve.h"` Include dependency graph for linear_solve.c:



Functions

- void forward_substitution_ones (size_t n, double ∗∗const L, double ∗const x, double ∗const b)

  Forward substitution for lower triangular matrices (Lx = b) where the diagonal of L are 1's.
- void ∗ forward_substitution (size_t n, double ∗∗const L, double ∗const x, double ∗const b, double tol)

Forward substitution for lower triangular matrices (Lx = b)

- void * back_substitution (size_t n, double **const U, double *const x, double *const b, double tol)

  Back substitution for upper triangular matrices (Ux = b)
- void * solve_qr (size_t m, size_t n, double **const A, double *const x, double *const b, double tol)

  QR solve linear system (QRx = b)
- void * solve_lu (size_t m, size_t n, double **const A, double *const x, double *const b, double tol)

  LU solve linear system (LUx = b)
- void * solve (size_t m, size_t n, double **const A, double *const x, double *const b, double tol, void *(*s)(size_t, size_t, double **const, double *const, double *const, double))

  solve a linear system (Ax = b) with the pointer to a solve function
- solve_ptr get_solve_with_lu ()
- solve_ptr get_solve_with_qr ()

7.12.1 Detailed Description

Set of different function for solving linear systems.

Author

Joan Gimeno

Date

22/11/2013 (start)

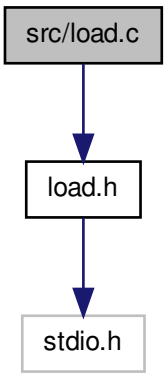
Version

1.0

7.13 src/load.c File Reference

Implements functions that load a vector or a matrix in some file.

`#include "load.h"` Include dependency graph for load.c:



Functions

- int load_vector (double ∗const b, FILE ∗const f)

  Function that loads the values of a vector in a given file.
- int loadas_vector (double ∗const b, char ∗const name)

  Function that loads the values of a vector in a given filename.
- int load_matrix (double ∗∗const A, FILE ∗const f)

  Function that loads the values of a matrix in a given file.
- int loadas_matrix (double ∗∗const A, char ∗const name)

  Function that loads the values of a matrix in a given filename.

7.13.1 Detailed Description

Implements functions that load a vector or a matrix in some file.

Author

Joan Gimeno

Date

22/05/2014 (start)

Version

1.0

7.13.2 Function Documentation

7.13.2.1 int load_matrix (double **const *A*, FILE *const *f*)

Function that loads the values of a matrix in a given file.

Parameters

in	*A*	Pointer to a matrix
in	*f*	Output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

Here is the caller graph for this function:

7.13.2.2 int load_vector (double *const b, FILE *const f)

Function that loads the values of a vector in a given file.

Parameters

in	*b*	Pointer to a vector
in	*f*	Output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

Here is the caller graph for this function:

7.13.2.3 int loadas_matrix (double **const A, char *const name)

Function that loads the values of a matrix in a given filename.

Parameters

in	*A*	Pointer to a matrix
in	*name*	Filename

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

References load_matrix().

Here is the call graph for this function:

7.13.2.4 int loadas_vector (double *const b, char *const name)

Function that loads the values of a vector in a given filename.

Parameters

in	*b*	Pointer to a vector
in	*name*	Filename

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

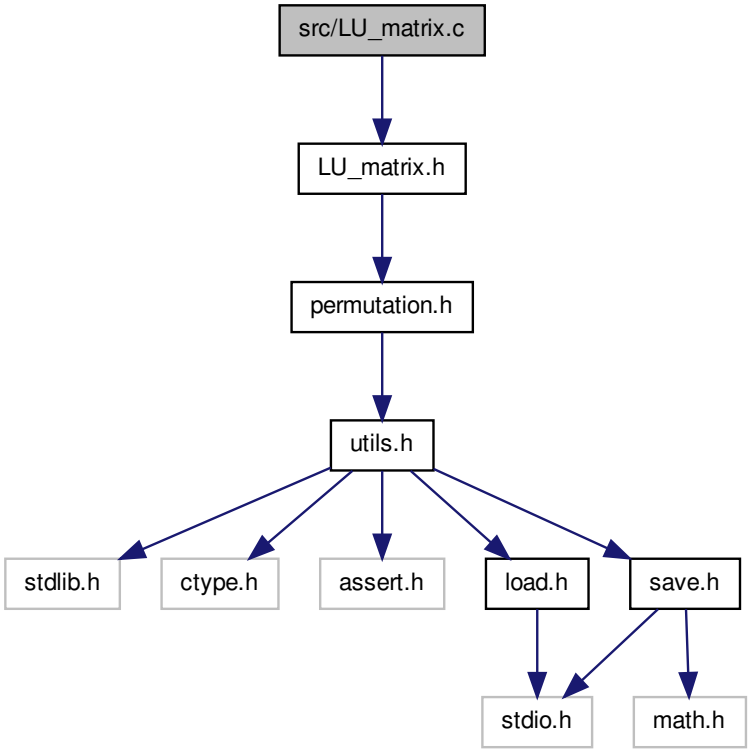
References load_vector().

Here is the call graph for this function:

7.14 src/LU_matrix.c File Reference

Functions for computing LU decomposition of any square matrix.

`#include "LU_matrix.h"` Include dependency graph for LU_matrix.c:



Defines

- #define _OPENMP_LU_DIMENSION 1e2

  Minimum value for using the openmp library in LU decomposition.

Functions

- double max_row (size_t m, double ∗∗const A, size_t idx, size_t ∗const row)

  Find the maximum value of a column based on the same row index.
- double max_col (size_t n, double ∗∗const A, size_t idx, size_t ∗const col)

  Find the first maximum value of a column based on the same column index.
- double max_row_col (size_t m, size_t n, double ∗∗const A, size_t idx, size_t ∗const row, size_t ∗const col)

Find the maximum value of a column and row based on the same column and row index.

- void swap_bit (size_t i, size_t j, size_t *const x, size_t *const y)

  Swap two values from vector x to y.
- void swap_row (size_t row1, size_t row2, size_t n, double **const A, double **const B)

  Swap two rows from matrix A to B based on the same minimum row index.
- void swap_col (size_t col1, size_t col2, size_t m, double **const A, double **const B)

  Swap two columns from matrix A to B based on the same minimum column index.
- void * lu_flag (size_t m, size_t n, double **const A, double **const LU, size_t *const pr, size_t *const pc, double tol, unsigned int flag)

  Compute the LU factorization of a matrix m-by-n with a specific pivoting.
- void * lu (size_t m, size_t n, double **const A, double **const LU, size_t *const pr, size_t *const pc, double tol)

  Compute the LU factorization of a matrix n-by-n.

7.14.1 Detailed Description

Functions for computing LU decomposition of any square matrix.

Author

Joan

Date

19/01/2014 (start)

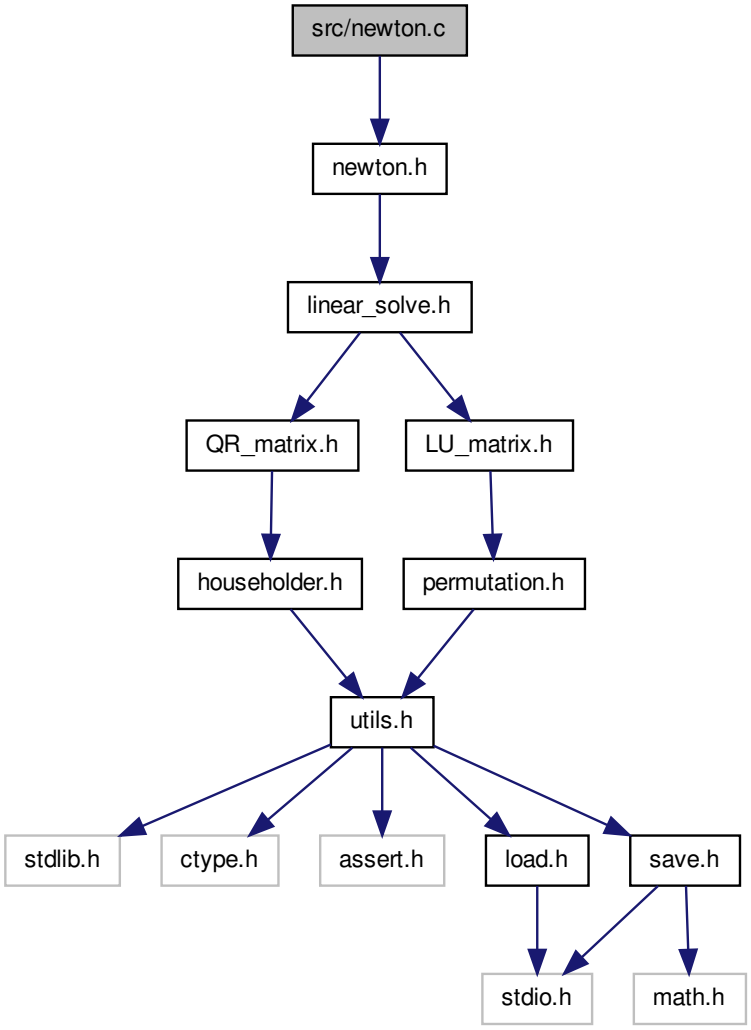
Version

1.0

7.15 src/newton.c File Reference

Implements the functions for a Newton's method.

`#include "newton.h"` Include dependency graph for newton.c:



Defines

- #define _DEFAULT_NEWTON_MAX_ITERATIONS 10

  Maximum number of newton iterations.

Functions

- unsigned int get_newton_max_iterations ()

  Get the maximum number of newton iterations.
- unsigned int set_newton_max_iterations (unsigned int newton_max_iter)

  Set the maximum number of newton iterations.
- double * malloc_vector2 (size_t n)
- int newton (size_t m, size_t n, double *const x, double *const f, double **const df, void *(*solve)(size_t, size_t, double **const, double *const, double *const, double), double tol)

  Implements the compute of a newton method's iteration.
- void newton_differential_block (size_t m, size_t n, double *const x, double *const f, double **const df, double w, double t)

  Compute a block of the differential for the Newton's method.
- void dft_r2r_1d (size_t n, size_t N, double *const dft, double *const x, double theta)

  Compute the Discrete Fourier Transform.
- void newton_differential (size_t n, size_t N, double *const x, double *const X, double **const DF, void(*f)(size_t, size_t, double *const, double *const, double **const, double, double), double w, double *h)

  Compute the differential for the Newton's method.
- void * newton_execute (size_t n, size_t N, double *const x, double *const X, double **const DF, double w, double tol, void(*f)(size_t, size_t, double *const, double *const, double **const, double, double), void *(*solve)(size_t, size_t, double **const, double *const, double *const, double), short unsigned int autonomous)

  Implement the execution of a Newton's method.

7.15.1 Detailed Description

Implements the functions for a Newton's method.

Author

Joan Gimeno

Date

22/01/2014 (start)

Version

1.0

7.15.2 Define Documentation

7.15.2.1 #define _DEFAULT_NEWTON_MAX_ITERATIONS 10

Maximum number of newton iterations.

7.15.3 Function Documentation

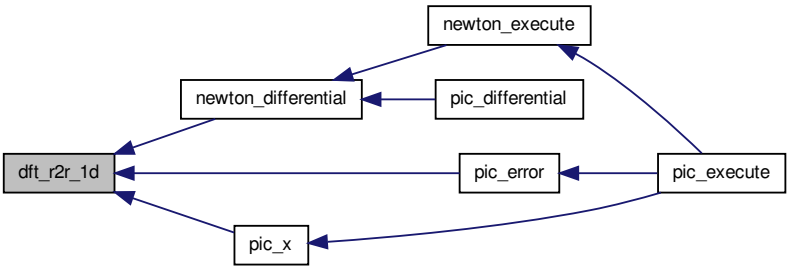
7.15.3.1 void dft_r2r_1d (size_t *n*, size_t *N*, double ∗const *dft*, double ∗const *x*, double *theta*)

Compute the Discrete Fourier Transform.

Parameters

in	*n*	Space's dimension
in	*N*	Mesh size
out	*x*	Discrete Fourier Transform
in	*X*	Fourier coefficients
in	*theta*	Angle evaluation

Here is the caller graph for this function:



7.15.3.2 unsigned int get_newton_max_iterations ( )

Get the maximum number of newton iterations.

Returns

Maximum number iterations for Newton's method

7.15.3.3 double∗ malloc_vector2 (size_t *n*)

7.15.3.4 int newton (size_t *m*, size_t *n*, double ∗const *x*, double ∗const *f*, double ∗∗const *df*, void ∗(∗)(size_t, size_t, double ∗∗const, double ∗const, double ∗const, double) *solve*, double *tol*)

Implements the compute of a newton method's iteration.

Parameters

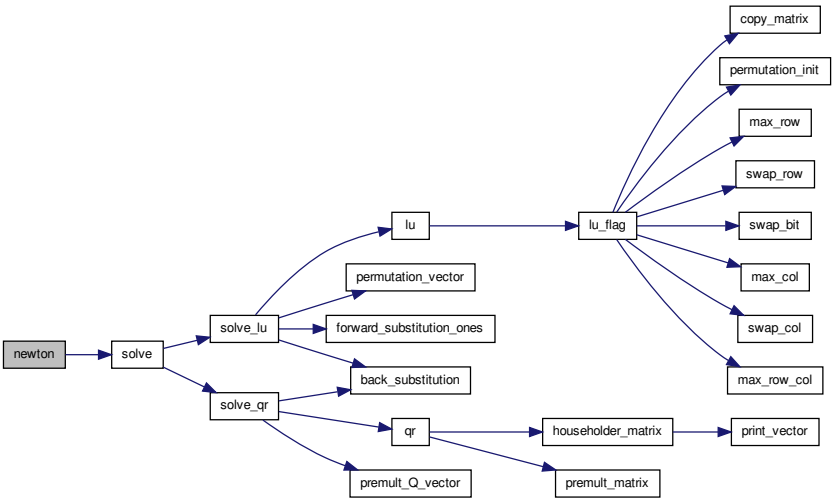
in	*m*	Rows of the matrix
in	*n*	Columns of the matrix
in	*x*	Current iteration value
out	*x*	Next iteration value
in	*f*	Constant terms
in	*df*	Matrix of the linear system
in	∗*solve*	Pointer to solver linear systems of equations
in	*tol*	Tolerance

Return values

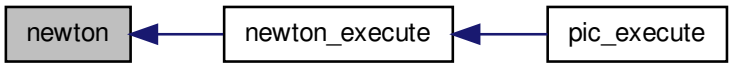
0	Some problem has ocurred
1	When the solution x is accepted by the tolerance given

References solve().

Here is the call graph for this function:



Here is the caller graph for this function:



7.15.3.5 void newton_differential (size_t *n*, size_t *N*, double *const *x*, double *const *X*, double **const *DF*, void(*)(size_t, size_t, double *const, double *const, double **const, double, double) *f*, double *w*, double * *h*)

Compute the differential for the Newton's method.

Parameters

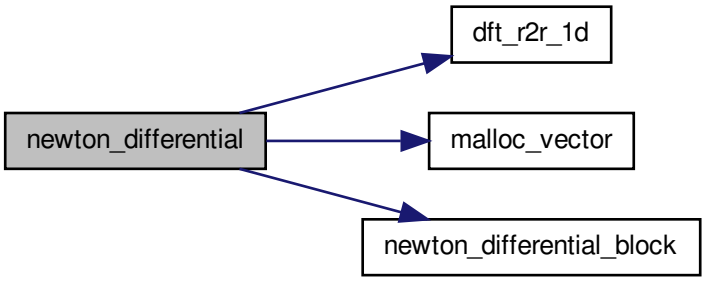
in	*n*	Space's dimension
in	*N*	Mesh size
in,out	*x*	Discrete Fourier Transform for each mesh's element, that is, n(2N+1) components
in	*X*	Fourier coefficients
in	*DF*	Matrix of the linear system
in	*f*	Pointer to function given by the user
in	*w*	Parameter
in	*h*	Auxiliar vector

Returns

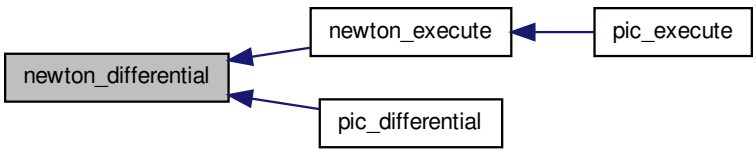
Number of iterations have been performed or NULL if some problem has ocurred

References dft_r2r_1d(), malloc_vector(), and newton_differential_block().

Here is the call graph for this function:



Here is the caller graph for this function:



7.15.3.6 void newton_differential_block (size_t *m*, size_t *n*, double *const *x*, double *const *f*, double **const *df*, double *w*, double *t*)

Compute a block of the differential for the Newton's method.

Parameters

in	*m*	Rows
in	*n*	Columns
in	*x*	
in	*f*	
out	*f*	
out	*df*	Block matrix of the differential for the Newton's method
in	*w*	Parameter
in	*t*	Angle of evaulation

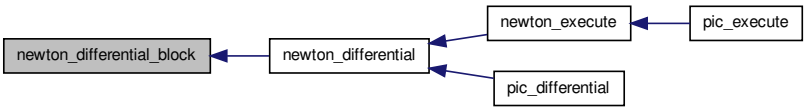
Precondition

The pointers x and f are not NULL

Note

If df is NULL, the differential will not be compute

Here is the caller graph for this function:



7.15.3.7 void* newton_execute (size_t *n*, size_t *N*, double *const *x*, double *const *X*, double **const *DF*, double *w*, double *tol*, void(*)(size_t, size_t, double *const, double *const, double **const, double, double) *f*, void *(*)(size_t, size_t, double **const, double *const, double *const, double) *solve*, short unsigned int *autonomous*)

Implement the execution of a Newton's method.

Parameters

in	*n*	Space's dimension
in	*N*	Mesh size
in,out	*x*	Discrete Fourier Transform for each mesh's element, that is, n(2N+1) components
in	*X*	Fourier coefficients
in	*DF*	Matrix of the linear system
in	*w*	Parameter
in	*tol*	Tolerance
in	*f*	Pointer to function given by the user
in	*solve*	Pointer to solver linear systems of equations

Returns

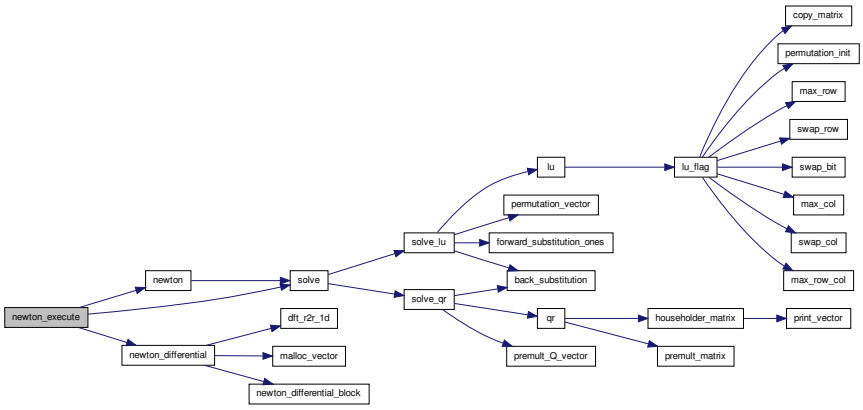
Number of iterations have been performed or NULL if some problem has ocurred

See also

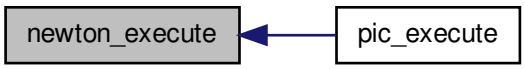
newton_max_iterations

References newton(), newton_differential(), and solve().

Here is the call graph for this function:



Here is the caller graph for this function:



7.15.3.8 unsigned int set_newton_max_iterations (unsigned int *newton_max_iter*)

Set the maximum number of newton iterations.

Parameters

in	*newton_max_iter*	New value

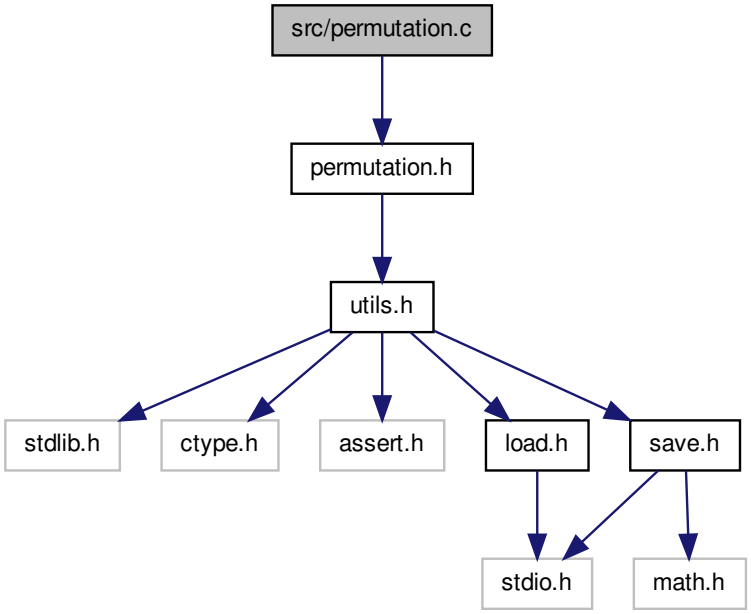
Returns

The old maximum number iterations for Newton's method

7.16 src/permutation.c File Reference

Permutation operation with a vector.

`#include "permutation.h"` Include dependency graph for permutation.c:



Defines

- #define _OPENMP_PERMUTATION_DIMENSION 1e2

  Minimum value for using the openmp library in permutation.

Functions

- void permutation_init (size_t n, size_t ∗const p)

  Initialize a identity vector permutation.
- void permutation_vector (size_t n, size_t ∗const p, double ∗const x, double ∗const px)

  Permute the values of the vector x into px based on the vector permutation p.

7.16.1 Detailed Description

Permutation operation with a vector.

Author

Joan

Date

20/01/2014 (start)

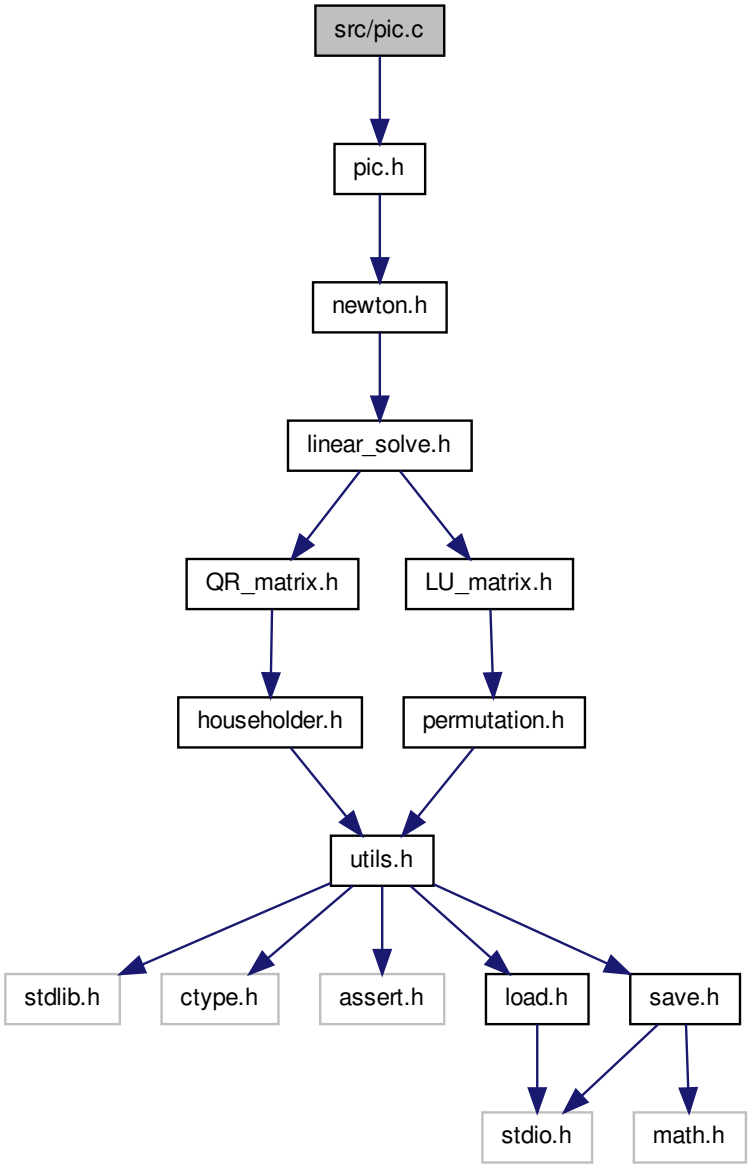
Version

1.0

7.17 src/pic.c File Reference

Implements differents functions for the pic struct.

`#include "pic.h"` Include dependency graph for pic.c:



Defines

- #define _DEFAULT_MAX_ITERATIONS 80

Maximum number of newton iterations.

- #define _MAX_LENGTH_STRING_INPUT 128

Functions

- unsigned int pic_get_max_iterations ()

  Get maximum number of iterations.

- unsigned int pic_set_max_iterations (unsigned int max_iter)

  Set maximum number of iterations.

- size_t pic_get_mesh_size (struct pic *const p)
- size_t pic_get_dimension_size (struct pic *const p)
- int pic_set_function_pointer (struct pic *const p, void(*f)(size_t, size_t, double *const, double *const, double **const, double, double))
- int pic_set_function_solve (struct pic *const p, _SOLVE_PROTOTYPE)
- struct pic * pic_create (size_t n, size_t N, double w, double *const X, _FUNCTION_PROTOTYPE, _SOLVE_PROTOTYPE)

  Function that initializes a struct pic.

- void * _pic_realloc (struct pic *const p, size_t n, size_t N, short unsigned int autonomous)
- int pic_set_mesh_size (struct pic *const p, size_t N)

  Set the mesh size.

- int pic_set_dimension_size (struct pic *const p, size_t n)

  Set the dimension.

- void * _init_pic (struct pic *const p)
- void pic_destroy (struct pic *p)

  Function that frees the memory space pointed to by the differents pointers of struct pic.

- int _get_fourier_coef_index (char **const c)
- double * pic_load_fourier_coef (size_t n, size_t N, FILE *const f)

  Function that load the Fourier coefficients from a given file.

- double * pic_loadas_fourier_coef (size_t n, size_t N, char *const name)

  Function that load the Fourier coefficients from a given file by name.

- int pic_save_fourier_coef (struct pic *const p, FILE *const f)

  Function that saves the Fourier coefficients in a given file.

- int pic_saveas_fourier_coef (struct pic *const p, char *const name)

  Function that saves the Fourier coefficients in a given file by name.

- int pic_save_curve (struct pic *const p, FILE *const f)

  Function that saves the values of the solution curve in a given file.

- int pic_saveas_curve (struct pic *const p, char *const name)

  Function that saves the values of the solution curve in a given file by name.

- int pic_saveas_differential (struct pic *const p, char *const name)

  Function that saves the values of the differential of Newton method in a given file by name.

- int pic_save_differential (struct pic *const p, FILE *const f)

Function that saves the values of the differential of Newton method in a given file.

- int pic_saveas_differential_indexs (struct pic ∗const p, char ∗const name)

  Function that saves the values of the differential of Newton method in a given file by name from its indexs.

- int pic_save_differential_indexs (struct pic ∗const p, FILE ∗const f)

  Function that saves the values of the differential of Newton method in a given file from its indexs.

- void ∗ pic_error (struct pic ∗const p, size_t N, size_t ∗const idx, double ∗const max)

  Function that compute the maximum error for a solution previously computed.

- void pic_x (size_t n, size_t N, double ∗const X, double ∗const x)

  Function that compute the Discrete Fourier Transform.

- size_t _pic_estimate_mesh_size (struct pic ∗const p, double tol, double ∗∗A, double ∗h)
- size_t pic_estimate_mesh_size (struct pic ∗const p, double tol)
- void ∗ pic_execute (struct pic ∗const p, double tol)

  Function that compute the Discrete Fourier Transform.

- double ∗∗ pic_differential (struct pic ∗const p, size_t ∗const m, size_t ∗const n)

  Return a differential matrix for the current pic struct's values.

- size_t pic_get_n (struct pic ∗const p)
- size_t pic_get_N (struct pic ∗const p)
- double ∗ pic_get_x (struct pic ∗const p)
- double ∗ pic_get_X (struct pic ∗const p)
- double ∗∗ pic_get_DF (struct pic ∗const p)
- double pic_get_w (struct pic ∗const p)

7.17.1 Detailed Description

Implements differents functions for the pic struct.

Author

Joan Gimeno

Date

22/01/2014 (start)

Version

1.0

7.17.2 Define Documentation

7.17.2.1 #define _DEFAULT_MAX_ITERATIONS 80

Maximum number of newton iterations.

7.17.2.2 #define _MAX_LENGTH_STRING_INPUT 128

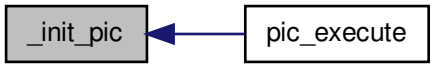
7.17.3 Function Documentation

7.17.3.1 int _get_fourier_coef_index (char ∗∗const c)

7.17.3.2 void∗ _init_pic (struct pic ∗const p)

References pic::autonomous, pic::DF, pic::n, pic::N, pic::x, and pic::X.

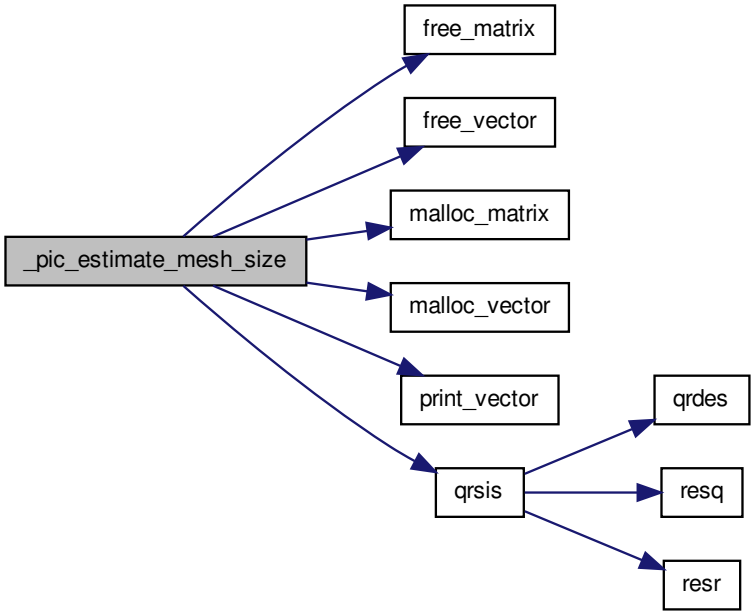
Here is the caller graph for this function:



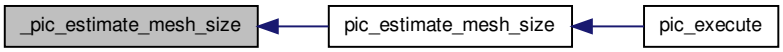
7.17.3.3 size_t _pic_estimate_mesh_size (struct pic ∗const p, double tol, double ∗∗ A, double ∗ h)

References free_matrix(), free_vector(), malloc_matrix(), malloc_vector(), pic::n, pic::N, print_vector(), qrsis(), and pic::X.

Here is the call graph for this function:



Here is the caller graph for this function:



7.17.3.4 void * _pic_realloc (struct pic *const *p*, size_t *n*, size_t *N*, short unsigned int *autonomous*)

Parameters

in	*p*	Pointer to a pic struct
in	*n*	Space's dimension of the function $f: \mathbb{R}^n \to \mathbb{R}^n$
in	*N*	Mesh size of the $\mathbb{T}^1$, 2N+1
in	*autonomous*	

Return values

NULL	If memory space has not been reallocated or the pointer to a pic struct is NULL
1	Otherwise

Precondition

The mesh size and space's dimension are not zeros

Postcondition

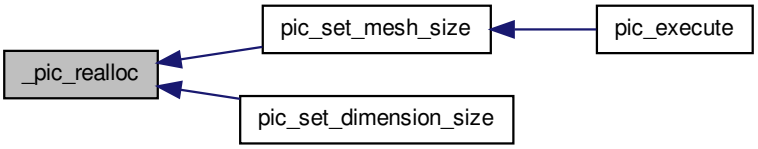
- The p::x,p::X and p::DF have been changed the size of the memory block pointed
- The field pic::N and pic::n have been assigned with the new values given by parameter

Note

- The new positons in pic::X will be assigned with zero values
- The new rows of pic::DF will be allocated

References pic::autonomous, pic::DF, pic::n, pic::N, pic::x, and pic::X.

Here is the caller graph for this function:



7.17.3.5 struct pic* pic_create (size_t *n*, size_t *N*, double *w*, double *const *X*, _FUNCTION_PROTOTYPE , _SOLVE_PROTOTYPE) [read]

Function that initializes a struct pic.

Parameters

in	*n*	Space's dimension of the function $f: \mathbb{R}^n \to \mathbb{R}^n$
in	*N*	Mesh size of the $\mathbb{T}^1$, 2N+1
in	*w*	Real number densely filling the circumference
in	*X*	Seed of the newton method. The size has to be n(2N+1)
in	*f*	Function pointer
in	*solve*	Function pointer for solves a linear system equations

Return values

NULL	If memory space has not been allocated
p	Pointer to struct pic

Precondition

The space's dimension is not zero

Postcondition

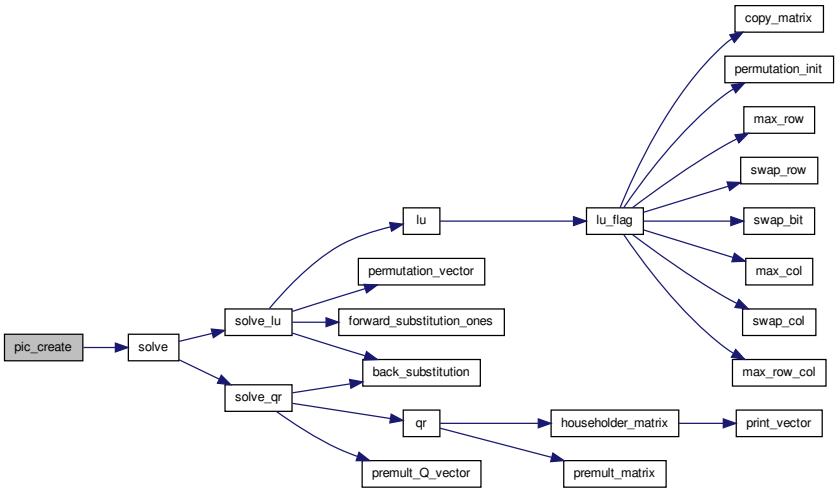
The struct pic has been initialized

Note

If the seed of newton method is NULL, the default seed will be the vector zero

References pic::autonomous, pic::DF, pic::f, pic::n, pic::N, PIC_NAUTONOMOUS, pic::solve, solve(), SOLVE_WITH_LU, pic::w, pic::x, and pic::X.

Here is the call graph for this function:



7.17.3.6 void pic_destroy (struct pic * p)

Function that frees the memory space pointed to by the differents pointers of struct pic.

Parameters

in	*p*	Pointer to periodic invariant curve

Postcondition

The memory space has been freed

References pic::autonomous, pic::DF, pic::n, pic::N, pic::x, and pic::X.

7.17.3.7 double** pic_differential (struct pic *const *p*, size_t *const *m*, size_t *const *n*)

Return a differential matrix for the current pic struct's values.

Parameters

in	*p*	Pointer to a pic struct
out	*m*	Number of differential matrix's rows
out	*n*	Number of differential matrix's columns

Returns

Pointer to initial position of the matrix differential or NULL if some problem occurred

Precondition

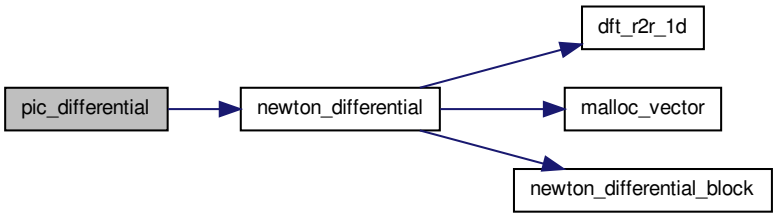
The pointer to a pic struct is not NULL

Note

The pointers for rows and columns can be NULL

References pic::autonomous, pic::DF, pic::f, pic::n, pic::N, newton_differential(), pic::w, pic::x, and pic::X.

Here is the call graph for this function:



7.17.3.8 void * pic_error (struct pic *const *p*, size_t *N*, size_t *const *idx*, double *const *max*)

Function that compute the maximum error for a solution previously computed.

Parameters

in	*p*	Pointer to struct pic
in	*N*	Mesh size that we want to check the error of the solution (e.g. 2∗(p->N))
out	*idx*	Index where the maximum is reached
out	*max*	Maximum value of error

Return values

NULL	if some problem occurred
1	otherwise

Precondition

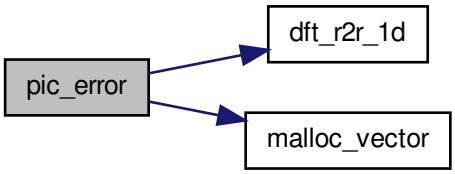
- The struct pointer, the index and the maximum are not NULL
- The fields of the pointer to struct pic are not NULL

Postcondition

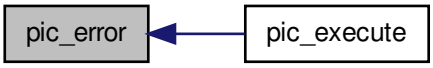
The idx and max have been changed

References dft_r2r_1d(), pic::f, malloc_vector(), pic::n, pic::N, pic::w, and pic::X.

Here is the call graph for this function:



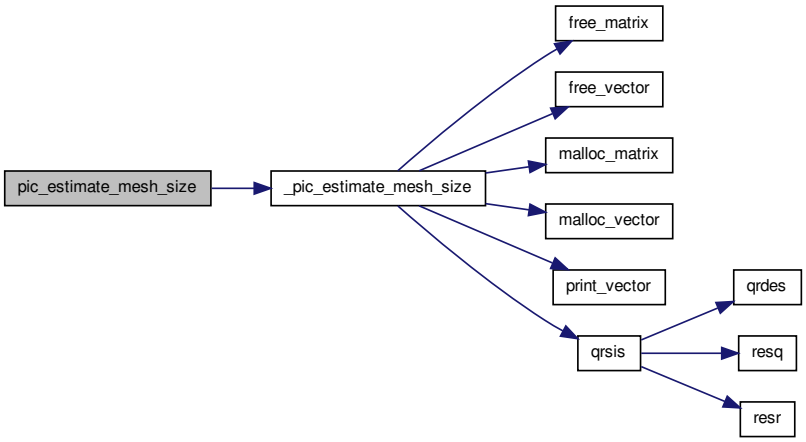
Here is the caller graph for this function:



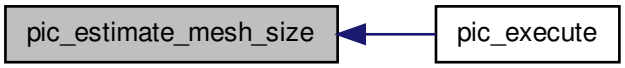
7.17.3.9 size_t pic_estimate_mesh_size (struct pic *const p, double tol)

References _pic_estimate_mesh_size().

Here is the call graph for this function:



Here is the caller graph for this function:



7.17.3.10 void* pic_execute (struct pic *const p, double tol)

Function that compute the Discrete Fourier Transform.

Parameters

in	*p*	Pointer to the pic struct
in	*tol*	Tolerance for the result

Returns

NULL if some problem ocurrer. Otherwise the number of iterations performed until the result (minimum 1)

Note

The mesh size of pic struct can have been changed. Use

See also

pic_get_mesh_size

Precondition

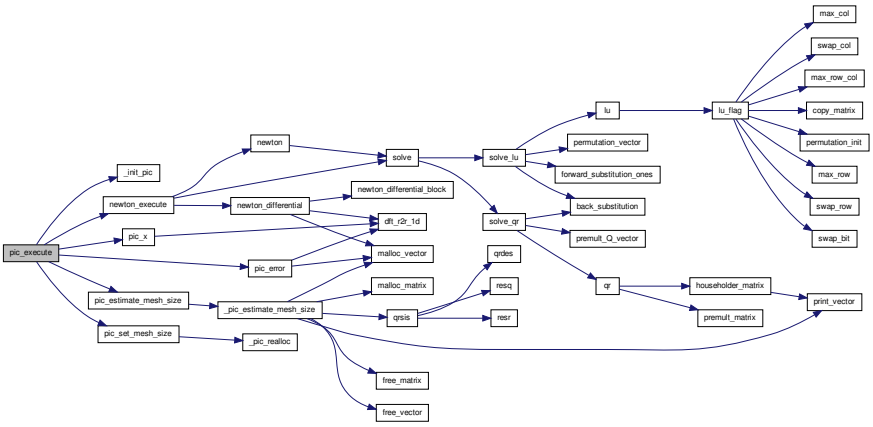
The pointers are not NULL

Postcondition

The vector p->x has the solution

References _init_pic(), pic::autonomous, pic::DF, pic::f, pic::n, pic::N, newton_execute(), pic_error(), pic_estimate_mesh_size(), pic_set_mesh_size(), pic_x(), pic::solve, pic::w, pic::x, and pic::X.

Here is the call graph for this function:



7.17.3.11 double** pic_get_DF (struct pic *const p)

Parameters

in	*p*	Pointer to a pic struct

Returns

The pointer to the differential of the pic struct

Precondition

The pointer is not NULL

References pic::DF.

7.17.3.12 size_t pic_get_dimension_size (struct pic *const p)

Parameters

in	*p*	Pointer to a pic struct

Returns

Space's dimension of the pic struct

Precondition

The pointer is not NULL

References pic::n.

7.17.3.13 unsigned int pic_get_max_iterations ( )

Get maximum number of iterations.

Returns

Maximum iterations that increase the mesh size

7.17.3.14 size_t pic_get_mesh_size (struct pic *const p)

Parameters

in	*p*	Pointer to a pic struct

Returns

Mesh size of the pic struct

Precondition

The pointer is not NULL

References pic::N.

7.17.3.15 size_t pic_get_n (struct pic *const p)

Parameters

in	p	Pointer to a pic struct

Returns

Space's dimension of the pic struct

Precondition

The pointer is not NULL

References pic::n.

7.17.3.16 size_t pic_get_N (struct pic *const p)

Parameters

in	p	Pointer to a pic struct

Returns

The mesh size of the pic struct

Precondition

The pointer is not NULL

References pic::N.

7.17.3.17 double pic_get_w (struct pic *const p)

Parameters

in	p	Pointer to a pic struct

Returns

The real number of the pic struct

Precondition

The pointer is not NULL

References pic::w.

7.17.3.18 double* pic_get_x (struct pic *const p)

Parameters

in	*p*	Pointer to a pic struct

Returns

The value table of the curve solution of the pic struct

Precondition

The pointer is not NULL

References pic::x.

7.17.3.19 double* pic_get_X (struct pic *const *p*)

Parameters

in	*p*	Pointer to a pic struct

Returns

The Fourier's coefficients of the pic struct

Precondition

The pointer is not NULL

References pic::X.

7.17.3.20 double* pic_load_fourier_coef (size_t *n*, size_t *N*, FILE *const *f*)

Function that load the Fourier coefficients from a given file.

Parameters

in	*n*	Dimension of each coefficient
in	*N*	Number of coefficient
in	*f*	Pointer to a file

Returns

n*(2N+1)-dimensional vector with the Fourier coefficients saved

Precondition

The pointer is not NULL

Postcondition

The user has to free the memory for the pointer returned

Note

The structure of the vector is a_0, a_1, b_1, ..., a_N, b_N where each element is a n-dimensional vector

References _MAX_LENGTH_STRING_INPUT, and pic::n.

Here is the caller graph for this function:

7.17.3.21 double* pic_loadas_fourier_coef (size_t *n*, size_t *N*, char *const *name*)

Function that load the Fourier coefficients from a given file by name.

Parameters

in	*n*	Dimension of each coefficient
in	*N*	Number of coefficient
in	*f*	Pointer to a file

Returns

n*(2N+1)-dimensional vector with the Fourier coefficients saved

Postcondition

The user has to free the memory for the pointer returned

Precondition

The pointer is not NULL

Note

The structure of the vector is a_0, a_1, b_1, ..., a_N, b_N where each element is a n-dimensional vector

References pic::f, and pic_load_fourier_coef().

Here is the call graph for this function:

7.17.3.22 int pic_save_curve (struct pic *const p, FILE *const f)

Function that saves the values of the solution curve in a given file.

Parameters

in	*p*	Pointer to struct pic
in	*f*	Output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

References pic::n, pic::N, and pic::x.

Here is the caller graph for this function:

7.17.3.23 int pic_save_differential (struct pic *const p, FILE *const f)

Function that saves the values of the differential of Newton method in a given file.

Parameters

in	*p*	Pointer to struct pic
in	*f*	Output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

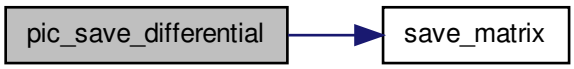
The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

References pic::DF, pic::n, pic::N, and save_matrix().

Here is the call graph for this function:



7.17.3.24 int pic_save_differential_indexs (struct pic *const p, FILE *const f)

Function that saves the values of the differential of Newton method in a given file from its indexs.

Parameters

in	*p*	Pointer to struct pic
in	*f*	Output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

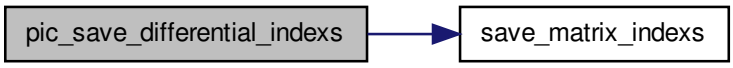
The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

References pic::DF, pic::n, pic::N, and save_matrix_indexs().

Here is the call graph for this function:



7.17.3.25 int pic_save_fourier_coef (struct pic *const p, FILE *const f)

Function that saves the Fourier coefficients in a given file.

Parameters

in	*p*	Pointer to struct pic
in	*f*	Output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The pointers are not NULL

Postcondition

The file has been created or modified with the result

References pic::n, pic::N, and pic::X.

Here is the caller graph for this function:

7.17.3.26 int pic_saveas_curve (struct pic *const p, char *const name)

Function that saves the values of the solution curve in a given file by name.

Parameters

in	*p*	Pointer to struct pic
in	*name*	Name of the output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The struct pointer is not NULL

Postcondition

The file with name has been created or modified with the result

References pic::f, and pic_save_curve().

Here is the call graph for this function:

7.17.3.27 int pic_saveas_differential (struct pic *const p, char *const name)

Function that saves the values of the differential of Newton method in a given file by name.

Parameters

in	*p*	Pointer to struct pic
in	*name*	Name of the output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

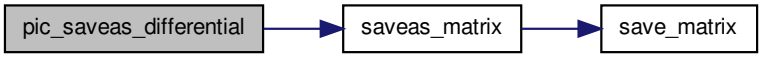
The struct pointer is not NULL

Postcondition

The file with name has been created or modified with the result

References pic::DF, pic::n, pic::N, and saveas_matrix().

Here is the call graph for this function:



7.17.3.28 int pic_saveas_differential_indexs (struct pic ∗const *p*, char ∗const *name*)

Function that saves the values of the differential of Newton method in a given file by name from its indexs.

Parameters

in	*p*	Pointer to struct pic
in	*name*	Name of the output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

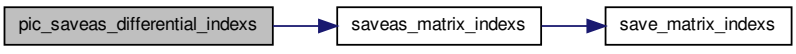
The struct pointer is not NULL

Postcondition

The file with name has been created or modified with the result

References pic::DF, pic::n, pic::N, and saveas_matrix_indexs().

Here is the call graph for this function:



7.17.3.29 int pic_saveas_fourier_coef (struct pic ∗const *p*, char ∗const *name*)

Function that saves the Fourier coefficients in a given file by name.

Parameters

in	*p*	Pointer to struct pic
in	*f*	Output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The pointers are not NULL

Postcondition

The file has been created or modified with the result

References pic::f, and pic_save_fourier_coef().

Here is the call graph for this function:

7.17.3.30 int pic_set_dimension_size (struct pic ∗const *p*, size_t *n*)

Set the dimension.

Parameters

in	*p*	Pointer to a pic struct
in	*N*	New dimension

Returns

0 if some error occurred or the old dimension

Precondition

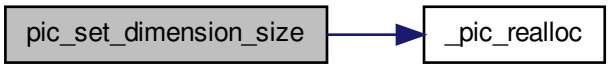
The new mesh size is not 0 and the pointer is not NULL

Postcondition

The pointer memory has been reallocated with the new dimension

References _pic_realloc(), pic::autonomous, pic::n, and pic::N.

Here is the call graph for this function:



7.17.3.31 int pic_set_function_pointer (struct pic *const p, void(*)(size_t, size_t, double *const, double *const, double **const, double, double) f)

References pic::f.

7.17.3.32 int pic_set_function_solve (struct pic *const p, _SOLVE_PROTOTYPE)

Parameters

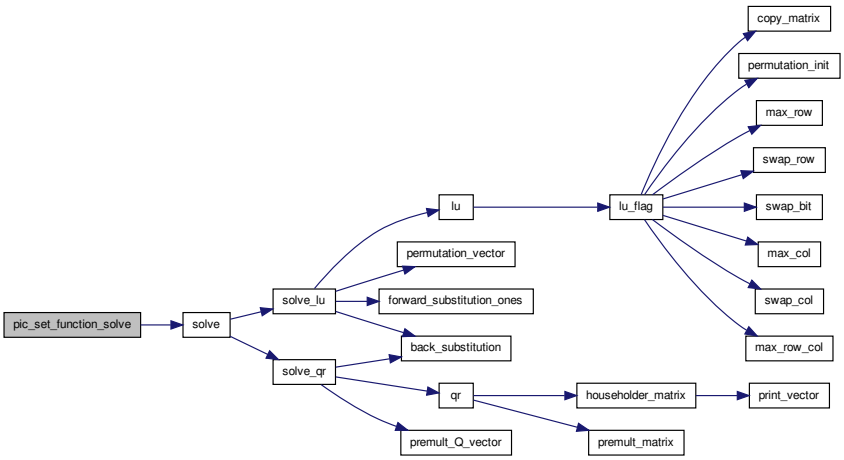
in	*p*	Pointer to a pic struct
in	*f*	Pointer to a function pointer

Return values

0	if some argument is NULL
1	Otherwise

References pic::solve, and solve().

Here is the call graph for this function:



7.17.3.33 unsigned int pic_set_max_iterations (unsigned int *max_iter*)

Set maximum number of iterations.

Parameters

in	*max_iter*	New value

Returns

The old maximum iterations that increase the mesh size

7.17.3.34 int pic_set_mesh_size (struct pic *const *p*, size_t *N*)

Set the mesh size.

Parameters

in	*p*	Pointer to a pic struct
in	*N*	New mesh size

Returns

0 if some error occurred or the old mesh size

Precondition

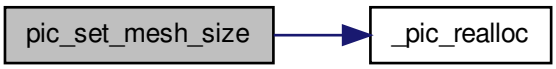
The new mesh size is not 0 and the pointer is not NULL

Postcondition

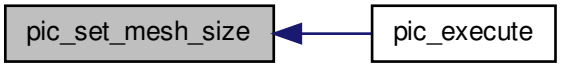
The pointer memory has been reallocated with the new mesh size

References _pic_realloc(), pic::autonomous, pic::n, and pic::N.

Here is the call graph for this function:



Here is the caller graph for this function:



7.17.3.35 void pic_x (size_t *n*, size_t *N*, double *const *X*, double *const *x*)

Function that compute the Discrete Fourier Transform.

Parameters

in	*n*	Space's dimension
in	*N*	Mesh size
in	*X*	Fourier coefficients, that is, n(2N+1) components
in	*x*	Discrete Fourier Transform for each mesh's element, that is, n(2N+1) components

Precondition

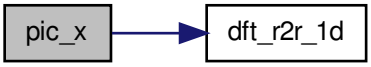
The pointers are not NULL

Postcondition

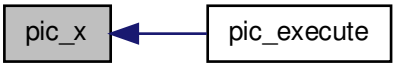
The vector x has been changed

References dft_r2r_1d().

Here is the call graph for this function:

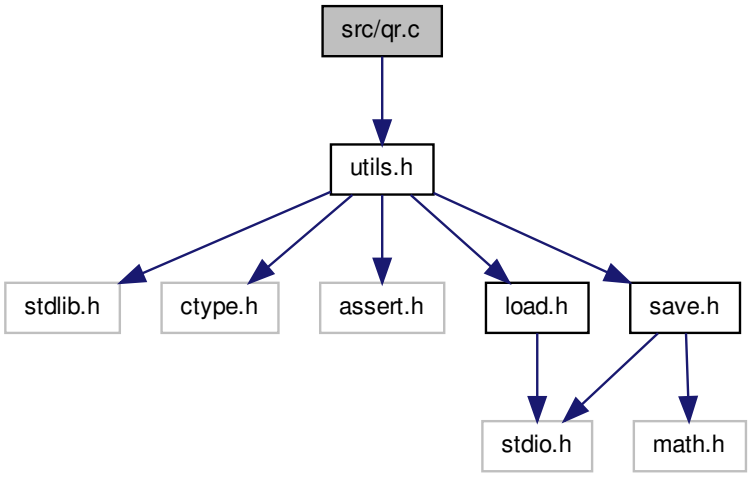


Here is the caller graph for this function:



7.18 src/qr.c File Reference

`#include "utils.h"` Include dependency graph for qr.c:



Functions

- int qrsis (int n, int m, double ∗∗a, double ∗b, double tol)
- int resq (double ∗∗a, int n, double ∗d, double ∗b, double tol)
- int resr (double ∗∗a, int m, double ∗d, double ∗b, double tol)
- void qrdes (double ∗∗a, int n, int m, double ∗d)

7.18.1 Function Documentation

7.18.1.1 void qrdes (double ∗∗ *a,* int *n,* int *m,* double ∗ *d*)

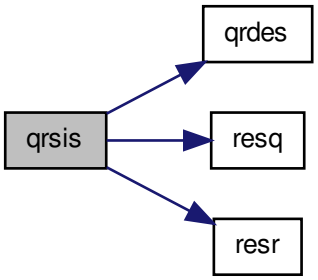
References pic::n, and pic::w.

Here is the caller graph for this function:

7.18.1.2 int qrsis (int *n*, int *m*, double ** *a*, double * *b*, double *tol*)

References pic::n, qrdes(), resq(), and resr().

Here is the call graph for this function:

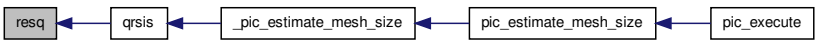


Here is the caller graph for this function:

7.18.1.3 int resq (double ** *a*, int *n*, double * *d*, double * *b*, double *tol*)

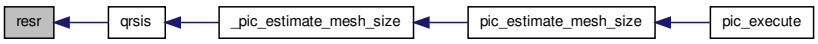
References pic::n.

Here is the caller graph for this function:



7.18.1.4 int resr (double ** a, int m, double * d, double * b, double tol)

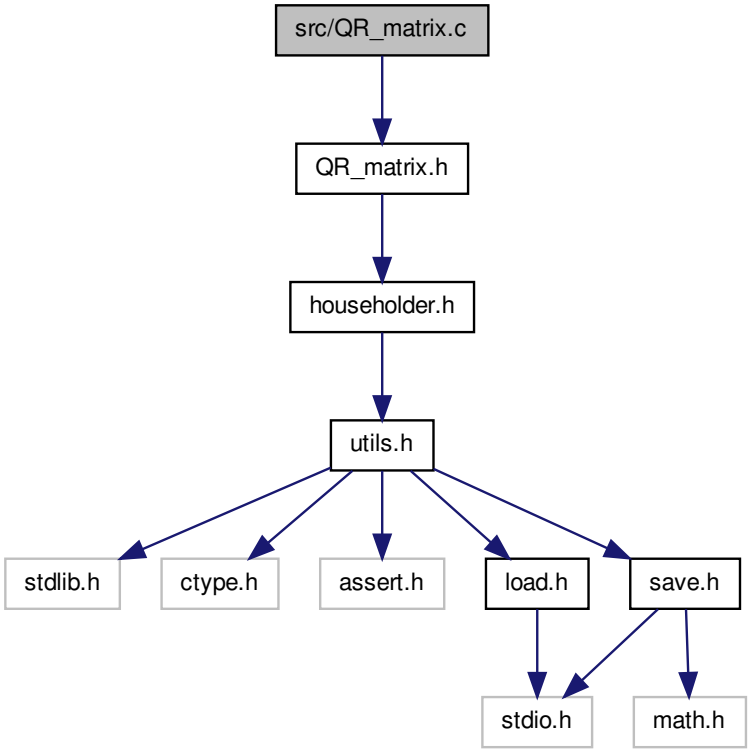
Here is the caller graph for this function:



7.19 src/QR_matrix.c File Reference

Functions for computing QR decomposition of any matrix.

`#include "QR_matrix.h"` Include dependency graph for QR_matrix.c:



Defines

- #define _OPENMP_QR_DIMENSION 1e2

  Minimum value for using the openmp library in QR decomposition.

Functions

- void premult_Q_vector (size_t m, size_t n, double **const QR, double *const b)

  Compute $P_1 \cdots P_n b$ where b and P_i are respectively m-by-1 and m-by-m matrixs.
- void * qr (size_t m, size_t n, double **const A, double **const QR)

  Compute the QR factorization of a matrix m-by-n.

7.19.1 Detailed Description

Functions for computing QR decomposition of any matrix.

Author

Joan

Date

10/12/2013 (start)

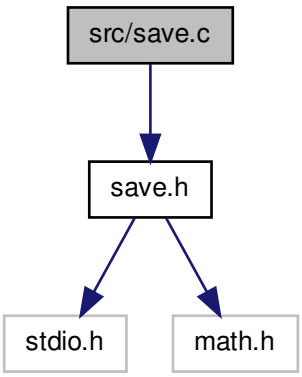
Version

1.0

7.20 src/save.c File Reference

Implements functions that save a vector or a matrix in some file.

`#include "save.h"` Include dependency graph for save.c:



Defines

- #define EPS 1e-10

Functions

- int save_vector (size_t idx, size_t n, double ∗const b, FILE ∗const f)

  Function that saves the values of a vector in a given file.
- int saveas_vector (size_t idx, size_t n, double ∗const b, char ∗const name)

  Function that saves the values of a vector in a given filename.
- int save_vector_indexs (size_t idx, size_t n, double ∗const b, FILE ∗const f)

  Function that saves the values not zero of a vector with its indexs in a given file.
- int saveas_vector_indexs (size_t idx, size_t n, double ∗const b, char ∗const name)

  Function that saves the values not zero of a vector with its indexs in a given filename.
- int save_matrix (size_t idx_row, size_t m, size_t idx_col, size_t n, double ∗∗const A, FILE ∗const f)

Function that saves the values of a matrix in a given file.

- int saveas_matrix (size_t idx_row, size_t m, size_t idx_col, size_t n, double **const A, char *const name)

  Function that saves the values of a matrix in a given filename.
- int save_matrix_indexs (size_t idx_row, size_t m, size_t idx_col, size_t n, double **const A, FILE *const f)

  Function that saves the values not zero of a matrix with its indexs in a given file.
- int saveas_matrix_indexs (size_t idx_row, size_t m, size_t idx_col, size_t n, double **const A, char *const name)

  Function that saves the values not zero of a matrix with its indexs in a given filename.

7.20.1 Detailed Description

Implements functions that save a vector or a matrix in some file.

Author

Joan Gimeno

Date

22/05/2014 (start)

Version

1.0

7.20.2 Define Documentation

7.20.2.1 #define EPS 1e-10

7.20.3 Function Documentation

7.20.3.1 int save_matrix (size_t *idx_row*, size_t *m*, size_t *idx_col*, size_t *n*, double **const *A*, FILE *const *f*)

Function that saves the values of a matrix in a given file.

Parameters

in	*idx_row*	Start index row
in	*m*	Rows
in	*idx_col*	Start index column
in	*n*	Columns
in	*A*	Pointer to a matrix
in	*f*	Output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

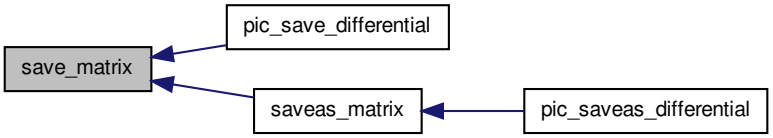
The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

References pic::n.

Here is the caller graph for this function:



7.20.3.2 int save_matrix_indexs (size_t *idx_row*, size_t *m*, size_t *idx_col*, size_t *n*, double **const *A*, FILE *const *f*)

Function that saves the values not zero of a matrix with its indexs in a given file.

Parameters

in	*idx_row*	Start index row
in	*m*	Rows
in	*idx_col*	Start index column
in	*n*	Columns
in	*A*	Pointer to a matrix
in	*f*	Output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

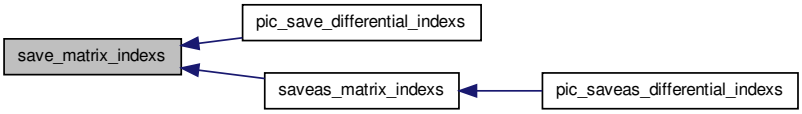
The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

References EPS, and pic::n.

Here is the caller graph for this function:



7.20.3.3 int save_vector (size_t *idx*, size_t *n*, double *const *b*, FILE *const *f*)

Function that saves the values of a vector in a given file.

Parameters

in	*idx*	Start index
in	*n*	Dimension
in	*b*	Pointer to a vector
in	*f*	Output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

References pic::n.

Here is the caller graph for this function:

7.20.3.4 int save_vector_indexs (size_t *idx*, size_t *n*, double *const *b*, FILE *const *f*)

Function that saves the values not zero of a vector with its indexs in a given file.

Parameters

in	*idx*	Start index
in	*n*	Dimension
in	*b*	Pointer to a vector
in	*f*	Output file

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

References EPS, and pic::n.

Here is the caller graph for this function:

7.20.3.5 int saveas_matrix (size_t *idx_row*, size_t *m*, size_t *idx_col*, size_t *n*, double **const *A*, char *const *name*)

Function that saves the values of a matrix in a given filename.

Parameters

in	*idx_row*	Start index row
in	*m*	Rows
in	*idx_col*	Start index column
in	*n*	Columns
in	*A*	Pointer to a matrix
in	*name*	Filename

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

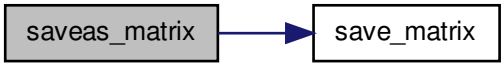
The pointers are not NULL

Postcondition

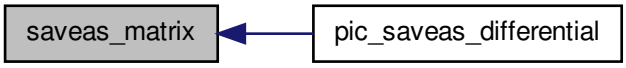
The file with name has been created or modified with the result

References pic::f, and save_matrix().

Here is the call graph for this function:



Here is the caller graph for this function:



7.20.3.6 int saveas_matrix_indexs (size_t *idx_row*, size_t *m*, size_t *idx_col*, size_t *n*, double **const *A*, char *const *name*)

Function that saves the values not zero of a matrix with its indexs in a given filename.

Parameters

in	*idx_row*	Start index row
in	*m*	Rows
in	*idx_col*	Start index column
in	*n*	Columns
in	*A*	Pointer to a matrix
in	*name*	Filename

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The pointers are not NULL

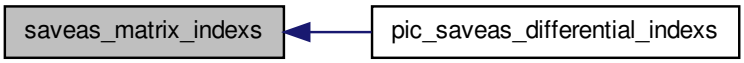
Postcondition

The file with name has been created or modified with the result

References pic::f, and save_matrix_indexs().

Here is the call graph for this function:

Here is the caller graph for this function:



7.20.3.7 int saveas_vector (size_t *idx*, size_t *n*, double *const *b*, char *const *name*)

Function that saves the values of a vector in a given filename.

Parameters

in	*idx*	Start index
in	*n*	Dimension
in	*b*	Pointer to a vector
in	*name*	Filename

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

References pic::f, and save_vector().

Here is the call graph for this function:

7.20.3.8 int saveas_vector_indexs (size_t *idx*, size_t *n*, double *const *b*, char *const *name*)

Function that saves the values not zero of a vector with its indexs in a given filename.

Parameters

in	*idx*	Start index
in	*n*	Dimension
in	*b*	Pointer to a vector
in	*name*	Filename

Return values

0	If some error has occurred in some moment of the process
1	Otherwise

Precondition

The pointers are not NULL

Postcondition

The file with name has been created or modified with the result

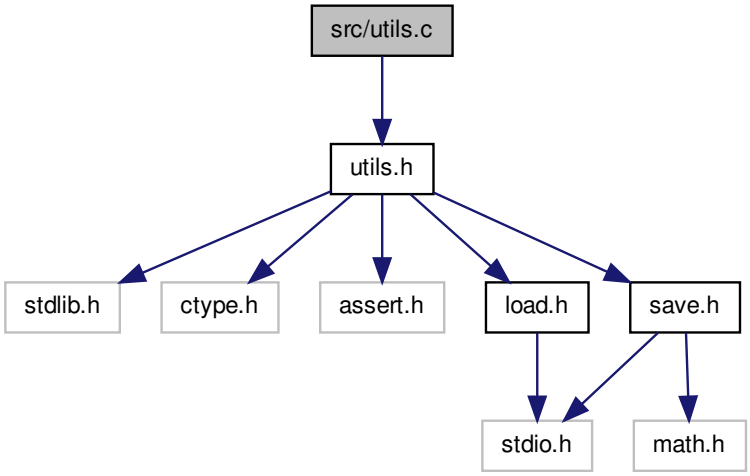
References pic::f, and save_vector_indexs().

Here is the call graph for this function:

7.21 src/utils.c File Reference

Set of different function for linear systems.

`#include "utils.h"` Include dependency graph for utils.c:



Functions

- double euclidian_norm (size_t n, double ∗const x)

  Derive the euclidian norm of a vector, that is, $||x||_2 = \left(\sum_{i=0}^{n-1} x_i^2\right)^{\frac{1}{2}}$.

- double inner_product (size_t n, double ∗const x)

Derive the ordinary inner product of a vector, that is, $\langle x,x\rangle = \sum_{i=0}^{n-1} x_i^2$.

- void copy_vector (size_t n, double *const x, double *const y)

  Copy the vector x in y.
- void copy_matrix (size_t m, size_t n, double **const A, double **const B)

  Copy the matrix A in B.
- void print_matrix (size_t m, size_t n, double **const A, char *const name)

  Display in stdout a matrix m-by-n.
- void print_vector (size_t idx, size_t n, double *const v, char *const name)

  Display in stdout a n-dimensional vector starting by an index.
- double * malloc_vector (size_t n)

  Malloc a n-dimensional vector.
- void free_vector (double *p)

  Free a vector.
- double ** malloc_matrix (size_t m, size_t n)

  Malloc a m-by-n matrix.
- void free_matrix (size_t m, double **p)

  Free a matrix with m rows.
- void allocated_vector (void *p, size_t n)
- void free_vector2 (void *p, size_t n)
- void allocated_matrix (void **p, size_t m, size_t n)
- void free_matrix2 (void **p, size_t m, size_t n)

7.21.1 Detailed Description

Set of different function for linear systems.

Author

Joan

Date

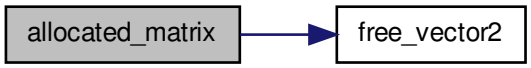
04/12/2013 (start)

7.21.2 Function Documentation

7.21.2.1 void allocated_matrix (void ** *p*, size_t *m*, size_t *n*)

References free_vector2().

Here is the call graph for this function:



7.21.2.2 void allocated_vector (void * p, size_t n)

7.21.2.3 void copy_matrix (size_t m, size_t n, double **const A, double **const B)

Copy the matrix A in B.

Parameters

in	*m*	Rows
in	*m*	Columns
in	*A*	Source matrix
out	*B*	Destination matrix

Precondition

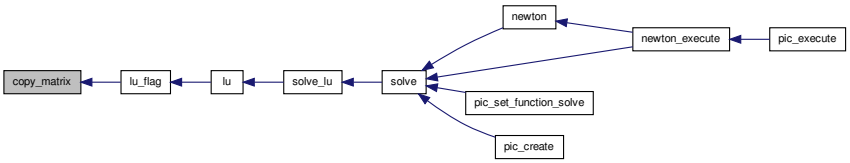
The pointers are not NULL

Warning

The pointers should not be the same

References pic::n.

Here is the caller graph for this function:



7.21.2.4 void copy_vector (size_t *n*, double *const *x*, double *const *y*)

Copy the vector x in y.

Parameters

in	*n*	Dimension
in	*x*	Source vector
out	*y*	Destination vector

Precondition

The pointers are not NULL

Warning

The pointers should not be the same

References pic::n.

7.21.2.5 double euclidian_norm (size_t *n*, double *const *x*)

Derive the euclidian norm of a vector, that is, $||x||_2 = \left(\sum_{i=0}^{n-1} x_i^2\right)^{\frac{1}{2}}$.

Parameters

in	*n*	Dimension of vector
in	*x*	Input vector

Returns

Euclidian norm

Postcondition

Thre result is greater than or equal to zero

Note

If the pointer is NULL, the result is zero

References pic::n.

Here is the caller graph for this function:

7.21.2.6 void free_matrix (size_t *m*, double ** *p*)

Free a matrix with m rows.

Parameters

in	*m*	Rows

Here is the caller graph for this function:

7.21.2.7 void free_matrix2 (void ** *p*, size_t *m*, size_t *n*)

References free_vector2().

Here is the call graph for this function:

7.21.2.8 void free_vector (double * p)

Free a vector.

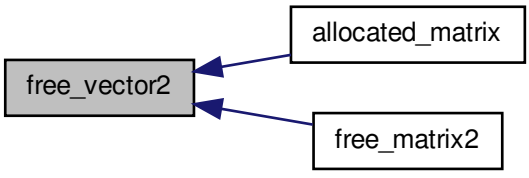
Parameters

in	p	Pointer

Here is the caller graph for this function:

7.21.2.9 void free_vector2 (void * p, size_t n)

Here is the caller graph for this function:



7.21.2.10 double inner_product (size_t *n*, double *const *x*)

Derive the ordinary inner product of a vector, that is, $\langle x,x\rangle = \sum_{i=0}^{n-1} x_i^2$.

Parameters

in	*n*	Dimension of vector
in	*x*	Input vector

Returns

Inner product

Postcondition

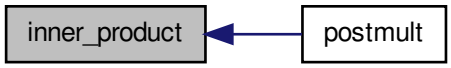
The result is greater than or equal to zero

Note

If the pointer is NULL, the result is zero

References pic::n.

Here is the caller graph for this function:



7.21.2.11 double** malloc_matrix (size_t *m*, size_t *n*)

Malloc a m-by-n matrix.

Parameters

in	*m*	Rows
in	*n*	Columns

Returns

Pointer to a matrix

Note

The return value could be NULL

Here is the caller graph for this function:

7.21.2.12 double* malloc_vector (size_t *n*)

Malloc a n-dimensional vector.

Parameters

in	*n*	End index

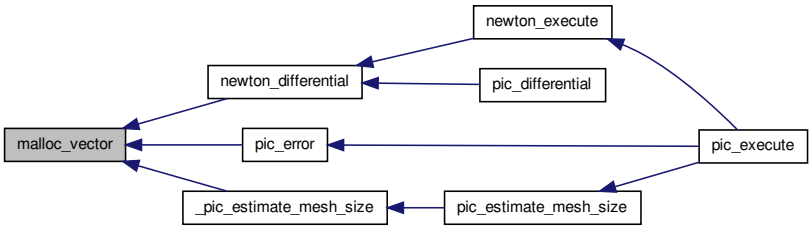
Returns

Pointer to a malloc vector

Note

The return value could be NULL

Here is the caller graph for this function:



7.21.2.13 void print_matrix (size_t *m*, size_t *n*, double ∗∗const *A*, char ∗const *name*)

Display in stdout a matrix m-by-n.

Parameters

in	*m*	Rows
in	*n*	Columns
in	*A*	Matrix m-by-n
in	*name*	Name of the matrix in stdout

Precondition

The dimension are not zeros and the pointers are not NULL

References pic::n.

7.21.2.14 void print_vector (size_t *idx*, size_t *n*, double ∗const *v*, char ∗const *name*)

Display in stdout a n-dimensional vector starting by an index.

Parameters

in	*idx*	Start index
in	*n*	End index
in	*v*	Vector
in	*name*	Name of the vector in stdout

Precondition

The idx is less than n and the pointers are not NULL

References pic::n.

Here is the caller graph for this function: