C++ Library of the Linear Conjugate Gradient Methods (LibLCG)

Yi Zhang

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 CLCG_CUDA_Solver Class Reference

Complex linear conjugate gradient solver class.

```
#include <solver_cuda.h>
```

Public Member Functions

- CLCG_CUDA_Solver ()
- virtual ~CLCG_CUDA_Solver ()
- virtual void AxProduct (cublasHandle_t cub_handle, cusparseHandle_t cus_handle, cusparseDnVecDescr↩ _t x, cusparseDnVecDescr_t prod_Ax, const int n_size, const int nz_size, cusparseOperation_t oper_t)=0

  Virtual function of the product of A∗x.
- virtual void MxProduct (cublasHandle_t cub_handle, cusparseHandle_t cus_handle, cusparseDnVecDescr↩ _t x, cusparseDnVecDescr_t prod_Mx, const int n_size, const int nz_size, cusparseOperation_t oper_t)=0

  Virtual function of the product of M^-1∗x.
- virtual int Progress (const cuDoubleComplex ∗m, const lcg_float converge, const clcg_para ∗param, const int n_size, const int nz_size, const int k)

  Virtual function of the process monitoring.
- void silent ()

  Do not report any processes.
- void set_report_interval (unsigned int inter)

  Set the interval to run the process monitoring function.
- void set_clcg_parameter (const clcg_para &in_param)

  Set the parameters of the algorithms.
- void Minimize (cublasHandle_t cub_handle, cusparseHandle_t cus_handle, cuDoubleComplex ∗x, cu↩ DoubleComplex ∗b, const int n_size, const int nz_size, clcg_solver_enum solver_id=CLCG_BICG, bool verbose=true, bool er_throw=false)

  Run the constrained minimizing process.
- void MinimizePreconditioned (cublasHandle_t cub_handle, cusparseHandle_t cus_handle, cuDouble↩ Complex ∗x, cuDoubleComplex ∗b, const int n_size, const int nz_size, clcg_solver_enum solver_↩ id=CLCG_PCG, bool verbose=true, bool er_throw=false)

  Run the preconditioned minimizing process.

Static Public Member Functions

- static void _AxProduct (void ∗instance, cublasHandle_t cub_handle, cusparseHandle_t cus_handle, cusparseDnVecDescr_t x, cusparseDnVecDescr_t prod_Ax, const int n_size, const int nz_size, cusparse←Operation_t oper_t)

  Interface of the virtual function of the product of A∗x.
- static void _MxProduct (void ∗instance, cublasHandle_t cub_handle, cusparseHandle_t cus_handle, cusparseDnVecDescr_t x, cusparseDnVecDescr_t prod_Mx, const int n_size, const int nz_size, cusparse←Operation_t oper_t)

  Interface of the virtual function of the product of M^-1∗x.
- static int _Progress (void ∗instance, const cuDoubleComplex ∗m, const lcg_float converge, const clcg_para ∗param, const int n_size, const int nz_size, const int k)

  Interface of the virtual function of the process monitoring.

Protected Attributes

- clcg_para param_
- unsigned int inter_
- bool silent_

3.1.1 Detailed Description

Complex linear conjugate gradient solver class.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 CLCG_CUDA_Solver()

```
CLCG_CUDA_Solver::CLCG_CUDA_Solver ( )
```

3.1.2.2 ∼CLCG_CUDA_Solver()

```
virtual CLCG_CUDA_Solver::∼CLCG_CUDA_Solver ( )  [inline], [virtual]
```

3.1.3 Member Function Documentation

3.1.3.1 _AxProduct()

```
static void CLCG_CUDA_Solver::_AxProduct (
            void * instance,
            cublasHandle_t cub_handle,
            cusparseHandle_t cus_handle,
            cusparseDnVecDescr_t x,
            cusparseDnVecDescr_t prod_Ax,
            const int n_size,
            const int nz_size,
            cusparseOperation_t oper_t )  [inline], [static]
```

Interface of the virtual function of the product of A∗x.

Parameters

instance	User data sent to identify the function address
cub_handle	Handler of the CuBLAS library
cus_handle	Handler of the CuSparse library
x[in]	Pointer of the multiplier
prod_Ax[out]	Pointer of the product
n_size	Size of the solution
nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages
oper_t	Cusparse operator. This parameter is not need by the algorithm. It is passed for CUDA usages

3.1.3.2 _MxProduct()

```
static void CLCG_CUDA_Solver::_MxProduct (
            void * instance,
            cublasHandle_t cub_handle,
            cusparseHandle_t cus_handle,
            cusparseDnVecDescr_t x,
            cusparseDnVecDescr_t prod_Mx,
            const int n_size,
            const int nz_size,
            cusparseOperation_t oper_t )  [inline], [static]
```

Interface of the virtual function of the product of M^-1∗x.

Parameters

instance	User data sent to identify the function address
cub_handle	Handler of the CuBLAS library
cus_handle	Handler of the CuSparse library
x[in]	Pointer of the multiplier
prod_Mx[out]	Pointer of the product
n_size	Size of the solution
nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages
oper_t	Cusparse operator. This parameter is not need by the algorithm. It is passed for CUDA usages

3.1.3.3 _Progress()

```
static int CLCG_CUDA_Solver::_Progress (
            void * instance,
            const cuDoubleComplex * m,
            const lcg_float converge,
            const clcg_para * param,
```

```
            const int n_size,
            const int nz_size,
            const int k )  [inline], [static]
```

Interface of the virtual function of the process monitoring.

Parameters

instance	User data sent to identify the function address
m	Pointer of the current solution
converge	Current value of the convergence
param	Pointer of the parameters used in the algorithms
n_size	Size of the solution
nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages
k	Current iteration times

Returns

int Status of the process

3.1.3.4 AxProduct()

```
virtual void CLCG_CUDA_Solver::AxProduct (
            cublasHandle_t cub_handle,
            cusparseHandle_t cus_handle,
            cusparseDnVecDescr_t x,
            cusparseDnVecDescr_t prod_Ax,
            const int n_size,
            const int nz_size,
            cusparseOperation_t oper_t )  [pure virtual]
```

Virtual function of the product of A∗x.

Parameters

cub_handle	Handler of the CuBLAS library
cus_handle	Handler of the CuSparse library
x[in]	Pointer of the multiplier
prod_Ax[out]	Pointer of the product
n_size	Size of the solution
nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages
oper_t	Cusparse operator. This parameter is not need by the algorithm. It is passed for CUDA usages

3.1.3.5 Minimize()

```
void CLCG_CUDA_Solver::Minimize (
            cublasHandle_t cub_handle,
            cusparseHandle_t cus_handle,
            cuDoubleComplex * x,
            cuDoubleComplex * b,
            const int n_size,
            const int nz_size,
            clcg_solver_enum solver_id = CLCG_BICG,
            bool verbose = true,
            bool er_throw = false )
```

Run the constrained minimizing process.

Parameters

cub_handle	Handler of the CuBLAS library
cus_handle	Handler of the CuSparse library
x	Pointer of the solution vector
b	Pointer of the targeting vector
n_size	Size of the solution vector
nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages
solver_id	Solver type
verbose	Report more information of the full process
er_throw	Instead of showing error messages on screen, throw them out using std::exception

3.1.3.6 MinimizePreconditioned()

```
void CLCG_CUDA_Solver::MinimizePreconditioned (
            cublasHandle_t cub_handle,
            cusparseHandle_t cus_handle,
            cuDoubleComplex * x,
            cuDoubleComplex * b,
            const int n_size,
            const int nz_size,
            clcg_solver_enum solver_id = CLCG_PCG,
            bool verbose = true,
            bool er_throw = false )
```

Run the preconditioned minimizing process.

Parameters

cub_handle	Handler of the CuBLAS library
cus_handle	Handler of the CuSparse library
x	Pointer of the solution vector
b	Pointer of the targeting vector
n_size	Size of the solution vector

Parameters

nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages
solver_id	Solver type
verbose	Report more information of the full process
er_throw	Instead of showing error messages on screen, throw them out using std::exception

3.1.3.7 MxProduct()

```
virtual void CLCG_CUDA_Solver::MxProduct (
            cublasHandle_t cub_handle,
            cusparseHandle_t cus_handle,
            cusparseDnVecDescr_t x,
            cusparseDnVecDescr_t prod_Mx,
            const int n_size,
            const int nz_size,
            cusparseOperation_t oper_t )  [pure virtual]
```

Virtual function of the product of M^-1*x.

Parameters

cub_handle	Handler of the CuBLAS library
cus_handle	Handler of the CuSparse library
x[in]	Pointer of the multiplier
prod_Mx[out]	Pointer of the product
n_size	Size of the solution
nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages
oper_t	Cusparse operator. This parameter is not need by the algorithm. It is passed for CUDA usages

3.1.3.8 Progress()

```
virtual int CLCG_CUDA_Solver::Progress (
            const cuDoubleComplex * m,
            const lcg_float converge,
            const clcg_para * param,
            const int n_size,
            const int nz_size,
            const int k )  [virtual]
```

Virtual function of the process monitoring.

Parameters

m	Pointer of the current solution
converge	Current value of the convergence
param	Pointer of the parameters used in the algorithms
n_size	Size of the solution
nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages
k	Current iteration times

Returns

int Status of the process

3.1.3.9 set_clcg_parameter()

```
void CLCG_CUDA_Solver::set_clcg_parameter (
            const clcg_para & in_param )
```

Set the parameters of the algorithms.

Parameters

in_param	the input parameters

3.1.3.10 set_report_interval()

```
void CLCG_CUDA_Solver::set_report_interval (
            unsigned int inter )
```

Set the interval to run the process monitoring function.

Parameters

inter	the interval

3.1.3.11 silent()

```
void CLCG_CUDA_Solver::silent ( )
```

Do not report any processes.

3.1.4 Field Documentation

3.1.4.1 inter_

`unsigned int CLCG_CUDA_Solver::inter_ [protected]`

3.1.4.2 param_

`clcg_para CLCG_CUDA_Solver::param_ [protected]`

3.1.4.3 silent_

`bool CLCG_CUDA_Solver::silent_ [protected]`

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

- solver_cuda.h

3.2 CLCG_CUDAF_Solver Class Reference

Complex linear conjugate gradient solver class.

`#include <solver_cuda.h>`

Public Member Functions

- CLCG_CUDAF_Solver ()
- virtual ∼CLCG_CUDAF_Solver ()
- virtual void AxProduct (cublasHandle_t cub_handle, cusparseHandle_t cus_handle, cusparseDnVecDescr←_t x, cusparseDnVecDescr_t prod_Ax, const int n_size, const int nz_size, cusparseOperation_t oper_t)=0

    Virtual function of the product of A∗x.
- virtual void MxProduct (cublasHandle_t cub_handle, cusparseHandle_t cus_handle, cusparseDnVecDescr←_t x, cusparseDnVecDescr_t prod_Mx, const int n_size, const int nz_size, cusparseOperation_t oper_t)=0

    Virtual function of the product of M^-1∗x.
- virtual int Progress (const cuComplex ∗m, const float converge, const clcg_para ∗param, const int n_size, const int nz_size, const int k)

    Virtual function of the process monitoring.
- void silent ()

    Do not report any processes.
- void set_report_interval (unsigned int inter)

    Set the interval to run the process monitoring function.
- void set_clcg_parameter (const clcg_para &in_param)

    Set the parameters of the algorithms.
- void Minimize (cublasHandle_t cub_handle, cusparseHandle_t cus_handle, cuComplex ∗x, cuComplex ∗b, const int n_size, const int nz_size, clcg_solver_enum solver_id=CLCG_BICG, bool verbose=true, bool er_←throw=false)

    Run the constrained minimizing process.
- void MinimizePreconditioned (cublasHandle_t cub_handle, cusparseHandle_t cus_handle, cuComplex ∗x, cuComplex ∗b, const int n_size, const int nz_size, clcg_solver_enum solver_id=CLCG_PCG, bool verbose=true, bool er_throw=false)

    Run the preconditioned minimizing process.

Static Public Member Functions

- static void _AxProduct (void *instance, cublasHandle_t cub_handle, cusparseHandle_t cus_handle, cusparseDnVecDescr_t x, cusparseDnVecDescr_t prod_Ax, const int n_size, const int nz_size, cusparse↩ Operation_t oper_t)

  *Interface of the virtual function of the product of A*x.*
- static void _MxProduct (void *instance, cublasHandle_t cub_handle, cusparseHandle_t cus_handle, cusparseDnVecDescr_t x, cusparseDnVecDescr_t prod_Mx, const int n_size, const int nz_size, cusparse↩ Operation_t oper_t)

  *Interface of the virtual function of the product of M^-1*x.*
- static int _Progress (void *instance, const cuComplex *m, const float converge, const clcg_para *param, const int n_size, const int nz_size, const int k)

  Interface of the virtual function of the process monitoring.

Protected Attributes

- clcg_para param_
- unsigned int inter_
- bool silent_

3.2.1 Detailed Description

Complex linear conjugate gradient solver class.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 CLCG_CUDAF_Solver()

```
CLCG_CUDAF_Solver::CLCG_CUDAF_Solver ( )
```

3.2.2.2 ∼CLCG_CUDAF_Solver()

```
virtual CLCG_CUDAF_Solver::∼CLCG_CUDAF_Solver ( ) [inline], [virtual]
```

3.2.3 Member Function Documentation

3.2.3.1 _AxProduct()

```
static void CLCG_CUDAF_Solver::_AxProduct (
            void * instance,
            cublasHandle_t cub_handle,
            cusparseHandle_t cus_handle,
            cusparseDnVecDescr_t x,
            cusparseDnVecDescr_t prod_Ax,
            const int n_size,
            const int nz_size,
            cusparseOperation_t oper_t ) [inline], [static]
```

Interface of the virtual function of the product of A*x.

Parameters

instance	User data sent to identify the function address
cub_handle	Handler of the CuBLAS library
cus_handle	Handler of the CuSparse library
x[in]	Pointer of the multiplier
prod_Ax[out]	Pointer of the product
n_size	Size of the solution
nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages
oper_t	Cusparse operator. This parameter is not need by the algorithm. It is passed for CUDA usages

3.2.3.2 _MxProduct()

```
static void CLCG_CUDAF_Solver::_MxProduct (
            void * instance,
            cublasHandle_t cub_handle,
            cusparseHandle_t cus_handle,
            cusparseDnVecDescr_t x,
            cusparseDnVecDescr_t prod_Mx,
            const int n_size,
            const int nz_size,
            cusparseOperation_t oper_t )  [inline], [static]
```

Interface of the virtual function of the product of M^-1∗x.

Parameters

instance	User data sent to identify the function address
cub_handle	Handler of the CuBLAS library
cus_handle	Handler of the CuSparse library
x[in]	Pointer of the multiplier
prod_Mx[out]	Pointer of the product
n_size	Size of the solution
nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages
oper_t	Cusparse operator. This parameter is not need by the algorithm. It is passed for CUDA usages

3.2.3.3 _Progress()

```
static int CLCG_CUDAF_Solver::_Progress (
            void * instance,
            const cuComplex * m,
            const float converge,
            const clcg_para * param,
```

```
            const int n_size,
            const int nz_size,
            const int k )  [inline], [static]
```

Interface of the virtual function of the process monitoring.

Parameters

instance	User data sent to identify the function address
m	Pointer of the current solution
converge	Current value of the convergence
param	Pointer of the parameters used in the algorithms
n_size	Size of the solution
nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages
k	Current iteration times

Returns

int Status of the process

3.2.3.4 AxProduct()

```
virtual void CLCG_CUDAF_Solver::AxProduct (
            cublasHandle_t cub_handle,
            cusparseHandle_t cus_handle,
            cusparseDnVecDescr_t x,
            cusparseDnVecDescr_t prod_Ax,
            const int n_size,
            const int nz_size,
            cusparseOperation_t oper_t )  [pure virtual]
```

Virtual function of the product of A∗x.

Parameters

cub_handle	Handler of the CuBLAS library
cus_handle	Handler of the CuSparse library
x[in]	Pointer of the multiplier
prod_Ax[out]	Pointer of the product
n_size	Size of the solution
nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages
oper_t	Cusparse operator. This parameter is not need by the algorithm. It is passed for CUDA usages

3.2.3.5 Minimize()

```
void CLCG_CUDAF_Solver::Minimize (
            cublasHandle_t cub_handle,
            cusparseHandle_t cus_handle,
            cuComplex * x,
            cuComplex * b,
            const int n_size,
            const int nz_size,
            clcg_solver_enum solver_id = CLCG_BICG,
            bool verbose = true,
            bool er_throw = false )
```

Run the constrained minimizing process.

Parameters

cub_handle	Handler of the CuBLAS library
cus_handle	Handler of the CuSparse library
x	Pointer of the solution vector
b	Pointer of the targeting vector
n_size	Size of the solution vector
nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages
solver_id	Solver type
verbose	Report more information of the full process
er_throw	Instead of showing error messages on screen, throw them out using std::exception

3.2.3.6 MinimizePreconditioned()

```
void CLCG_CUDAF_Solver::MinimizePreconditioned (
            cublasHandle_t cub_handle,
            cusparseHandle_t cus_handle,
            cuComplex * x,
            cuComplex * b,
            const int n_size,
            const int nz_size,
            clcg_solver_enum solver_id = CLCG_PCG,
            bool verbose = true,
            bool er_throw = false )
```

Run the preconditioned minimizing process.

Parameters

cub_handle	Handler of the CuBLAS library
cus_handle	Handler of the CuSparse library
x	Pointer of the solution vector
b	Pointer of the targeting vector
n_size	Size of the solution vector

Parameters

nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages
solver_id	Solver type
verbose	Report more information of the full process
er_throw	Instead of showing error messages on screen, throw them out using std::exception

3.2.3.7 MxProduct()

```
virtual void CLCG_CUDAF_Solver::MxProduct (
             cublasHandle_t cub_handle,
             cusparseHandle_t cus_handle,
             cusparseDnVecDescr_t x,
             cusparseDnVecDescr_t prod_Mx,
             const int n_size,
             const int nz_size,
             cusparseOperation_t oper_t )  [pure virtual]
```

Virtual function of the product of M^-1*x.

Parameters

cub_handle	Handler of the CuBLAS library
cus_handle	Handler of the CuSparse library
x[in]	Pointer of the multiplier
prod_Mx[out]	Pointer of the product
n_size	Size of the solution
nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages
oper_t	Cusparse operator. This parameter is not need by the algorithm. It is passed for CUDA usages

3.2.3.8 Progress()

```
virtual int CLCG_CUDAF_Solver::Progress (
             const cuComplex * m,
             const float converge,
             const clcg_para * param,
             const int n_size,
             const int nz_size,
             const int k )  [virtual]
```

Virtual function of the process monitoring.

Parameters

m	Pointer of the current solution
converge	Current value of the convergence
param	Pointer of the parameters used in the algorithms
n_size	Size of the solution
nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages
k	Current iteration times

Returns

int Status of the process

3.2.3.9 set_clcg_parameter()

```
void CLCG_CUDAF_Solver::set_clcg_parameter (
            const clcg_para & in_param )
```

Set the parameters of the algorithms.

Parameters

in_param	the input parameters

3.2.3.10 set_report_interval()

```
void CLCG_CUDAF_Solver::set_report_interval (
            unsigned int inter )
```

Set the interval to run the process monitoring function.

Parameters

inter	the interval

3.2.3.11 silent()

```
void CLCG_CUDAF_Solver::silent ( )
```

Do not report any processes.

3.2.4 Field Documentation

3.2.4.1 inter_

```
unsigned int CLCG_CUDAF_Solver::inter_ [protected]
```

3.2.4.2 param_

```
clcg_para CLCG_CUDAF_Solver::param_ [protected]
```

3.2.4.3 silent_

```
bool CLCG_CUDAF_Solver::silent_ [protected]
```

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

- solver_cuda.h

3.3 CLCG_EIGEN_Solver Class Reference

Complex linear conjugate gradient solver class.

```
#include <solver_eigen.h>
```

Public Member Functions

- CLCG_EIGEN_Solver ()
- virtual ∼CLCG_EIGEN_Solver ()
- virtual void AxProduct (const Eigen::VectorXcd &x, Eigen::VectorXcd &prod_Ax, lcg_matrix_e layout, clcg_complex_e conjugate)=0

  Interface of the virtual function of the product of A∗x.
- virtual void MxProduct (const Eigen::VectorXcd &x, Eigen::VectorXcd &prod_Mx, lcg_matrix_e layout, clcg_complex_e conjugate)=0

  Interface of the virtual function of the product of M^-1∗x.
- virtual int Progress (const Eigen::VectorXcd ∗m, const lcg_float converge, const clcg_para ∗param, const int k)

  Virtual function of the process monitoring.
- void silent ()

  Do not report any processes.
- void set_report_interval (unsigned int inter)

  Set the interval to run the process monitoring function.
- void set_clcg_parameter (const clcg_para &in_param)

  Set the interval to run the process monitoring function.
- void Minimize (Eigen::VectorXcd &m, const Eigen::VectorXcd &b, clcg_solver_enum solver_id=CLCG_CGS, bool verbose=true, bool er_throw=false)

  Run the minimizing process.
- void MinimizePreconditioned (Eigen::VectorXcd &m, const Eigen::VectorXcd &b, clcg_solver_enum solver_id=CLCG_PBICG, bool verbose=true, bool er_throw=false)

  Run the preconitioned minimizing process.

Static Public Member Functions

- static void _AxProduct (void ∗instance, const Eigen::VectorXcd &x, Eigen::VectorXcd &prod_Ax, lcg_matrix_e layout, clcg_complex_e conjugate)

    Interface of the virtual function of the product of A∗x.
- static void _MxProduct (void ∗instance, const Eigen::VectorXcd &x, Eigen::VectorXcd &prod_Mx, lcg_matrix_e layout, clcg_complex_e conjugate)

    Interface of the virtual function of the product of M^-1∗x.
- static int _Progress (void ∗instance, const Eigen::VectorXcd ∗m, const lcg_float converge, const clcg_para ∗param, const int k)

    Interface of the virtual function of the process monitoring.

Protected Attributes

- clcg_para param_
- unsigned int inter_
- bool silent_

3.3.1 Detailed Description

Complex linear conjugate gradient solver class.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 CLCG_EIGEN_Solver()

```
CLCG_EIGEN_Solver::CLCG_EIGEN_Solver ( )
```

3.3.2.2 ∼CLCG_EIGEN_Solver()

```
virtual CLCG_EIGEN_Solver::∼CLCG_EIGEN_Solver ( ) [inline], [virtual]
```

3.3.3 Member Function Documentation

3.3.3.1 _AxProduct()

```
static void CLCG_EIGEN_Solver::_AxProduct (
            void * instance,
            const Eigen::VectorXcd & x,
            Eigen::VectorXcd & prod_Ax,
            lcg_matrix_e layout,
            clcg_complex_e conjugate ) [inline], [static]
```

Interface of the virtual function of the product of A∗x.

Parameters

instance	User data sent to identify the function address
x[in]	Pointer of the multiplier
prod_Ax[out]	Pointer of the product
layout	Layout of the kernel matrix. This is passed for the clcg_matvec() function
conjugate	Welther to use conjugate of the kernel matrix. This is passed for the clcg_matvec() function

3.3.3.2 _MxProduct()

```
static void CLCG_EIGEN_Solver::_MxProduct (
             void * instance,
             const Eigen::VectorXcd & x,
             Eigen::VectorXcd & prod_Mx,
             lcg_matrix_e layout,
             clcg_complex_e conjugate ) [inline], [static]
```

Interface of the virtual function of the product of M^-1∗x.

Parameters

instance	User data sent to identify the function address
x[in]	Pointer of the multiplier
prod_Mx[out]	Pointer of the product
layout	Layout of the kernel matrix. This is passed for the clcg_matvec() function
conjugate	Welther to use conjugate of the kernel matrix. This is passed for the clcg_matvec() function

3.3.3.3 _Progress()

```
static int CLCG_EIGEN_Solver::_Progress (
             void * instance,
             const Eigen::VectorXcd * m,
             const lcg_float converge,
             const clcg_para * param,
             const int k ) [inline], [static]
```

Interface of the virtual function of the process monitoring.

Parameters

instance	User data sent to identify the function address
m	Pointer of the current solution
converge	Current value of the convergence
param	Pointer of the parameters used in the algorithms
k	Current iteration times

Returns

int Status of the process

3.3.3.4 AxProduct()

```
virtual void CLCG_EIGEN_Solver::AxProduct (
            const Eigen::VectorXcd & x,
            Eigen::VectorXcd & prod_Ax,
            lcg_matrix_e layout,
            clcg_complex_e conjugate )  [pure virtual]
```

Interface of the virtual function of the product of A∗x.

Parameters

x[in]	Pointer of the multiplier
prod_Ax[out]	Pointer of the product
layout	Layout of the kernel matrix. This is passed for the clcg_matvec() function
conjugate	Welther to use conjugate of the kernel matrix. This is passed for the clcg_matvec() function

3.3.3.5 Minimize()

```
void CLCG_EIGEN_Solver::Minimize (
            Eigen::VectorXcd & m,
            const Eigen::VectorXcd & b,
            clcg_solver_enum solver_id = CLCG_CGS,
            bool verbose = true,
            bool er_throw = false )
```

Run the minimizing process.

Parameters

m	Pointer of the solution vector
b	Pointer of the targeting vector
solver←_id	Solver type
verbose	Report more information of the full process
er_throw	Instead of showing error messages on screen, throw them out using std::exception

3.3.3.6 MinimizePreconditioned()

```
void CLCG_EIGEN_Solver::MinimizePreconditioned (
```

```
Eigen::VectorXcd & m,
const Eigen::VectorXcd & b,
clcg_solver_enum solver_id = CLCG_PBICG,
bool verbose = true,
bool er_throw = false )
```

Run the preconitioned minimizing process.

Parameters

m	Pointer of the solution vector
b	Pointer of the targeting vector
solver←_id	Solver type
verbose	Report more information of the full process
er_throw	Instead of showing error messages on screen, throw them out using std::exception

3.3.3.7 MxProduct()

```
virtual void CLCG_EIGEN_Solver::MxProduct (
        const Eigen::VectorXcd & x,
        Eigen::VectorXcd & prod_Mx,
        lcg_matrix_e layout,
        clcg_complex_e conjugate ) [pure virtual]
```

Interface of the virtual function of the product of M^-1∗x.

Parameters

x[in]	Pointer of the multiplier
prod_Mx[out]	Pointer of the product
layout	Layout of the kernel matrix. This is passed for the clcg_matvec() function
conjugate	Welther to use conjugate of the kernel matrix. This is passed for the clcg_matvec() function

3.3.3.8 Progress()

```
int CLCG_EIGEN_Solver::Progress (
        const Eigen::VectorXcd * m,
        const lcg_float converge,
        const clcg_para * param,
        const int k ) [virtual]
```

Virtual function of the process monitoring.

Parameters

m	Pointer of the current solution

Parameters

converge	Current value of the convergence
param	Pointer of the parameters used in the algorithms
k	Current iteration times

Returns

int Status of the process

3.3.3.9 set_clcg_parameter()

```
void CLCG_EIGEN_Solver::set_clcg_parameter (
            const clcg_para & in_param )
```

Set the interval to run the process monitoring function.

Parameters

inter	the interval

3.3.3.10 set_report_interval()

```
void CLCG_EIGEN_Solver::set_report_interval (
            unsigned int inter )
```

Set the interval to run the process monitoring function.

Parameters

inter	the interval

3.3.3.11 silent()

```
void CLCG_EIGEN_Solver::silent ( )
```

Do not report any processes.

3.3.4 Field Documentation

3.3.4.1 inter_

```
unsigned int CLCG_EIGEN_Solver::inter_ [protected]
```

3.3.4.2 param_

```
clcg_para CLCG_EIGEN_Solver::param_ [protected]
```

3.3.4.3 silent_

```
bool CLCG_EIGEN_Solver::silent_ [protected]
```

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

- solver_eigen.h
- solver_eigen.cpp

3.4 clcg_para Struct Reference

Parameters of the conjugate gradient methods.

```
#include <util.h>
```

Data Fields

- int max_iterations
- lcg_float epsilon
- int abs_diff

3.4.1 Detailed Description

Parameters of the conjugate gradient methods.

3.4.2 Field Documentation

3.4.2.1 abs_diff

```
int clcg_para::abs_diff
```

Whether to use absolute mean differences (AMD) between |Ax - B| to evaluate the process. The default value is false which means the gradient based evaluating method is used. The AMD based method will be used if this variable is set to true. This parameter is only applied to the non-constrained methods.

3.4.2.2 epsilon

```
lcg_float clcg_para::epsilon
```

Epsilon for convergence test. This parameter determines the accuracy with which the solution is to be found. A minimization terminates when ||g||/max(||x||, 1.0) <= epsilon or |Ax - B| <= epsilon for the lcg_solver() function, where ||.|| denotes the Euclidean (L2) norm and | | denotes the L1 norm. The default value of epsilon is 1e-6. For box-constrained methods, the convergence test is implemented using ||P(m-g) - m|| <= epsilon, in which P is the projector that transfers m into the constrained domain.

3.4.2.3 max_iterations

```
int clcg_para::max_iterations
```

Maximal iteration times. The process will continue till the convergance is met if this option is set to zero (default).

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

- util.h

3.5 CLCG_Solver Class Reference

Complex linear conjugate gradient solver class.

```
#include <solver.h>
```

Public Member Functions

- CLCG_Solver ()
- virtual ~CLCG_Solver ()
- virtual void AxProduct (const lcg_complex *x, lcg_complex *prod_Ax, const int x_size, lcg_matrix_e layout, clcg_complex_e conjugate)=0

  *Interface of the virtual function of the product of A*x.*
- virtual int Progress (const lcg_complex *m, const lcg_float converge, const clcg_para *param, const int n_size, const int k)

  Interface of the virtual function of the process monitoring.
- void silent ()

  Do not report any processes.
- void set_report_interval (unsigned int inter)

  Set the interval to run the process monitoring function.
- void set_clcg_parameter (const clcg_para &in_param)

  Set the parameters of the algorithms.
- void Minimize (lcg_complex *m, const lcg_complex *b, int x_size, clcg_solver_enum solver_id=CLCG_CGS, bool verbose=true, bool er_throw=false)

  Run the minimizing process.

Static Public Member Functions

- static void _AxProduct (void *instance, const lcg_complex *x, lcg_complex *prod_Ax, const int x_size, lcg_matrix_e layout, clcg_complex_e conjugate)

  *Interface of the virtual function of the product of A*x.*
- static int _Progress (void *instance, const lcg_complex *m, const lcg_float converge, const clcg_para *param, const int n_size, const int k)

  Interface of the virtual function of the process monitoring.

Protected Attributes

- clcg_para param_
- unsigned int inter_
- bool silent_

3.5.1 Detailed Description

Complex linear conjugate gradient solver class.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 CLCG_Solver()

```
CLCG_Solver::CLCG_Solver ( )
```

3.5.2.2 ∼CLCG_Solver()

```
virtual CLCG_Solver::~CLCG_Solver ( )  [inline], [virtual]
```

3.5.3 Member Function Documentation

3.5.3.1 _AxProduct()

```
static void CLCG_Solver::_AxProduct (
            void * instance,
            const lcg_complex * x,
            lcg_complex * prod_Ax,
            const int x_size,
            lcg_matrix_e layout,
            clcg_complex_e conjugate )  [inline], [static]
```

Interface of the virtual function of the product of A*x.

Parameters

instance	User data sent to identify the function address
x[in]	Pointer of the multiplier
prod_Ax[out]	Pointer of the product
x_size	Size of the array
layout	Layout of the kernel matrix. This is passed for the clcg_matvec() function
conjugate	Welther to use conjugate of the kernel matrix. This is passed for the clcg_matvec() function

3.5.3.2 _Progress()

```
static int CLCG_Solver::_Progress (
            void * instance,
            const lcg_complex * m,
            const lcg_float converge,
            const clcg_para * param,
            const int n_size,
            const int k )  [inline], [static]
```

Interface of the virtual function of the process monitoring.

Parameters

instance	User data sent to identify the function address
m	Pointer of the current solution
converge	Current value of the convergence
param	Pointer of the parameters used in the algorithms
n_size	Size of the solution
k	Current iteration times

Returns

int Status of the process

3.5.3.3 AxProduct()

```
virtual void CLCG_Solver::AxProduct (
            const lcg_complex * x,
            lcg_complex * prod_Ax,
            const int x_size,
            lcg_matrix_e layout,
            clcg_complex_e conjugate )  [pure virtual]
```

Interface of the virtual function of the product of A∗x.

Parameters

x[in]	Pointer of the multiplier
prod_Ax[out]	Pointer of the product
x_size	Size of the array
layout	Layout of the kernel matrix. This is passed for the clcg_matvec() function
conjugate	Welther to use conjugate of the kernel matrix. This is passed for the clcg_matvec() function

3.5.3.4 Minimize()

```
void CLCG_Solver::Minimize (
            lcg_complex * m,
            const lcg_complex * b,
            int x_size,
            clcg_solver_enum solver_id = CLCG_CGS,
            bool verbose = true,
            bool er_throw = false )
```

Run the minimizing process.

Parameters

m	Pointer of the solution vector
b	Pointer of the targeting vector
x_size	Size of the solution vector
solver←_id	Solver type
verbose	Report more information of the full process
er_throw	Instead of showing error messages on screen, throw them out using std::exception

3.5.3.5 Progress()

```
int CLCG_Solver::Progress (
            const lcg_complex * m,
            const lcg_float converge,
            const clcg_para * param,
            const int n_size,
            const int k ) [virtual]
```

Interface of the virtual function of the process monitoring.

Parameters

m	Pointer of the current solution
converge	Current value of the convergence
param	Pointer of the parameters used in the algorithms
n_size	Size of the solution
k	Current iteration times

Returns

int Status of the process

3.5.3.6 set_clcg_parameter()

```
void CLCG_Solver::set_clcg_parameter (
            const clcg_para & in_param )
```

Set the parameters of the algorithms.

Parameters

in_param	the input parameters

3.5.3.7 set_report_interval()

```
void CLCG_Solver::set_report_interval (
            unsigned int inter )
```

Set the interval to run the process monitoring function.

Parameters

inter	the interval

3.5.3.8 silent()

```
void CLCG_Solver::silent ( )
```

Do not report any processes.

3.5.4 Field Documentation

3.5.4.1 inter_

```
unsigned int CLCG_Solver::inter_ [protected]
```

3.5.4.2 param_

```
clcg_para CLCG_Solver::param_ [protected]
```

3.5.4.3 silent_

```
bool CLCG_Solver::silent_ [protected]
```

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

- solver.h
- solver.cpp

3.6 LCG_CUDA_Solver Class Reference

Linear conjugate gradient solver class.

```
#include <solver_cuda.h>
```

Public Member Functions

- LCG_CUDA_Solver ()
- virtual ~LCG_CUDA_Solver ()
- virtual void AxProduct (cublasHandle_t cub_handle, cusparseHandle_t cus_handle, cusparseDnVecDescr←_t x, cusparseDnVecDescr_t prod_Ax, const int n_size, const int nz_size)=0

  Virtual function of the product of A∗x.
- virtual void MxProduct (cublasHandle_t cub_handle, cusparseHandle_t cus_handle, cusparseDnVecDescr←_t x, cusparseDnVecDescr_t prod_Mx, const int n_size, const int nz_size)=0

  Virtual function of the product of M^-1∗x.
- virtual int Progress (const lcg_float ∗m, const lcg_float converge, const lcg_para ∗param, const int n_size, const int nz_size, const int k)

  Virtual function of the process monitoring.
- void silent ()

  Do not report any processes.
- void set_report_interval (unsigned int inter)

  Set the interval to run the process monitoring function.
- void set_lcg_parameter (const lcg_para &in_param)

  Set the parameters of the algorithms.
- void Minimize (cublasHandle_t cub_handle, cusparseHandle_t cus_handle, lcg_float ∗x, lcg_float ∗b, const int n_size, const int nz_size, lcg_solver_enum solver_id=LCG_CG, bool verbose=true, bool er_throw=false)

  Run the constrained minimizing process.
- void MinimizePreconditioned (cublasHandle_t cub_handle, cusparseHandle_t cus_handle, lcg_float ∗x, lcg_float ∗b, const int n_size, const int nz_size, lcg_solver_enum solver_id=LCG_CG, bool verbose=true, bool er_throw=false)

  Run the preconditioned minimizing process.
- void MinimizeConstrained (cublasHandle_t cub_handle, cusparseHandle_t cus_handle, lcg_float ∗x, const lcg_float ∗b, const lcg_float ∗low, const lcg_float ∗hig, const int n_size, const int nz_size, lcg_solver_enum solver_id=LCG_PG, bool verbose=true, bool er_throw=false)

  Run the constrained minimizing process.

Static Public Member Functions

- static void _AxProduct (void ∗instance, cublasHandle_t cub_handle, cusparseHandle_t cus_handle, cusparseDnVecDescr_t x, cusparseDnVecDescr_t prod_Ax, const int n_size, const int nz_size)

  Interface of the virtual function of the product of A∗x.
- static void _MxProduct (void ∗instance, cublasHandle_t cub_handle, cusparseHandle_t cus_handle, cusparseDnVecDescr_t x, cusparseDnVecDescr_t prod_Mx, const int n_size, const int nz_size)

  Interface of the virtual function of the product of M^-1∗x.
- static int _Progress (void ∗instance, const lcg_float ∗m, const lcg_float converge, const lcg_para ∗param, const int n_size, const int nz_size, const int k)

  Interface of the virtual function of the process monitoring.

Protected Attributes

- lcg_para param_
- unsigned int inter_
- bool silent_

3.6.1 Detailed Description

Linear conjugate gradient solver class.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 LCG_CUDA_Solver()

```
LCG_CUDA_Solver::LCG_CUDA_Solver ( )
```

3.6.2.2 ∼LCG_CUDA_Solver()

```
virtual LCG_CUDA_Solver::∼LCG_CUDA_Solver ( ) [inline], [virtual]
```

3.6.3 Member Function Documentation

3.6.3.1 _AxProduct()

```
static void LCG_CUDA_Solver::_AxProduct (
            void ∗ instance,
            cublasHandle_t cub_handle,
            cusparseHandle_t cus_handle,
            cusparseDnVecDescr_t x,
            cusparseDnVecDescr_t prod_Ax,
            const int n_size,
            const int nz_size ) [inline], [static]
```

Interface of the virtual function of the product of A∗x.

Parameters

instance	User data sent to identify the function address
cub_handle	Handler of the CuBLAS library
cus_handle	Handler of the CuSparse library
x[in]	Pointer of the multiplier
prod_Ax[out]	Pointer of the product
n_size	Size of the solution
nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages

3.6.3.2 _MxProduct()

```
static void LCG_CUDA_Solver::_MxProduct (
            void * instance,
            cublasHandle_t cub_handle,
            cusparseHandle_t cus_handle,
            cusparseDnVecDescr_t x,
            cusparseDnVecDescr_t prod_Mx,
            const int n_size,
            const int nz_size )  [inline], [static]
```

Interface of the virtual function of the product of M^-1∗x.

Parameters

instance	User data sent to identify the function address
cub_handle	Handler of the CuBLAS library
cus_handle	Handler of the CuSparse library
x[in]	Pointer of the multiplier
prod_Mx[out]	Pointer of the product
n_size	Size of the solution
nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages

3.6.3.3 _Progress()

```
static int LCG_CUDA_Solver::_Progress (
            void * instance,
            const lcg_float * m,
            const lcg_float converge,
            const lcg_para * param,
            const int n_size,
            const int nz_size,
            const int k )  [inline], [static]
```

Interface of the virtual function of the process monitoring.

Parameters

instance	User data sent to identify the function address
m	Pointer of the current solution
converge	Current value of the convergence
param	Pointer of the parameters used in the algorithms
n_size	Size of the solution
nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages
k	Current iteration times

Returns

int Status of the process

3.6.3.4 AxProduct()

```
virtual void LCG_CUDA_Solver::AxProduct (
            cublasHandle_t cub_handle,
            cusparseHandle_t cus_handle,
            cusparseDnVecDescr_t x,
            cusparseDnVecDescr_t prod_Ax,
            const int n_size,
            const int nz_size ) [pure virtual]
```

Virtual function of the product of A∗x.

Parameters

cub_handle	Handler of the CuBLAS library
cus_handle	Handler of the CuSparse library
x[in]	Pointer of the multiplier
prod_Ax[out]	Pointer of the product
n_size	Size of the solution
nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages

3.6.3.5 Minimize()

```
void LCG_CUDA_Solver::Minimize (
            cublasHandle_t cub_handle,
            cusparseHandle_t cus_handle,
            lcg_float * x,
            lcg_float * b,
            const int n_size,
```

```
              const int nz_size,
              lcg_solver_enum solver_id = LCG_CG,
              bool verbose = true,
              bool er_throw = false )
```

Run the constrained minimizing process.

Parameters

cub_handle	Handler of the CuBLAS library
cus_handle	Handler of the CuSparse library
x	Pointer of the solution vector
b	Pointer of the targeting vector
n_size	Size of the solution vector
nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages
solver_id	Solver type
verbose	Report more information of the full process
er_throw	Instead of showing error messages on screen, throw them out using std::exception

3.6.3.6 MinimizeConstrained()

```
void LCG_CUDA_Solver::MinimizeConstrained (
              cublasHandle_t cub_handle,
              cusparseHandle_t cus_handle,
              lcg_float * x,
              const lcg_float * b,
              const lcg_float * low,
              const lcg_float * hig,
              const int n_size,
              const int nz_size,
              lcg_solver_enum solver_id = LCG_PG,
              bool verbose = true,
              bool er_throw = false )
```

Run the constrained minimizing process.

Parameters

cub_handle	Handler of the CuBLAS library
cus_handle	Handler of the CuSparse library
x	Pointer of the solution vector
b	Pointer of the targeting vector
low	Lower bound of the solution vector
hig	Higher bound of the solution vector
n_size	Size of the solution vector
nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages
solver_id	Solver type
verbose	Report more information of the full process
er_throw	Instead of showing error messages on screen, throw them out using std::exception

3.6.3.7 MinimizePreconditioned()

```
void LCG_CUDA_Solver::MinimizePreconditioned (
            cublasHandle_t cub_handle,
            cusparseHandle_t cus_handle,
            lcg_float * x,
            lcg_float * b,
            const int n_size,
            const int nz_size,
            lcg_solver_enum solver_id = LCG_CG,
            bool verbose = true,
            bool er_throw = false )
```

Run the preconditioned minimizing process.

Parameters

cub_handle	Handler of the CuBLAS library
cus_handle	Handler of the CuSparse library
x	Pointer of the solution vector
b	Pointer of the targeting vector
n_size	Size of the solution vector
nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages
solver_id	Solver type
verbose	Report more information of the full process
er_throw	Instead of showing error messages on screen, throw them out using std::exception

3.6.3.8 MxProduct()

```
virtual void LCG_CUDA_Solver::MxProduct (
            cublasHandle_t cub_handle,
            cusparseHandle_t cus_handle,
            cusparseDnVecDescr_t x,
            cusparseDnVecDescr_t prod_Mx,
            const int n_size,
            const int nz_size )  [pure virtual]
```

Virtual function of the product of M^-1∗x.

Parameters

cub_handle	Handler of the CuBLAS library
cus_handle	Handler of the CuSparse library
x[in]	Pointer of the multiplier
prod_Mx[out]	Pointer of the product
n_size	Size of the solution
nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages

3.6.3.9 Progress()

```
virtual int LCG_CUDA_Solver::Progress (
            const lcg_float * m,
            const lcg_float converge,
            const lcg_para * param,
            const int n_size,
            const int nz_size,
            const int k )  [virtual]
```

Virtual function of the process monitoring.

Parameters

m	Pointer of the current solution
converge	Current value of the convergence
param	Pointer of the parameters used in the algorithms
n_size	Size of the solution
nz_size	Non-zero size of the sparse kernel matrix. This parameter is not need by the algorithm. It is passed for CUDA usages
k	Current iteration times

Returns

int Status of the process

3.6.3.10 set_lcg_parameter()

```
void LCG_CUDA_Solver::set_lcg_parameter (
            const lcg_para & in_param )
```

Set the parameters of the algorithms.

Parameters

in_param	the input parameters

3.6.3.11 set_report_interval()

```
void LCG_CUDA_Solver::set_report_interval (
            unsigned int inter )
```

Set the interval to run the process monitoring function.

Parameters

inter	the interval

3.6.3.12 silent()

```
void LCG_CUDA_Solver::silent ( )
```

Do not report any processes.

3.6.4 Field Documentation

3.6.4.1 inter_

```
unsigned int LCG_CUDA_Solver::inter_  [protected]
```

3.6.4.2 param_

```
lcg_para LCG_CUDA_Solver::param_  [protected]
```

3.6.4.3 silent_

```
bool LCG_CUDA_Solver::silent_  [protected]
```

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

- solver_cuda.h

3.7 LCG_EIGEN_Solver Class Reference

Linear conjugate gradient solver class.

```
#include <solver_eigen.h>
```

Public Member Functions

- LCG_EIGEN_Solver ()
- virtual ∼LCG_EIGEN_Solver ()
- virtual void AxProduct (const Eigen::VectorXd &x, Eigen::VectorXd &prod_Ax)=0

    Virtual function of the product of A∗x.
- virtual void MxProduct (const Eigen::VectorXd &x, Eigen::VectorXd &prod_Mx)=0

    Virtual function of the product of M^-1∗x.
- virtual int Progress (const Eigen::VectorXd ∗m, const lcg_float converge, const lcg_para ∗param, const int k)

    Virtual function of the process monitoring.
- void silent ()

    Do not report any processes.
- void set_report_interval (unsigned int inter)

    Set the interval to run the process monitoring function.
- void set_lcg_parameter (const lcg_para &in_param)

    Set the parameters of the algorithms.
- void Minimize (Eigen::VectorXd &m, const Eigen::VectorXd &b, lcg_solver_enum solver_id=LCG_CG, bool verbose=true, bool er_throw=false)

    Run the minimizing process.
- void MinimizePreconditioned (Eigen::VectorXd &m, const Eigen::VectorXd &b, lcg_solver_enum solver_↩ id=LCG_PCG, bool verbose=true, bool er_throw=false)

    Run the preconitioned minimizing process.
- void MinimizeConstrained (Eigen::VectorXd &m, const Eigen::VectorXd &B, const Eigen::VectorXd &low, const Eigen::VectorXd &hig, lcg_solver_enum solver_id=LCG_PG, bool verbose=true, bool er_throw=false)

    Run the constrained minimizing process.

Static Public Member Functions

- static void _AxProduct (void ∗instance, const Eigen::VectorXd &x, Eigen::VectorXd &prod_Ax)

    Interface of the virtual function of the product of A∗x.
- static void _MxProduct (void ∗instance, const Eigen::VectorXd &x, Eigen::VectorXd &prod_Mx)

    Interface of the virtual function of the product of M^-1∗x.
- static int _Progress (void ∗instance, const Eigen::VectorXd ∗m, const lcg_float converge, const lcg_para ∗param, const int k)

    Interface of the virtual function of the process monitoring.

Protected Attributes

- lcg_para param_
- unsigned int inter_
- bool silent_

3.7.1 Detailed Description

Linear conjugate gradient solver class.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 LCG_EIGEN_Solver()

```
LCG_EIGEN_Solver::LCG_EIGEN_Solver ( )
```

3.7.2.2 ∼LCG_EIGEN_Solver()

```
virtual LCG_EIGEN_Solver::∼LCG_EIGEN_Solver ( )  [inline], [virtual]
```

3.7.3 Member Function Documentation

3.7.3.1 _AxProduct()

```
static void LCG_EIGEN_Solver::_AxProduct (
             void * instance,
             const Eigen::VectorXd & x,
             Eigen::VectorXd & prod_Ax )  [inline], [static]
```

Interface of the virtual function of the product of A∗x.

Parameters

instance	User data sent to identify the function address
x[in]	Pointer of the multiplier
prod_Ax[out]	Pointer of the product

3.7.3.2 _MxProduct()

```
static void LCG_EIGEN_Solver::_MxProduct (
             void * instance,
             const Eigen::VectorXd & x,
             Eigen::VectorXd & prod_Mx )  [inline], [static]
```

Interface of the virtual function of the product of M^-1∗x.

Parameters

instance	User data sent to identify the function address
x[in]	Pointer of the multiplier
prod_Mx[out]	Pointer of the product

3.7.3.3 _Progress()

```
static int LCG_EIGEN_Solver::_Progress (
            void * instance,
            const Eigen::VectorXd * m,
            const lcg_float converge,
            const lcg_para * param,
            const int k )  [inline], [static]
```

Interface of the virtual function of the process monitoring.

Parameters

instance	User data sent to identify the function address
m	Pointer of the current solution
converge	Current value of the convergence
param	Pointer of the parameters used in the algorithms
k	Current iteration times

Returns

int Status of the process

3.7.3.4 AxProduct()

```
virtual void LCG_EIGEN_Solver::AxProduct (
            const Eigen::VectorXd & x,
            Eigen::VectorXd & prod_Ax )  [pure virtual]
```

Virtual function of the product of A*x.

Parameters

x[in]	Pointer of the multiplier
prod_Ax[out]	Pointer of the product

3.7.3.5 Minimize()

```
void LCG_EIGEN_Solver::Minimize (
            Eigen::VectorXd & m,
            const Eigen::VectorXd & b,
            lcg_solver_enum solver_id = LCG_CG,
```

```
              bool verbose = true,
              bool er_throw = false )
```

Run the minimizing process.

Parameters

m	Pointer of the solution vector
b	Pointer of the targeting vector
solver_id	Solver type
verbose	Report more information of the full process
er_throw	Instead of showing error messages on screen, throw them out using std::exception

3.7.3.6 MinimizeConstrained()

```
void LCG_EIGEN_Solver::MinimizeConstrained (
              Eigen::VectorXd & m,
              const Eigen::VectorXd & B,
              const Eigen::VectorXd & low,
              const Eigen::VectorXd & hig,
              lcg_solver_enum solver_id = LCG_PG,
              bool verbose = true,
              bool er_throw = false )
```

Run the constrained minimizing process.

Parameters

m	Pointer of the solution vector
b	Pointer of the targeting vector
low	Lower bound of the solution vector
hig	Higher bound of the solution vector
solver_id	Solver type
verbose	Report more information of the full process
er_throw	Instead of showing error messages on screen, throw them out using std::exception

3.7.3.7 MinimizePreconditioned()

```
void LCG_EIGEN_Solver::MinimizePreconditioned (
              Eigen::VectorXd & m,
              const Eigen::VectorXd & b,
              lcg_solver_enum solver_id = LCG_PCG,
              bool verbose = true,
              bool er_throw = false )
```

Run the preconitioned minimizing process.

Parameters

m	Pointer of the solution vector
b	Pointer of the targeting vector
solver←↩ _id	Solver type
verbose	Report more information of the full process
er_throw	Instead of showing error messages on screen, throw them out using std::exception

3.7.3.8 MxProduct()

```
virtual void LCG_EIGEN_Solver::MxProduct (
             const Eigen::VectorXd & x,
             Eigen::VectorXd & prod_Mx ) [pure virtual]
```

Virtual function of the product of M^-1∗x.

Parameters

x[in]	Pointer of the multiplier
prod_Mx[out]	Pointer of the product

3.7.3.9 Progress()

```
int LCG_EIGEN_Solver::Progress (
             const Eigen::VectorXd * m,
             const lcg_float converge,
             const lcg_para * param,
             const int k ) [virtual]
```

Virtual function of the process monitoring.

Parameters

m	Pointer of the current solution
converge	Current value of the convergence
param	Pointer of the parameters used in the algorithms
k	Current iteration times

Returns

int Status of the process

3.7.3.10 set_lcg_parameter()

```
void LCG_EIGEN_Solver::set_lcg_parameter (
            const lcg_para & in_param )
```

Set the parameters of the algorithms.

Parameters

in_param	the input parameters

3.7.3.11 set_report_interval()

```
void LCG_EIGEN_Solver::set_report_interval (
            unsigned int inter )
```

Set the interval to run the process monitoring function.

Parameters

inter	the interval

3.7.3.12 silent()

```
void LCG_EIGEN_Solver::silent ( )
```

Do not report any processes.

3.7.4 Field Documentation

3.7.4.1 inter_

```
unsigned int LCG_EIGEN_Solver::inter_  [protected]
```

3.7.4.2 param_

```
lcg_para LCG_EIGEN_Solver::param_  [protected]
```

3.7.4.3 silent_

```
bool LCG_EIGEN_Solver::silent_  [protected]
```

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

- solver_eigen.h
- solver_eigen.cpp

3.8 lcg_para Struct Reference

Parameters of the conjugate gradient methods.

```
#include <util.h>
```

Data Fields

- int max_iterations
- lcg_float epsilon
- int abs_diff
- lcg_float restart_epsilon
- lcg_float step
- lcg_float sigma
- lcg_float beta
- int maxi_m

3.8.1 Detailed Description

Parameters of the conjugate gradient methods.

3.8.2 Field Documentation

3.8.2.1 abs_diff

```
int lcg_para::abs_diff
```

Whether to use absolute mean differences (AMD) between |Ax - B| to evaluate the process. The default value is false which means the gradient based evaluating method is used. The AMD based method will be used if this variable is set to true. This parameter is only applied to the non-constrained methods.

3.8.2.2 beta

```
lcg_float lcg_para::beta
```

descending ratio for conducting the non-monotonic linear search. The range of this variable is (0, 1). The default is given as 0.9

3.8.2.3 epsilon

```
lcg_float lcg_para::epsilon
```

Epsilon for convergence test. This parameter determines the accuracy with which the solution is to be found. A minimization terminates when ||g||/max(||x||, 1.0) <= epsilon or sqrt(||g||)/N <= epsilon for the lcg_solver() function, where ||.|| denotes the Euclidean (L2) norm. The default value of epsilon is 1e-6.

3.8.2.4 max_iterations

```
int lcg_para::max_iterations
```

Maximal iteration times. The process will continue till the convergance is met if this option is set to zero (default).

3.8.2.5 maxi_m

```
int lcg_para::maxi_m
```

The maximal record times of the objective values for the SPG method. The method use the objective values from the most recent maxi_m times to preform the non-monotonic linear search. The default value is 10.

3.8.2.6 restart_epsilon

```
lcg_float lcg_para::restart_epsilon
```

Restart epsilon for the LCG_BICGSTAB2 algorithm. The default value is 1e-6

3.8.2.7 sigma

```
lcg_float lcg_para::sigma
```

multiplier for updating solutions with the spectral projected gradient method. The range of this variable is (0, 1). The default is given as 0.95

3.8.2.8 step

```
lcg_float lcg_para::step
```

Initial step length for the project gradient method. The default is 1.0

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

- util.h

3.9 LCG_Solver Class Reference

Linear conjugate gradient solver class.

```
#include <solver.h>
```

Public Member Functions

- LCG_Solver ()
- virtual ~LCG_Solver ()
- virtual void AxProduct (const lcg_float *a, lcg_float *b, const int num)=0

    *Virtual function of the product of A*x.*
- virtual void MxProduct (const lcg_float *a, lcg_float *b, const int num)=0

    *Virtual function of the product of M^-1*x.*
- virtual int Progress (const lcg_float *m, const lcg_float converge, const lcg_para *param, const int n_size, const int k)

    Virtual function of the process monitoring.
- void silent ()

    Do not report any processes.
- void set_report_interval (unsigned int inter)

    Set the interval to run the process monitoring function.
- void set_lcg_parameter (const lcg_para &in_param)

    Set the parameters of the algorithms.
- void Minimize (lcg_float *m, const lcg_float *b, int x_size, lcg_solver_enum solver_id=LCG_CG, bool verbose=true, bool er_throw=false)

    Run the minimizing process.
- void MinimizePreconditioned (lcg_float *m, const lcg_float *b, int x_size, lcg_solver_enum solver_id=LCG_PCG, bool verbose=true, bool er_throw=false)

    Run the preconitioned minimizing process.
- void MinimizeConstrained (lcg_float *m, const lcg_float *b, const lcg_float *low, const lcg_float *hig, int x_size, lcg_solver_enum solver_id=LCG_PG, bool verbose=true, bool er_throw=false)

    Run the constrained minimizing process.

Static Public Member Functions

- static void _AxProduct (void *instance, const lcg_float *a, lcg_float *b, const int num)

    *Interface of the virtual function of the product of A*x.*
- static void _MxProduct (void *instance, const lcg_float *a, lcg_float *b, const int num)

    *Interface of the virtual function of the product of M^-1*x.*
- static int _Progress (void *instance, const lcg_float *m, const lcg_float converge, const lcg_para *param, const int n_size, const int k)

    Interface of the virtual function of the process monitoring.

Protected Attributes

- lcg_para param_
- unsigned int inter_
- bool silent_

3.9.1 Detailed Description

Linear conjugate gradient solver class.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 LCG_Solver()

```
LCG_Solver::LCG_Solver ( )
```

3.9.2.2 ∼LCG_Solver()

```
virtual LCG_Solver::∼LCG_Solver ( ) [inline], [virtual]
```

3.9.3 Member Function Documentation

3.9.3.1 _AxProduct()

```
static void LCG_Solver::_AxProduct (
            void ∗ instance,
            const lcg_float ∗ a,
            lcg_float ∗ b,
            const int num ) [inline], [static]
```

Interface of the virtual function of the product of A∗x.

Parameters

instance	User data sent to identify the function address
a[in]	Pointer of the multiplier
b[out]	Pointer of the product
num	Size of the array

3.9.3.2 _MxProduct()

```
static void LCG_Solver::_MxProduct (
            void ∗ instance,
```

```
            const lcg_float * a,
            lcg_float * b,
            const int num )  [inline], [static]
```

Interface of the virtual function of the product of M^-1∗x.

Parameters

instance	User data sent to identify the function address
a[in]	Pointer of the multiplier
b[out]	Pointer of the product
num	Size of the array

3.9.3.3 _Progress()

```
static int LCG_Solver::_Progress (
            void * instance,
            const lcg_float * m,
            const lcg_float converge,
            const lcg_para * param,
            const int n_size,
            const int k )  [inline], [static]
```

Interface of the virtual function of the process monitoring.

Parameters

instance	User data sent to identify the function address
m	Pointer of the current solution
converge	Current value of the convergence
param	Pointer of the parameters used in the algorithms
n_size	Size of the solution
k	Current iteration times

Returns

int Status of the process

3.9.3.4 AxProduct()

```
virtual void LCG_Solver::AxProduct (
            const lcg_float * a,
            lcg_float * b,
            const int num )  [pure virtual]
```

Virtual function of the product of A∗x.

Parameters

a[in]	Pointer of the multiplier
b[out]	Pointer of the product
num	Size of the array

3.9.3.5 Minimize()

```
void LCG_Solver::Minimize (
            lcg_float * m,
            const lcg_float * b,
            int x_size,
            lcg_solver_enum solver_id = LCG_CG,
            bool verbose = true,
            bool er_throw = false )
```

Run the minimizing process.

Parameters

m	Pointer of the solution vector
b	Pointer of the targeting vector
x_size	Size of the solution vector
solver←_id	Solver type
verbose	Report more information of the full process
er_throw	Instead of showing error messages on screen, throw them out using std::exception

3.9.3.6 MinimizeConstrained()

```
void LCG_Solver::MinimizeConstrained (
            lcg_float * m,
            const lcg_float * b,
            const lcg_float * low,
            const lcg_float * hig,
            int x_size,
            lcg_solver_enum solver_id = LCG_PG,
            bool verbose = true,
            bool er_throw = false )
```

Run the constrained minimizing process.

Parameters

m	Pointer of the solution vector
b	Pointer of the targeting vector

Parameters

low	Lower bound of the solution vector
hig	Higher bound of the solution vector
x_size	Size of the solution vector
solver←↩ _id	Solver type
verbose	Report more information of the full process
er_throw	Instead of showing error messages on screen, throw them out using std::exception

3.9.3.7 MinimizePreconditioned()

```
void LCG_Solver::MinimizePreconditioned (
            lcg_float * m,
            const lcg_float * b,
            int x_size,
            lcg_solver_enum solver_id = LCG_PCG,
            bool verbose = true,
            bool er_throw = false )
```

Run the preconitioned minimizing process.

Parameters

m	Pointer of the solution vector
b	Pointer of the targeting vector
x_size	Size of the solution vector
solver←↩ _id	Solver type
verbose	Report more information of the full process
er_throw	Instead of showing error messages on screen, throw them out using std::exception

3.9.3.8 MxProduct()

```
virtual void LCG_Solver::MxProduct (
            const lcg_float * a,
            lcg_float * b,
            const int num ) [pure virtual]
```

Virtual function of the product of M^-1∗x.

Parameters

a[in]	Pointer of the multiplier
b[out]	Pointer of the product
num	Size of the array

3.9.3.9 Progress()

```
int LCG_Solver::Progress (
            const lcg_float * m,
            const lcg_float converge,
            const lcg_para * param,
            const int n_size,
            const int k ) [virtual]
```

Virtual function of the process monitoring.

Parameters

m	Pointer of the current solution
converge	Current value of the convergence
param	Pointer of the parameters used in the algorithms
n_size	Size of the solution
k	Current iteration times

Returns

int Status of the process

3.9.3.10 set_lcg_parameter()

```
void LCG_Solver::set_lcg_parameter (
            const lcg_para & in_param )
```

Set the parameters of the algorithms.

Parameters

in_param	the input parameters

3.9.3.11 set_report_interval()

```
void LCG_Solver::set_report_interval (
            unsigned int inter )
```

Set the interval to run the process monitoring function.

Parameters

inter	the interval

3.9.3.12 silent()

```
void LCG_Solver::silent ( )
```

Do not report any processes.

3.9.4 Field Documentation

3.9.4.1 inter_

```
unsigned int LCG_Solver::inter_ [protected]
```

3.9.4.2 param_

```
lcg_para LCG_Solver::param_ [protected]
```

3.9.4.3 silent_

```
bool LCG_Solver::silent_ [protected]
```

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

- solver.h
- solver.cpp

Chapter 4

File Documentation

4.1 algebra.cpp File Reference

```
#include "ctime"
#include "random"
#include "algebra.h"
#include "omp.h"
```

Functions

- lcg_float lcg_abs (lcg_float a)

  Return absolute value.
- lcg_float lcg_max (lcg_float a, lcg_float b)

  Return the bigger value.
- lcg_float lcg_min (lcg_float a, lcg_float b)

  Return the smaller value.
- lcg_float lcg_set2box (lcg_float low, lcg_float hig, lcg_float a, bool low_bound, bool hig_bound)

  Set the input value within a box constraint.
- lcg_float ∗ lcg_malloc (int n)

  Locate memory for a lcg_float pointer type.
- lcg_float ∗∗ lcg_malloc (int m, int n)

  Locate memory for a lcg_float second pointer type.
- void lcg_free (lcg_float ∗x)

  Destroy memory used by the lcg_float type array.
- void lcg_free (lcg_float ∗∗x, int m)

  Destroy memory used by the 2D lcg_float type array.
- void lcg_vecset (lcg_float ∗a, lcg_float b, int size)

  set a vector's value
- void lcg_vecset (lcg_float ∗∗a, lcg_float b, int m, int n)

  set a 2d vector's value
- void lcg_vecrnd (lcg_float ∗a, lcg_float l, lcg_float h, int size)

  set a vector using random values
- void lcg_vecrnd (lcg_float ∗∗a, lcg_float l, lcg_float h, int m, int n)

  set a 2D vector using random values

- double lcg_squaredl2norm (lcg_float ∗a, int n)

    calculate the squared L2 norm of the input vector
- void lcg_dot (lcg_float &ret, const lcg_float ∗a, const lcg_float ∗b, int size)

    calculate dot product of two real vectors
- void lcg_matvec (lcg_float ∗∗A, const lcg_float ∗x, lcg_float ∗Ax, int m_size, int n_size, lcg_matrix_e layout)

    calculate product of a real matrix and a vector
- void lcg_matvec_coo (const int ∗row, const int ∗col, const lcg_float ∗Mat, const lcg_float ∗V, lcg_float ∗p, int M, int N, int nz_size, bool pre_position)

    Calculate the product of a sparse matrix multipled by a vector. The matrix is stored in the COO format.

4.1.1 Function Documentation

4.1.1.1 lcg_abs()

```
lcg_float lcg_abs (
            lcg_float a )
```

Return absolute value.

Parameters

in	*a*	input value

Returns

The absolute value

4.1.1.2 lcg_dot()

```
void lcg_dot (
            lcg_float & ret,
            const lcg_float * a,
            const lcg_float * b,
            int size )
```

calculate dot product of two real vectors

Parameters

in	*a*	pointer of the vector a
in	*b*	pointer of the vector b
in	*size*	size of the vector

Returns

dot product

4.1.1.3 lcg_free() [1/2]

```
void lcg_free (
            lcg_float ** x,
            int m )
```

Destroy memory used by the 2D lcg_float type array.

Parameters

x	Pointer of the array.

4.1.1.4 lcg_free() [2/2]

```
void lcg_free (
            lcg_float * x )
```

Destroy memory used by the lcg_float type array.

Parameters

x	Pointer of the array.

4.1.1.5 lcg_malloc() [1/2]

```
lcg_float** lcg_malloc (
            int m,
            int n )
```

Locate memory for a lcg_float second pointer type.

Parameters

in	*n*	Size of the lcg_float array.

Returns

Pointer of the array's location.

4.1.1.6 lcg_malloc() [2/2]

```
lcg_float* lcg_malloc (
            int n )
```

Locate memory for a lcg_float pointer type.

Parameters

in	*n*	Size of the lcg_float array.

Returns

Pointer of the array's location.

4.1.1.7 lcg_matvec()

```
void lcg_matvec (
            lcg_float ** A,
            const lcg_float * x,
            lcg_float * Ax,
            int m_size,
            int n_size,
            lcg_matrix_e layout = MatNormal )
```

calculate product of a real matrix and a vector

Different configurations: layout=Normal -> A layout=Transpose -> A^T

Parameters

	A	matrix A
in	*x*	vector x
	Ax	product of Ax
in	*m_size*	row size of A
in	*n_size*	column size of A
in	*layout*	layout of A used for multiplication. Must be Normal or Transpose

4.1.1.8 lcg_matvec_coo()

```
void lcg_matvec_coo (
            const int * row,
```

```
            const int * col,
            const lcg_float * Mat,
            const lcg_float * V,
            lcg_float * p,
            int M,
            int N,
            int nz_size,
            bool pre_position = false )
```

Calculate the product of a sparse matrix multipled by a vector. The matrix is stored in the COO format.

Parameters

row	Row index of the input sparse matrix.
col	Column index of the input sparse matrix.
Mat	Non-zero values of the input sparse matrix.
V	Multipler vector
p	Output prodcut
M	Row number of the sparse matrix
N	Column number of the sparse matrix
nz_size	Non-zero size of the matrix
pre_position	If ture, the multipler is seen as a row vector. Otherwise, it is treated as a column vector.

4.1.1.9 lcg_max()

```
lcg_float lcg_max (
            lcg_float a,
            lcg_float b )
```

Return the bigger value.

Parameters

in	*a*	input value
in	*b*	input value

Returns

The bigger value

4.1.1.10 lcg_min()

```
lcg_float lcg_min (
            lcg_float a,
            lcg_float b )
```

Return the smaller value.

Parameters

in	*a*	input value
in	*b*	input value

Returns

The smaller value

4.1.1.11 lcg_set2box()

```
lcg_float lcg_set2box (
            lcg_float low,
            lcg_float hig,
            lcg_float a,
            bool low_bound = true,
            bool hig_bound = true )
```

Set the input value within a box constraint.

Parameters

a	low boundary
b	high boundary
in	input value
low_bound	Whether to include the low boundary value
hig_bound	Whether to include the high boundary value

Returns

box constrained value

4.1.1.12 lcg_squaredl2norm()

```
double lcg_squaredl2norm (
            lcg_float * a,
            int n )
```

calculate the squared L2 norm of the input vector

Parameters

a	pointer of the vector
n	size of the vector

Returns

double L2 norm

4.1.1.13 lcg_vecrnd() [1/2]

```
void lcg_vecrnd (
            lcg_float ** a,
            lcg_float l,
            lcg_float h,
            int m,
            int n )
```

set a 2D vector using random values

Parameters

	a	pointer of the vector
in	*l*	the lower bound of random values
in	*h*	the higher bound of random values
in	*m*	row size of the vector
in	*n*	column size of the vector

4.1.1.14 lcg_vecrnd() [2/2]

```
void lcg_vecrnd (
            lcg_float * a,
            lcg_float l,
            lcg_float h,
            int size )
```

set a vector using random values

Parameters

	a	pointer of the vector
in	*l*	the lower bound of random values
in	*h*	the higher bound of random values
in	*size*	size of the vector

4.1.1.15 lcg_vecset() [1/2]

```
void lcg_vecset (
            lcg_float ** a,
```

```
            lcg_float b,
            int m,
            int n )
```

set a 2d vector's value

Parameters

	a	pointer of the matrix
in	*b*	initial value
in	*m*	row size of the matrix
in	*n*	column size of the matrix

4.1.1.16 lcg_vecset() [2/2]

```
void lcg_vecset (
            lcg_float * a,
            lcg_float b,
            int size )
```

set a vector's value

Parameters

	a	pointer of the vector
in	*b*	initial value
in	*size*	vector size

4.2 algebra.h File Reference

`#include "config.h"`

Typedefs

- typedef double lcg_float

  A simple definition of the float type we use here. Easy to change in the future. Right now it is just an alias of double.

Enumerations

- enum lcg_matrix_e { MatNormal, MatTranspose }

  Matrix layouts.
- enum clcg_complex_e { NonConjugate, Conjugate }

  Conjugate types for a complex number.

Functions

- lcg_float lcg_abs (lcg_float a)

    Return absolute value.
- lcg_float lcg_max (lcg_float a, lcg_float b)

    Return the bigger value.
- lcg_float lcg_min (lcg_float a, lcg_float b)

    Return the smaller value.
- lcg_float lcg_set2box (lcg_float low, lcg_float hig, lcg_float a, bool low_bound=true, bool hig_bound=true)

    Set the input value within a box constraint.
- lcg_float ∗ lcg_malloc (int n)

    Locate memory for a lcg_float pointer type.
- lcg_float ∗∗ lcg_malloc (int m, int n)

    Locate memory for a lcg_float second pointer type.
- void lcg_free (lcg_float ∗x)

    Destroy memory used by the lcg_float type array.
- void lcg_free (lcg_float ∗∗x, int m)

    Destroy memory used by the 2D lcg_float type array.
- void lcg_vecset (lcg_float ∗a, lcg_float b, int size)

    set a vector's value
- void lcg_vecset (lcg_float ∗∗a, lcg_float b, int m, int n)

    set a 2d vector's value
- void lcg_vecrnd (lcg_float ∗a, lcg_float l, lcg_float h, int size)

    set a vector using random values
- void lcg_vecrnd (lcg_float ∗∗a, lcg_float l, lcg_float h, int m, int n)

    set a 2D vector using random values
- double lcg_squaredl2norm (lcg_float ∗a, int n)

    calculate the squared L2 norm of the input vector
- void lcg_dot (lcg_float &ret, const lcg_float ∗a, const lcg_float ∗b, int size)

    calculate dot product of two real vectors
- void lcg_matvec (lcg_float ∗∗A, const lcg_float ∗x, lcg_float ∗Ax, int m_size, int n_size, lcg_matrix_e layout=MatNormal)

    calculate product of a real matrix and a vector
- void lcg_matvec_coo (const int ∗row, const int ∗col, const lcg_float ∗Mat, const lcg_float ∗V, lcg_float ∗p, int M, int N, int nz_size, bool pre_position=false)

    Calculate the product of a sparse matrix multipled by a vector. The matrix is stored in the COO format.

4.2.1 Typedef Documentation

4.2.1.1 lcg_float

```
typedef double lcg_float
```

A simple definition of the float type we use here. Easy to change in the future. Right now it is just an alias of double.

4.2.2 Enumeration Type Documentation

4.2.2.1 clcg_complex_e

```
enum clcg_complex_e
```

Conjugate types for a complex number.

Enumerator

NonConjugate	
Conjugate	

4.2.2.2 lcg_matrix_e

```
enum lcg_matrix_e
```

Matrix layouts.

Enumerator

MatNormal	
MatTranspose	

4.2.3 Function Documentation

4.2.3.1 lcg_abs()

```
lcg_float lcg_abs (
            lcg_float a )
```

Return absolute value.

Parameters

in	*a*	input value

Returns

The absolute value

4.2.3.2 lcg_dot()

```
void lcg_dot (
            lcg_float & ret,
            const lcg_float * a,
            const lcg_float * b,
            int size )
```

calculate dot product of two real vectors

Parameters

in	*a*	pointer of the vector a
in	*b*	pointer of the vector b
in	*size*	size of the vector

Returns

dot product

4.2.3.3 lcg_free() [1/2]

```
void lcg_free (
            lcg_float ** x,
            int m )
```

Destroy memory used by the 2D lcg_float type array.

Parameters

x	Pointer of the array.

4.2.3.4 lcg_free() [2/2]

```
void lcg_free (
            lcg_float * x )
```

Destroy memory used by the lcg_float type array.

Parameters

x	Pointer of the array.

4.2.3.5 lcg_malloc() [1/2]

```
lcg_float** lcg_malloc (
            int m,
            int n )
```

Locate memory for a lcg_float second pointer type.

Parameters

in	*n*	Size of the lcg_float array.

Returns

Pointer of the array's location.

4.2.3.6 lcg_malloc() [2/2]

```
lcg_float* lcg_malloc (
            int n )
```

Locate memory for a lcg_float pointer type.

Parameters

in	*n*	Size of the lcg_float array.

Returns

Pointer of the array's location.

4.2.3.7 lcg_matvec()

```
void lcg_matvec (
            lcg_float ** A,
            const lcg_float * x,
            lcg_float * Ax,
```

```
int m_size,
int n_size,
lcg_matrix_e layout = MatNormal )
```

calculate product of a real matrix and a vector

Different configurations: layout=Normal -> A layout=Transpose -> A^T

Parameters

	A	matrix A
in	*x*	vector x
	Ax	product of Ax
in	*m_size*	row size of A
in	*n_size*	column size of A
in	*layout*	layout of A used for multiplication. Must be Normal or Transpose

4.2.3.8 lcg_matvec_coo()

```
void lcg_matvec_coo (
        const int * row,
        const int * col,
        const lcg_float * Mat,
        const lcg_float * V,
        lcg_float * p,
        int M,
        int N,
        int nz_size,
        bool pre_position = false )
```

Calculate the product of a sparse matrix multipled by a vector. The matrix is stored in the COO format.

Parameters

row	Row index of the input sparse matrix.
col	Column index of the input sparse matrix.
Mat	Non-zero values of the input sparse matrix.
V	Multipler vector
p	Output prodcut
M	Row number of the sparse matrix
N	Column number of the sparse matrix
nz_size	Non-zero size of the matrix
pre_position	If ture, the multipler is seen as a row vector. Otherwise, it is treated as a column vector.

4.2.3.9 lcg_max()

```
lcg_float lcg_max (
```

```
            lcg_float a,
            lcg_float b )
```

Return the bigger value.

Parameters

in	*a*	input value
in	*b*	input value

Returns

The bigger value

4.2.3.10 lcg_min()

```
lcg_float lcg_min (
            lcg_float a,
            lcg_float b )
```

Return the smaller value.

Parameters

in	*a*	input value
in	*b*	input value

Returns

The smaller value

4.2.3.11 lcg_set2box()

```
lcg_float lcg_set2box (
            lcg_float low,
            lcg_float hig,
            lcg_float a,
            bool low_bound = true,
            bool hig_bound = true )
```

Set the input value within a box constraint.

Parameters

a	low boundary
b	high boundary
in	input value
low_bound	Whether to include the low boundary value
hig_bound	Whether to include the high boundary value

Returns

box constrained value

4.2.3.12 lcg_squaredl2norm()

```
double lcg_squaredl2norm (
            lcg_float * a,
            int n )
```

calculate the squared L2 norm of the input vector

Parameters

a	pointer of the vector
n	size of the vector

Returns

double L2 norm

4.2.3.13 lcg_vecrnd() [1/2]

```
void lcg_vecrnd (
            lcg_float ** a,
            lcg_float l,
            lcg_float h,
            int m,
            int n )
```

set a 2D vector using random values

Parameters

	a	pointer of the vector
in	*l*	the lower bound of random values
in	*h*	the higher bound of random values
in	*m*	row size of the vector
in	*n*	column size of the vector

4.2.3.14 lcg_vecrnd() [2/2]

```
void lcg_vecrnd (
            lcg_float * a,
```

```
            lcg_float l,
            lcg_float h,
            int size )
```

set a vector using random values

Parameters

	a	pointer of the vector
in	*l*	the lower bound of random values
in	*h*	the higher bound of random values
in	*size*	size of the vector

4.2.3.15 lcg_vecset() [1/2]

```
void lcg_vecset (
            lcg_float ** a,
            lcg_float b,
            int m,
            int n )
```

set a 2d vector's value

Parameters

	a	pointer of the matrix
in	*b*	initial value
in	*m*	row size of the matrix
in	*n*	column size of the matrix

4.2.3.16 lcg_vecset() [2/2]

```
void lcg_vecset (
            lcg_float * a,
            lcg_float b,
            int size )
```

set a vector's value

Parameters

	a	pointer of the vector
in	*b*	initial value
in	*size*	vector size

4.3 algebra_cuda.h File Reference

```
#include "algebra.h"
#include <cuda_runtime.h>
```

Functions

- void lcg_set2box_cuda (const lcg_float *low, const lcg_float *hig, lcg_float *a, int n, bool low_bound=true, bool hig_bound=true)

    Set the input value within a box constraint.
- void lcg_smDcsr_get_diagonal (const int *A_ptr, const int *A_col, const lcg_float *A_val, const int A_len, lcg_float *A_diag, int bk_size=1024)

    Extract diagonal elements from a square CUDA sparse matrix that is formatted in the CSR format.
- void lcg_vecMvecD_element_wise (const lcg_float *a, const lcg_float *b, lcg_float *c, int n, int bk_size=1024)

    Element-wise muplication between two CUDA arries.
- void lcg_vecDvecD_element_wise (const lcg_float *a, const lcg_float *b, lcg_float *c, int n, int bk_size=1024)

    Element-wise division between two CUDA arries.

4.3.1 Function Documentation

4.3.1.1 lcg_set2box_cuda()

```
void lcg_set2box_cuda (
            const lcg_float * low,
            const lcg_float * hig,
            lcg_float * a,
            int n,
            bool low_bound = true,
            bool hig_bound = true )
```

Set the input value within a box constraint.

Parameters

a	low boundary
b	high boundary
in	input value
low_bound	Whether to include the low boundary value
hig_bound	Whether to include the high boundary value

Returns

box constrained value

4.3.1.2 lcg_smDcsr_get_diagonal()

```
void lcg_smDcsr_get_diagonal (
            const int * A_ptr,
            const int * A_col,
            const lcg_float * A_val,
            const int A_len,
            lcg_float * A_diag,
            int bk_size = 1024 )
```

Extract diagonal elements from a square CUDA sparse matrix that is formatted in the CSR format.

Note

This is a device side function. All memories must be allocated on the GPU device.

Parameters

in	*A_ptr*	Row index pointer
in	*A_col*	Column index
in	*A_val*	Non-zero values of the matrix
in	*A_len*	Dimension of the matrix
	A_diag	Output digonal elements
in	*bk_size*	Default CUDA block size.

4.3.1.3 lcg_vecDvecD_element_wise()

```
void lcg_vecDvecD_element_wise (
            const lcg_float * a,
            const lcg_float * b,
            lcg_float * c,
            int n,
            int bk_size = 1024 )
```

Element-wise division between two CUDA arries.

Note

This is a device side function. All memories must be allocated on the GPU device.

Parameters

in	*a*	Pointer of the input array
in	*b*	Pointer of the input array
	c	Pointer of the output array
in	*n*	Length of the arraies
in	*bk_size*	Default CUDA block size.

4.3.1.4 lcg_vecMvecD_element_wise()

```
void lcg_vecMvecD_element_wise (
            const lcg_float * a,
            const lcg_float * b,
            lcg_float * c,
            int n,
            int bk_size = 1024 )
```

Element-wise muplication between two CUDA arries.

Note

This is a device side function. All memories must be allocated on the GPU device.

Parameters

in	*a*	Pointer of the input array
in	*b*	Pointer of the input array
	c	Pointer of the output array
in	*n*	Length of the arraies
in	*bk_size*	Default CUDA block size.

4.4 algebra_eigen.cpp File Reference

```
#include "algebra_eigen.h"
```

Functions

- void lcg_set2box_eigen (const Eigen::VectorXd &low, const Eigen::VectorXd &hig, Eigen::VectorXd m)

    Set the input value within a box constraint.

4.4.1 Function Documentation

4.4.1.1 lcg_set2box_eigen()

```
void lcg_set2box_eigen (
            const Eigen::VectorXd & low,
            const Eigen::VectorXd & hig,
            Eigen::VectorXd m )
```

Set the input value within a box constraint.

Parameters

low_bound	Whether to include the low boundary value
hig_bound	Whether to include the high boundary value
m	Returned values

4.5 algebra_eigen.h File Reference

```
#include "algebra.h"
#include "Eigen/Dense"
```

Functions

- void lcg_set2box_eigen (const Eigen::VectorXd &low, const Eigen::VectorXd &hig, Eigen::VectorXd m)

  Set the input value within a box constraint.

4.5.1 Function Documentation

4.5.1.1 lcg_set2box_eigen()

```
void lcg_set2box_eigen (
            const Eigen::VectorXd & low,
            const Eigen::VectorXd & hig,
            Eigen::VectorXd m )
```

Set the input value within a box constraint.

Parameters

low_bound	Whether to include the low boundary value
hig_bound	Whether to include the high boundary value
m	Returned values

4.6 clcg.cpp File Reference

```
#include "clcg.h"
#include "cmath"
#include "config.h"
#include "omp.h"
```

Typedefs

- typedef int(∗ clcg_solver_ptr) (clcg_axfunc_ptr Afp, clcg_progress_ptr Pfp, lcg_complex ∗m, const lcg_complex ∗B, const int n_size, const clcg_para ∗param, void ∗instance)

Functions

- int clbicg (clcg_axfunc_ptr Afp, clcg_progress_ptr Pfp, lcg_complex ∗m, const lcg_complex ∗B, const int n←_size, const clcg_para ∗param, void ∗instance)
- int clbicg_symmetric (clcg_axfunc_ptr Afp, clcg_progress_ptr Pfp, lcg_complex ∗m, const lcg_complex ∗B, const int n_size, const clcg_para ∗param, void ∗instance)
- int clcgs (clcg_axfunc_ptr Afp, clcg_progress_ptr Pfp, lcg_complex ∗m, const lcg_complex ∗B, const int n_←size, const clcg_para ∗param, void ∗instance)
- int clbicgstab (clcg_axfunc_ptr Afp, clcg_progress_ptr Pfp, lcg_complex ∗m, const lcg_complex ∗B, const int n_size, const clcg_para ∗param, void ∗instance)
- int cltfqmr (clcg_axfunc_ptr Afp, clcg_progress_ptr Pfp, lcg_complex ∗m, const lcg_complex ∗B, const int n_size, const clcg_para ∗param, void ∗instance)
- int clcg_solver (clcg_axfunc_ptr Afp, clcg_progress_ptr Pfp, lcg_complex ∗m, const lcg_complex ∗B, const int n_size, const clcg_para ∗param, void ∗instance, clcg_solver_enum solver_id)

  A combined complex conjugate gradient solver function.

4.6.1 Typedef Documentation

4.6.1.1 clcg_solver_ptr

```
typedef int(* clcg_solver_ptr) (clcg_axfunc_ptr Afp, clcg_progress_ptr Pfp, lcg_complex *m,
const lcg_complex *B, const int n_size, const clcg_para *param, void *instance)
```

4.6.2 Function Documentation

4.6.2.1 clbicg()

```
int clbicg (
            clcg_axfunc_ptr Afp,
            clcg_progress_ptr Pfp,
            lcg_complex * m,
            const lcg_complex * B,
            const int n_size,
            const clcg_para * param,
            void * instance )
```

4.6.2.2 clbicg_symmetric()

```
int clbicg_symmetric (
            clcg_axfunc_ptr Afp,
            clcg_progress_ptr Pfp,
            lcg_complex * m,
            const lcg_complex * B,
            const int n_size,
            const clcg_para * param,
            void * instance )
```

4.6.2.3 clbicgstab()

```
int clbicgstab (
            clcg_axfunc_ptr Afp,
            clcg_progress_ptr Pfp,
            lcg_complex * m,
            const lcg_complex * B,
            const int n_size,
            const clcg_para * param,
            void * instance )
```

4.6.2.4 clcg_solver()

```
int clcg_solver (
            clcg_axfunc_ptr Afp,
            clcg_progress_ptr Pfp,
            lcg_complex * m,
            const lcg_complex * B,
            const int n_size,
            const clcg_para * param,
            void * instance,
            clcg_solver_enum solver_id = CLCG_BICG )
```

A combined complex conjugate gradient solver function.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'NULL' for global functions.
	solver_id	Solver type used to solve the linear system. The default value is LCG_CGS.

Returns

Status of the function.

4.6.2.5 clcgs()

```
int clcgs (
            clcg_axfunc_ptr Afp,
            clcg_progress_ptr Pfp,
            lcg_complex * m,
            const lcg_complex * B,
            const int n_size,
            const clcg_para * param,
            void * instance )
```

4.6.2.6 cltfqmr()

```
int cltfqmr (
            clcg_axfunc_ptr Afp,
            clcg_progress_ptr Pfp,
            lcg_complex * m,
            const lcg_complex * B,
            const int n_size,
            const clcg_para * param,
            void * instance )
```

4.7 clcg.h File Reference

```
#include "lcg_complex.h"
#include "util.h"
```

Typedefs

- typedef void(* clcg_axfunc_ptr) (void *instance, const lcg_complex *x, lcg_complex *prod_Ax, const int x←_size, lcg_matrix_e layout, clcg_complex_e conjugate)

  *Callback interface for calculating the complex product of a N*N matrix 'A' multiplied by a complex vertical vector 'x'.*
- typedef int(* clcg_progress_ptr) (void *instance, const lcg_complex *m, const lcg_float converge, const clcg_para *param, const int n_size, const int k)

  Callback interface for monitoring the progress and terminate the iteration if necessary.

Functions

- int clcg_solver (clcg_axfunc_ptr Afp, clcg_progress_ptr Pfp, lcg_complex *m, const lcg_complex *B, const int n_size, const clcg_para *param, void *instance, clcg_solver_enum solver_id=CLCG_BICG)

  A combined complex conjugate gradient solver function.

4.7.1 Typedef Documentation

4.7.1.1 clcg_axfunc_ptr

```
typedef void(* clcg_axfunc_ptr) (void *instance, const lcg_complex *x, lcg_complex *prod_Ax,
const int x_size, lcg_matrix_e layout, clcg_complex_e conjugate)
```

Callback interface for calculating the complex product of a N∗N matrix 'A' multiplied by a complex vertical vector 'x'.

Parameters

instance	The user data sent for the clcg_solver() functions by the client.
x	Multiplier of the Ax product.
Ax	Product of A multiplied by x.
x_size	Size of x and column/row numbers of A.
layout	Whether to use the transpose of A for calculation.
conjugate	Whether to use the conjugate of A for calculation.

4.7.1.2 clcg_progress_ptr

```
typedef int(* clcg_progress_ptr) (void *instance, const lcg_complex *m, const lcg_float converge,
const clcg_para *param, const int n_size, const int k)
```

Callback interface for monitoring the progress and terminate the iteration if necessary.

Parameters

instance	The user data sent for the clcg_solver() functions by the client.
m	The current solutions.
converge	The current value evaluating the iteration progress.
n_size	The size of the variables
k	The iteration count.

Return values

int	Zero to continue the optimization process. Returning a non-zero value will terminate the optimization process.

4.7.2 Function Documentation

4.7.2.1 clcg_solver()

```
int clcg_solver (
            clcg_axfunc_ptr Afp,
            clcg_progress_ptr Pfp,
            lcg_complex * m,
            const lcg_complex * B,
            const int n_size,
            const clcg_para * param,
            void * instance,
            clcg_solver_enum solver_id = CLCG_BICG )
```

A combined complex conjugate gradient solver function.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'NULL' for global functions.
	solver↩_id	Solver type used to solve the linear system. The default value is LCG_CGS.

Returns

Status of the function.

4.8 clcg_cuda.h File Reference

```
#include "util.h"
#include "lcg_complex_cuda.h"
#include <cublas_v2.h>
#include <cusparse_v2.h>
```

Typedefs

- typedef void(∗ clcg_axfunc_cuda_ptr) (void ∗instance, cublasHandle_t cub_handle, cusparseHandle_t cus↩_handle, cusparseDnVecDescr_t x, cusparseDnVecDescr_t prod_Ax, const int n_size, const int nz_size, cusparseOperation_t oper_t)

  Callback interface for calculating the product of a N∗N matrix 'A' multiplied by a vertical vector 'x'. Note that both A and x are hosted on the GPU device.
- typedef int(∗ clcg_progress_cuda_ptr) (void ∗instance, const cuDoubleComplex ∗m, const lcg_float converge, const clcg_para ∗param, const int n_size, const int nz_size, const int k)

  Callback interface for monitoring the progress and terminate the iteration if necessary. Note that m is hosted on the GPU device.

Functions

- int clcg_solver_cuda (clcg_axfunc_cuda_ptr Afp, clcg_progress_cuda_ptr Pfp, cuDoubleComplex *m, const cuDoubleComplex *B, const int n_size, const int nz_size, const clcg_para *param, void *instance, cublas←Handle_t cub_handle, cusparseHandle_t cus_handle, clcg_solver_enum solver_id=CLCG_BICG)

    A combined conjugate gradient solver function. Note that both m and B are hosted on the GPU device.
- int clcg_solver_preconditioned_cuda (clcg_axfunc_cuda_ptr Afp, clcg_axfunc_cuda_ptr Mfp, clcg_progress_cuda_ptr Pfp, cuDoubleComplex *m, const cuDoubleComplex *B, const int n_size, const int nz_size, const clcg_para *param, void *instance, cublasHandle_t cub_handle, cusparseHandle_t cus_handle, clcg_solver_enum solver_id=CLCG_PCG)

    A combined conjugate gradient solver function. Note that both m and B are hosted on the GPU device.

4.8.1 Typedef Documentation

4.8.1.1 clcg_axfunc_cuda_ptr

```
typedef void(* clcg_axfunc_cuda_ptr) (void *instance, cublasHandle_t cub_handle, cusparse←
Handle_t cus_handle, cusparseDnVecDescr_t x, cusparseDnVecDescr_t prod_Ax, const int n_size,
const int nz_size, cusparseOperation_t oper_t)
```

Callback interface for calculating the product of a N*N matrix 'A' multiplied by a vertical vector 'x'. Note that both A and x are hosted on the GPU device.

Parameters

instance	The user data sent for the lcg_solver_cuda() functions by the client.
cub_handle	Handler of the cublas object.
cus_handle	Handlee of the cusparse object.
x	Multiplier of the Ax product.
Ax	Product of A multiplied by x.
n_size	Size of x and column/row numbers of A.

4.8.1.2 clcg_progress_cuda_ptr

```
typedef int(* clcg_progress_cuda_ptr) (void *instance, const cuDoubleComplex *m, const lcg_float
converge, const clcg_para *param, const int n_size, const int nz_size, const int k)
```

Callback interface for monitoring the progress and terminate the iteration if necessary. Note that m is hosted on the GPU device.

Parameters

instance	The user data sent for the lcg_solver() functions by the client.
m	The current solutions.
converge	The current value evaluating the iteration progress.
n_size	The size of the variables
k	The iteration count.

Return values

int	Zero to continue the optimization process. Returning a non-zero value will terminate the optimization process.

4.8.2 Function Documentation

4.8.2.1 clcg_solver_cuda()

```
int clcg_solver_cuda (
            clcg_axfunc_cuda_ptr Afp,
            clcg_progress_cuda_ptr Pfp,
            cuDoubleComplex * m,
            const cuDoubleComplex * B,
            const int n_size,
            const int nz_size,
            const clcg_para * param,
            void * instance,
            cublasHandle_t cub_handle,
            cusparseHandle_t cus_handle,
            clcg_solver_enum solver_id = CLCG_BICG )
```

A combined conjugate gradient solver function. Note that both m and B are hosted on the GPU device.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client.
	cub_handle	Handler of the cublas object.
	cus_handle	Handlee of the cusparse object. This variable is either 'this' for class member functions or 'NULL' for global functions.
	solver_id	Solver type used to solve the linear system. The default value is LCG_BICG.

Returns

Status of the function.

4.8.2.2 clcg_solver_preconditioned_cuda()

```
int clcg_solver_preconditioned_cuda (
            clcg_axfunc_cuda_ptr Afp,
```

```
clcg_axfunc_cuda_ptr Mfp,
clcg_progress_cuda_ptr Pfp,
cuDoubleComplex * m,
const cuDoubleComplex * B,
const int n_size,
const int nz_size,
const clcg_para * param,
void * instance,
cublasHandle_t cub_handle,
cusparseHandle_t cus_handle,
clcg_solver_enum solver_id = CLCG_PCG )
```

A combined conjugate gradient solver function. Note that both m and B are hosted on the GPU device.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Mfp*	Callback function for calculating the product of 'Mx' for preconditioning.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client.
	cub_handle	Handler of the cublas object.
	cus_handle	Handlee of the cusparse object. This variable is either 'this' for class member functions or 'NULL' for global functions.
	solver_id	Solver type used to solve the linear system. The default value is LCG_CGS.

Returns

Status of the function.

4.9 clcg_cudaf.h File Reference

```
#include "util.h"
#include "lcg_complex_cuda.h"
#include <cublas_v2.h>
#include <cusparse_v2.h>
```

Typedefs

- typedef void(∗ clcg_axfunc_cudaf_ptr) (void ∗instance, cublasHandle_t cub_handle, cusparseHandle_t cus←_handle, cusparseDnVecDescr_t x, cusparseDnVecDescr_t prod_Ax, const int n_size, const int nz_size, cusparseOperation_t oper_t)

  Callback interface for calculating the product of a N∗N matrix 'A' multiplied by a vertical vector 'x'. Note that both A and x are hosted on the GPU device.
- typedef int(∗ clcg_progress_cudaf_ptr) (void ∗instance, const cuComplex ∗m, const float converge, const clcg_para ∗param, const int n_size, const int nz_size, const int k)

  Callback interface for monitoring the progress and terminate the iteration if necessary. Note that m is hosted on the GPU device.

Functions

- int clcg_solver_cuda (clcg_axfunc_cudaf_ptr Afp, clcg_progress_cudaf_ptr Pfp, cuComplex ∗m, const cu←Complex ∗B, const int n_size, const int nz_size, const clcg_para ∗param, void ∗instance, cublasHandle_t cub_handle, cusparseHandle_t cus_handle, clcg_solver_enum solver_id=CLCG_BICG)

  A combined conjugate gradient solver function. Note that both m and B are hosted on the GPU device.
- int clcg_solver_preconditioned_cuda (clcg_axfunc_cudaf_ptr Afp, clcg_axfunc_cudaf_ptr Mfp, clcg_progress_cudaf_ptr Pfp, cuComplex ∗m, const cuComplex ∗B, const int n_size, const int nz_size, const clcg_para ∗param, void ∗instance, cublasHandle_t cub_handle, cusparseHandle_t cus_handle, clcg_solver_enum solver_←id=CLCG_PCG)

  A combined conjugate gradient solver function. Note that both m and B are hosted on the GPU device.

4.9.1 Typedef Documentation

4.9.1.1 clcg_axfunc_cudaf_ptr

```
typedef void(* clcg_axfunc_cudaf_ptr) (void *instance, cublasHandle_t cub_handle, cusparse←
Handle_t cus_handle, cusparseDnVecDescr_t x, cusparseDnVecDescr_t prod_Ax, const int n_size,
const int nz_size, cusparseOperation_t oper_t)
```

Callback interface for calculating the product of a N∗N matrix 'A' multiplied by a vertical vector 'x'. Note that both A and x are hosted on the GPU device.

Parameters

instance	The user data sent for the lcg_solver_cuda() functions by the client.
cub_handle	Handler of the cublas object.
cus_handle	Handlee of the cusparse object.
x	Multiplier of the Ax product.
Ax	Product of A multiplied by x.
n_size	Size of x and column/row numbers of A.

4.9.1.2 clcg_progress_cudaf_ptr

```
typedef int(* clcg_progress_cudaf_ptr) (void *instance, const cuComplex *m, const float converge,
const clcg_para *param, const int n_size, const int nz_size, const int k)
```

Callback interface for monitoring the progress and terminate the iteration if necessary. Note that m is hosted on the GPU device.

Parameters

instance	The user data sent for the lcg_solver() functions by the client.
m	The current solutions.
converge	The current value evaluating the iteration progress.
n_size	The size of the variables
k	The iteration count.

Return values

int	Zero to continue the optimization process. Returning a non-zero value will terminate the optimization process.

4.9.2 Function Documentation

4.9.2.1 clcg_solver_cuda()

```
int clcg_solver_cuda (
            clcg_axfunc_cudaf_ptr Afp,
            clcg_progress_cudaf_ptr Pfp,
            cuComplex * m,
            const cuComplex * B,
            const int n_size,
            const int nz_size,
            const clcg_para * param,
            void * instance,
            cublasHandle_t cub_handle,
            cusparseHandle_t cus_handle,
            clcg_solver_enum solver_id = CLCG_BICG )
```

A combined conjugate gradient solver function. Note that both m and B are hosted on the GPU device.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client.
	cub_handle	Handler of the cublas object.
	cus_handle	Handlee of the cusparse object. This variable is either 'this' for class member functions or 'NULL' for global functions.
	solver_id	Solver type used to solve the linear system. The default value is LCG_BICG.

Returns

Status of the function.

4.9.2.2 clcg_solver_preconditioned_cuda()

```
int clcg_solver_preconditioned_cuda (
            clcg_axfunc_cudaf_ptr Afp,
```

```
clcg_axfunc_cudaf_ptr Mfp,
clcg_progress_cudaf_ptr Pfp,
cuComplex * m,
const cuComplex * B,
const int n_size,
const int nz_size,
const clcg_para * param,
void * instance,
cublasHandle_t cub_handle,
cusparseHandle_t cus_handle,
clcg_solver_enum solver_id = CLCG_PCG )
```

A combined conjugate gradient solver function. Note that both m and B are hosted on the GPU device.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Mfp*	Callback function for calculating the product of 'Mx' for preconditioning.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client.
	cub_handle	Handler of the cublas object.
	cus_handle	Handlee of the cusparse object. This variable is either 'this' for class member functions or 'NULL' for global functions.
	solver_id	Solver type used to solve the linear system. The default value is LCG_CGS.

Returns

Status of the function.

4.10 clcg_eigen.cpp File Reference

```
#include "cmath"
#include "ctime"
#include "iostream"
#include "clcg_eigen.h"
#include "config.h"
#include "omp.h"
```

Typedefs

- typedef int(∗ eigen_solver_ptr) (clcg_axfunc_eigen_ptr Afp, clcg_progress_eigen_ptr Pfp, Eigen::VectorXcd &m, const Eigen::VectorXcd &B, const clcg_para ∗param, void ∗instance)
- typedef int(∗ eigen_preconditioned_solver_ptr) (clcg_axfunc_eigen_ptr Afp, clcg_axfunc_eigen_ptr Mfp, clcg_progress_eigen_ptr Pfp, Eigen::VectorXcd &m, const Eigen::VectorXcd &B, const clcg_para ∗param, void ∗instance)

Functions

- int clbicg (clcg_axfunc_eigen_ptr Afp, clcg_progress_eigen_ptr Pfp, Eigen::VectorXcd &m, const Eigen::↩ VectorXcd &B, const clcg_para ∗param, void ∗instance)
- int clbicg_symmetric (clcg_axfunc_eigen_ptr Afp, clcg_progress_eigen_ptr Pfp, Eigen::VectorXcd &m, const Eigen::VectorXcd &B, const clcg_para ∗param, void ∗instance)
- int clcgs (clcg_axfunc_eigen_ptr Afp, clcg_progress_eigen_ptr Pfp, Eigen::VectorXcd &m, const Eigen::↩ VectorXcd &B, const clcg_para ∗param, void ∗instance)
- int cltfqmr (clcg_axfunc_eigen_ptr Afp, clcg_progress_eigen_ptr Pfp, Eigen::VectorXcd &m, const Eigen::↩ VectorXcd &B, const clcg_para ∗param, void ∗instance)
- int clcg_solver_eigen (clcg_axfunc_eigen_ptr Afp, clcg_progress_eigen_ptr Pfp, Eigen::VectorXcd &m, const Eigen::VectorXcd &B, const clcg_para ∗param, void ∗instance, clcg_solver_enum solver_id)

    A combined conjugate gradient solver function.
- int clpcg (clcg_axfunc_eigen_ptr Afp, clcg_axfunc_eigen_ptr Mfp, clcg_progress_eigen_ptr Pfp, Eigen::↩ VectorXcd &m, const Eigen::VectorXcd &B, const clcg_para ∗param, void ∗instance)
- int clpbicg (clcg_axfunc_eigen_ptr Afp, clcg_axfunc_eigen_ptr Mfp, clcg_progress_eigen_ptr Pfp, Eigen::↩ VectorXcd &m, const Eigen::VectorXcd &B, const clcg_para ∗param, void ∗instance)
- int clcg_solver_preconditioned_eigen (clcg_axfunc_eigen_ptr Afp, clcg_axfunc_eigen_ptr Mfp, clcg_progress_eigen_ptr Pfp, Eigen::VectorXcd &m, const Eigen::VectorXcd &B, const clcg_para ∗param, void ∗instance, clcg_solver_enum solver_id)

    A combined conjugate gradient solver function.

4.10.1 Typedef Documentation

4.10.1.1 eigen_preconditioned_solver_ptr

```
typedef int(∗ eigen_preconditioned_solver_ptr) (clcg_axfunc_eigen_ptr Afp, clcg_axfunc_eigen_ptr
Mfp, clcg_progress_eigen_ptr Pfp, Eigen::VectorXcd &m, const Eigen::VectorXcd &B, const clcg_para
∗param, void ∗instance)
```

4.10.1.2 eigen_solver_ptr

```
typedef int(∗ eigen_solver_ptr) (clcg_axfunc_eigen_ptr Afp, clcg_progress_eigen_ptr Pfp, Eigen↩
::VectorXcd &m, const Eigen::VectorXcd &B, const clcg_para ∗param, void ∗instance)
```

4.10.2 Function Documentation

4.10.2.1 clbicg()

```
int clbicg (
            clcg_axfunc_eigen_ptr Afp,
            clcg_progress_eigen_ptr Pfp,
            Eigen::VectorXcd & m,
            const Eigen::VectorXcd & B,
            const clcg_para ∗ param,
            void ∗ instance )
```

4.10.2.2 clbicg_symmetric()

```
int clbicg_symmetric (
            clcg_axfunc_eigen_ptr Afp,
            clcg_progress_eigen_ptr Pfp,
            Eigen::VectorXcd & m,
            const Eigen::VectorXcd & B,
            const clcg_para * param,
            void * instance )
```

4.10.2.3 clcg_solver_eigen()

```
int clcg_solver_eigen (
            clcg_axfunc_eigen_ptr Afp,
            clcg_progress_eigen_ptr Pfp,
            Eigen::VectorXcd & m,
            const Eigen::VectorXcd & B,
            const clcg_para * param,
            void * instance,
            clcg_solver_enum solver_id = CLCG_CGS )
```

A combined conjugate gradient solver function.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the solver function by the client. This variable is either 'this' for class member functions or 'nullptr' for global functions.
	solver↵_id	Solver type used to solve the linear system. The default value is CLCG_CGS.

Returns

Status of the function.

4.10.2.4 clcg_solver_preconditioned_eigen()

```
int clcg_solver_preconditioned_eigen (
            clcg_axfunc_eigen_ptr Afp,
            clcg_axfunc_eigen_ptr Mfp,
            clcg_progress_eigen_ptr Pfp,
            Eigen::VectorXcd & m,
            const Eigen::VectorXcd & B,
```

```
const clcg_para * param,
void * instance,
clcg_solver_enum solver_id = CLCG_PBICG )
```

A combined conjugate gradient solver function.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Mfp*	Callback function for calculating the product of 'M^{-1}x', in which M is the preconditioning matrix
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the solver function by the client. This variable is either 'this' for class member functions or 'nullptr' for global functions.
	solver_id	Solver type used to solve the linear system. the value must CLCG_PBICG (default) or CLCG_PCG.

Returns

Status of the function.

4.10.2.5 clcgs()

```
int clcgs (
    clcg_axfunc_eigen_ptr Afp,
    clcg_progress_eigen_ptr Pfp,
    Eigen::VectorXcd & m,
    const Eigen::VectorXcd & B,
    const clcg_para * param,
    void * instance )
```

4.10.2.6 clpbicg()

```
int clpbicg (
    clcg_axfunc_eigen_ptr Afp,
    clcg_axfunc_eigen_ptr Mfp,
    clcg_progress_eigen_ptr Pfp,
    Eigen::VectorXcd & m,
    const Eigen::VectorXcd & B,
    const clcg_para * param,
    void * instance )
```

4.10.2.7 clpcg()

```
int clpcg (
            clcg_axfunc_eigen_ptr Afp,
            clcg_axfunc_eigen_ptr Mfp,
            clcg_progress_eigen_ptr Pfp,
            Eigen::VectorXcd & m,
            const Eigen::VectorXcd & B,
            const clcg_para * param,
            void * instance )
```

4.10.2.8 cltfqmr()

```
int cltfqmr (
            clcg_axfunc_eigen_ptr Afp,
            clcg_progress_eigen_ptr Pfp,
            Eigen::VectorXcd & m,
            const Eigen::VectorXcd & B,
            const clcg_para * param,
            void * instance )
```

4.11 clcg_eigen.h File Reference

```
#include "util.h"
#include "complex"
#include "Eigen/Dense"
```

Typedefs

- typedef void(∗ clcg_axfunc_eigen_ptr) (void ∗instance, const Eigen::VectorXcd &x, Eigen::VectorXcd &prod↩_Ax, lcg_matrix_e layout, clcg_complex_e conjugate)

  Callback interface for calculating the product of a N∗N matrix 'A' multiplied by a vertical vector 'x'.
- typedef int(∗ clcg_progress_eigen_ptr) (void ∗instance, const Eigen::VectorXcd ∗m, const lcg_float converge, const clcg_para ∗param, const int k)

  Callback interface for monitoring the progress and terminate the iteration if necessary.

Functions

- int clcg_solver_eigen (clcg_axfunc_eigen_ptr Afp, clcg_progress_eigen_ptr Pfp, Eigen::VectorXcd &m, const Eigen::VectorXcd &B, const clcg_para ∗param, void ∗instance, clcg_solver_enum solver_id=CLCG_CGS)

  A combined conjugate gradient solver function.
- int clcg_solver_preconditioned_eigen (clcg_axfunc_eigen_ptr Afp, clcg_axfunc_eigen_ptr Mfp, clcg_progress_eigen_ptr Pfp, Eigen::VectorXcd &m, const Eigen::VectorXcd &B, const clcg_para ∗param, void ∗instance, clcg_solver_enum solver_id=CLCG_PBICG)

  A combined conjugate gradient solver function.

4.11.1 Typedef Documentation

4.11.1.1 clcg_axfunc_eigen_ptr

```
typedef void(* clcg_axfunc_eigen_ptr) (void *instance, const Eigen::VectorXcd &x, Eigen::↩
VectorXcd &prod_Ax, lcg_matrix_e layout, clcg_complex_e conjugate)
```

Callback interface for calculating the product of a N∗N matrix 'A' multiplied by a vertical vector 'x'.

Parameters

instance	The user data sent for the solver functions by the client.
x	Multiplier of the Ax product.
Ax	Product of A multiplied by x.
layout	layout information of the matrix A passed by the solver functions.
conjugate	Layout information of the matrix A passed by the solver functions.

4.11.1.2 clcg_progress_eigen_ptr

```
typedef int(* clcg_progress_eigen_ptr) (void *instance, const Eigen::VectorXcd *m, const lcg_float
converge, const clcg_para *param, const int k)
```

Callback interface for monitoring the progress and terminate the iteration if necessary.

Parameters

instance	The user data sent for the solver functions by the client.
m	The current solutions.
converge	The current value evaluating the iteration progress.
param	The parameter object passed by the solver functions.
k	The iteration count.

Return values

int	Zero to continue the optimization process. Returning a non-zero value will terminate the optimization process.

4.11.2 Function Documentation

4.11.2.1 clcg_solver_eigen()

```
int clcg_solver_eigen (
            clcg_axfunc_eigen_ptr Afp,
            clcg_progress_eigen_ptr Pfp,
            Eigen::VectorXcd & m,
            const Eigen::VectorXcd & B,
            const clcg_para * param,
            void * instance,
            clcg_solver_enum solver_id = CLCG_CGS )
```

A combined conjugate gradient solver function.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the solver function by the client. This variable is either 'this' for class member functions or 'nullptr' for global functions.
	solver_id	Solver type used to solve the linear system. The default value is CLCG_CGS.

Returns

Status of the function.

4.11.2.2 clcg_solver_preconditioned_eigen()

```
int clcg_solver_preconditioned_eigen (
            clcg_axfunc_eigen_ptr Afp,
            clcg_axfunc_eigen_ptr Mfp,
            clcg_progress_eigen_ptr Pfp,
            Eigen::VectorXcd & m,
            const Eigen::VectorXcd & B,
            const clcg_para * param,
            void * instance,
            clcg_solver_enum solver_id = CLCG_PBICG )
```

A combined conjugate gradient solver function.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Mfp*	Callback function for calculating the product of 'M^{-1}x', in which M is the preconditioning matrix
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.

Parameters

	B	Objective vector of the linear system.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the solver function by the client. This variable is either 'this' for class member functions or 'nullptr' for global functions.
	solver↩ _id	Solver type used to solve the linear system. the value must CLCG_PBICG (default) or CLCG_PCG.

Returns

Status of the function.

4.12 config.h File Reference

Macros

- #define LibLCG_OPENMP
- #define LibLCG_EIGEN
- #define LibLCG_STD_COMPLEX
- #define LibLCG_CUDA

4.12.1 Macro Definition Documentation

4.12.1.1 LibLCG_CUDA

```
#define LibLCG_CUDA
```

4.12.1.2 LibLCG_EIGEN

```
#define LibLCG_EIGEN
```

4.12.1.3 LibLCG_OPENMP

```
#define LibLCG_OPENMP
```

4.12.1.4 LibLCG_STD_COMPLEX

```
#define LibLCG_STD_COMPLEX
```

4.13 lcg.cpp File Reference

```
#include "lcg.h"
#include "cmath"
#include "config.h"
#include "omp.h"
```

Typedefs

- typedef int(∗ lcg_solver_ptr) (lcg_axfunc_ptr Afp, lcg_progress_ptr Pfp, lcg_float ∗m, const lcg_float ∗B, const int n_size, const lcg_para ∗param, void ∗instance)

  Callback interface of the conjugate gradient solver.
- typedef int(∗ lcg_solver_ptr2) (lcg_axfunc_ptr Afp, lcg_progress_ptr Pfp, lcg_float ∗m, const lcg_float ∗B, const lcg_float ∗low, const lcg_float ∗hig, const int n_size, const lcg_para ∗param, void ∗instance)

  A combined conjugate gradient solver function.

Functions

- int lbicgstab (lcg_axfunc_ptr Afp, lcg_progress_ptr Pfp, lcg_float ∗m, const lcg_float ∗B, const int n_size, const lcg_para ∗param, void ∗instance)

  Biconjugate gradient method.
- int lbicgstab2 (lcg_axfunc_ptr Afp, lcg_progress_ptr Pfp, lcg_float ∗m, const lcg_float ∗B, const int n_size, const lcg_para ∗param, void ∗instance)

  Biconjugate gradient method 2.
- int lcg_solver (lcg_axfunc_ptr Afp, lcg_progress_ptr Pfp, lcg_float ∗m, const lcg_float ∗B, const int n_size, const lcg_para ∗param, void ∗instance, lcg_solver_enum solver_id)

  A combined conjugate gradient solver function.
- int lpcg (lcg_axfunc_ptr Afp, lcg_axfunc_ptr Mfp, lcg_progress_ptr Pfp, lcg_float ∗m, const lcg_float ∗B, const int n_size, const lcg_para ∗param, void ∗instance)

  Preconditioned conjugate gradient method.
- int lcg_solver_preconditioned (lcg_axfunc_ptr Afp, lcg_axfunc_ptr Mfp, lcg_progress_ptr Pfp, lcg_float ∗m, const lcg_float ∗B, const int n_size, const lcg_para ∗param, void ∗instance, lcg_solver_enum solver_id)

  A combined conjugate gradient solver function.
- int lpg (lcg_axfunc_ptr Afp, lcg_progress_ptr Pfp, lcg_float ∗m, const lcg_float ∗B, const lcg_float ∗low, const lcg_float ∗hig, const int n_size, const lcg_para ∗param, void ∗instance)

  Conjugate gradient method with projected gradient for inequality constraints.
- int lspg (lcg_axfunc_ptr Afp, lcg_progress_ptr Pfp, lcg_float ∗m, const lcg_float ∗B, const lcg_float ∗low, const lcg_float ∗hig, const int n_size, const lcg_para ∗param, void ∗instance)

  Conjugate gradient method with projected gradient for inequality constraints.
- int lcg_solver_constrained (lcg_axfunc_ptr Afp, lcg_progress_ptr Pfp, lcg_float ∗m, const lcg_float ∗B, const lcg_float ∗low, const lcg_float ∗hig, const int n_size, const lcg_para ∗param, void ∗instance, lcg_solver_enum solver_id)

  A combined conjugate gradient solver function with inequality constraints.
- int lcg (lcg_axfunc_ptr Afp, lcg_progress_ptr Pfp, lcg_float ∗m, const lcg_float ∗B, const int n_size, const lcg_para ∗param, void ∗instance, lcg_float ∗Gk, lcg_float ∗Dk, lcg_float ∗ADk)

  Standalone function of the Linear Conjugate Gradient algorithm.
- int lcgs (lcg_axfunc_ptr Afp, lcg_progress_ptr Pfp, lcg_float ∗m, const lcg_float ∗B, const int n_size, const lcg_para ∗param, void ∗instance, lcg_float ∗RK, lcg_float ∗R0T, lcg_float ∗PK, lcg_float ∗AX, lcg_float ∗UK, lcg_float ∗QK, lcg_float ∗WK)

  Standalone function of the Conjugate Gradient Squared algorithm.

4.13.1 Typedef Documentation

4.13.1.1 lcg_solver_ptr

```
typedef int(* lcg_solver_ptr) (lcg_axfunc_ptr Afp, lcg_progress_ptr Pfp, lcg_float *m, const
lcg_float *B, const int n_size, const lcg_para *param, void *instance)
```

Callback interface of the conjugate gradient solver.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'nullptr' for global functions.

Returns

Status of the function.

4.13.1.2 lcg_solver_ptr2

```
typedef int(* lcg_solver_ptr2) (lcg_axfunc_ptr Afp, lcg_progress_ptr Pfp, lcg_float *m, const
lcg_float *B, const lcg_float *low, const lcg_float *hig, const int n_size, const lcg_para
*param, void *instance)
```

A combined conjugate gradient solver function.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*low*	The lower boundary of the acceptable solution.
in	*hig*	The higher boundary of the acceptable solution.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'nullptr' for global functions.
	solver_id	Solver type used to solve the linear system. The default value is LCG_CGS.
	P	Precondition vector (optional expect for the LCG_PCG method). The default value is nullptr.

Returns

Status of the function.

4.13.2 Function Documentation

4.13.2.1 lbicgstab()

```
int lbicgstab (
            lcg_axfunc_ptr Afp,
            lcg_progress_ptr Pfp,
            lcg_float * m,
            const lcg_float * B,
            const int n_size,
            const lcg_para * param,
            void * instance )
```

Biconjugate gradient method.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'nullptr' for global functions.
	P	Precondition vector (optional expect for the LCG_PCG method). The default value is nullptr.

Returns

Status of the function.

4.13.2.2 lbicgstab2()

```
int lbicgstab2 (
            lcg_axfunc_ptr Afp,
            lcg_progress_ptr Pfp,
            lcg_float * m,
            const lcg_float * B,
            const int n_size,
            const lcg_para * param,
            void * instance )
```

Biconjugate gradient method 2.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'nullptr' for global functions.
	P	Precondition vector (optional expect for the LCG_PCG method). The default value is nullptr.

Returns

Status of the function.

4.13.2.3 lcg()

```
int lcg (
            lcg_axfunc_ptr Afp,
            lcg_progress_ptr Pfp,
            lcg_float * m,
            const lcg_float * B,
            const int n_size,
            const lcg_para * param,
            void * instance,
            lcg_float * Gk = nullptr,
            lcg_float * Dk = nullptr,
            lcg_float * ADk = nullptr )
```

Standalone function of the Linear Conjugate Gradient algorithm.

Note

To use the lcg() function for massive inversions, it is better to provide external vectors Gk, Dk and ADk to avoid allocating and destroying temporary vectors.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector of the size n_size
in	*B*	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
in	*param*	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg() function by the client. This variable is either 'this' for class member functions or 'NULL' for global functions.
	Gk	Conjugate gradient vector of the size n_size. If this pointer is null, the function will create an internal vector instead.
	Dk	Directional gradient vector of the size n_size. If this pointer is null, the function will create an internal vector instead.
	ADk	Intermediate vector of the size n_size. If this pointer is null, the function will create an internal vector instead.

Returns

Status of the function.

4.13.2.4 lcg_solver()

```
int lcg_solver (
            lcg_axfunc_ptr Afp,
            lcg_progress_ptr Pfp,
            lcg_float * m,
            const lcg_float * B,
            const int n_size,
            const lcg_para * param,
            void * instance,
            lcg_solver_enum solver_id = LCG_CGS )
```

A combined conjugate gradient solver function.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'NULL' for global functions.
	solver↩_id	Solver type used to solve the linear system. The default value is LCG_CGS.

Returns

Status of the function.

4.13.2.5 lcg_solver_constrained()

```
int lcg_solver_constrained (
            lcg_axfunc_ptr Afp,
            lcg_progress_ptr Pfp,
            lcg_float * m,
            const lcg_float * B,
            const lcg_float * low,
            const lcg_float * hig,
            const int n_size,
            const lcg_para * param,
            void * instance,
            lcg_solver_enum solver_id = LCG_PG )
```

A combined conjugate gradient solver function with inequality constraints.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*low*	The lower boundary of the acceptable solution.
in	*hig*	The higher boundary of the acceptable solution.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'NULL' for global functions.
	solver←_id	Solver type used to solve the linear system. The default value is LCG_CGS.
	P	Precondition vector (optional expect for the LCG_PCG method). The default value is NULL.

Returns

Status of the function.

4.13.2.6 lcg_solver_preconditioned()

```
int lcg_solver_preconditioned (
            lcg_axfunc_ptr Afp,
            lcg_axfunc_ptr Mfp,
            lcg_progress_ptr Pfp,
            lcg_float * m,
            const lcg_float * B,
            const int n_size,
            const lcg_para * param,
            void * instance,
            lcg_solver_enum solver_id = LCG_PCG )
```

A combined conjugate gradient solver function.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Mfp*	Callback function for calculating the product of 'M^{-1}x', in which M is the preconditioning matrix.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'NULL' for global functions.
	solver←_id	Solver type used to solve the linear system. The default value is LCG_PCG.

Returns

Status of the function.

4.13.2.7 lcgs()

```
int lcgs (
            lcg_axfunc_ptr Afp,
            lcg_progress_ptr Pfp,
            lcg_float * m,
            const lcg_float * B,
            const int n_size,
            const lcg_para * param,
            void * instance,
            lcg_float * RK = nullptr,
            lcg_float * R0T = nullptr,
            lcg_float * PK = nullptr,
            lcg_float * AX = nullptr,
            lcg_float * UK = nullptr,
            lcg_float * QK = nullptr,
            lcg_float * WK = nullptr )
```

Standalone function of the Conjugate Gradient Squared algorithm.

Note

Algorithm 2 in "Generalized conjugate gradient method" by Fokkema et al. (1996).

To use the lcgs() function for massive inversions, it is better to provide external vectors RK, R0T, PK, AX, UK, QK, and WK to avoid allocating and destroying temporary vectors.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'nullptr' for global functions.
	RK	Intermediate vector of the size n_size. If this pointer is null, the function will create an internal vector instead.
	R0T	Intermediate vector of the size n_size. If this pointer is null, the function will create an internal vector instead.
	PK	Intermediate vector of the size n_size. If this pointer is null, the function will create an internal vector instead.
	AX	Intermediate vector of the size n_size. If this pointer is null, the function will create an internal vector instead.
	UK	Intermediate vector of the size n_size. If this pointer is null, the function will create an internal vector instead.
	QK	Intermediate vector of the size n_size. If this pointer is null, the function will create an internal vector instead.
	WK	Intermediate vector of the size n_size. If this pointer is null, the function will create an internal vector instead.

Returns

Status of the function.

4.13.2.8 lpcg()

```
int lpcg (
            lcg_axfunc_ptr Afp,
            lcg_axfunc_ptr Mfp,
            lcg_progress_ptr Pfp,
            lcg_float * m,
            const lcg_float * B,
            const int n_size,
            const lcg_para * param,
            void * instance )
```

Preconditioned conjugate gradient method.

Note

Algorithm 1 in "Preconditioned conjugate gradients for singular systems" by Kaasschieter (1988).

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Mfp*	Callback function for calculating the product of 'M^{-1}x', in which M is the preconditioning matrix.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'nullptr' for global functions.

Returns

Status of the function.

4.13.2.9 lpg()

```
int lpg (
            lcg_axfunc_ptr Afp,
            lcg_progress_ptr Pfp,
            lcg_float * m,
            const lcg_float * B,
```

```
            const lcg_float * low,
            const lcg_float * hig,
            const int n_size,
            const lcg_para * param,
            void * instance )
```

Conjugate gradient method with projected gradient for inequality constraints.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*low*	The lower boundary of the acceptable solution.
in	*hig*	The higher boundary of the acceptable solution.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'nullptr' for global functions.
	solver↩_id	Solver type used to solve the linear system. The default value is LCG_CGS.
	P	Precondition vector (optional expect for the LCG_PCG method). The default value is nullptr.

Returns

Status of the function.

4.13.2.10 lspg()

```
int lspg (
            lcg_axfunc_ptr Afp,
            lcg_progress_ptr Pfp,
            lcg_float * m,
            const lcg_float * B,
            const lcg_float * low,
            const lcg_float * hig,
            const int n_size,
            const lcg_para * param,
            void * instance )
```

Conjugate gradient method with projected gradient for inequality constraints.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*low*	The lower boundary of the acceptable solution.

Parameters

in	*hig*	The higher boundary of the acceptable solution.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'nullptr' for global functions.
	solver↩ _id	Solver type used to solve the linear system. The default value is LCG_CGS.
	P	Precondition vector (optional expect for the LCG_PCG method). The default value is nullptr.

Returns

Status of the function.

4.14 lcg.h File Reference

```
#include "util.h"
```

Typedefs

- typedef void(∗ lcg_axfunc_ptr) (void ∗instance, const lcg_float ∗x, lcg_float ∗prod_Ax, const int n_size)

  Callback interface for calculating the product of a N∗N matrix 'A' multiplied by a vertical vector 'x'.
- typedef int(∗ lcg_progress_ptr) (void ∗instance, const lcg_float ∗m, const lcg_float converge, const lcg_para ∗param, const int n_size, const int k)

  Callback interface for monitoring the progress and terminate the iteration if necessary.

Functions

- int lcg_solver (lcg_axfunc_ptr Afp, lcg_progress_ptr Pfp, lcg_float ∗m, const lcg_float ∗B, const int n_size, const lcg_para ∗param, void ∗instance, lcg_solver_enum solver_id=LCG_CGS)

  A combined conjugate gradient solver function.
- int lcg_solver_preconditioned (lcg_axfunc_ptr Afp, lcg_axfunc_ptr Mfp, lcg_progress_ptr Pfp, lcg_float ∗m, const lcg_float ∗B, const int n_size, const lcg_para ∗param, void ∗instance, lcg_solver_enum solver_↩ id=LCG_PCG)

  A combined conjugate gradient solver function.
- int lcg_solver_constrained (lcg_axfunc_ptr Afp, lcg_progress_ptr Pfp, lcg_float ∗m, const lcg_float ∗B, const lcg_float ∗low, const lcg_float ∗hig, const int n_size, const lcg_para ∗param, void ∗instance, lcg_solver_enum solver_id=LCG_PG)

  A combined conjugate gradient solver function with inequality constraints.
- int lcg (lcg_axfunc_ptr Afp, lcg_progress_ptr Pfp, lcg_float ∗m, const lcg_float ∗B, const int n_size, const lcg_para ∗param, void ∗instance, lcg_float ∗Gk=nullptr, lcg_float ∗Dk=nullptr, lcg_float ∗ADk=nullptr)

  Standalone function of the Linear Conjugate Gradient algorithm.
- int lcgs (lcg_axfunc_ptr Afp, lcg_progress_ptr Pfp, lcg_float ∗m, const lcg_float ∗B, const int n_size, const lcg_para ∗param, void ∗instance, lcg_float ∗RK=nullptr, lcg_float ∗R0T=nullptr, lcg_float ∗PK=nullptr, lcg_float ∗AX=nullptr, lcg_float ∗UK=nullptr, lcg_float ∗QK=nullptr, lcg_float ∗WK=nullptr)

  Standalone function of the Conjugate Gradient Squared algorithm.

4.14.1 Typedef Documentation

4.14.1.1 lcg_axfunc_ptr

```
typedef void(* lcg_axfunc_ptr) (void *instance, const lcg_float *x, lcg_float *prod_Ax, const int n_size)
```

Callback interface for calculating the product of a N∗N matrix 'A' multiplied by a vertical vector 'x'.

Parameters

instance	The user data sent for the lcg_solver() functions by the client.
x	Multiplier of the Ax product.
Ax	Product of A multiplied by x.
n_size	Size of x and column/row numbers of A.

4.14.1.2 lcg_progress_ptr

```
typedef int(* lcg_progress_ptr) (void *instance, const lcg_float *m, const lcg_float converge, const lcg_para *param, const int n_size, const int k)
```

Callback interface for monitoring the progress and terminate the iteration if necessary.

Parameters

instance	The user data sent for the lcg_solver() functions by the client.
m	The current solutions.
converge	The current value evaluating the iteration progress.
n_size	The size of the variables
k	The iteration count.

Return values

int	Zero to continue the optimization process. Returning a non-zero value will terminate the optimization process.

4.14.2 Function Documentation

4.14.2.1 lcg()

```
int lcg (
            lcg_axfunc_ptr Afp,
```

```
            lcg_progress_ptr Pfp,
            lcg_float * m,
            const lcg_float * B,
            const int n_size,
            const lcg_para * param,
            void * instance,
            lcg_float * Gk = nullptr,
            lcg_float * Dk = nullptr,
            lcg_float * ADk = nullptr )
```

Standalone function of the Linear Conjugate Gradient algorithm.

Note

To use the lcg() function for massive inversions, it is better to provide external vectors Gk, Dk and ADk to avoid allocating and destroying temporary vectors.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector of the size n_size
in	*B*	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
in	*param*	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg() function by the client. This variable is either 'this' for class member functions or 'NULL' for global functions.
	Gk	Conjugate gradient vector of the size n_size. If this pointer is null, the function will create an internal vector instead.
	Dk	Directional gradient vector of the size n_size. If this pointer is null, the function will create an internal vector instead.
	ADk	Intermediate vector of the size n_size. If this pointer is null, the function will create an internal vector instead.

Returns

Status of the function.

4.14.2.2 lcg_solver()

```
int lcg_solver (
            lcg_axfunc_ptr Afp,
            lcg_progress_ptr Pfp,
            lcg_float * m,
            const lcg_float * B,
            const int n_size,
            const lcg_para * param,
            void * instance,
            lcg_solver_enum solver_id = LCG_CGS )
```

A combined conjugate gradient solver function.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'NULL' for global functions.
	solver↩ _id	Solver type used to solve the linear system. The default value is LCG_CGS.

Returns

Status of the function.

4.14.2.3 lcg_solver_constrained()

```
int lcg_solver_constrained (
            lcg_axfunc_ptr Afp,
            lcg_progress_ptr Pfp,
            lcg_float * m,
            const lcg_float * B,
            const lcg_float * low,
            const lcg_float * hig,
            const int n_size,
            const lcg_para * param,
            void * instance,
            lcg_solver_enum solver_id = LCG_PG )
```

A combined conjugate gradient solver function with inequality constraints.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*low*	The lower boundary of the acceptable solution.
in	*hig*	The higher boundary of the acceptable solution.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'NULL' for global functions.
	solver↩ _id	Solver type used to solve the linear system. The default value is LCG_CGS.
	P	Precondition vector (optional expect for the LCG_PCG method). The default value is NULL.

Returns

Status of the function.

4.14.2.4 lcg_solver_preconditioned()

```
int lcg_solver_preconditioned (
            lcg_axfunc_ptr Afp,
            lcg_axfunc_ptr Mfp,
            lcg_progress_ptr Pfp,
            lcg_float * m,
            const lcg_float * B,
            const int n_size,
            const lcg_para * param,
            void * instance,
            lcg_solver_enum solver_id = LCG_PCG )
```

A combined conjugate gradient solver function.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Mfp*	Callback function for calculating the product of 'M^{-1}x', in which M is the preconditioning matrix.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'NULL' for global functions.
	solver↩_id	Solver type used to solve the linear system. The default value is LCG_PCG.

Returns

Status of the function.

4.14.2.5 lcgs()

```
int lcgs (
            lcg_axfunc_ptr Afp,
            lcg_progress_ptr Pfp,
            lcg_float * m,
            const lcg_float * B,
            const int n_size,
            const lcg_para * param,
```

```
void * instance,
lcg_float * RK = nullptr,
lcg_float * R0T = nullptr,
lcg_float * PK = nullptr,
lcg_float * AX = nullptr,
lcg_float * UK = nullptr,
lcg_float * QK = nullptr,
lcg_float * WK = nullptr )
```

Standalone function of the Conjugate Gradient Squared algorithm.

Note

Algorithm 2 in "Generalized conjugate gradient method" by Fokkema et al. (1996).

To use the lcgs() function for massive inversions, it is better to provide external vectors RK, R0T, PK, AX, UK, QK, and WK to avoid allocating and destroying temporary vectors.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'nullptr' for global functions.
	RK	Intermediate vector of the size n_size. If this pointer is null, the function will create an internal vector instead.
	R0T	Intermediate vector of the size n_size. If this pointer is null, the function will create an internal vector instead.
	PK	Intermediate vector of the size n_size. If this pointer is null, the function will create an internal vector instead.
	AX	Intermediate vector of the size n_size. If this pointer is null, the function will create an internal vector instead.
	UK	Intermediate vector of the size n_size. If this pointer is null, the function will create an internal vector instead.
	QK	Intermediate vector of the size n_size. If this pointer is null, the function will create an internal vector instead.
	WK	Intermediate vector of the size n_size. If this pointer is null, the function will create an internal vector instead.

Returns

Status of the function.

4.15 lcg_complex.cpp File Reference

```
#include "cmath"
#include "ctime"
#include "random"
#include "lcg_complex.h"
#include "omp.h"
```

Functions

- lcg_complex ∗ clcg_malloc (int n)
    Locate memory for a lcg_complex pointer type.
- lcg_complex ∗∗ clcg_malloc (int m, int n)
    Locate memory for a lcg_complex second pointer type.
- void clcg_free (lcg_complex ∗x)
    Destroy memory used by the lcg_complex type array.
- void clcg_free (lcg_complex ∗∗x, int m)
    Destroy memory used by the 2D lcg_complex type array.
- void clcg_vecset (lcg_complex ∗a, lcg_complex b, int size)
    set a complex vector's value
- void clcg_vecset (lcg_complex ∗∗a, lcg_complex b, int m, int n)
    set a 2d complex vector's value
- void clcg_set (lcg_complex ∗a, lcg_float r, lcg_float i)
    setup a complex number
- lcg_float clcg_square (const lcg_complex ∗a)
    Calculate the squared module of a complex number.
- lcg_float clcg_module (const lcg_complex ∗a)
    Calculate the module of a complex number.
- lcg_complex clcg_conjugate (const lcg_complex ∗a)
    Calculate the conjugate of a complex number.
- void clcg_vecrnd (lcg_complex ∗a, lcg_complex l, lcg_complex h, int size)
    set a complex vector using random values
- void clcg_vecrnd (lcg_complex ∗∗a, lcg_complex l, lcg_complex h, int m, int n)
    set a 2D complex vector using random values
- void clcg_dot (lcg_complex &ret, const lcg_complex ∗a, const lcg_complex ∗b, int size)
    calculate dot product of two complex vectors
- void clcg_inner (lcg_complex &ret, const lcg_complex ∗a, const lcg_complex ∗b, int size)
    calculate inner product of two complex vectors
- void clcg_matvec (lcg_complex ∗∗A, const lcg_complex ∗x, lcg_complex ∗Ax, int m_size, int n_size, lcg_matrix_e layout, clcg_complex_e conjugate)
    calculate product of a complex matrix and a complex vector

4.15.1 Function Documentation

4.15.1.1 clcg_conjugate()

```
lcg_complex clcg_conjugate (
            const lcg_complex * a )
```

Calculate the conjugate of a complex number.

Returns

The complex conjugate.

4.15.1.2 clcg_dot()

```
void clcg_dot (
            lcg_complex & ret,
            const lcg_complex * a,
            const lcg_complex * b,
            int size )
```

calculate dot product of two complex vectors

the product of two complex vectors are defined as <a, b> = \sum{a_i \cdot b_i}

Parameters

in	*a*	complex vector a
in	*b*	complex vector b
in	*x_size*	size of the vector

Returns

product

4.15.1.3 clcg_free() [1/2]

```
void clcg_free (
            lcg_complex ** x,
            int m )
```

Destroy memory used by the 2D lcg_complex type array.

Parameters

x	Pointer of the array.

4.15.1.4 clcg_free() [2/2]

```
void clcg_free (
            lcg_complex * x )
```

Destroy memory used by the lcg_complex type array.

Parameters

x	Pointer of the array.

4.15.1.5 clcg_inner()

```
void clcg_inner (
            lcg_complex & ret,
            const lcg_complex * a,
            const lcg_complex * b,
            int size )
```

calculate inner product of two complex vectors

the product of two complex vectors are defined as <a, b> = \sum{\bar{a_i} \cdot b_i}

Parameters

in	*a*	complex vector a
in	*b*	complex vector b
in	*x_size*	size of the vector

Returns

product

4.15.1.6 clcg_malloc() [1/2]

```
lcg_complex** clcg_malloc (
            int m,
            int n )
```

Locate memory for a lcg_complex second pointer type.

Parameters

in	*n*	Size of the lcg_float array.

Returns

Pointer of the array's location.

4.15.1.7 clcg_malloc() [2/2]

```
lcg_complex* clcg_malloc (
            int n )
```

Locate memory for a lcg_complex pointer type.

Parameters

in	*n*	Size of the lcg_float array.

Returns

Pointer of the array's location.

4.15.1.8 clcg_matvec()

```
void clcg_matvec (
            lcg_complex ** A,
            const lcg_complex * x,
            lcg_complex * Ax,
            int m_size,
            int n_size,
            lcg_matrix_e layout = MatNormal,
            clcg_complex_e conjugate = NonConjugate )
```

calculate product of a complex matrix and a complex vector

the product of two complex vectors are defined as <a, b> = \sum{\bar{a_i}\cdot\b_i}. Different configurations: layout=Normal,conjugate=false -> A layout=Transpose,conjugate=false -> A^T layout=Normal,conjugate=true -> \bar{A} layout=Transpose,conjugate=true -> A^H

Parameters

	A	complex matrix A
in	*x*	complex vector x
	Ax	product of Ax
in	*m_size*	row size of A
in	*n_size*	column size of A
in	*layout*	layout of A used for multiplication. Must be Normal or Transpose
in	*conjugate*	whether to use the complex conjugate of A for calculation

4.15.1.9 clcg_module()

```
lcg_float clcg_module (
            const lcg_complex * a )
```

Calculate the module of a complex number.

Returns

The module

4.15.1.10 clcg_set()

```
void clcg_set (
            lcg_complex * a,
            lcg_float r,
            lcg_float i )
```

setup a complex number

Parameters

in	*r*	The real part of the complex number
in	*i*	The imaginary part of the complex number

4.15.1.11 clcg_square()

```
lcg_float clcg_square (
            const lcg_complex * a )
```

Calculate the squared module of a complex number.

Returns

The module

4.15.1.12 clcg_vecrnd() [1/2]

```
void clcg_vecrnd (
            lcg_complex ** a,
            lcg_complex l,
            lcg_complex h,
            int m,
            int n )
```

set a 2D complex vector using random values

Parameters

	a	pointer of the vector
in	*l*	the lower bound of random values
in	*h*	the higher bound of random values
in	*m*	row size of the vector
in	*n*	column size of the vector

4.15.1.13 clcg_vecrnd() [2/2]

```
void clcg_vecrnd (
            lcg_complex * a,
            lcg_complex l,
            lcg_complex h,
            int size )
```

set a complex vector using random values

Parameters

	a	pointer of the vector
in	*l*	the lower bound of random values
in	*h*	the higher bound of random values
in	*size*	size of the vector

4.15.1.14 clcg_vecset() [1/2]

```
void clcg_vecset (
            lcg_complex ** a,
            lcg_complex b,
            int m,
            int n )
```

set a 2d complex vector's value

Parameters

	a	pointer of the matrix
in	*b*	initial value
in	*m*	row size of the matrix
in	*n*	column size of the matrix

4.15.1.15 clcg_vecset() [2/2]

```
void clcg_vecset (
            lcg_complex * a,
            lcg_complex b,
            int size )
```

set a complex vector's value

Parameters

	a	pointer of the vector
in	*b*	initial value
in	*size*	vector size

4.16 lcg_complex.h File Reference

```
#include "iostream"
#include "algebra.h"
#include "complex"
```

Typedefs

- typedef std::complex< lcg_float > lcg_complex

Functions

- lcg_complex ∗ clcg_malloc (int n)

  Locate memory for a lcg_complex pointer type.
- lcg_complex ∗∗ clcg_malloc (int m, int n)

  Locate memory for a lcg_complex second pointer type.
- void clcg_free (lcg_complex ∗x)

  Destroy memory used by the lcg_complex type array.
- void clcg_free (lcg_complex ∗∗x, int m)

  Destroy memory used by the 2D lcg_complex type array.
- void clcg_vecset (lcg_complex ∗a, lcg_complex b, int size)

  set a complex vector's value
- void clcg_vecset (lcg_complex ∗∗a, lcg_complex b, int m, int n)

  set a 2d complex vector's value
- void clcg_set (lcg_complex ∗a, lcg_float r, lcg_float i)

  setup a complex number
- lcg_float clcg_square (const lcg_complex ∗a)

  Calculate the squared module of a complex number.
- lcg_float clcg_module (const lcg_complex ∗a)

  Calculate the module of a complex number.
- lcg_complex clcg_conjugate (const lcg_complex ∗a)

  Calculate the conjugate of a complex number.
- void clcg_vecrnd (lcg_complex ∗a, lcg_complex l, lcg_complex h, int size)

  set a complex vector using random values
- void clcg_vecrnd (lcg_complex ∗∗a, lcg_complex l, lcg_complex h, int m, int n)

  set a 2D complex vector using random values
- void clcg_dot (lcg_complex &ret, const lcg_complex ∗a, const lcg_complex ∗b, int size)

  calculate dot product of two complex vectors
- void clcg_inner (lcg_complex &ret, const lcg_complex ∗a, const lcg_complex ∗b, int size)

  calculate inner product of two complex vectors
- void clcg_matvec (lcg_complex ∗∗A, const lcg_complex ∗x, lcg_complex ∗Ax, int m_size, int n_size, lcg_matrix_e layout=MatNormal, clcg_complex_e conjugate=NonConjugate)

  calculate product of a complex matrix and a complex vector

4.16.1 Typedef Documentation

4.16.1.1 lcg_complex

```
typedef std::complex<lcg_float> lcg_complex
```

4.16.2 Function Documentation

4.16.2.1 clcg_conjugate()

```
lcg_complex clcg_conjugate (
            const lcg_complex * a )
```

Calculate the conjugate of a complex number.

Returns

The complex conjugate.

4.16.2.2 clcg_dot()

```
void clcg_dot (
            lcg_complex & ret,
            const lcg_complex * a,
            const lcg_complex * b,
            int size )
```

calculate dot product of two complex vectors

the product of two complex vectors are defined as <a, b> = \sum{a_i \cdot b_i}

Parameters

in	*a*	complex vector a
in	*b*	complex vector b
in	*x_size*	size of the vector

Returns

product

4.16.2.3 clcg_free() [1/2]

```
void clcg_free (
```

```
            lcg_complex ** x,
            int m )
```

Destroy memory used by the 2D lcg_complex type array.

Parameters

x	Pointer of the array.

4.16.2.4 clcg_free() [2/2]

```
void clcg_free (
            lcg_complex * x )
```

Destroy memory used by the lcg_complex type array.

Parameters

x	Pointer of the array.

4.16.2.5 clcg_inner()

```
void clcg_inner (
            lcg_complex & ret,
            const lcg_complex * a,
            const lcg_complex * b,
            int size )
```

calculate inner product of two complex vectors

the product of two complex vectors are defined as <a, b> = \sum{\bar{a_i} \cdot b_i}

Parameters

in	*a*	complex vector a
in	*b*	complex vector b
in	*x_size*	size of the vector

Returns

product

4.16.2.6 clcg_malloc() [1/2]

```
lcg_complex** clcg_malloc (
              int m,
              int n )
```

Locate memory for a lcg_complex second pointer type.

Parameters

in	*n*	Size of the lcg_float array.

Returns

Pointer of the array's location.

4.16.2.7 clcg_malloc() [2/2]

```
lcg_complex* clcg_malloc (
              int n )
```

Locate memory for a lcg_complex pointer type.

Parameters

in	*n*	Size of the lcg_float array.

Returns

Pointer of the array's location.

4.16.2.8 clcg_matvec()

```
void clcg_matvec (
              lcg_complex ** A,
              const lcg_complex * x,
              lcg_complex * Ax,
              int m_size,
              int n_size,
              lcg_matrix_e layout = MatNormal,
              clcg_complex_e conjugate = NonConjugate )
```

calculate product of a complex matrix and a complex vector

the product of two complex vectors are defined as <a, b> = \sum{\bar{a_i}\cdot\b_i}. Different configurations: layout=Normal,conjugate=false -> A layout=Transpose,conjugate=false -> A^T layout=Normal,conjugate=true -> \bar{A} layout=Transpose,conjugate=true -> A^H

Parameters

	A	complex matrix A
in	*x*	complex vector x
	Ax	product of Ax
in	*m_size*	row size of A
in	*n_size*	column size of A
in	*layout*	layout of A used for multiplication. Must be Normal or Transpose
in	*conjugate*	whether to use the complex conjugate of A for calculation

4.16.2.9 clcg_module()

```
lcg_float clcg_module (
            const lcg_complex * a )
```

Calculate the module of a complex number.

Returns

The module

4.16.2.10 clcg_set()

```
void clcg_set (
            lcg_complex * a,
            lcg_float r,
            lcg_float i )
```

setup a complex number

Parameters

in	*r*	The real part of the complex number
in	*i*	The imaginary part of the complex number

4.16.2.11 clcg_square()

```
lcg_float clcg_square (
            const lcg_complex * a )
```

Calculate the squared module of a complex number.

Returns

The module

4.16.2.12 clcg_vecrnd() [1/2]

```
void clcg_vecrnd (
            lcg_complex ** a,
            lcg_complex l,
            lcg_complex h,
            int m,
            int n )
```

set a 2D complex vector using random values

Parameters

	a	pointer of the vector
in	*l*	the lower bound of random values
in	*h*	the higher bound of random values
in	*m*	row size of the vector
in	*n*	column size of the vector

4.16.2.13 clcg_vecrnd() [2/2]

```
void clcg_vecrnd (
            lcg_complex * a,
            lcg_complex l,
            lcg_complex h,
            int size )
```

set a complex vector using random values

Parameters

	a	pointer of the vector
in	*l*	the lower bound of random values
in	*h*	the higher bound of random values
in	*size*	size of the vector

4.16.2.14 clcg_vecset() [1/2]

```
void clcg_vecset (
            lcg_complex ** a,
```

```
            lcg_complex b,
            int m,
            int n )
```

set a 2d complex vector's value

Parameters

	a	pointer of the matrix
in	*b*	initial value
in	*m*	row size of the matrix
in	*n*	column size of the matrix

4.16.2.15 clcg_vecset() [2/2]

```
void clcg_vecset (
            lcg_complex * a,
            lcg_complex b,
            int size )
```

set a complex vector's value

Parameters

	a	pointer of the vector
in	*b*	initial value
in	*size*	vector size

4.17 lcg_complex_cuda.h File Reference

```
#include "lcg_complex.h"
#include <cuda_runtime.h>
#include <cuComplex.h>
```

Functions

- lcg_complex cuda2lcg_complex (cuDoubleComplex a)

  Convert cuda complex number to lcg complex number.
- cuDoubleComplex lcg2cuda_complex (lcg_complex a)

  Convert lcg complex number to CUDA complex number.
- cuDoubleComplex ∗ clcg_malloc_cuda (size_t n)

  Locate memory for a cuDoubleComplex pointer type.
- void clcg_free_cuda (cuDoubleComplex ∗x)

  Destroy memory used by the cuDoubleComplex type array.
- void clcg_vecset_cuda (cuDoubleComplex ∗a, cuDoubleComplex b, size_t size)

set a complex vector's value

- cuComplex clcg_Cscale (lcg_float s, cuComplex a)

  Host side function for scale a cuDoubleComplex object.
- cuComplex clcg_Csum (cuComplex a, cuComplex b)

  Calculate the sum of two cuda complex number. This is a host side function.
- cuComplex clcg_Cdiff (cuComplex a, cuComplex b)

  Calculate the difference of two cuda complex number. This is a host side function.
- cuComplex clcg_Csqrt (cuComplex a)

  Calculate the sqrt() of a cuda complex number.
- cuDoubleComplex clcg_Zscale (lcg_float s, cuDoubleComplex a)

  Host side function for scale a cuDoubleComplex object.
- cuDoubleComplex clcg_Zsum (cuDoubleComplex a, cuDoubleComplex b)

  Calculate the sum of two cuda complex number. This is a host side function.
- cuDoubleComplex clcg_Zdiff (cuDoubleComplex a, cuDoubleComplex b)

  Calculate the difference of two cuda complex number. This is a host side function.
- cuDoubleComplex clcg_Zsqrt (cuDoubleComplex a)

  Calculate the sqrt() of a cuda complex number.
- void clcg_smCcoo_row2col (const int ∗A_row, const int ∗A_col, const cuComplex ∗A, int N, int nz, int ∗Ac↩ _row, int ∗Ac_col, cuComplex ∗Ac_val)

  Convert the indexing sequence of a sparse matrix from the row-major to col-major format.
- void clcg_smZcoo_row2col (const int ∗A_row, const int ∗A_col, const cuDoubleComplex ∗A, int N, int nz, int ∗Ac_row, int ∗Ac_col, cuDoubleComplex ∗Ac_val)

  Convert the indexing sequence of a sparse matrix from the row-major to col-major format.
- void clcg_smCcsr_get_diagonal (const int ∗A_ptr, const int ∗A_col, const cuComplex ∗A_val, const int A_len, cuComplex ∗A_diag, int bk_size=1024)

  Extract diagonal elements from a square CUDA sparse matrix that is formatted in the CSR format.
- void clcg_smZcsr_get_diagonal (const int ∗A_ptr, const int ∗A_col, const cuDoubleComplex ∗A_val, const int A_len, cuDoubleComplex ∗A_diag, int bk_size=1024)

  Extract diagonal elements from a square CUDA sparse matrix that is formatted in the CSR format.
- void clcg_vecMvecC_element_wise (const cuComplex ∗a, const cuComplex ∗b, cuComplex ∗c, int n, int bk↩ _size=1024)

  Element-wise muplication between two CUDA arries.
- void clcg_vecMvecZ_element_wise (const cuDoubleComplex ∗a, const cuDoubleComplex ∗b, cuDouble↩ Complex ∗c, int n, int bk_size=1024)

  Element-wise muplication between two CUDA arries.
- void clcg_vecDvecC_element_wise (const cuComplex ∗a, const cuComplex ∗b, cuComplex ∗c, int n, int bk↩ _size=1024)

  Element-wise division between two CUDA arries.
- void clcg_vecDvecZ_element_wise (const cuDoubleComplex ∗a, const cuDoubleComplex ∗b, cuDouble↩ Complex ∗c, int n, int bk_size=1024)

  Element-wise division between two CUDA arries.
- void clcg_vecC_conjugate (const cuComplex ∗a, cuComplex ∗ca, int n, int bk_size=1024)

  Return complex conjugates of an input CUDA complex array.
- void clcg_vecZ_conjugate (const cuDoubleComplex ∗a, cuDoubleComplex ∗ca, int n, int bk_size=1024)

  Return complex conjugates of an input CUDA complex array.

4.17.1 Function Documentation

4.17.1.1 clcg_Cdiff()

```
cuComplex clcg_Cdiff (
            cuComplex a,
            cuComplex b )
```

Calculate the difference of two cuda complex number. This is a host side function.

Parameters

a	Complex number
b	Complex number

Returns

cuComplex Difference of the input complex number

4.17.1.2 clcg_Cscale()

```
cuComplex clcg_Cscale (
            lcg_float s,
            cuComplex a )
```

Host side function for scale a cuDoubleComplex object.

Parameters

s	scale factor
a	Complex number

Returns

cuComplex scaled complex number

4.17.1.3 clcg_Csqrt()

```
cuComplex clcg_Csqrt (
            cuComplex a )
```

Calculate the sqrt() of a cuda complex number.

Parameters

a	Complex number

Returns

cuComplex root value

4.17.1.4 clcg_Csum()

```
cuComplex clcg_Csum (
            cuComplex a,
            cuComplex b )
```

Calculate the sum of two cuda complex number. This is a host side function.

Parameters

a	Complex number
b	Complex number

Returns

cuComplex Sum of the input complex number

4.17.1.5 clcg_free_cuda()

```
void clcg_free_cuda (
            cuDoubleComplex * x )
```

Destroy memory used by the cuDoubleComplex type array.

Parameters

x	Pointer of the array.

4.17.1.6 clcg_malloc_cuda()

```
cuDoubleComplex* clcg_malloc_cuda (
            size_t n )
```

Locate memory for a cuDoubleComplex pointer type.

Parameters

in	*n*	Size of the lcg_float array.

Returns

Pointer of the array's location.

4.17.1.7 clcg_smCcoo_row2col()

```
void clcg_smCcoo_row2col (
            const int * A_row,
            const int * A_col,
            const cuComplex * A,
            int N,
            int nz,
            int * Ac_row,
            int * Ac_col,
            cuComplex * Ac_val )
```

Convert the indexing sequence of a sparse matrix from the row-major to col-major format.

Note

The sparse matrix is stored in the COO foramt. This is a host side function.

Parameters

A_row	Row index
A_col	Column index
A	Non-zero values of the matrix
N	Row/column length of A
nz	Number of the non-zero values in A
Ac_row	Output row index
Ac_col	Output column index
Ac_val	Non-zero values of the output matrix

4.17.1.8 clcg_smCcsr_get_diagonal()

```
void clcg_smCcsr_get_diagonal (
            const int * A_ptr,
            const int * A_col,
            const cuComplex * A_val,
            const int A_len,
            cuComplex * A_diag,
            int bk_size = 1024 )
```

Extract diagonal elements from a square CUDA sparse matrix that is formatted in the CSR format.

Note

This is a device side function. All memories must be allocated on the GPU device.

Parameters

in	*A_ptr*	Row index pointer
in	*A_col*	Column index
in	*A_val*	Non-zero values of the matrix
in	*A_len*	Dimension of the matrix
	A_diag	Output digonal elements
in	*bk_size*	Default CUDA block size.

4.17.1.9 clcg_smZcoo_row2col()

```
void clcg_smZcoo_row2col (
            const int * A_row,
            const int * A_col,
            const cuDoubleComplex * A,
            int N,
            int nz,
            int * Ac_row,
            int * Ac_col,
            cuDoubleComplex * Ac_val )
```

Convert the indexing sequence of a sparse matrix from the row-major to col-major format.

Note

The sparse matrix is stored in the COO foramt. This is a host side function.

Parameters

A_row	Row index
A_col	Column index
A	Non-zero values of the matrix
N	Row/column length of A
nz	Number of the non-zero values in A
Ac_row	Output row index
Ac_col	Output column index
Ac_val	Non-zero values of the output matrix

4.17.1.10 clcg_smZcsr_get_diagonal()

```
void clcg_smZcsr_get_diagonal (
            const int * A_ptr,
            const int * A_col,
            const cuDoubleComplex * A_val,
            const int A_len,
```

```
            cuDoubleComplex * A_diag,
            int bk_size = 1024 )
```

Extract diagonal elements from a square CUDA sparse matrix that is formatted in the CSR format.

Note

This is a device side function. All memories must be allocated on the GPU device.

Parameters

in	*A_ptr*	Row index pointer
in	*A_col*	Column index
in	*A_val*	Non-zero values of the matrix
in	*A_len*	Dimension of the matrix
	A_diag	Output digonal elements
in	*bk_size*	Default CUDA block size.

4.17.1.11 clcg_vecC_conjugate()

```
void clcg_vecC_conjugate (
            const cuComplex * a,
            cuComplex * ca,
            int n,
            int bk_size = 1024 )
```

Return complex conjugates of an input CUDA complex array.

Parameters

	a	Pointer of the input arra
	ca	Pointer of the output array
	n	Length of the arraies
in	*bk_size*	Default CUDA block size.

4.17.1.12 clcg_vecDvecC_element_wise()

```
void clcg_vecDvecC_element_wise (
            const cuComplex * a,
            const cuComplex * b,
            cuComplex * c,
            int n,
            int bk_size = 1024 )
```

Element-wise division between two CUDA arries.

Note

This is a device side function. All memories must be allocated on the GPU device.

Parameters

in	*a*	Pointer of the input array
in	*b*	Pointer of the input array
	c	Pointer of the output array
in	*n*	Length of the arraies
in	*bk_size*	Default CUDA block size.

4.17.1.13 clcg_vecDvecZ_element_wise()

```
void clcg_vecDvecZ_element_wise (
            const cuDoubleComplex * a,
            const cuDoubleComplex * b,
            cuDoubleComplex * c,
            int n,
            int bk_size = 1024 )
```

Element-wise division between two CUDA arries.

Note

This is a device side function. All memories must be allocated on the GPU device.

Parameters

in	*a*	Pointer of the input array
in	*b*	Pointer of the input array
	c	Pointer of the output array
in	*n*	Length of the arraies
in	*bk_size*	Default CUDA block size.

4.17.1.14 clcg_vecMvecC_element_wise()

```
void clcg_vecMvecC_element_wise (
            const cuComplex * a,
            const cuComplex * b,
            cuComplex * c,
            int n,
            int bk_size = 1024 )
```

Element-wise muplication between two CUDA arries.

Note

This is a device side function. All memories must be allocated on the GPU device.

Parameters

in	*a*	Pointer of the input array
in	*b*	Pointer of the input array
	c	Pointer of the output array
in	*n*	Length of the arraies
in	*bk_size*	Default CUDA block size.

4.17.1.15 clcg_vecMvecZ_element_wise()

```
void clcg_vecMvecZ_element_wise (
            const cuDoubleComplex * a,
            const cuDoubleComplex * b,
            cuDoubleComplex * c,
            int n,
            int bk_size = 1024 )
```

Element-wise muplication between two CUDA arries.

Note

This is a device side function. All memories must be allocated on the GPU device.

Parameters

in	*a*	Pointer of the input array
in	*b*	Pointer of the input array
	c	Pointer of the output array
in	*n*	Length of the arraies
in	*bk_size*	Default CUDA block size.

4.17.1.16 clcg_vecset_cuda()

```
void clcg_vecset_cuda (
            cuDoubleComplex * a,
            cuDoubleComplex b,
            size_t size )
```

set a complex vector's value

Parameters

	a	pointer of the vector
in	*b*	initial value
in	*size*	vector size

4.17.1.17 clcg_vecZ_conjugate()

```
void clcg_vecZ_conjugate (
            const cuDoubleComplex * a,
            cuDoubleComplex * ca,
            int n,
            int bk_size = 1024 )
```

Return complex conjugates of an input CUDA complex array.

Parameters

	a	Pointer of the input arra
	ca	Pointer of the output array
	n	Length of the arraies
in	*bk_size*	Default CUDA block size.

4.17.1.18 clcg_Zdiff()

```
cuDoubleComplex clcg_Zdiff (
            cuDoubleComplex a,
            cuDoubleComplex b )
```

Calculate the difference of two cuda complex number. This is a host side function.

Parameters

a	Complex number
b	Complex number

Returns

cuDoubleComplex Difference of the input complex number

4.17.1.19 clcg_Zscale()

```
cuDoubleComplex clcg_Zscale (
            lcg_float s,
            cuDoubleComplex a )
```

Host side function for scale a cuDoubleComplex object.

Parameters

s	scale factor
a	Complex number

Returns

cuDoubleComplex scaled complex number

4.17.1.20 clcg_Zsqrt()

```
cuDoubleComplex clcg_Zsqrt (
            cuDoubleComplex a )
```

Calculate the sqrt() of a cuda complex number.

Parameters

a	Complex number

Returns

cuDoubleComplex root value

4.17.1.21 clcg_Zsum()

```
cuDoubleComplex clcg_Zsum (
            cuDoubleComplex a,
            cuDoubleComplex b )
```

Calculate the sum of two cuda complex number. This is a host side function.

Parameters

a	Complex number
b	Complex number

Returns

cuDoubleComplex Sum of the input complex number

4.17.1.22 cuda2lcg_complex()

```
lcg_complex cuda2lcg_complex (
             cuDoubleComplex a )
```

Convert cuda complex number to lcg complex number.

Parameters

a	CUDA complex number

Returns

lcg_complex lcg complex number

4.17.1.23 lcg2cuda_complex()

```
cuDoubleComplex lcg2cuda_complex (
             lcg_complex a )
```

Convert lcg complex number to CUDA complex number.

Parameters

a	lcg complex number

Returns

cuDoubleComplex CUDA complex number

4.18 lcg_cuda.h File Reference

```
#include "util.h"
#include "algebra_cuda.h"
#include <cublas_v2.h>
#include <cusparse_v2.h>
```

Typedefs

- typedef void(∗ lcg_axfunc_cuda_ptr) (void ∗instance, cublasHandle_t cub_handle, cusparseHandle_t cus_↩ handle, cusparseDnVecDescr_t x, cusparseDnVecDescr_t prod_Ax, const int n_size, const int nz_size)

  Callback interface for calculating the product of a N∗N matrix 'A' multiplied by a vertical vector 'x'. Note that both A and x are hosted on the GPU device.
- typedef int(∗ lcg_progress_cuda_ptr) (void ∗instance, const lcg_float ∗m, const lcg_float converge, const lcg_para ∗param, const int n_size, const int nz_size, const int k)

  Callback interface for monitoring the progress and terminate the iteration if necessary. Note that m is hosted on the GPU device.

Functions

- int lcg_solver_cuda (lcg_axfunc_cuda_ptr Afp, lcg_progress_cuda_ptr Pfp, lcg_float ∗m, const lcg_float ∗B, const int n_size, const int nz_size, const lcg_para ∗param, void ∗instance, cublasHandle_t cub_handle, cusparseHandle_t cus_handle, lcg_solver_enum solver_id=LCG_CG)

  A combined conjugate gradient solver function. Note that both m and B are hosted on the GPU device.
- int lcg_solver_preconditioned_cuda (lcg_axfunc_cuda_ptr Afp, lcg_axfunc_cuda_ptr Mfp, lcg_progress_cuda_ptr Pfp, lcg_float ∗m, const lcg_float ∗B, const int n_size, const int nz_size, const lcg_para ∗param, void ∗instance, cublasHandle_t cub_handle, cusparseHandle_t cus_handle, lcg_solver_enum solver_↩ id=LCG_PCG)

  A combined conjugate gradient solver function. Note that both m and B are hosted on the GPU device.
- int lcg_solver_constrained_cuda (lcg_axfunc_cuda_ptr Afp, lcg_progress_cuda_ptr Pfp, lcg_float ∗m, const lcg_float ∗B, const lcg_float ∗low, const lcg_float ∗hig, const int n_size, const int nz_size, const lcg_para ∗param, void ∗instance, cublasHandle_t cub_handle, cusparseHandle_t cus_handle, lcg_solver_enum solver_id=LCG_PG)

  A combined conjugate gradient solver function. Note that both m and B are hosted on the GPU device.

4.18.1 Typedef Documentation

4.18.1.1 lcg_axfunc_cuda_ptr

```
typedef void(∗ lcg_axfunc_cuda_ptr) (void ∗instance, cublasHandle_t cub_handle, cusparse↩
Handle_t cus_handle, cusparseDnVecDescr_t x, cusparseDnVecDescr_t prod_Ax, const int n_size,
const int nz_size)
```

Callback interface for calculating the product of a N∗N matrix 'A' multiplied by a vertical vector 'x'. Note that both A and x are hosted on the GPU device.

Parameters

instance	The user data sent for the lcg_solver_cuda() functions by the client.
cub_handle	Handler of the cublas object.
cus_handle	Handlee of the cusparse object.
x	Multiplier of the Ax product.
Ax	Product of A multiplied by x.
n_size	Size of x and column/row numbers of A.

4.18.1.2 lcg_progress_cuda_ptr

```
typedef int(* lcg_progress_cuda_ptr) (void *instance, const lcg_float *m, const lcg_float converge,
const lcg_para *param, const int n_size, const int nz_size, const int k)
```

Callback interface for monitoring the progress and terminate the iteration if necessary. Note that m is hosted on the GPU device.

Parameters

instance	The user data sent for the lcg_solver() functions by the client.
m	The current solutions.
converge	The current value evaluating the iteration progress.
n_size	The size of the variables
k	The iteration count.

Return values

int	Zero to continue the optimization process. Returning a non-zero value will terminate the optimization process.

4.18.2 Function Documentation

4.18.2.1 lcg_solver_constrained_cuda()

```
int lcg_solver_constrained_cuda (
            lcg_axfunc_cuda_ptr Afp,
            lcg_progress_cuda_ptr Pfp,
            lcg_float * m,
            const lcg_float * B,
            const lcg_float * low,
            const lcg_float * hig,
            const int n_size,
            const int nz_size,
            const lcg_para * param,
            void * instance,
            cublasHandle_t cub_handle,
            cusparseHandle_t cus_handle,
            lcg_solver_enum solver_id = LCG_PG )
```

A combined conjugate gradient solver function. Note that both m and B are hosted on the GPU device.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Mfp*	Callback function for calculating the product of 'Mx' for preconditioning.

Parameters

in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	low	Lower bound of the acceptable solution.
	hig	Higher bound of the acceptable solution.
	B	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
in	*nz_size*	Size of the non-zero element of a cusparse object.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client.
	cub_handle	Handler of the cublas object.
	cus_handle	Handlee of the cusparse object. This variable is either 'this' for class member functions or 'NULL' for global functions.
	solver_id	Solver type used to solve the linear system. The default value is LCG_CGS.

Returns

Status of the function.

4.18.2.2 lcg_solver_cuda()

```
int lcg_solver_cuda (
            lcg_axfunc_cuda_ptr Afp,
            lcg_progress_cuda_ptr Pfp,
            lcg_float * m,
            const lcg_float * B,
            const int n_size,
            const int nz_size,
            const lcg_para * param,
            void * instance,
            cublasHandle_t cub_handle,
            cusparseHandle_t cus_handle,
            lcg_solver_enum solver_id = LCG_CG )
```

A combined conjugate gradient solver function. Note that both m and B are hosted on the GPU device.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client.
	cub_handle	Handler of the cublas object.
	cus_handle	Handlee of the cusparse object. This variable is either 'this' for class member functions or 'NULL' for global functions.
	solver_id	Solver type used to solve the linear system. The default value is LCG_CGS.

Returns

Status of the function.

4.18.2.3 lcg_solver_preconditioned_cuda()

```
int lcg_solver_preconditioned_cuda (
            lcg_axfunc_cuda_ptr Afp,
            lcg_axfunc_cuda_ptr Mfp,
            lcg_progress_cuda_ptr Pfp,
            lcg_float * m,
            const lcg_float * B,
            const int n_size,
            const int nz_size,
            const lcg_para * param,
            void * instance,
            cublasHandle_t cub_handle,
            cusparseHandle_t cus_handle,
            lcg_solver_enum solver_id = LCG_PCG )
```

A combined conjugate gradient solver function. Note that both m and B are hosted on the GPU device.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Mfp*	Callback function for calculating the product of 'Mx' for preconditioning.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
in	*nz_size*	Size of the non-zero element of a cusparse object.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client.
	cub_handle	Handler of the cublas object.
	cus_handle	Handlee of the cusparse object. This variable is either 'this' for class member functions or 'NULL' for global functions.
	solver_id	Solver type used to solve the linear system. The default value is LCG_CGS.

Returns

Status of the function.

4.19 lcg_eigen.cpp File Reference

```
#include "lcg_eigen.h"
#include "cmath"
#include "config.h"
#include "omp.h"
```

Typedefs

- typedef int(∗ eigen_solver_ptr) (lcg_axfunc_eigen_ptr Afp, lcg_progress_eigen_ptr Pfp, Eigen::VectorXd &m, const Eigen::VectorXd &B, const lcg_para ∗param, void ∗instance)

    Callback interface of the conjugate gradient solver.
- typedef int(∗ eigen_solver_ptr2) (lcg_axfunc_eigen_ptr Afp, lcg_progress_eigen_ptr Pfp, Eigen::VectorXd &m, const Eigen::VectorXd &B, const Eigen::VectorXd &low, const Eigen::VectorXd &hig, const lcg_para ∗param, void ∗instance)

    A combined conjugate gradient solver function.

Functions

- int lcg (lcg_axfunc_eigen_ptr Afp, lcg_progress_eigen_ptr Pfp, Eigen::VectorXd &m, const Eigen::VectorXd &B, const lcg_para ∗param, void ∗instance)

    Conjugate gradient method.
- int lcgs (lcg_axfunc_eigen_ptr Afp, lcg_progress_eigen_ptr Pfp, Eigen::VectorXd &m, const Eigen::VectorXd &B, const lcg_para ∗param, void ∗instance)

    Conjugate gradient squared method.
- int lbicgstab (lcg_axfunc_eigen_ptr Afp, lcg_progress_eigen_ptr Pfp, Eigen::VectorXd &m, const Eigen::↩ VectorXd &B, const lcg_para ∗param, void ∗instance)

    Biconjugate gradient method.
- int lbicgstab2 (lcg_axfunc_eigen_ptr Afp, lcg_progress_eigen_ptr Pfp, Eigen::VectorXd &m, const Eigen::↩ VectorXd &B, const lcg_para ∗param, void ∗instance)

    Biconjugate gradient method 2.
- int lcg_solver_eigen (lcg_axfunc_eigen_ptr Afp, lcg_progress_eigen_ptr Pfp, Eigen::VectorXd &m, const Eigen::VectorXd &B, const lcg_para ∗param, void ∗instance, lcg_solver_enum solver_id)

    A combined conjugate gradient solver function.
- int lpcg (lcg_axfunc_eigen_ptr Afp, lcg_axfunc_eigen_ptr Mfp, lcg_progress_eigen_ptr Pfp, Eigen::VectorXd &m, const Eigen::VectorXd &B, const lcg_para ∗param, void ∗instance)

    Preconditioned conjugate gradient method.
- int lcg_solver_preconditioned_eigen (lcg_axfunc_eigen_ptr Afp, lcg_axfunc_eigen_ptr Mfp, lcg_progress_eigen_ptr Pfp, Eigen::VectorXd &m, const Eigen::VectorXd &B, const lcg_para ∗param, void ∗instance, lcg_solver_enum solver_id)

    A combined conjugate gradient solver function.
- int lpg (lcg_axfunc_eigen_ptr Afp, lcg_progress_eigen_ptr Pfp, Eigen::VectorXd &m, const Eigen::VectorXd &B, const Eigen::VectorXd &low, const Eigen::VectorXd &hig, const lcg_para ∗param, void ∗instance)

    Conjugate gradient method with projected gradient for inequality constraints.
- int lspg (lcg_axfunc_eigen_ptr Afp, lcg_progress_eigen_ptr Pfp, Eigen::VectorXd &m, const Eigen::VectorXd &B, const Eigen::VectorXd &low, const Eigen::VectorXd &hig, const lcg_para ∗param, void ∗instance)

    Conjugate gradient method with projected gradient for inequality constraints.
- int lcg_solver_constrained_eigen (lcg_axfunc_eigen_ptr Afp, lcg_progress_eigen_ptr Pfp, Eigen::VectorXd &m, const Eigen::VectorXd &B, const Eigen::VectorXd &low, const Eigen::VectorXd &hig, const lcg_para ∗param, void ∗instance, lcg_solver_enum solver_id)

    A combined conjugate gradient solver function with inequality constraints.

4.19.1 Typedef Documentation

4.19.1.1 eigen_solver_ptr

```
typedef int(* eigen_solver_ptr) (lcg_axfunc_eigen_ptr Afp, lcg_progress_eigen_ptr Pfp, Eigen↩
::VectorXd &m, const Eigen::VectorXd &B, const lcg_para *param, void *instance)
```

Callback interface of the conjugate gradient solver.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'nullptr' for global functions.

Returns

Status of the function.

4.19.1.2 eigen_solver_ptr2

```
typedef int(* eigen_solver_ptr2) (lcg_axfunc_eigen_ptr Afp, lcg_progress_eigen_ptr Pfp, Eigen↩
::VectorXd &m, const Eigen::VectorXd &B, const Eigen::VectorXd &low, const Eigen::VectorXd
&hig, const lcg_para *param, void *instance)
```

A combined conjugate gradient solver function.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*low*	The lower boundary of the acceptable solution.
in	*hig*	The higher boundary of the acceptable solution.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'nullptr' for global functions.
	solver↩_id	Solver type used to solve the linear system. The default value is LCG_CGS.
	P	Precondition vector (optional expect for the LCG_PCG method). The default value is nullptr.

Returns

Status of the function.

4.19.2 Function Documentation

4.19.2.1 lbicgstab()

```
int lbicgstab (
            lcg_axfunc_eigen_ptr Afp,
            lcg_progress_eigen_ptr Pfp,
            Eigen::VectorXd & m,
            const Eigen::VectorXd & B,
            const lcg_para * param,
            void * instance )
```

Biconjugate gradient method.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'nullptr' for global functions.
	P	Precondition vector (optional expect for the LCG_PCG method). The default value is nullptr.

Returns

Status of the function.

4.19.2.2 lbicgstab2()

```
int lbicgstab2 (
            lcg_axfunc_eigen_ptr Afp,
            lcg_progress_eigen_ptr Pfp,
            Eigen::VectorXd & m,
            const Eigen::VectorXd & B,
            const lcg_para * param,
            void * instance )
```

Biconjugate gradient method 2.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'nullptr' for global functions.
	P	Precondition vector (optional expect for the LCG_PCG method). The default value is nullptr.

Returns

Status of the function.

4.19.2.3 lcg()

```
int lcg (
            lcg_axfunc_eigen_ptr Afp,
            lcg_progress_eigen_ptr Pfp,
            Eigen::VectorXd & m,
            const Eigen::VectorXd & B,
            const lcg_para * param,
            void * instance )
```

Conjugate gradient method.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'nullptr' for global functions.
	P	Precondition vector (optional expect for the LCG_PCG method). The default value is nullptr.

Returns

Status of the function.

4.19.2.4 lcg_solver_constrained_eigen()

```
int lcg_solver_constrained_eigen (
            lcg_axfunc_eigen_ptr Afp,
            lcg_progress_eigen_ptr Pfp,
            Eigen::VectorXd & m,
            const Eigen::VectorXd & B,
            const Eigen::VectorXd & low,
            const Eigen::VectorXd & hig,
            const lcg_para * param,
            void * instance,
            lcg_solver_enum solver_id = LCG_PG )
```

A combined conjugate gradient solver function with inequality constraints.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*low*	The lower boundary of the acceptable solution.
in	*hig*	The higher boundary of the acceptable solution.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'NULL' for global functions.
	solver_id	Solver type used to solve the linear system. The default value is LCG_CGS.
	P	Precondition vector (optional expect for the LCG_PCG method). The default value is NULL.

Returns

Status of the function.

4.19.2.5 lcg_solver_eigen()

```
int lcg_solver_eigen (
            lcg_axfunc_eigen_ptr Afp,
            lcg_progress_eigen_ptr Pfp,
            Eigen::VectorXd & m,
            const Eigen::VectorXd & B,
            const lcg_para * param,
            void * instance,
            lcg_solver_enum solver_id = LCG_CG )
```

A combined conjugate gradient solver function.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'NULL' for global functions.
	solver_id	Solver type used to solve the linear system. The default value is LCG_CGS.

Returns

Status of the function.

4.19.2.6 lcg_solver_preconditioned_eigen()

```
int lcg_solver_preconditioned_eigen (
            lcg_axfunc_eigen_ptr Afp,
            lcg_axfunc_eigen_ptr Mfp,
            lcg_progress_eigen_ptr Pfp,
            Eigen::VectorXd & m,
            const Eigen::VectorXd & B,
            const lcg_para * param,
            void * instance,
            lcg_solver_enum solver_id = LCG_PCG )
```

A combined conjugate gradient solver function.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Mfp*	Callback function for calculating the product of 'M^{-1}x', in which M is the preconditioning matrix.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'NULL' for global functions.
	solver_id	Solver type used to solve the linear system. The default value is LCG_PCG.

Returns

Status of the function.

4.19.2.7 lcgs()

```
int lcgs (
            lcg_axfunc_eigen_ptr Afp,
            lcg_progress_eigen_ptr Pfp,
            Eigen::VectorXd & m,
            const Eigen::VectorXd & B,
            const lcg_para * param,
            void * instance )
```

Conjugate gradient squared method.

Note

Algorithm 2 in "Generalized conjugate gradient method" by Fokkema et al. (1996).

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'nullptr' for global functions.
	P	Precondition vector (optional expect for the LCG_PCG method). The default value is nullptr.

Returns

Status of the function.

4.19.2.8 lpcg()

```
int lpcg (
            lcg_axfunc_eigen_ptr Afp,
            lcg_axfunc_eigen_ptr Mfp,
            lcg_progress_eigen_ptr Pfp,
            Eigen::VectorXd & m,
            const Eigen::VectorXd & B,
            const lcg_para * param,
            void * instance )
```

Preconditioned conjugate gradient method.

Note

Algorithm 1 in "Preconditioned conjugate gradients for singular systems" by Kaasschieter (1988).

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*n_size*	Size of the solution vector and objective vector.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'nullptr' for global functions.
	P	Precondition vector (optional expect for the LCG_PCG method). The default value is nullptr.

Returns

Status of the function.

4.19.2.9 lpg()

```
int lpg (
            lcg_axfunc_eigen_ptr Afp,
            lcg_progress_eigen_ptr Pfp,
            Eigen::VectorXd & m,
            const Eigen::VectorXd & B,
            const Eigen::VectorXd & low,
            const Eigen::VectorXd & hig,
            const lcg_para * param,
            void * instance )
```

Conjugate gradient method with projected gradient for inequality constraints.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*low*	The lower boundary of the acceptable solution.
in	*hig*	The higher boundary of the acceptable solution.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'nullptr' for global functions.
	solver↩ *_id*	Solver type used to solve the linear system. The default value is LCG_CGS.
	P	Precondition vector (optional expect for the LCG_PCG method). The default value is nullptr.

Returns

Status of the function.

4.19.2.10 lspg()

```
int lspg (
            lcg_axfunc_eigen_ptr Afp,
            lcg_progress_eigen_ptr Pfp,
            Eigen::VectorXd & m,
            const Eigen::VectorXd & B,
            const Eigen::VectorXd & low,
            const Eigen::VectorXd & hig,
            const lcg_para * param,
            void * instance )
```

Conjugate gradient method with projected gradient for inequality constraints.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*low*	The lower boundary of the acceptable solution.
in	*hig*	The higher boundary of the acceptable solution.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'nullptr' for global functions.
	solver←_id	Solver type used to solve the linear system. The default value is LCG_CGS.
	P	Precondition vector (optional expect for the LCG_PCG method). The default value is nullptr.

Returns

Status of the function.

4.20 lcg_eigen.h File Reference

```
#include "util.h"
#include "algebra_eigen.h"
```

Typedefs

- typedef void(∗ lcg_axfunc_eigen_ptr) (void ∗instance, const Eigen::VectorXd &x, Eigen::VectorXd &prod_Ax)

  Callback interface for calculating the product of a N∗N matrix 'A' multiplied by a vertical vector 'x'.
- typedef int(∗ lcg_progress_eigen_ptr) (void ∗instance, const Eigen::VectorXd ∗m, const lcg_float converge, const lcg_para ∗param, const int k)

  Callback interface for monitoring the progress and terminate the iteration if necessary.

Functions

- int lcg_solver_eigen (lcg_axfunc_eigen_ptr Afp, lcg_progress_eigen_ptr Pfp, Eigen::VectorXd &m, const Eigen::VectorXd &B, const lcg_para ∗param, void ∗instance, lcg_solver_enum solver_id=LCG_CG)

  A combined conjugate gradient solver function.
- int lcg_solver_preconditioned_eigen (lcg_axfunc_eigen_ptr Afp, lcg_axfunc_eigen_ptr Mfp, lcg_progress_eigen_ptr Pfp, Eigen::VectorXd &m, const Eigen::VectorXd &B, const lcg_para ∗param, void ∗instance, lcg_solver_enum solver_id=LCG_PCG)

  A combined conjugate gradient solver function.
- int lcg_solver_constrained_eigen (lcg_axfunc_eigen_ptr Afp, lcg_progress_eigen_ptr Pfp, Eigen::VectorXd &m, const Eigen::VectorXd &B, const Eigen::VectorXd &low, const Eigen::VectorXd &hig, const lcg_para ∗param, void ∗instance, lcg_solver_enum solver_id=LCG_PG)

  A combined conjugate gradient solver function with inequality constraints.

4.20.1 Typedef Documentation

4.20.1.1 lcg_axfunc_eigen_ptr

```
typedef void(* lcg_axfunc_eigen_ptr) (void *instance, const Eigen::VectorXd &x, Eigen::VectorXd
&prod_Ax)
```

Callback interface for calculating the product of a N∗N matrix 'A' multiplied by a vertical vector 'x'.

Parameters

instance	The user data sent for the lcg_solver() functions by the client.
x	Multiplier of the Ax product.
Ax	Product of A multiplied by x.

4.20.1.2 lcg_progress_eigen_ptr

```
typedef int(* lcg_progress_eigen_ptr) (void *instance, const Eigen::VectorXd *m, const lcg_float
converge, const lcg_para *param, const int k)
```

Callback interface for monitoring the progress and terminate the iteration if necessary.

Parameters

instance	The user data sent for the lcg_solver() functions by the client.
m	The current solutions.
converge	The current value evaluating the iteration progress.
k	The iteration count.

Return values

int	Zero to continue the optimization process. Returning a non-zero value will terminate the optimization process.

4.20.2 Function Documentation

4.20.2.1 lcg_solver_constrained_eigen()

```
int lcg_solver_constrained_eigen (
            lcg_axfunc_eigen_ptr Afp,
```

```
            lcg_progress_eigen_ptr Pfp,
            Eigen::VectorXd & m,
            const Eigen::VectorXd & B,
            const Eigen::VectorXd & low,
            const Eigen::VectorXd & hig,
            const lcg_para * param,
            void * instance,
            lcg_solver_enum solver_id = LCG_PG )
```

A combined conjugate gradient solver function with inequality constraints.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
in	*low*	The lower boundary of the acceptable solution.
in	*hig*	The higher boundary of the acceptable solution.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'NULL' for global functions.
	solver_id	Solver type used to solve the linear system. The default value is LCG_CGS.
	P	Precondition vector (optional expect for the LCG_PCG method). The default value is NULL.

Returns

Status of the function.

4.20.2.2 lcg_solver_eigen()

```
int lcg_solver_eigen (
            lcg_axfunc_eigen_ptr Afp,
            lcg_progress_eigen_ptr Pfp,
            Eigen::VectorXd & m,
            const Eigen::VectorXd & B,
            const lcg_para * param,
            void * instance,
            lcg_solver_enum solver_id = LCG_CG )
```

A combined conjugate gradient solver function.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'NULL' for global functions.
	solver_id	Solver type used to solve the linear system. The default value is LCG_CGS.

Returns

Status of the function.

4.20.2.3 lcg_solver_preconditioned_eigen()

```
int lcg_solver_preconditioned_eigen (
            lcg_axfunc_eigen_ptr Afp,
            lcg_axfunc_eigen_ptr Mfp,
            lcg_progress_eigen_ptr Pfp,
            Eigen::VectorXd & m,
            const Eigen::VectorXd & B,
            const lcg_para * param,
            void * instance,
            lcg_solver_enum solver_id = LCG_PCG )
```

A combined conjugate gradient solver function.

Parameters

in	*Afp*	Callback function for calculating the product of 'Ax'.
in	*Mfp*	Callback function for calculating the product of 'M^{-1}x', in which M is the preconditioning matrix.
in	*Pfp*	Callback function for monitoring the iteration progress.
	m	Initial solution vector.
	B	Objective vector of the linear system.
	param	Parameter setup for the conjugate gradient methods.
	instance	The user data sent for the lcg_solver() function by the client. This variable is either 'this' for class member functions or 'NULL' for global functions.
	solver↔_id	Solver type used to solve the linear system. The default value is LCG_PCG.

Returns

Status of the function.

4.21 preconditioner.cpp File Reference

```
#include "preconditioner.h"
#include "cmath"
#include "map"
```

Functions

- void lcg_incomplete_Cholesky_half_buffsize_coo (const int *row, const int *col, int nz_size, int *lnz_size)

  Return the number of non-zero elements in the lower triangular part of the input matrix.

- void lcg_incomplete_Cholesky_half_coo (const int ∗row, const int ∗col, const lcg_float ∗val, int N, int nz_size, int lnz_size, int ∗IC_row, int ∗IC_col, lcg_float ∗IC_val)

  Preform the incomplete Cholesky factorization for a sparse matrix that is saved in the COO format.
- void lcg_incomplete_Cholesky_full_coo (const int ∗row, const int ∗col, const lcg_float ∗val, int N, int nz_size, int ∗IC_row, int ∗IC_col, lcg_float ∗IC_val)

  Preform the incomplete Cholesky factorization for a sparse matrix that is saved in the COO format.
- void lcg_solve_upper_triangle_coo (const int ∗row, const int ∗col, const lcg_float ∗U, const lcg_float ∗B, lcg_float ∗x, int N, int nz_size)

  Solve the linear system Ux = B, in which U is a upper triangle matrix.
- void lcg_solve_lower_triangle_coo (const int ∗row, const int ∗col, const lcg_float ∗L, const lcg_float ∗B, lcg_float ∗x, int N, int nz_size)

  Solve the linear system Lx = B, in which L is a lower triangle matrix.
- bool lcg_full_rank_coo (const int ∗row, const int ∗col, const lcg_float ∗M, int N, int nz_size)

  Check to see if a square matrix is full ranked or not. The sparse matrix is stored in the COO format.

4.21.1 Function Documentation

4.21.1.1 lcg_full_rank_coo()

```
bool lcg_full_rank_coo (
            const int * row,
            const int * col,
            const lcg_float * M,
            int N,
            int nz_size )
```

Check to see if a square matrix is full ranked or not. The sparse matrix is stored in the COO format.

Parameters

row	Row index of the input sparse matrix.
col	Column index of the input sparse matrix.
M	Non-zero values of the input sparse matrix.
N	Row/Column size of the sparse matrix.
nz_size	Length of the non-zeor elements.

Returns

true The matrix is full ranked.

false The matrix is not full ranked.

4.21.1.2 lcg_incomplete_Cholesky_full_coo()

```
void lcg_incomplete_Cholesky_full_coo (
            const int * row,
```

```
            const int * col,
            const lcg_float * val,
            int N,
            int nz_size,
            int * IC_row,
            int * IC_col,
            lcg_float * IC_val )
```

Preform the incomplete Cholesky factorization for a sparse matrix that is saved in the COO format.

Note

The factorized lower and upper triangular matrixes are stored in the lower and upper triangular parts of the output matrix accordingly.

Parameters

row	Row index of the input sparse matrix.
col	Column index of the input sparse matrix.
val	Non-zero values of the input sparse matrix.
N	Row/Column size of the sparse matrix.
nz_size	Length of the non-zeor elements.
IC_row	Row index of the factorized triangular sparse matrix.
IC_col	Column index of the factorized triangular sparse matrix.
IC_val	Non-zero values of the factorized triangular sparse matrix.

4.21.1.3 lcg_incomplete_Cholesky_half_buffsize_coo()

```
void lcg_incomplete_Cholesky_half_buffsize_coo (
            const int * row,
            const int * col,
            int nz_size,
            int * lnz_size )
```

Return the number of non-zero elements in the lower triangular part of the input matrix.

Parameters

row[in]	Row index of the input sparse matrix.
col[in]	Column index of the input sparse matrix.
nz_size[in]	Length of the non-zero elements.
lnz_size[out]	Legnth of the non-zero elements in the lower triangle

4.21.1.4 lcg_incomplete_Cholesky_half_coo()

```
void lcg_incomplete_Cholesky_half_coo (
```

```
                const int * row,
                const int * col,
                const lcg_float * val,
                int N,
                int nz_size,
                int lnz_size,
                int * IC_row,
                int * IC_col,
                lcg_float * IC_val )
```

Preform the incomplete Cholesky factorization for a sparse matrix that is saved in the COO format.

Note

Only the factorized lower triangular matrix is stored in the lower part of the output matrix accordingly.

Parameters

row	Row index of the input sparse matrix.
col	Column index of the input sparse matrix.
val	Non-zero values of the input sparse matrix.
N	Row/Column size of the sparse matrix.
nz_size	Length of the non-zero elements.
lnz_size	Legnth of the non-zero elements in the lower triangle
IC_row	Row index of the factorized triangular sparse matrix.
IC_col	Column index of the factorized triangular sparse matrix.
IC_val	Non-zero values of the factorized triangular sparse matrix.

4.21.1.5 lcg_solve_lower_triangle_coo()

```
void lcg_solve_lower_triangle_coo (
                const int * row,
                const int * col,
                const lcg_float * L,
                const lcg_float * B,
                lcg_float * x,
                int N,
                int nz_size )
```

Solve the linear system Lx = B, in which L is a lower triangle matrix.

Parameters

row	Row index of the input sparse matrix.
col	Column index of the input sparse matrix.
L	Non-zero values of the input sparse matrix.
B	Object array.
x	The returned solution.
N	Row/Column size of the sparse matrix.
nz_size	Length of the non-zeor elements.

4.21.1.6 lcg_solve_upper_triangle_coo()

```
void lcg_solve_upper_triangle_coo (
            const int * row,
            const int * col,
            const lcg_float * U,
            const lcg_float * B,
            lcg_float * x,
            int N,
            int nz_size )
```

Solve the linear system Ux = B, in which U is a upper triangle matrix.

Parameters

row	Row index of the input sparse matrix.
col	Column index of the input sparse matrix.
U	Non-zero values of the input sparse matrix.
B	Object array.
x	The returned solution.
N	Row/Column size of the sparse matrix.
nz_size	Length of the non-zeor elements.

4.22 preconditioner.h File Reference

```
#include "algebra.h"
```

Functions

- void lcg_incomplete_Cholesky_half_buffsize_coo (const int ∗row, const int ∗col, int nz_size, int ∗lnz_size)

  Return the number of non-zero elements in the lower triangular part of the input matrix.
- void lcg_incomplete_Cholesky_half_coo (const int ∗row, const int ∗col, const lcg_float ∗val, int N, int nz_size, int lnz_size, int ∗IC_row, int ∗IC_col, lcg_float ∗IC_val)

  Preform the incomplete Cholesky factorization for a sparse matrix that is saved in the COO format.
- void lcg_incomplete_Cholesky_full_coo (const int ∗row, const int ∗col, const lcg_float ∗val, int N, int nz_size, int ∗IC_row, int ∗IC_col, lcg_float ∗IC_val)

  Preform the incomplete Cholesky factorization for a sparse matrix that is saved in the COO format.
- void lcg_solve_upper_triangle_coo (const int ∗row, const int ∗col, const lcg_float ∗U, const lcg_float ∗B, lcg_float ∗x, int N, int nz_size)

  Solve the linear system Ux = B, in which U is a upper triangle matrix.
- void lcg_solve_lower_triangle_coo (const int ∗row, const int ∗col, const lcg_float ∗L, const lcg_float ∗B, lcg_float ∗x, int N, int nz_size)

  Solve the linear system Lx = B, in which L is a lower triangle matrix.
- bool lcg_full_rank_coo (const int ∗row, const int ∗col, const lcg_float ∗M, int N, int nz_size)

  Check to see if a square matrix is full ranked or not. The sparse matrix is stored in the COO format.

4.22.1 Function Documentation

4.22.1.1 lcg_full_rank_coo()

```
bool lcg_full_rank_coo (
            const int * row,
            const int * col,
            const lcg_float * M,
            int N,
            int nz_size )
```

Check to see if a square matrix is full ranked or not. The sparse matrix is stored in the COO format.

Parameters

row	Row index of the input sparse matrix.
col	Column index of the input sparse matrix.
M	Non-zero values of the input sparse matrix.
N	Row/Column size of the sparse matrix.
nz_size	Length of the non-zeor elements.

Returns

true The matrix is full ranked.

false The matrix is not full ranked.

4.22.1.2 lcg_incomplete_Cholesky_full_coo()

```
void lcg_incomplete_Cholesky_full_coo (
            const int * row,
            const int * col,
            const lcg_float * val,
            int N,
            int nz_size,
            int * IC_row,
            int * IC_col,
            lcg_float * IC_val )
```

Preform the incomplete Cholesky factorization for a sparse matrix that is saved in the COO format.

Note

The factorized lower and upper triangular matrixes are stored in the lower and upper triangular parts of the output matrix accordingly.

Parameters

row	Row index of the input sparse matrix.
col	Column index of the input sparse matrix.
val	Non-zero values of the input sparse matrix.
N	Row/Column size of the sparse matrix.
nz_size	Length of the non-zeor elements.
IC_row	Row index of the factorized triangular sparse matrix.
IC_col	Column index of the factorized triangular sparse matrix.
IC_val	Non-zero values of the factorized triangular sparse matrix.

4.22.1.3 lcg_incomplete_Cholesky_half_buffsize_coo()

```
void lcg_incomplete_Cholesky_half_buffsize_coo (
            const int * row,
            const int * col,
            int nz_size,
            int * lnz_size )
```

Return the number of non-zero elements in the lower triangular part of the input matrix.

Parameters

row[in]	Row index of the input sparse matrix.
col[in]	Column index of the input sparse matrix.
nz_size[in]	Length of the non-zero elements.
lnz_size[out]	Legnth of the non-zero elements in the lower triangle

4.22.1.4 lcg_incomplete_Cholesky_half_coo()

```
void lcg_incomplete_Cholesky_half_coo (
            const int * row,
            const int * col,
            const lcg_float * val,
            int N,
            int nz_size,
            int lnz_size,
            int * IC_row,
            int * IC_col,
            lcg_float * IC_val )
```

Preform the incomplete Cholesky factorization for a sparse matrix that is saved in the COO format.

Note

Only the factorized lower triangular matrix is stored in the lower part of the output matrix accordingly.

Parameters

row	Row index of the input sparse matrix.
col	Column index of the input sparse matrix.
val	Non-zero values of the input sparse matrix.
N	Row/Column size of the sparse matrix.
nz_size	Length of the non-zero elements.
lnz_size	Legnth of the non-zero elements in the lower triangle
IC_row	Row index of the factorized triangular sparse matrix.
IC_col	Column index of the factorized triangular sparse matrix.
IC_val	Non-zero values of the factorized triangular sparse matrix.

4.22.1.5 lcg_solve_lower_triangle_coo()

```
void lcg_solve_lower_triangle_coo (
            const int * row,
            const int * col,
            const lcg_float * L,
            const lcg_float * B,
            lcg_float * x,
            int N,
            int nz_size )
```

Solve the linear system Lx = B, in which L is a lower triangle matrix.

Parameters

row	Row index of the input sparse matrix.
col	Column index of the input sparse matrix.
L	Non-zero values of the input sparse matrix.
B	Object array.
x	The returned solution.
N	Row/Column size of the sparse matrix.
nz_size	Length of the non-zeor elements.

4.22.1.6 lcg_solve_upper_triangle_coo()

```
void lcg_solve_upper_triangle_coo (
            const int * row,
            const int * col,
            const lcg_float * U,
            const lcg_float * B,
            lcg_float * x,
            int N,
            int nz_size )
```

Solve the linear system Ux = B, in which U is a upper triangle matrix.

Parameters

row	Row index of the input sparse matrix.
col	Column index of the input sparse matrix.
U	Non-zero values of the input sparse matrix.
B	Object array.
x	The returned solution.
N	Row/Column size of the sparse matrix.
nz_size	Length of the non-zeor elements.

4.23 preconditioner_cuda.h File Reference

```
#include "lcg_complex_cuda.h"
```

Functions

- void clcg_incomplete_Cholesky_cuda_half_buffsize (const int ∗row, const int ∗col, int nz_size, int ∗lnz_size)

  Return the number of non-zero elements in the lower triangular part of the input matrix.
- void clcg_incomplete_Cholesky_cuda_half (const int ∗row, const int ∗col, const cuComplex ∗val, int N, int nz_size, int lnz_size, int ∗IC_row, int ∗IC_col, cuComplex ∗IC_val)

  Preform the incomplete Cholesky factorization for a sparse matrix that is saved in the COO format.
- void clcg_incomplete_Cholesky_cuda_half (const int ∗row, const int ∗col, const cuDoubleComplex ∗val, int N, int nz_size, int lnz_size, int ∗IC_row, int ∗IC_col, cuDoubleComplex ∗IC_val)

  Preform the incomplete Cholesky factorization for a sparse matrix that is saved in the COO format.
- void clcg_incomplete_Cholesky_cuda_full (const int ∗row, const int ∗col, const cuDoubleComplex ∗val, int N, int nz_size, int ∗IC_row, int ∗IC_col, cuDoubleComplex ∗IC_val)

  Preform the incomplete Cholesky factorization for a sparse matrix that is saved in the COO format.

4.23.1 Function Documentation

4.23.1.1 clcg_incomplete_Cholesky_cuda_full()

```
void clcg_incomplete_Cholesky_cuda_full (
            const int ∗ row,
            const int ∗ col,
            const cuDoubleComplex ∗ val,
            int N,
            int nz_size,
            int ∗ IC_row,
            int ∗ IC_col,
            cuDoubleComplex ∗ IC_val )
```

Preform the incomplete Cholesky factorization for a sparse matrix that is saved in the COO format.

Note

The factorized lower and upper triangular matrixes are stored in the lower and upper triangular parts of the output matrix accordingly.

Parameters

row	Row index of the input sparse matrix.
col	Column index of the input sparse matrix.
val	Non-zero values of the input sparse matrix.
N	Row/Column size of the sparse matrix.
nz_size	Length of the non-zeor elements.
IC_row	Row index of the factorized triangular sparse matrix.
IC_col	Column index of the factorized triangular sparse matrix.
IC_val	Non-zero values of the factorized triangular sparse matrix.

4.23.1.2 clcg_incomplete_Cholesky_cuda_half() [1/2]

```
void clcg_incomplete_Cholesky_cuda_half (
            const int * row,
            const int * col,
            const cuComplex * val,
            int N,
            int nz_size,
            int lnz_size,
            int * IC_row,
            int * IC_col,
            cuComplex * IC_val )
```

Preform the incomplete Cholesky factorization for a sparse matrix that is saved in the COO format.

Note

Only the factorized lower triangular matrix is stored in the lower part of the output matrix accordingly.

Parameters

row	Row index of the input sparse matrix.
col	Column index of the input sparse matrix.
val	Non-zero values of the input sparse matrix.
N	Row/Column size of the sparse matrix.
nz_size	Length of the non-zero elements.
lnz_size	Legnth of the non-zero elements in the lower triangle
IC_row	Row index of the factorized triangular sparse matrix.
IC_col	Column index of the factorized triangular sparse matrix.
IC_val	Non-zero values of the factorized triangular sparse matrix.

4.23.1.3 clcg_incomplete_Cholesky_cuda_half() [2/2]

```
void clcg_incomplete_Cholesky_cuda_half (
```

```
            const int * row,
            const int * col,
            const cuDoubleComplex * val,
            int N,
            int nz_size,
            int lnz_size,
            int * IC_row,
            int * IC_col,
            cuDoubleComplex * IC_val )
```

Preform the incomplete Cholesky factorization for a sparse matrix that is saved in the COO format.

Note

Only the factorized lower triangular matrix is stored in the lower part of the output matrix accordingly.

Parameters

row	Row index of the input sparse matrix.
col	Column index of the input sparse matrix.
val	Non-zero values of the input sparse matrix.
N	Row/Column size of the sparse matrix.
nz_size	Length of the non-zero elements.
lnz_size	Legnth of the non-zero elements in the lower triangle
IC_row	Row index of the factorized triangular sparse matrix.
IC_col	Column index of the factorized triangular sparse matrix.
IC_val	Non-zero values of the factorized triangular sparse matrix.

4.23.1.4 clcg_incomplete_Cholesky_cuda_half_buffsize()

```
void clcg_incomplete_Cholesky_cuda_half_buffsize (
            const int * row,
            const int * col,
            int nz_size,
            int * lnz_size )
```

Return the number of non-zero elements in the lower triangular part of the input matrix.

Parameters

row[in]	Row index of the input sparse matrix.
col[in]	Column index of the input sparse matrix.
nz_size[in]	Length of the non-zero elements.
lnz_size[out]	Legnth of the non-zero elements in the lower triangle

4.24 preconditioner_eigen.cpp File Reference

```
#include "preconditioner_eigen.h"
#include "exception"
#include "stdexcept"
#include "vector"
#include "iostream"
```

Typedefs

- typedef Eigen::Triplet< int > triplet_bl
- typedef Eigen::Triplet< double > triplet_d
- typedef Eigen::Triplet< std::complex< double > > triplet_cd

Functions

- void lcg_Cholesky (const Eigen::MatrixXd &A, Eigen::MatrixXd &L)

  Perform the Cholesky decomposition and return the lower triangular matrix.
- void clcg_Cholesky (const Eigen::MatrixXcd &A, Eigen::MatrixXcd &L)

  Perform the Cholesky decomposition and return the lower triangular matrix.
- void lcg_invert_lower_triangle (const Eigen::MatrixXd &L, Eigen::MatrixXd &Linv)

  Calculate the invert of a lower triangle matrix (Full rank only).
- void lcg_invert_upper_triangle (const Eigen::MatrixXd &U, Eigen::MatrixXd &Uinv)

  Calculate the invert of a upper triangle matrix (Full rank only).
- void clcg_invert_lower_triangle (const Eigen::MatrixXcd &L, Eigen::MatrixXcd &Linv)

  Calculate the invert of a lower triangle matrix (Full rank only).
- void clcg_invert_upper_triangle (const Eigen::MatrixXcd &U, Eigen::MatrixXcd &Uinv)

  Calculate the invert of a upper triangle matrix (Full rank only).
- void lcg_incomplete_Cholesky (const Eigen::SparseMatrix< double, Eigen::RowMajor > &A, Eigen::SparseMatrix< double, Eigen::RowMajor > &L, size_t fill)

  Calculate the incomplete Cholesky decomposition and return the lower triangular matrix.
- void clcg_incomplete_Cholesky (const Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > &A, Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > &L, size_t fill)

  Calculate the incomplete Cholesky decomposition and return the lower triangular matrix.
- void lcg_incomplete_LU (const Eigen::SparseMatrix< double, Eigen::RowMajor > &A, Eigen::SparseMatrix< double, Eigen::RowMajor > &L, Eigen::SparseMatrix< double, Eigen::RowMajor > &U, size_t fill)

  Calculate the incomplete LU factorizations.
- void clcg_incomplete_LU (const Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > &A, Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > &L, Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > &U, size_t fill)

  Calculate the incomplete LU factorizations.
- void lcg_solve_lower_triangle (const Eigen::SparseMatrix< double, Eigen::RowMajor > &L, const Eigen::VectorXd &B, Eigen::VectorXd &X)

  Solve the linear system Lx = B, in which L is a lower triangle matrix.
- void lcg_solve_upper_triangle (const Eigen::SparseMatrix< double, Eigen::RowMajor > &U, const Eigen::VectorXd &B, Eigen::VectorXd &X)

  Solve the linear system Ux = B, in which U is a upper triangle matrix.
- void clcg_solve_lower_triangle (const Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > &L, const Eigen::VectorXcd &B, Eigen::VectorXcd &X)

  Solve the linear system Lx = B, in which L is a lower triangle matrix.
- void clcg_solve_upper_triangle (const Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > &U, const Eigen::VectorXcd &B, Eigen::VectorXcd &X)

  Solve the linear system Ux = B, in which U is a upper triangle matrix.

4.24.1 Typedef Documentation

4.24.1.1 triplet_bl

```
typedef Eigen::Triplet<int> triplet_bl
```

4.24.1.2 triplet_cd

```
typedef Eigen::Triplet<std::complex<double> > triplet_cd
```

4.24.1.3 triplet_d

```
typedef Eigen::Triplet<double> triplet_d
```

4.24.2 Function Documentation

4.24.2.1 clcg_Cholesky()

```
void clcg_Cholesky (
            const Eigen::MatrixXcd & A,
            Eigen::MatrixXcd & L )
```

Perform the Cholesky decomposition and return the lower triangular matrix.

Note

This could serve as a direct solver.

Parameters

in	*A*	The input matrix. Must be full rank and symmetric (aka. A = A^T)
	L	The output low triangular matrix

4.24.2.2 clcg_incomplete_Cholesky()

```
void clcg_incomplete_Cholesky (
            const Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > & A,
            Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > & L,
            size_t fill = 0 )
```

Calculate the incomplete Cholesky decomposition and return the lower triangular matrix.

Parameters

in	*A*	The input sparse matrix. Must be full rank and symmetric (aka. A = A^T)
	L	The output lower triangular matrix
	fill	The fill-in number of the output sparse matrix. No fill-in reduction will be processed if this variable is set to zero.

4.24.2.3 clcg_incomplete_LU()

```
void clcg_incomplete_LU (
            const Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > & A,
            Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > & L,
            Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > & U,
            size_t fill = 0 )
```

Calculate the incomplete LU factorizations.

Parameters

A	The input sparse matrix. Must be full rank.
L	The output lower triangular matrix.
U	The output upper triangular matrix.
fill	The fill-in number of the output sparse matrix. No fill-in reduction will be processed if this variable is set to zero.

4.24.2.4 clcg_invert_lower_triangle()

```
void clcg_invert_lower_triangle (
            const Eigen::MatrixXcd & L,
            Eigen::MatrixXcd & Linv )
```

Calculate the invert of a lower triangle matrix (Full rank only).

Parameters

L	The operating lower triangle matrix
Linv	The inverted lower triangle matrix

4.24.2.5 clcg_invert_upper_triangle()

```
void clcg_invert_upper_triangle (
            const Eigen::MatrixXcd & U,
            Eigen::MatrixXcd & Uinv )
```

Calculate the invert of a upper triangle matrix (Full rank only).

Parameters

U	The operating upper triangle matrix
Uinv	The inverted upper triangle matrix

4.24.2.6 clcg_solve_lower_triangle()

```
void clcg_solve_lower_triangle (
            const Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > & L,
            const Eigen::VectorXcd & B,
            Eigen::VectorXcd & X )
```

Solve the linear system Lx = B, in which L is a lower triangle matrix.

Parameters

L	The input lower triangle matrix
B	The object vector
X	The solution vector

4.24.2.7 clcg_solve_upper_triangle()

```
void clcg_solve_upper_triangle (
            const Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > & U,
            const Eigen::VectorXcd & B,
            Eigen::VectorXcd & X )
```

Solve the linear system Ux = B, in which U is a upper triangle matrix.

Parameters

U	The input upper triangle matrix
B	The object vector
X	The solution vector

4.24.2.8 lcg_Cholesky()

```
void lcg_Cholesky (
            const Eigen::MatrixXd & A,
            Eigen::MatrixXd & L )
```

Perform the Cholesky decomposition and return the lower triangular matrix.

Note

This could serve as a direct solver.

Parameters

A	The input matrix. Must be full rank and symmetric (aka. A = A^T)
L	The output low triangular matrix

4.24.2.9 lcg_incomplete_Cholesky()

```
void lcg_incomplete_Cholesky (
            const Eigen::SparseMatrix< double, Eigen::RowMajor > & A,
            Eigen::SparseMatrix< double, Eigen::RowMajor > & L,
            size_t fill = 0 )
```

Calculate the incomplete Cholesky decomposition and return the lower triangular matrix.

Parameters

in	*A*	The input sparse matrix. Must be full rank and symmetric (aka. A = A^T)
	L	The output lower triangular matrix
	fill	The fill-in number of the output sparse matrix. No fill-in reduction will be processed if this variable is set to zero.

4.24.2.10 lcg_incomplete_LU()

```
void lcg_incomplete_LU (
            const Eigen::SparseMatrix< double, Eigen::RowMajor > & A,
            Eigen::SparseMatrix< double, Eigen::RowMajor > & L,
            Eigen::SparseMatrix< double, Eigen::RowMajor > & U,
            size_t fill = 0 )
```

Calculate the incomplete LU factorizations.

Parameters

A	The input sparse matrix. Must be full rank.
L	The output lower triangular matrix.
U	The output upper triangular matrix.
fill	The fill-in number of the output sparse matrix. No fill-in reduction will be processed if this variable is set to zero.

4.24.2.11 lcg_invert_lower_triangle()

```
void lcg_invert_lower_triangle (
            const Eigen::MatrixXd & L,
            Eigen::MatrixXd & Linv )
```

Calculate the invert of a lower triangle matrix (Full rank only).

Parameters

L	The operating lower triangle matrix
Linv	The inverted lower triangle matrix

4.24.2.12 lcg_invert_upper_triangle()

```
void lcg_invert_upper_triangle (
            const Eigen::MatrixXd & U,
            Eigen::MatrixXd & Uinv )
```

Calculate the invert of a upper triangle matrix (Full rank only).

Parameters

U	The operating upper triangle matrix
Uinv	The inverted upper triangle matrix

4.24.2.13 lcg_solve_lower_triangle()

```
void lcg_solve_lower_triangle (
            const Eigen::SparseMatrix< double, Eigen::RowMajor > & L,
            const Eigen::VectorXd & B,
            Eigen::VectorXd & X )
```

Solve the linear system Lx = B, in which L is a lower triangle matrix.

Parameters

L	The input lower triangle matrix
B	The object vector
X	The solution vector

4.24.2.14 lcg_solve_upper_triangle()

```
void lcg_solve_upper_triangle (
            const Eigen::SparseMatrix< double, Eigen::RowMajor > & U,
            const Eigen::VectorXd & B,
            Eigen::VectorXd & X )
```

Solve the linear system Ux = B, in which U is a upper triangle matrix.

Parameters

U	The input upper triangle matrix
B	The object vector
X	The solution vector

4.25 preconditioner_eigen.h File Reference

```
#include "complex"
#include "Eigen/Dense"
#include "Eigen/SparseCore"
```

Functions

- void lcg_Cholesky (const Eigen::MatrixXd &A, Eigen::MatrixXd &L)

  Perform the Cholesky decomposition and return the lower triangular matrix.
- void clcg_Cholesky (const Eigen::MatrixXcd &A, Eigen::MatrixXcd &L)

  Perform the Cholesky decomposition and return the lower triangular matrix.
- void lcg_invert_lower_triangle (const Eigen::MatrixXd &L, Eigen::MatrixXd &Linv)

  Calculate the invert of a lower triangle matrix (Full rank only).
- void lcg_invert_upper_triangle (const Eigen::MatrixXd &U, Eigen::MatrixXd &Uinv)

  Calculate the invert of a upper triangle matrix (Full rank only).
- void clcg_invert_lower_triangle (const Eigen::MatrixXcd &L, Eigen::MatrixXcd &Linv)

  Calculate the invert of a lower triangle matrix (Full rank only).
- void clcg_invert_upper_triangle (const Eigen::MatrixXcd &U, Eigen::MatrixXcd &Uinv)

  Calculate the invert of a upper triangle matrix (Full rank only).
- void lcg_incomplete_Cholesky (const Eigen::SparseMatrix< double, Eigen::RowMajor > &A, Eigen::↩ SparseMatrix< double, Eigen::RowMajor > &L, size_t fill=0)

  Calculate the incomplete Cholesky decomposition and return the lower triangular matrix.

- void clcg_incomplete_Cholesky (const Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > &A, Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > &L, size_t fill=0)

    Calculate the incomplete Cholesky decomposition and return the lower triangular matrix.
- void lcg_incomplete_LU (const Eigen::SparseMatrix< double, Eigen::RowMajor > &A, Eigen::Sparse↩ Matrix< double, Eigen::RowMajor > &L, Eigen::SparseMatrix< double, Eigen::RowMajor > &U, size_t fill=0)

    Calculate the incomplete LU factorizations.
- void clcg_incomplete_LU (const Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > &A, Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > &L, Eigen::SparseMatrix< std↩ ::complex< double >, Eigen::RowMajor > &U, size_t fill=0)

    Calculate the incomplete LU factorizations.
- void lcg_solve_lower_triangle (const Eigen::SparseMatrix< double, Eigen::RowMajor > &L, const Eigen::↩ VectorXd &B, Eigen::VectorXd &X)

    Solve the linear system Lx = B, in which L is a lower triangle matrix.
- void lcg_solve_upper_triangle (const Eigen::SparseMatrix< double, Eigen::RowMajor > &U, const Eigen::↩ VectorXd &B, Eigen::VectorXd &X)

    Solve the linear system Ux = B, in which U is a upper triangle matrix.
- void clcg_solve_lower_triangle (const Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > &L, const Eigen::VectorXcd &B, Eigen::VectorXcd &X)

    Solve the linear system Lx = B, in which L is a lower triangle matrix.
- void clcg_solve_upper_triangle (const Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > &U, const Eigen::VectorXcd &B, Eigen::VectorXcd &X)

    Solve the linear system Ux = B, in which U is a upper triangle matrix.

4.25.1 Function Documentation

4.25.1.1 clcg_Cholesky()

```
void clcg_Cholesky (
            const Eigen::MatrixXcd & A,
            Eigen::MatrixXcd & L )
```

Perform the Cholesky decomposition and return the lower triangular matrix.

Note

This could serve as a direct solver.

Parameters

in	*A*	The input matrix. Must be full rank and symmetric (aka. A = A^T)
	L	The output low triangular matrix

4.25.1.2 clcg_incomplete_Cholesky()

```
void clcg_incomplete_Cholesky (
            const Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > & A,
```

```
Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > & L,
size_t fill = 0 )
```

Calculate the incomplete Cholesky decomposition and return the lower triangular matrix.

Parameters

in	*A*	The input sparse matrix. Must be full rank and symmetric (aka. A = A^T)
	L	The output lower triangular matrix
	fill	The fill-in number of the output sparse matrix. No fill-in reduction will be processed if this variable is set to zero.

4.25.1.3 clcg_incomplete_LU()

```
void clcg_incomplete_LU (
            const Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > & A,
            Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > & L,
            Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > & U,
            size_t fill = 0 )
```

Calculate the incomplete LU factorizations.

Parameters

A	The input sparse matrix. Must be full rank.
L	The output lower triangular matrix.
U	The output upper triangular matrix.
fill	The fill-in number of the output sparse matrix. No fill-in reduction will be processed if this variable is set to zero.

4.25.1.4 clcg_invert_lower_triangle()

```
void clcg_invert_lower_triangle (
            const Eigen::MatrixXcd & L,
            Eigen::MatrixXcd & Linv )
```

Calculate the invert of a lower triangle matrix (Full rank only).

Parameters

L	The operating lower triangle matrix
Linv	The inverted lower triangle matrix

4.25.1.5 clcg_invert_upper_triangle()

```
void clcg_invert_upper_triangle (
            const Eigen::MatrixXcd & U,
            Eigen::MatrixXcd & Uinv )
```

Calculate the invert of a upper triangle matrix (Full rank only).

Parameters

U	The operating upper triangle matrix
Uinv	The inverted upper triangle matrix

4.25.1.6 clcg_solve_lower_triangle()

```
void clcg_solve_lower_triangle (
            const Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > & L,
            const Eigen::VectorXcd & B,
            Eigen::VectorXcd & X )
```

Solve the linear system Lx = B, in which L is a lower triangle matrix.

Parameters

L	The input lower triangle matrix
B	The object vector
X	The solution vector

4.25.1.7 clcg_solve_upper_triangle()

```
void clcg_solve_upper_triangle (
            const Eigen::SparseMatrix< std::complex< double >, Eigen::RowMajor > & U,
            const Eigen::VectorXcd & B,
            Eigen::VectorXcd & X )
```

Solve the linear system Ux = B, in which U is a upper triangle matrix.

Parameters

U	The input upper triangle matrix
B	The object vector
X	The solution vector

4.25.1.8 lcg_Cholesky()

```
void lcg_Cholesky (
            const Eigen::MatrixXd & A,
            Eigen::MatrixXd & L )
```

Perform the Cholesky decomposition and return the lower triangular matrix.

Note

This could serve as a direct solver.

Parameters

A	The input matrix. Must be full rank and symmetric (aka. A = A^T)
L	The output low triangular matrix

4.25.1.9 lcg_incomplete_Cholesky()

```
void lcg_incomplete_Cholesky (
            const Eigen::SparseMatrix< double, Eigen::RowMajor > & A,
            Eigen::SparseMatrix< double, Eigen::RowMajor > & L,
            size_t fill = 0 )
```

Calculate the incomplete Cholesky decomposition and return the lower triangular matrix.

Parameters

in	*A*	The input sparse matrix. Must be full rank and symmetric (aka. A = A^T)
	L	The output lower triangular matrix
	fill	The fill-in number of the output sparse matrix. No fill-in reduction will be processed if this variable is set to zero.

4.25.1.10 lcg_incomplete_LU()

```
void lcg_incomplete_LU (
            const Eigen::SparseMatrix< double, Eigen::RowMajor > & A,
            Eigen::SparseMatrix< double, Eigen::RowMajor > & L,
            Eigen::SparseMatrix< double, Eigen::RowMajor > & U,
            size_t fill = 0 )
```

Calculate the incomplete LU factorizations.

Parameters

A	The input sparse matrix. Must be full rank.

Parameters

L	The output lower triangular matrix.
U	The output upper triangular matrix.
fill	The fill-in number of the output sparse matrix. No fill-in reduction will be processed if this variable is set to zero.

4.25.1.11 lcg_invert_lower_triangle()

```
void lcg_invert_lower_triangle (
            const Eigen::MatrixXd & L,
            Eigen::MatrixXd & Linv )
```

Calculate the invert of a lower triangle matrix (Full rank only).

Parameters

L	The operating lower triangle matrix
Linv	The inverted lower triangle matrix

4.25.1.12 lcg_invert_upper_triangle()

```
void lcg_invert_upper_triangle (
            const Eigen::MatrixXd & U,
            Eigen::MatrixXd & Uinv )
```

Calculate the invert of a upper triangle matrix (Full rank only).

Parameters

U	The operating upper triangle matrix
Uinv	The inverted upper triangle matrix

4.25.1.13 lcg_solve_lower_triangle()

```
void lcg_solve_lower_triangle (
            const Eigen::SparseMatrix< double, Eigen::RowMajor > & L,
            const Eigen::VectorXd & B,
            Eigen::VectorXd & X )
```

Solve the linear system Lx = B, in which L is a lower triangle matrix.

Parameters

L	The input lower triangle matrix
B	The object vector
X	The solution vector

4.25.1.14 lcg_solve_upper_triangle()

```
void lcg_solve_upper_triangle (
            const Eigen::SparseMatrix< double, Eigen::RowMajor > & U,
            const Eigen::VectorXd & B,
            Eigen::VectorXd & X )
```

Solve the linear system Ux = B, in which U is a upper triangle matrix.

Parameters

U	The input upper triangle matrix
B	The object vector
X	The solution vector

4.26 solver.cpp File Reference

```
#include "solver.h"
#include "ctime"
#include "iostream"
#include "config.h"
#include "omp.h"
```

4.27 solver.h File Reference

```
#include "lcg.h"
#include "clcg.h"
```

Data Structures

- class LCG_Solver

  Linear conjugate gradient solver class.
- class CLCG_Solver

  Complex linear conjugate gradient solver class.

4.28 solver_cuda.h File Reference

```
#include "lcg_cuda.h"
#include "clcg_cuda.h"
#include "clcg_cudaf.h"
```

Data Structures

- class LCG_CUDA_Solver

  Linear conjugate gradient solver class.
- class CLCG_CUDAF_Solver

  Complex linear conjugate gradient solver class.
- class CLCG_CUDA_Solver

  Complex linear conjugate gradient solver class.

4.29 solver_eigen.cpp File Reference

```
#include "solver_eigen.h"
#include "cmath"
#include "ctime"
#include "iostream"
#include "config.h"
#include "omp.h"
```

4.30 solver_eigen.h File Reference

```
#include "lcg_eigen.h"
#include "clcg_eigen.h"
```

Data Structures

- class LCG_EIGEN_Solver

  Linear conjugate gradient solver class.
- class CLCG_EIGEN_Solver

  Complex linear conjugate gradient solver class.

4.31 util.cpp File Reference

```
#include "iostream"
#include "exception"
#include "stdexcept"
#include "util.h"
```

Functions

- lcg_para lcg_default_parameters ()

    Return a lcg_para type instance with default values.
- lcg_solver_enum lcg_select_solver (std::string slr_char)

    Select a type of solver according to the name.
- void lcg_error_str (int er_index, bool er_throw)

    Display or throw out a string explanation for the lcg_solver() function's return values.
- clcg_para clcg_default_parameters ()

    Return a clcg_para type instance with default values.
- clcg_solver_enum clcg_select_solver (std::string slr_char)

    Select a type of solver according to the name.
- void clcg_error_str (int er_index, bool er_throw)

    Display or throw out a string explanation for the clcg_solver() function's return values.

4.31.1 Function Documentation

4.31.1.1 clcg_default_parameters()

```
clcg_para clcg_default_parameters ( )
```

Return a clcg_para type instance with default values.

Users can use this function to get default parameters' value for the complex conjugate gradient methods.

Returns

A clcg_para type instance.

4.31.1.2 clcg_error_str()

```
void clcg_error_str (
            int er_index,
            bool er_throw = false )
```

Display or throw out a string explanation for the clcg_solver() function's return values.

Parameters

in	*er_index*	The error index returned by the lcg_solver() function.
in	*er_throw*	throw out a char string of the explanation.

Returns

A string explanation of the error.

4.31.1.3 clcg_select_solver()

```
clcg_solver_enum clcg_select_solver (
            std::string slr_char )
```

Select a type of solver according to the name.

Parameters

in	*slr_char*	Name of the solver

Returns

The clcg solver enum.

4.31.1.4 lcg_default_parameters()

```
lcg_para lcg_default_parameters ( )
```

Return a lcg_para type instance with default values.

Users can use this function to get default parameters' value for the conjugate gradient methods.

Returns

A lcg_para type instance.

4.31.1.5 lcg_error_str()

```
void lcg_error_str (
            int er_index,
            bool er_throw = false )
```

Display or throw out a string explanation for the lcg_solver() function's return values.

Parameters

in	*er_index*	The error index returned by the lcg_solver() function.
in	*er_throw*	throw out a char string of the explanation.

Returns

A string explanation of the error.

4.31.1.6 lcg_select_solver()

```
lcg_solver_enum lcg_select_solver (
            std::string slr_char )
```

Select a type of solver according to the name.

Parameters

in	*slr_char*	Name of the solver

Returns

The lcg solver enum.

4.32 util.h File Reference

```
#include "string"
#include "algebra.h"
```

Data Structures

- struct lcg_para

  Parameters of the conjugate gradient methods.
- struct clcg_para

  Parameters of the conjugate gradient methods.

Enumerations

- enum lcg_solver_enum {
  LCG_CG, LCG_PCG, LCG_CGS, LCG_BICGSTAB,
  LCG_BICGSTAB2, LCG_PG, LCG_SPG }

  Types of method that could be recognized by the lcg_solver() function.
- enum lcg_return_enum {
  LCG_SUCCESS = 0, LCG_CONVERGENCE = 0, LCG_STOP, LCG_ALREADY_OPTIMIZIED,
  LCG_UNKNOWN_ERROR = -1024, LCG_INVILAD_VARIABLE_SIZE, LCG_INVILAD_MAX_ITERATIONS,
  LCG_INVILAD_EPSILON,
  LCG_INVILAD_RESTART_EPSILON, LCG_REACHED_MAX_ITERATIONS, LCG_NULL_PRECONDITION_MATRIX,
  LCG_NAN_VALUE,
  LCG_INVALID_POINTER, LCG_INVALID_LAMBDA, LCG_INVALID_SIGMA, LCG_INVALID_BETA,
  LCG_INVALID_MAXIM, LCG_SIZE_NOT_MATCH }

  return value of the lcg_solver() function

- enum clcg_solver_enum {
  CLCG_BICG, CLCG_BICG_SYM, CLCG_CGS, CLCG_BICGSTAB,
  CLCG_TFQMR, CLCG_PCG, CLCG_PBICG }

  Types of method that could be recognized by the clcg_solver() function.
- enum clcg_return_enum {
  CLCG_SUCCESS = 0, CLCG_CONVERGENCE = 0, CLCG_STOP, CLCG_ALREADY_OPTIMIZIED,
  CLCG_UNKNOWN_ERROR = -1024, CLCG_INVILAD_VARIABLE_SIZE, CLCG_INVILAD_MAX_ITERATIONS,
  CLCG_INVILAD_EPSILON,
  CLCG_REACHED_MAX_ITERATIONS, CLCG_NAN_VALUE, CLCG_INVALID_POINTER, CLCG_SIZE_NOT_MATCH,
  CLCG_UNKNOWN_SOLVER }

  return value of the clcg_solver() function

Functions

- lcg_para lcg_default_parameters ()

  Return a lcg_para type instance with default values.
- lcg_solver_enum lcg_select_solver (std::string slr_char)

  Select a type of solver according to the name.
- void lcg_error_str (int er_index, bool er_throw=false)

  Display or throw out a string explanation for the lcg_solver() function's return values.
- clcg_para clcg_default_parameters ()

  Return a clcg_para type instance with default values.
- clcg_solver_enum clcg_select_solver (std::string slr_char)

  Select a type of solver according to the name.
- void clcg_error_str (int er_index, bool er_throw=false)

  Display or throw out a string explanation for the clcg_solver() function's return values.

4.32.1 Enumeration Type Documentation

4.32.1.1 clcg_return_enum

```
enum clcg_return_enum
```

return value of the clcg_solver() function

Enumerator

CLCG_SUCCESS	The solver function terminated successfully.
CLCG_CONVERGENCE	The iteration reached convergence.
CLCG_STOP	The iteration is stopped by the monitoring function.
CLCG_ALREADY_OPTIMIZIED	The initial solution is already optimized.
CLCG_UNKNOWN_ERROR	Unknown error.
CLCG_INVILAD_VARIABLE_SIZE	The variable size is negative.
CLCG_INVILAD_MAX_ITERATIONS	The maximal iteration times is negative.
CLCG_INVILAD_EPSILON	The epsilon is negative.
CLCG_REACHED_MAX_ITERATIONS	Iteration reached maximal limit.
CLCG_NAN_VALUE	Nan value.
CLCG_INVALID_POINTER	Invalid pointer.
CLCG_SIZE_NOT_MATCH	Sizes of m and B do not match.
CLCG_UNKNOWN_SOLVER	Unknown solver.

4.32.1.2 clcg_solver_enum

```
enum clcg_solver_enum
```

Types of method that could be recognized by the clcg_solver() function.

Enumerator

CLCG_BICG	Jacob's Bi-Conjugate Gradient Method
CLCG_BICG_SYM	Bi-Conjugate Gradient Method accelerated for complex symmetric A
CLCG_CGS	Conjugate Gradient Squared Method with real coefficients.
CLCG_BICGSTAB	Biconjugate gradient method.
CLCG_TFQMR	Quasi-Minimal Residual Method Transpose Free Quasi-Minimal Residual Method
CLCG_PCG	Preconditioned conjugate gradient
CLCG_PBICG	Preconditioned Bi-Conjugate Gradient Method

4.32.1.3 lcg_return_enum

```
enum lcg_return_enum
```

return value of the lcg_solver() function

Enumerator

LCG_SUCCESS	The solver function terminated successfully.
LCG_CONVERGENCE	The iteration reached convergence.
LCG_STOP	The iteration is stopped by the monitoring function.
LCG_ALREADY_OPTIMIZIED	The initial solution is already optimized.
LCG_UNKNOWN_ERROR	Unknown error.
LCG_INVILAD_VARIABLE_SIZE	The variable size is negative.
LCG_INVILAD_MAX_ITERATIONS	The maximal iteration times is negative.
LCG_INVILAD_EPSILON	The epsilon is negative.
LCG_INVILAD_RESTART_EPSILON	The restart epsilon is negative.
LCG_REACHED_MAX_ITERATIONS	Iteration reached maximal limit.
LCG_NULL_PRECONDITION_MATRIX	Null precondition matrix.
LCG_NAN_VALUE	Nan value.
LCG_INVALID_POINTER	Invalid pointer.
LCG_INVALID_LAMBDA	Invalid range for lambda.
LCG_INVALID_SIGMA	Invalid range for sigma.
LCG_INVALID_BETA	Invalid range for beta.
LCG_INVALID_MAXIM	Invalid range for maxi_m.
LCG_SIZE_NOT_MATCH	Sizes of m and B do not match.

4.32.1.4 lcg_solver_enum

```
enum lcg_solver_enum
```

Types of method that could be recognized by the lcg_solver() function.

Enumerator

LCG_CG	Conjugate gradient method.
LCG_PCG	Preconditioned conjugate gradient method.
LCG_CGS	Conjugate gradient squared method.
LCG_BICGSTAB	Biconjugate gradient method.
LCG_BICGSTAB2	Biconjugate gradient method with restart.
LCG_PG	Conjugate gradient method with projected gradient for inequality constraints. This algorithm comes without non-monotonic linear search for the step length.
LCG_SPG	Conjugate gradient method with spectral projected gradient for inequality constraints. This algorithm comes with non-monotonic linear search for the step length.

4.32.2 Function Documentation

4.32.2.1 clcg_default_parameters()

```
clcg_para clcg_default_parameters ( )
```

Return a clcg_para type instance with default values.

Users can use this function to get default parameters' value for the complex conjugate gradient methods.

Returns

A clcg_para type instance.

4.32.2.2 clcg_error_str()

```
void clcg_error_str (
            int er_index,
            bool er_throw = false )
```

Display or throw out a string explanation for the clcg_solver() function's return values.

Parameters

in	*er_index*	The error index returned by the lcg_solver() function.
in	*er_throw*	throw out a char string of the explanation.

Returns

A string explanation of the error.

4.32.2.3 clcg_select_solver()

```
clcg_solver_enum clcg_select_solver (
            std::string slr_char )
```

Select a type of solver according to the name.

Parameters

in	*slr_char*	Name of the solver

Returns

The clcg solver enum.

4.32.2.4 lcg_default_parameters()

```
lcg_para lcg_default_parameters ( )
```

Return a lcg_para type instance with default values.

Users can use this function to get default parameters' value for the conjugate gradient methods.

Returns

A lcg_para type instance.

4.32.2.5 lcg_error_str()

```
void lcg_error_str (
            int er_index,
            bool er_throw = false )
```

Display or throw out a string explanation for the lcg_solver() function's return values.

Parameters

in	*er_index*	The error index returned by the lcg_solver() function.
in	*er_throw*	throw out a char string of the explanation.

Returns

A string explanation of the error.

4.32.2.6 lcg_select_solver()

```
lcg_solver_enum lcg_select_solver (
            std::string slr_char )
```

Select a type of solver according to the name.

Parameters

in	*slr_char*	Name of the solver

Returns

The lcg solver enum.

Index