EasyPCG

1.0

Generated by Doxygen 1.8.8

Fri Oct 7 2016 13:25:28

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 CSC Struct Reference

```
#include <linalg_stuff.h>
```

Public Attributes

- int cols
- int * col_start
- int * indices
- double * elements

3.1.1 Detailed Description

Matrix type in Compressesed Sparse Column (CSC) format

see https://en.wikipedia.org/wiki/Sparse_matrix for details of the data format

3.1.2 Member Data Documentation

3.1.2.1 int * CSC::col_start

indices that point to the beginning of each row in indices

3.1.2.2 int CSC::cols

number of columns

3.1.2.3 double * CSC::elements

list of all values

3.1.2.4 int * CSC::indices

list of all indices

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

- easyPCG.c
- linalg_stuff.h

3.2 CSR Struct Reference

```
#include <linalg_stuff.h>
```

Public Attributes

- int rows
- int * row_start
- int * indices
- double * elements

3.2.1 Detailed Description

Matrix type in Compressesed Sparse Row (CSR) format

see `https://en.wikipedia.org/wiki/Sparse_matrix` for details of the data format

3.2.2 Member Data Documentation

3.2.2.1 double * CSR::elements

list of all values

3.2.2.2 int * CSR::indices

list of all indices

3.2.2.3 int * CSR::row_start

indices that point to the beginning of each row in indices

3.2.2.4 int CSR::rows

number of rows

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

- easyPCG.c
- linalg_stuff.h

Chapter 4

File Documentation

4.1 easyPCG.c File Reference

```
#include "easyPCG.h"
```

Classes

- struct CSR
- struct CSC

Typedefs

- typedef struct CSR CSR_matrix
- typedef struct CSC CSC_matrix

Functions

- void CSR_incompleteCholeski (CSR_matrix ∗A, CSR_matrix ∗L)

  Computes the incomplete Cholesky decomposition of a matrix.
- void dummy_precon (double ∗R, double ∗X, int n)

  Dummy preconditioner.
- void CSR_Jacobi_precon (double ∗R, double ∗X, int n)

  Simple Jacobi preconditioner.
- void CSR_GausSeidel_precon (double ∗R, double ∗X, int n)

  Gauss-Seidel preconditioner.
- void CSC_IC_precon (double ∗R, double ∗X, int n)

  Incomplete-Cholesky preconditioner.
- void PCG_set_default_matrix (void ∗A)

  Sets the system matrix A, for which A.X=B will be solved.
- void PCG_set_preconditioner_mode (void ∗System_matrix, preconditionerID mode)

  Sets the preconditioner.
- void PCG_clean ()

  Cleans internally allocated numerical objects used by the PCG algorithm.
- void PCG_set_precon (void(∗Precon)(double ∗R, double ∗X, int n))

  Sets a user-defined preconditioner.
- void PCG_set_mult (void(∗Mult)(double ∗R, double ∗X, int n))

Sets a user-defined multiplicatione routine.

- int PCG_solve (double *X, double *B, int n, double tol, int max_iter)

    Solves a linear equation with the PCG method.

4.1.1 Typedef Documentation

4.1.1.1 typedef struct CSC CSC_matrix

Matrix type in Compressesed Sparse Column (CSC) format

see `https://en.wikipedia.org/wiki/Sparse_matrix` for details of the data format

4.1.1.2 typedef struct CSR CSR_matrix

Matrix type in Compressesed Sparse Row (CSR) format

see `https://en.wikipedia.org/wiki/Sparse_matrix` for details of the data format

4.1.2 Function Documentation

4.1.2.1 void CSC_IC_precon (double * *R*, double * *X*, int *n*)

Incomplete-Cholesky preconditioner.

Parameters

*double**	R: pointer to memory where the result ist stored
*double**	X: pointer to vector in which the preconditioner is applied
int	n: dimension of vectors

This preconditioner assumes M=L.L^{T}, where L is the incomplete Cholesky factor of the system matrix. The preconditioning is basically forward/backward solving of a triangular system. The factor L and its transpose L^{T} must be stored in the global variables IL and ILT, respectively. The memory where R points to must be allocated before passing.

4.1.2.2 void CSR_GausSeidel_precon (double * *R*, double * *X*, int *n*)

Gauss-Seidel preconditioner.

Parameters

*double**	R: pointer to memory where the result ist stored
*double**	X: pointer to vector in which the preconditioner is applied
int	n: dimension of vectors

maybe useless

4.1.2.3 void CSR_incompleteCholeski (CSR_matrix * *A*, CSR_matrix * *L*)

Computes the incomplete Cholesky decomposition of a matrix.

Parameters

*CSR_matrix**	A: pointer to the matrix to be factorized

CSR_matrix∗	L: pointer where the resulting factor L.L^T=A is stored. Allocate in uninitialized CSR struct and pass the pointer.

This method computes the incomplete Cholesky decomposition A=L.L^T. The matrix to be factorized must be symmetric and positive definite to ensure existence of the factorization. For negative definete symmetric matrices multiply by -1 to obtain a positive definite matrix before decomposition.

4.1.2.4 void CSR_Jacobi_precon (double ∗ *R*, double ∗ *X*, int *n*)

Simple Jacobi preconditioner.

Parameters

double∗	R: pointer to memory where the result ist stored
double∗	X: pointer to vector in which the preconditioner is applied
int	n: dimension of vectors

This is the simplest preconditioner M.R=X. The preconditioning operation X=M^{-1}.R is performed by setting M=diagonal(A), where A is the system matrix. The system matrix must be stored in the global variable "default_A". The memory where R points to must be allocated before passing.

4.1.2.5 void dummy_precon (double ∗ *R*, double ∗ *X*, int *n*)

Dummy preconditioner.

Parameters

double∗	R: pointer to memory where the result ist stored
double∗	X: pointer to vector in which the preconditioner is applied
int	n: dimension of vectors

This is a dummy preconditioner, i.e. no preconditioning is applied. The memory where R points to must be allocated before passing.

4.1.2.6 void PCG_clean ( )

Cleans internally allocated numerical objects used by the PCG algorithm.

4.1.2.7 void PCG_set_default_matrix (void ∗ *A*)

Sets the system matrix A, for which A.X=B will be solved.

Parameters

void∗	A: pointer to the system matrix A

The system matrix is passed as a void pointer but make sure that the struct it points to has the same structure as CSR_matrix in external libraries.

4.1.2.8 void PCG_set_mult (void(∗)(double ∗R, double ∗X, int n) *Mult*)

Sets a user-defined multiplicatione routine.

Parameters

void	(*Mult)(double* R,double∗ X,int n): pointer to system matrix multiplication method

There is a default matrix-multiplication routine implemented in the code but the user might want to use its own, wich can be set with this function. If the preconditioner mode is also "USER_DEFINED", then no explicit system matrix must be set.

In the multiplicatione method the result is stored at pointer R and X represents the pointer to the vector that is multiplied by the system matrix. n denotes the dimension of the system.

4.1.2.9 void PCG_set_precon (void(∗)(double ∗R, double ∗X, int n) *Precon*)

Sets a user-defined preconditioner.

Parameters

void	(*Precon)(double* R,double∗ X,int n): pointer to the preconditioner method.

This function must be called when the preconditioner mode "USER_DEFINED" is set. The Argument Precon must perform a preconditioner operation on X. The result M^{-1}.X must stored at pointer R. n denotes the dimension of the vectors.

4.1.2.10 void PCG_set_preconditioner_mode (void ∗ *System_matrix,* preconditionerID *mode*)

Sets the preconditioner.

Parameters

void∗	System_matrix: pointer to the system matrix A, for which A.X=B will be solved
preconditionerID	mode: preconditioner-mode ID

This method sets the preconditioner that will be used for the PCG algorithm. Since for some preconditioners pre-computations must be performed, the system matrix can be passed, too. It is passed as a void pointer but make sure that the struct it points to, has the same structure as CSR_matrix in external libraries.

If System_matrix is set to NULL, the default matrix does not change. This can make sense, if one wants to change the preconditioner method but not the system to be solved.

If the System_matrix is not NULL the global variable default_A is set and it is not necessary to call "PCG_set_default_matrix" any more.

There are currently 4 preconditioner modes avialable:

1. mode==PCG_NONE: no preconditioner is used
2. PCG_USER_DEFINED: user defined preconditioner If this mode is used, the method "PCG_set_precon" must be called to set the preconditioner.
3. PCG_JACOBI: Jacobi preconditioner
4. PCG_ICHOL: Incomplete Cholesky preconditioner

4.1.2.11 int PCG_solve (double ∗ *X,* double ∗ *B,* int *n,* double *tol,* int *max_iter*)

Solves a linear equation with the PCG method.

Parameters

double∗	X: pointer to the vector where the solution of A.X=B is stored.
double∗	B: pointer to the vector representing the right-hand side if the equation A.X=B
int	n: dimension of the system
double	tol: tolerance for the residuum reductioen r/r0
int	max_iter: maximum number of iterations until termination

Returns

int: actual number of iterations for the PCG to reduce the residuum ratio to r/r0<tol

This is the solution method for the linear system A.X=B with help of the Preconditioned Conjugated Gradient (PCG) method. Before using this routine, one must set the preconditioner and the system matrix. Both can be done by calling "PCG_set_preconditioner_mode". Alternatively, one can specify the matrix-mutiplcation and preconditioning routine by hand if favored. In the latter case, one does not have to pass the system matrix explicitly.

When solving an equation, an initial guess X0 must be pointed to by X. After the algorithm has converged the solution is stored at X, and the initial guess is overwritten. Convergence is assumed when the ratio of the current residuum r/r0 = ||B-A.X||/||B-A.X0||<tol. If this condition is not fulfilled within max_iter iterations no convergence is assumed and the algorithm is terminated.

Note that the PCG algorithm works only for symmetric and positive-definite matrices. The "ICHOL" preconditioner might complain for negative-definite matrices. In this case (not for indefinite matrices), simply multiply the matrix by -1 (i.e. solve (-A).x=-B).

4.2 easyPCG.h File Reference

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
```

Typedefs

- typedef enum PREC_ID preconditionerID

Enumerations

- enum PREC_ID { PCG_NONE, PCG_USER_DEFINED, PCG_JACOBI, PCG_ICHOL }

Functions

- int PCG_solve (double ∗X, double ∗B, int n, double tol, int max_iter)

  Solves a linear equation with the PCG method.
- void PCG_set_preconditioner_mode (void ∗System_matrix, preconditionerID mode)

  Sets the preconditioner.
- void PCG_set_default_matrix (void ∗A)

  Sets the system matrix A, for which A.X=B will be solved.
- void PCG_set_precon (void(∗Precon)(double ∗R, double ∗X, int n))

  Sets a user-defined preconditioner.
- void PCG_set_mult (void(∗Mult)(double ∗R, double ∗X, int n))

  Sets a user-defined multiplicatione routine.
- void PCG_clean ()

  Cleans internally allocated numerical objects used by the PCG algorithm.

4.2.1 Typedef Documentation

4.2.1.1 typedef enum PREC_ID preconditionerID

ID for preconditioner used in the PCG algorithm

4.2.2 Enumeration Type Documentation

4.2.2.1 enum PREC_ID

ID for preconditioner used in the PCG algorithm

Enumerator

- ***PCG_NONE*** Preconditioner mode ID: no preconditioner used
- ***PCG_USER_DEFINED*** Preconditioner mode ID: user-defined preconditioner used
- ***PCG_JACOBI*** Preconditioner mode ID: Jacobi preconditioner used
- ***PCG_ICHOL*** Preconditioner mode ID: incomplete Cholesky preconditioner used

4.2.3 Function Documentation

4.2.3.1 void PCG_clean ( )

Cleans internally allocated numerical objects used by the PCG algorithm.

4.2.3.2 void PCG_set_default_matrix (void * *A*)

Sets the system matrix A, for which A.X=B will be solved.

Parameters

*void**	A: pointer to the system matrix A

The system matrix is passed as a void pointer but make sure that the struct it points to has the same structure as CSR_matrix in external libraries.

4.2.3.3 void PCG_set_mult (void(*)(double *R, double *X, int n) *Mult*)

Sets a user-defined multiplicatione routine.

Parameters

void	(*Mult)(double* R,double* X,int n): pointer to system matrix multiplication method

There is a default matrix-multiplication routine implemented in the code but the user might want to use its own, wich can be set with this function. If the preconditioner mode is also "USER_DEFINED", then no explicit system matrix must be set.

In the multiplicatione method the result is stored at pointer R and X represents the pointer to the vector that is multiplied by the system matrix. n denotes the dimension of the system.

4.2.3.4 void PCG_set_precon (void(*)(double *R, double *X, int n) *Precon*)

Sets a user-defined preconditioner.

Parameters

void	(*Precon)(double* R,double* X,int n): pointer to the preconditioner method.

This function must be called when the preconditioner mode "USER_DEFINED" is set. The Argument Precon must perform a preconditioner operation on X. The result M^{-1}.X must stored at pointer R. n denotes the dimension of the vectors.

4.2.3.5 void PCG_set_preconditioner_mode (void * *System_matrix,* preconditionerID *mode*)

Sets the preconditioner.

Parameters

void∗	System_matrix: pointer to the system matrix A, for which A.X=B will be solved
preconditionerID	mode: preconditioner-mode ID

This method sets the preconditioner that will be used for the PCG algorithm. Since for some preconditioners pre-computations must be performed, the system matrix can be passed, too. It is passed as a void pointer but make sure that the struct it points to, has the same structure as CSR_matrix in external libraries.

If System_matrix is set to NULL, the default matrix does not change. This can make sense, if one wants to change the preconditioner method but not the system to be solved.

If the System_matrix is not NULL the global variable default_A is set and it is not necessary to call "PCG_set_↩ default_matrix" any more.

There are currently 4 preconditioner modes avialable:

1. mode==PCG_NONE: no preconditioner is used
2. PCG_USER_DEFINED: user defined preconditioner If this mode is used, the method "PCG_set_precon" must be called to set the preconditioner.
3. PCG_JACOBI: Jacobi preconditioner
4. PCG_ICHOL: Incomplete Cholesky preconditioner

4.2.3.6 int PCG_solve (double ∗ *X*, double ∗ *B*, int *n*, double *tol*, int *max_iter*)

Solves a linear equation with the PCG method.

Parameters

double∗	X: pointer to the vector where the solution of A.X=B is stored.
double∗	B: pointer to the vector representing the right-hand side if the equation A.X=B
int	n: dimension of the system
double	tol: tolerance for the residuum reductioen r/r0
int	max_iter: maximum number of iterations until termination

Returns

int: actual number of iterations for the PCG to reduce the residuum ratio to r/r0<tol

This is the solution method for the linear system A.X=B with help of the Preconditioned Conjugated Gradient (PCG) method. Before using this routine, one must set the preconditioner and the system matrix. Both can be done by calling "PCG_set_preconditioner_mode". Alternatively, one can specify the matrix-mutiplcation and preconditioning routine by hand if favored. In the latter case, one does not have to pass the system matrix explicitly.

When solving an equation, an initial guess X0 must be pointed to by X. After the algorithm has converged the solution is stored at X, and the initial guess is overwritten. Convergence is assumed when the ratio of the current residuum r/r0 = ||B-A.X||/||B-A.X0||<tol. If this condition is not fulfilled within max_iter iterations no convergence is assumed and the algorithm is terminated.

Note that the PCG algorithm works only for symmetric and positive-definite matrices. The "ICHOL" preconditioner might complain for negative-definite matrices. In this case (not for indefinite matrices), simply multiply the matrix by -1 (i.e. solve (-A).x=-B).

4.3 linalg_stuff.c File Reference

```
#include "linalg_stuff.h"
```

Functions

- void print_vector (double ∗V, int n, const char ∗Name)

    Prints a vector to a text file.
- void print_list (int ∗V, int n, const char ∗Name)

    Prints an integer list to a text file.
- void print_CSR (CSR_matrix ∗A, const char ∗Name)

    Prints a CSR matrix to a text file.
- void print_CSC (CSC_matrix ∗A, const char ∗Name)

    Prints a CSC matrix to a text file.
- void print_CSR_matrix_market (CSR_matrix ∗A, const char ∗Name)

    Prints a CSR matrix to the Matrix Market format.
- void print_CSC_matrix_market (CSC_matrix ∗A, const char ∗Name)

    Prints a CSC matrix to the Matrix Market format.
- double ∗ zero_vector (int n)

    Allocates a list of doubles initialized by zero.
- void clear_vector (double ∗V, int n)

    Clears a vector by setting all elements to zero.
- void free_CSC_matrix (CSC_matrix ∗∗const A)

    Frees all memory of a CSC-matrix struct.
- void free_CSR_matrix (CSR_matrix ∗∗const A)

    Frees all memory of a CSR-matrix struct.
- void CSR_scale_matrix (CSR_matrix ∗A, double factor)

    Scales a CSR matrix by a factor.
- CSR_matrix ∗ matrix_product_to_CSR (CSR_matrix ∗A, CSC_matrix ∗B)

    Computes the matrix product.
- int ∗ zero_int_list (int n)

    Allocates a list of integers initialized by zero.

4.3.1 Function Documentation

4.3.1.1 void clear_vector (double ∗ *V*, int *n*)

Clears a vector by setting all elements to zero.

Parameters

double∗	V: Pointer to vector to be cleared)
int	n: dimension of vector

4.3.1.2 void CSR_scale_matrix (CSR_matrix ∗ *A*, double *factor*)

Scales a CSR matrix by a factor.

Parameters

CSR_matrix∗	A: pointer the the CSR-matrix to be scaled
double	factor: scaling factor

4.3.1.3 void free_CSC_matrix (CSC_matrix ∗∗const *A*)

Frees all memory of a CSC-matrix struct.

Parameters

CSC_matrix∗∗	A: pointer to the pointer where the matrix is stored

4.3.1.4 void free_CSR_matrix (CSR_matrix ∗∗const *A*)

Frees all memory of a CSR-matrix struct.

Parameters

CSR_matrix∗∗	A: pointer to the pointer where the matrix is stored

4.3.1.5 CSR_matrix∗ matrix_product_to_CSR (CSR_matrix ∗ *A*, CSC_matrix ∗ *B*)

Computes the matrix product.

Parameters

CSR_matrix∗	A: first factor of matrix product
CSC_matrix∗	B: second factor of matrix product

Returns

CSR_matrix∗: pointer to the newly allocated matrix product P=A.B

This method computes the matrix product P of the (n x k)-matrix A and the (k x m)-matrix B. The result is the (n x m)-matrix P=A.B. For faster computation the matrix A is given in row format and B in column format.

4.3.1.6 void print_CSC (CSC_matrix ∗ *A*, const char ∗ *Name*)

Prints a CSC matrix to a text file.

Parameters

CSC_matrix∗	A: pointer to a CSC matrix
char∗	Name: absolute filename of the output

This method prints a matrix in "Compressed Sparse Column"-format (CSC) to a textfile. Each column of the matrix A is represented by a line in the text file. The elements are tab-separated. Each element in a column *is represented by a pair (<row index="" j>="">,<A_{j,i}>)*

4.3.1.7 void print_CSC_matrix_market (CSC_matrix ∗ *A*, const char ∗ *Name*)

Prints a CSC matrix to the Matrix Market format.

Parameters

CSC_matrix∗	A: pointer to a CSC matrix
char∗	Name: absolute filename of the output

This method prints a matrix in "Compressed Sparse Column"-format (CSC) to a textfile in a "Matrix Marke exchange format". see `http://math.nist.gov/MatrixMarket/formats.html` for the structure. The file suffix is ".mtx".

4.3.1.8 void print_CSR (CSR_matrix ∗ *A*, const char ∗ *Name*)

Prints a CSR matrix to a text file.

Parameters

CSR_matrix∗	A: pointer to a CSR matrix
char∗	Name: absolute filename of the output

This method prints a matrix in "Compressed Sparse Row"-format (CSR) to a textfile. Each row of the matrix A is represented by a line in the text file. The elements are tab-separated. Each element in a row *is represented by a pair (<column index="" j>="">,<A_{i,j}>)*

4.3.1.9 void print_CSR_matrix_market (CSR_matrix ∗ *A*, const char ∗ *Name*)

Prints a CSR matrix to the Matrix Market format.

Parameters

CSR_matrix∗	A: pointer to a CSR matrix
char∗	Name: absolute filename of the output

This method prints a matrix in "Compressed Sparse Row"-format (CSR) to a textfile in a "Matrix Marke exchange format". see `http://math.nist.gov/MatrixMarket/formats.html` for the structure. The file suffix is ".mtx".

4.3.1.10 void print_list (int ∗ *V*, int *n*, const char ∗ *Name*)

Prints an integer list to a text file.

Parameters

int∗	V: pointer to a list of integer values
int	n: number of entries of the list
char∗	Name: absolute filename of the output

This method prints a list of integer values to a file, where all elements are line-separated.

4.3.1.11 void print_vector (double ∗ *V*, int *n*, const char ∗ *Name*)

Prints a vector to a text file.

Parameters

double∗	V: pointer to a list of double values
int	n: number of entries of the list
char∗	Name: absolute filename of the output

This method prints a list of double values to a file, where all elements are line-separated.

4.3.1.12 int∗ zero_int_list (int *n*)

Allocates a list of integers initialized by zero.

Parameters

int	n: number of entries of the list

Returns

int∗ : pointer to allocated list

4.3.1.13 double∗ zero_vector (int *n*)

Allocates a list of doubles initialized by zero.

Parameters

int	n: dimension of vector

Returns

double∗ : pointer to allocated list

4.4 linalg_stuff.h File Reference

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
```

Classes

- struct CSR
- struct CSC

Typedefs

- typedef struct CSR CSR_matrix
- typedef struct CSC CSC_matrix

Functions

- void print_vector (double ∗V, int n, const char ∗Name)

  Prints a vector to a text file.
- void print_list (int ∗V, int n, const char ∗Name)

  Prints an integer list to a text file.
- void print_CSR (CSR_matrix ∗A, const char ∗Name)

  Prints a CSR matrix to a text file.
- void print_CSC (CSC_matrix ∗A, const char ∗Name)

  Prints a CSC matrix to a text file.
- void print_CSR_matrix_market (CSR_matrix ∗A, const char ∗Name)

  Prints a CSR matrix to the Matrix Market format.
- void print_CSC_matrix_market (CSC_matrix ∗A, const char ∗Name)

  Prints a CSC matrix to the Matrix Market format.
- double ∗ zero_vector (int n)

  Allocates a list of doubles initialized by zero.
- void clear_vector (double ∗V, int n)

  Clears a vector by setting all elements to zero.
- void free_CSR_matrix (CSR_matrix ∗∗const A)

  Frees all memory of a CSR-matrix struct.
- void free_CSC_matrix (CSC_matrix ∗∗const A)

  Frees all memory of a CSC-matrix struct.
- void CSR_scale_matrix (CSR_matrix ∗A, double factor)

  Scales a CSR matrix by a factor.
- CSR_matrix ∗ matrix_product_to_CSR (CSR_matrix ∗A, CSC_matrix ∗B)

Computes the matrix product.

- int ∗ zero_int_list (int n)

  Allocates a list of integers initialized by zero.

4.4.1 Typedef Documentation

4.4.1.1 typedef struct CSC CSC_matrix

4.4.1.2 typedef struct CSR CSR_matrix

4.4.2 Function Documentation

4.4.2.1 void clear_vector (double ∗ *V*, int *n*)

Clears a vector by setting all elements to zero.

Parameters

double∗	V: Pointer to vector to be cleared)
int	n: dimension of vector

4.4.2.2 void CSR_scale_matrix (CSR_matrix ∗ *A*, double *factor*)

Scales a CSR matrix by a factor.

Parameters

CSR_matrix∗	A: pointer the the CSR-matrix to be scaled
double	factor: scaling factor

4.4.2.3 void free_CSC_matrix (CSC_matrix ∗∗const *A*)

Frees all memory of a CSC-matrix struct.

Parameters

CSC_matrix∗∗	A: pointer to the pointer where the matrix is stored

4.4.2.4 void free_CSR_matrix (CSR_matrix ∗∗const *A*)

Frees all memory of a CSR-matrix struct.

Parameters

CSR_matrix∗∗	A: pointer to the pointer where the matrix is stored

4.4.2.5 CSR_matrix∗ matrix_product_to_CSR (CSR_matrix ∗ *A*, CSC_matrix ∗ *B*)

Computes the matrix product.

Parameters

CSR_matrix∗	A: first factor of matrix product
CSC_matrix∗	B: second factor of matrix product

Returns

CSR_matrix∗: pointer to the newly allocated matrix product P=A.B

This method computes the matrix product P of the (n x k)-matrix A and the (k x m)-matrix B. The result is the (n x m)-matrix P=A.B. For faster computation the matrix A is given in row format and B in column format.

4.4.2.6 void print_CSC (**CSC_matrix** ∗ *A,* const char ∗ *Name*)

Prints a CSC matrix to a text file.

Parameters

CSC_matrix∗	A: pointer to a CSC matrix
char∗	Name: absolute filename of the output

This method prints a matrix in "Compressed Sparse Column"-format (CSC) to a textfile. Each column of the matrix A is represented by a line in the text file. The elements are tab-separated. Each element in a column *is represented by a pair (<row index="" j>="">,<A_{j,i}>)*

4.4.2.7 void print_CSC_matrix_market (**CSC_matrix** ∗ *A,* const char ∗ *Name*)

Prints a CSC matrix to the Matrix Market format.

Parameters

CSC_matrix∗	A: pointer to a CSC matrix
char∗	Name: absolute filename of the output

This method prints a matrix in "Compressed Sparse Column"-format (CSC) to a textfile in a "Matrix Marke exchange format". see `http://math.nist.gov/MatrixMarket/formats.html` for the structure. The file suffix is ".mtx".

4.4.2.8 void print_CSR (**CSR_matrix** ∗ *A,* const char ∗ *Name*)

Prints a CSR matrix to a text file.

Parameters

CSR_matrix∗	A: pointer to a CSR matrix
char∗	Name: absolute filename of the output

This method prints a matrix in "Compressed Sparse Row"-format (CSR) to a textfile. Each row of the matrix A is represented by a line in the text file. The elements are tab-separated. Each element in a row *is represented by a pair (<column index="" j>="">,<A_{i,j}>)*

4.4.2.9 void print_CSR_matrix_market (**CSR_matrix** ∗ *A,* const char ∗ *Name*)

Prints a CSR matrix to the Matrix Market format.

Parameters

CSR_matrix∗	A: pointer to a CSR matrix
char∗	Name: absolute filename of the output

This method prints a matrix in "Compressed Sparse Row"-format (CSR) to a textfile in a "Matrix Marke exchange format". see http://math.nist.gov/MatrixMarket/formats.html for the structure. The file suffix is ".mtx".

4.4.2.10 void print_list (int ∗ *V*, int *n*, const char ∗ *Name*)

Prints an integer list to a text file.

Parameters

int∗	V: pointer to a list of integer values
int	n: number of entries of the list
char∗	Name: absolute filename of the output

This method prints a list of integer values to a file, where all elements are line-separated.

4.4.2.11 void print_vector (double ∗ *V*, int *n*, const char ∗ *Name*)

Prints a vector to a text file.

Parameters

double∗	V: pointer to a list of double values
int	n: number of entries of the list
char∗	Name: absolute filename of the output

This method prints a list of double values to a file, where all elements are line-separated.

4.4.2.12 int∗ zero_int_list (int *n*)

Allocates a list of integers initialized by zero.

Parameters

int	n: number of entries of the list

Returns

int∗ : pointer to allocated list

4.4.2.13 double∗ zero_vector (int *n*)

Allocates a list of doubles initialized by zero.

Parameters

int	n: dimension of vector

Returns

double∗ : pointer to allocated list

4.5 testPCG.c File Reference

```
#include "easyPCG.h"
#include "linalg_stuff.h"
#include <stdlib.h>
#include <time.h>
```

Functions

- CSR_matrix ∗ Laplace1D_3point (int size)
- CSR_matrix ∗ Laplace2D_5point (int ∗size)
- double ∗ SourceTerm1D (int size, double left_val, double right_val)
- double ∗ SourceTerm2D (int size, double bound_val)
- int main (int argc, char ∗argv[])

4.5.1 Function Documentation

4.5.1.1 CSR_matrix∗ Laplace1D_3point (int *size*)

4.5.1.2 CSR_matrix∗ Laplace2D_5point (int ∗ *size*)

4.5.1.3 int main (int *argc,* char ∗ *argv[]*)

4.5.1.4 double∗ SourceTerm1D (int *size,* double *left_val,* double *right_val*)

4.5.1.5 double∗ SourceTerm2D (int *size,* double *bound_val*)

Index