libcasimir

v0.4

Generated by Doxygen 1.8.13

Contents

Chapter 1

libcasimir

1.1 Overview

libcasimir implements the numerics for the Casimir effect in the plane-sphere geometry for arbitrary materials at zero and finite temperature using the scattering approach.

This document describes the API of the libcasimir library. The compilation of the software and the usage of the programs are described in the user manual located in the directory manual/.

The library is tuned for high performance. Often, input parameters of functions are not checked. In contrast, library functions usually abort the program if results are wrong or look fishy.

Also, a user manual is available in manuael/.

1.2 Files

file	description
casimir.c	command line interface to libcasimir (see also user manual)
casimir_logdetD.c	command line interface to compute determinants of the scattering matrix (see also user manual)
cylinder.cpp	command line interface to compute Casimir interaction in the plane-cylinder geometry (see also user manual)
cquadpack/src/$*$.c	integration routines (CQUADPACK), see cquadpack/include/quadpack.h
libhodlr/src/hodlr.cpp	C wrapper for the HODLR library (see libhodlr/include/hodlr.h)
libcasimir.c	main part of the library
plm.c	routines to compute Legendre polynomials and associated Legendre polynomials
bessel.c	routines to compute modified Bessel functions
matrix.c	linear algebra functions; in particular computation of determinants
integration.c	routines to compute integrals that appear in the matrix elements of the round-trip operator
fcqs.c	integration routines using adapative convergent Fourier-Chebshev quadrature scheme
utils.c	wrappers for malloc, calloc realloc, and a few more useful functions
cache.c	implementation of a simple cache using a hash table
logfac.c	fast computation of $\log(n)$, $\log(n!)$, and $\log(n!!)$ for integer n
psd.c	weights and poles for Pade spectrum decomposition
misc.c	various mathematical functions
material.c	support for arbitrary dielectric functions
argparse.c	library to parse command line parameters

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

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

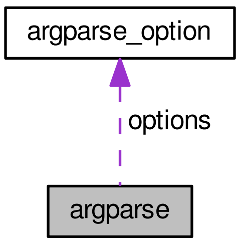
Chapter 4

Data Structure Documentation

4.1 argparse Struct Reference

```
#include <argparse.h>
```

Collaboration diagram for argparse:



Data Fields

- const struct argparse_option ∗ **options**
- const char ∗const ∗ **usages**
- int **flags**
- const char ∗ **description**
- const char ∗ **epilog**
- int **argc**
- const char ∗∗ **argv**
- const char ∗∗ **out**
- int **cpidx**
- const char ∗ **optvalue**

4.1.1 Detailed Description

argpparse

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

- include/argparse.h

4.2 argparse_option Struct Reference

`#include <argparse.h>`

Data Fields

- enum argparse_option_type **type**
- const char **short_name**
- const char ∗ **long_name**
- void ∗ **value**
- const char ∗ **help**
- argparse_callback ∗ **callback**
- intptr_t **data**
- int **flags**

4.2.1 Detailed Description

argparse option

`type`: holds the type of the option, you must have an ARGPARSE_OPT_END last in your array.

`short_name`: the character to use as a short option name, '\0' if none.

`long_name`: the long option name, without the leading dash, NULL if none.

`value`: stores pointer to the value to be filled.

`help`: the short help message associated to what the option does. Must never be NULL (except for ARGPARS←
E_OPT_END).

`callback`: function is called when corresponding argument is parsed.

`data`: associated data. Callbacks can use it like they want.

`flags`: option flags.

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

- include/argparse.h

4.3 buf Struct Reference

Data Fields

- size_t capacity
- size_t size
- char buffer [ ]

4.3.1 Field Documentation

4.3.1.1 buffer

```
char buf::buffer[]
```

buffer

4.3.1.2 capacity

```
size_t buf::capacity
```

total capacity of buffer

4.3.1.3 size

```
size_t buf::size
```

size / number of elements

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

- include/buf.h

4.4 cache_t Struct Reference

Data Fields

- int **head**
- int **tail**
- int **num_entries**
- int **num_lookup**
- uint64_t ∗ **keys**
- double ∗ **values**
- uint64_t ∗ **table**

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

- include/cache.h

4.5 casimir Struct Reference

```
#include <libcasimir.h>
```

Data Fields

geometry

- double L
- double R
- double calL
- double LbyR
- double y

dielectric function of the plate

- double(∗ **epsilonm1_plate**)(double xi_, void ∗userdata)
- void ∗ **userdata_plate**

dielectric function of the sphere

- double(∗ **epsilonm1_sphere**)(double xi_, void ∗userdata)
- void ∗ **userdata_sphere**

accuracy and numerical parameters

- int ldim
- double epsrel
- detalg_t detalg

4.5.1 Detailed Description

The Casimir object. This structure stores all essential information about temperature, geometry and the reflection properties of the mirrors.

Do not modify the attributes of the structure yourself!

4.5.2 Field Documentation

4.5.2.1 calL

```
double casimir::calL
```

$L+R$

4.5.2.2 detalg

```
detalg_t casimir::detalg
```

algorithm to calculate determinant

4.5.2.3 epsrel

```
double casimir::epsrel
```

relative error for integration

4.5.2.4 L

```
double casimir::L
```

separation of plane and sphere

4.5.2.5 LbyR

```
double casimir::LbyR
```

L/R

4.5.2.6 ldim

```
int casimir::ldim
```

truncation value for vector space ℓ_max

4.5.2.7 R

```
double casimir::R
```

radius of sphere

4.5.2.8 y

```
double casimir::y
```

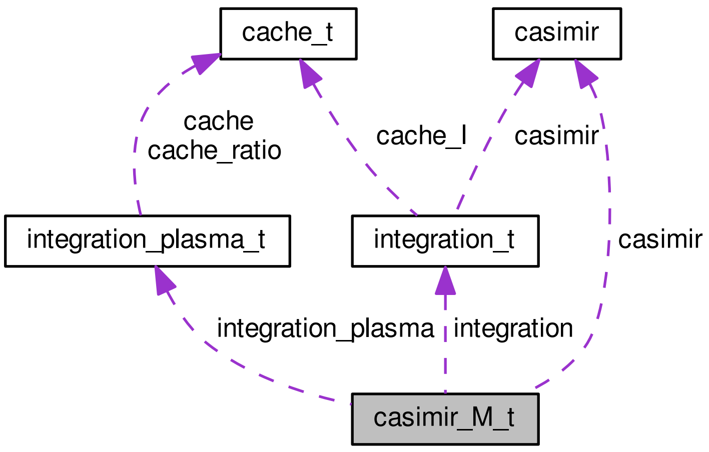
log(R/(R+L)/2)

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

- include/libcasimir.h

4.6 casimir_M_t Struct Reference

Collaboration diagram for casimir_M_t:



Data Fields

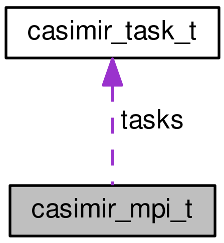
- casimir_t ∗ **casimir**
- int **m**
- int **lmin**
- integration_t ∗ **integration**
- integration_plasma_t ∗ **integration_plasma**
- double **xi_**
- double ∗ **al**
- double ∗ **bl**

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

- include/libcasimir.h

4.7 casimir_mpi_t Struct Reference

Collaboration diagram for casimir_mpi_t:



Data Fields

- double **L**
- double **R**
- double **T**
- double **omegap**
- double **gamma**
- double **cutoff**
- double **iepsrel**
- double **alpha**
- int **ldim**
- int **cores**
- bool **verbose**
- casimir_task_t ** **tasks**
- int **determinants**
- char **filename** [512]

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

- include/casimir.h

4.8 casimir_task_t Struct Reference

Data Fields

- int **index**
- int **m**
- double **xi_**
- double **recv**
- double **value**
- MPI_Request **request**
- int **state**

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

- include/casimir.h

4.9 integrand_plasma_t Struct Reference

Data Fields

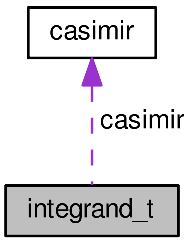
- int **nu**
- double **omegap**
- double **log_prefactor**

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

- integration.c

4.10 integrand_t Struct Reference

Collaboration diagram for integrand_t:



Data Fields

- int **nu**
- int **m**
- polarization_t **p**
- double **factor**
- double **alpha**
- double **log_normalization**
- casimir_t ∗ **casimir**

4.10.1 Detailed Description

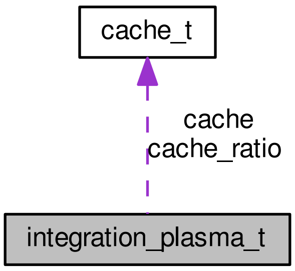
arguments for integrand in function K_integrand

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

- integration.c

4.11 integration_plasma_t Struct Reference

Collaboration diagram for integration_plasma_t:



Data Fields

- double **LbyR**
- double **alpha**
- double **omegap_**
- double **epsrel**
- cache_t ∗ **cache**
- cache_t ∗ **cache_ratio**

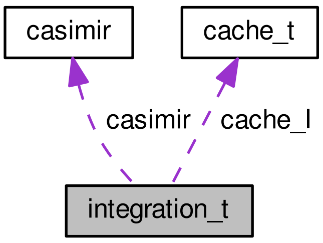
The documentation for this struct was generated from the following file:

- include/libcasimir.h

4.12 integration_t Struct Reference

```
#include <libcasimir.h>
```

Collaboration diagram for integration_t:



Data Fields

- casimir_t $*$ **casimir**
- int **m**
- double **alpha**
- double **epsrel**
- cache_t $*$ **cache_I**
- double $*$ **cache_K** [2]
- size_t **elems_cache_K**
- bool **is_pr**

4.12.1 Detailed Description

object for integration over k in matrix elements of round-trip operator

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

- include/libcasimir.h

4.13 kernel_args_t Struct Reference

Data Fields

- int **lmax**
- int **type**
- char **DN**
- double **alpha**
- double $*$ **cache_ratio**
- double $*$ **cache_K**

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

- include/cylinder.h

4.14 log_t Struct Reference

```
#include <misc.h>
```

Data Fields

- sign_t s
- double v

4.14.1 Detailed Description

represent number v by its sign and $\log|v|$

4.14.2 Field Documentation

4.14.2.1 s

```
sign_t log_t::s
```

sign of number

4.14.2.2 v

```
double log_t::v
```

logarithm of absolute value of number

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

- include/misc.h

4.15 material_t Struct Reference

```
#include <material.h>
```

Data Fields

- char filename [512]
- double calL
- double xi_min
- double xi_max
- size_t points
- double ∗ xi
- double ∗ epsm1
- double omegap_low
- double gamma_low
- double omegap_high
- double gamma_high

4.15.1 Detailed Description

material_t data type

4.15.2 Field Documentation

4.15.2.1 calL

```
double material_t::calL
```

$L + R$

4.15.2.2 epsm1

```
double* material_t::epsm1
```

tabulated dielectric function, $\epsilon(\mathrm{i}\xi) - 1$

4.15.2.3 filename

```
char material_t::filename[512]
```

material filename or \0\0...

4.15.2.4 gamma_high

```
double material_t::gamma_high
```

relaxation frequency for hight frequency extrapolation

4.15.2.5 gamma_low

```
double material_t::gamma_low
```

relaxation frequency for low frequency extrapolation

4.15.2.6 omegap_high

```
double material_t::omegap_high
```

plasma frequency for high frequency extrapolation

4.15.2.7 omegap_low

```
double material_t::omegap_low
```

plasma frequency for low frequency extrapolation

4.15.2.8 points

```
size_t material_t::points
```

number of points

4.15.2.9 xi

```
double* material_t::xi
```

tabulated frequencies ξ

4.15.2.10 xi_max

```
double material_t::xi_max
```

upper border of tabulated frequencies

4.15.2.11 xi_min

```
double material_t::xi_min
```

lower border of tabulated frequencies

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

- include/material.h

4.16 matrix_t Struct Reference

```
#include <matrix.h>
```

Data Fields

- size_t dim
- size_t dim2
- size_t lda
- double * M

4.16.1 Detailed Description

define matrix type

4.16.2 Field Documentation

4.16.2.1 dim

```
size_t matrix_t::dim
```

dimension of matrix

4.16.2.2 dim2

```
size_t matrix_t::dim2
```

square of dimension of matrix

4.16.2.3 lda

```
size_t matrix_t::lda
```

leading order

4.16.2.4 M

```
double* matrix_t::M
```

pointer to data

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

- include/matrix.h

Chapter 5

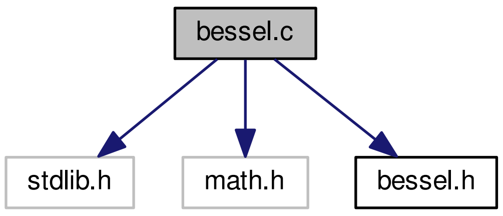
File Documentation

5.1 bessel.c File Reference

Computation of Bessel functions.

```
#include <stdlib.h>
#include <math.h>
#include "bessel.h"
```

Include dependency graph for bessel.c:



Functions

modified Bessel functions for integer orders

- double bessel_In (int n, double x)

  Modified Bessel function $I_n(x)$ for integer order n.
- double bessel_Kn (int n, double x)

  Modified Bessel function $K_n(x)$ for integer order n.
- double bessel_logKn_recursive (int n, double x)

  Logarithm of modified Bessel functions $K_n(x)$.
- double bessel_logKn (int n, double x)

  Logarithm of modified Bessel function $K_n(x)$ for integer order n.
- double bessel_logIn (int n, double x)

  Logarithm of modified Bessel function $I_n(x)$ for integer order n.

modified Bessel functions for arbitrary orders

- double bessel_ratioI (double nu, double x)

  Calculate $I_\nu(x)/I_{\nu+1}(x)$.
- double bessel_logInu_asymp (double nu, double x)

  Compute modified Bessel function $I_\nu(x)$ using asymptotic expansion.
- double bessel_logKnu_asymp (double nu, double x)

  Compute modified Bessel function $K_\nu(x)$ using asymptotic expansion.
- double bessel_logInu_series (double nu, double x)

  Compute modified Bessel functions $I_\nu(x)$ using series expansion.

modified Bessel functions for half-integer orders

- void bessel_logInKn_half (int n, const double x, double ∗logIn_p, double ∗logKn_p)

  Compute modified Bessel functions of first and second kind for half-integer orders.
- double bessel_logIn_half (int n, double x)

  Compute $\log I_{n+1/2}(x)$.
- double bessel_logKn_half (int n, double x)

  Compute $\log K_{n+1/2}(x)$.

modified Bessel functions for orders \f$n=0,1\f$

- static double I0_coeffs [ ]
- static double K0_coeffsA [ ]
- static double K0_coeffsB [ ]
- static double I1_coeffs [ ]
- static double K1_coeffsA [ ]
- static double K1_coeffsB [ ]
- static double chbevl (double x, double array[ ], int n)

  Evaluate Chebyshev series.
- double bessel_I0 (double x)

  Modified Bessel function $I_0(x)$.
- double bessel_logI0 (double x)

  Logarithm of modified Bessel function $I_0(x)$.
- double bessel_K0 (double x)

  Modified Bessel function $K_0(x)$.
- double bessel_logK0 (double x)

  Logarithm of modified Bessel function $K_0(x)$.
- double bessel_I1 (double x)

  Modified Bessel function $I_1(x)$.
- double bessel_logI1 (double x)

  Logarithm of modified Bessel function $I_1(x)$.
- double bessel_K1 (double x)

  Modified Bessel function $K_1(x)$.
- double bessel_logK1 (double x)

  Logarithm of modified Bessel function $K_1(x)$.

5.1.1 Detailed Description

Computation of Bessel functions.

Author

Stephen L. Moshier, Cephes Math Library Release 2.8, June 2000
Michael Hartmann `michael.hartmann@physik.uni-augsburg.de`

Date

October, 2019

5.1.2 Function Documentation

5.1.2.1 bessel_I0()

```
double bessel_I0 (
            double x )
```

Modified Bessel function $I_0(x)$.

See bessel_logI0.

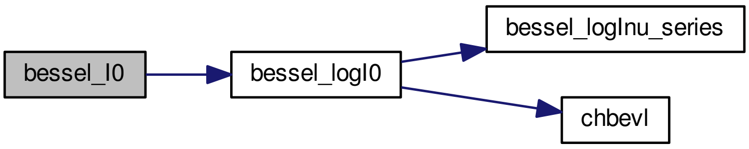
Parameters

in	*x*	argument

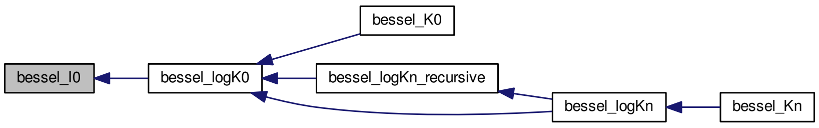
Return values

I0	$I_0(x)$

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.2 bessel_I1()

```
double bessel_I1 (
             double x )
```

Modified Bessel function $I_1(x)$.

See bessel_logI1.

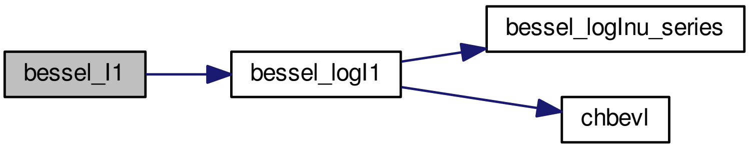
Parameters

in	*x*	argument

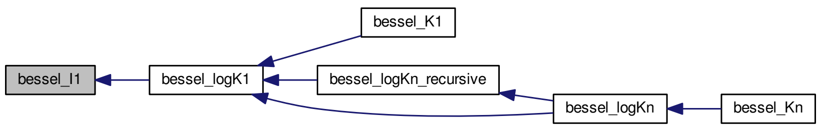
Return values

I1	$I_1(x)$

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.3 bessel_In()

```
double bessel_In (
             int n,
             double x )
```

Modified Bessel function $I_n(x)$ for integer order n.

See bessel_logIn.

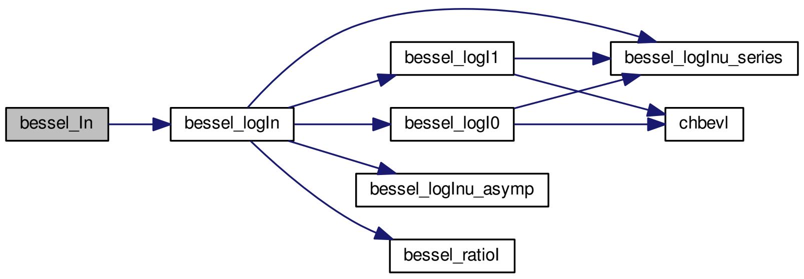
Parameters

in	*n*	order
in	*x*	argument

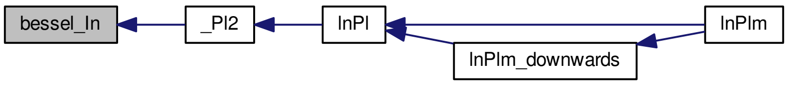
Return values

In	$I_n(x)$

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.4 bessel_K0()

```
double bessel_K0 (
             double x )
```

Modified Bessel function $K_0(x)$.

See bessel_logK0.

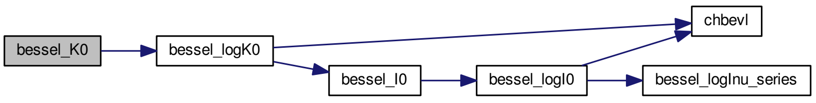
Parameters

in	*x*	argument

Return values

K0	$K_0(x)$

Here is the call graph for this function:



5.1.2.5 bessel_K1()

```
double bessel_K1 (
             double x )
```

Modified Bessel function $K_1(x)$.

See bessel_logK1.

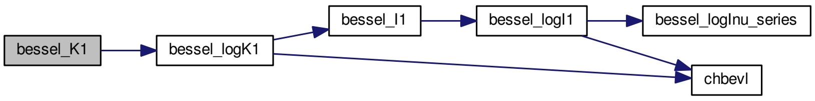
Parameters

`in`	*x*	argument

Return values

K1	$K_1(x)$

Here is the call graph for this function:



5.1.2.6 bessel_Kn()

```
double bessel_Kn (
              int n,
              double x )
```

Modified Bessel function $K_n(x)$ for integer order n.

See bessel_logKn.

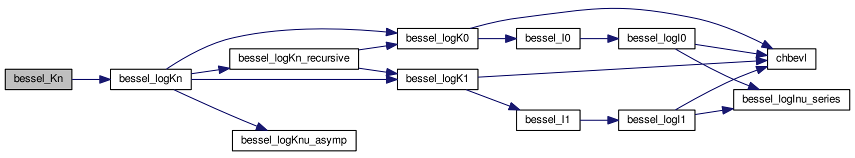
Parameters

`in`	*n*	order
`in`	*x*	argument

Return values

Kn	$K_n(x)$

Here is the call graph for this function:



5.1.2.7 bessel_logI0()

```
double bessel_logI0 (
             double x )
```

Logarithm of modified Bessel function $I_0(x)$.

- For $x < 0$ NAN (not a number) is returned.
- For $x = 0$ the value $\log I_0(0) = \log(1) = 0$ is returned.
- For $0 < x < 8$ a series expansion is used, see bessel_logInu_series.
- For $8 \leq x < 800$ a Chebychev expansion is used.
- For $x \geq 800$ the Hankel expansion

  $$I_0(x) \approx \frac{e^x}{\sqrt{2\pi x}} \left(1 + k + \frac{9}{2}k^2 + \frac{225}{6}k^3\right), \quad k = \frac{1}{8x}$$

  is used.

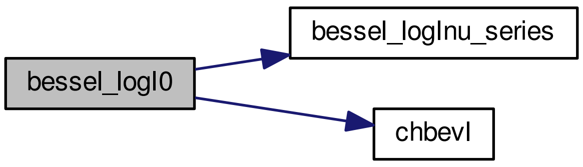
Parameters

in	*x*	argument

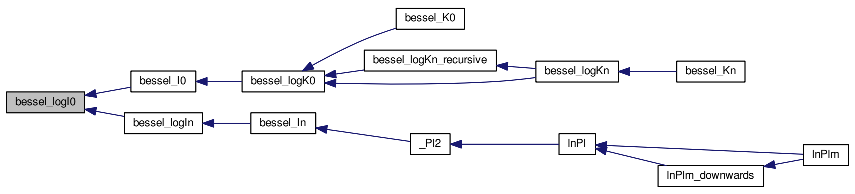
Return values

log←I0	$\log I_0(x)$

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.8 bessel_logI1()

```
double bessel_logI1 (
             double x )
```

Logarithm of modified Bessel function $I_1(x)$.

- For $x < 0$ NAN (not a number) is returned.
- For $0 < x < 8$ a series expansion is used, see bessel_logInu_series.
- For $8 \leq x < 800$ a Chebychev expansion is used.
- For $x \geq 800$ the Hankel expansion

$$I_0(x) \approx \frac{e^x}{\sqrt{2\pi x}} \left(1 - 3k - \frac{15}{2}k^2 - \frac{105}{2}k^3\right), \quad k = \frac{1}{8x}$$

  is used.

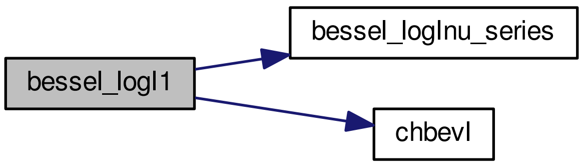
Parameters

`in`	*x*	argument

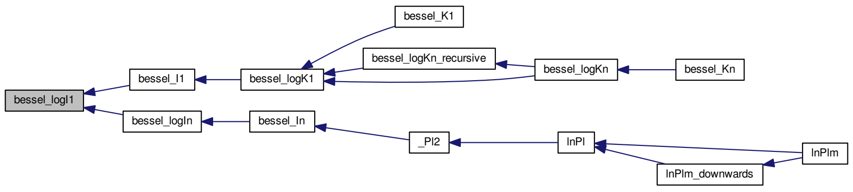
Return values

log↩ I1	$\log I_1(x)$

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.9 bessel_logIn()

```
double bessel_logIn (
             int n,
             double x )
```

Logarithm of modified Bessel function $I_n(x)$ for integer order n.

- For $n = 0$ and $n = 1$ the function calls bessel_logI0 or bessel_logI1.
- For $n \geq 100$ an asymptotic expansion is used, see bessel_logInu_asymp.
- For $n < 100$ and $x < 5\sqrt{n}$ a series expansion is used, see bessel_logInu_series.
- Otherwise, the function $I_n(x)$ is computed using the recurrence relation

$$I_{n-1}(x) = I_{n+1}(x) + \frac{2n}{x} I_n(x)$$

  in downwards direction using Miller's algorithm.

See also bessel_logI0, bessel_logI1, bessel_logInu_asymp, bessel_logInu_series, and bessel_ratioI.

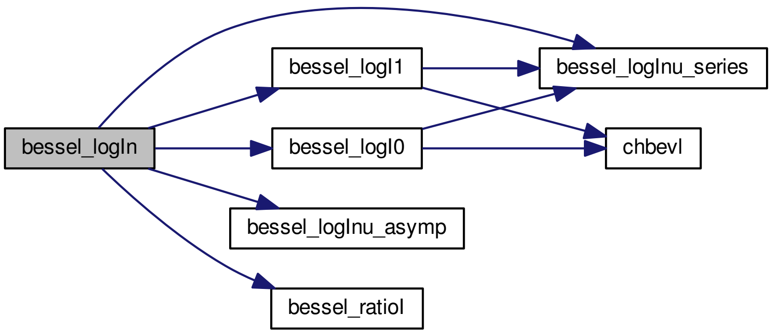
Parameters

in	*n*	order
in	*x*	argument

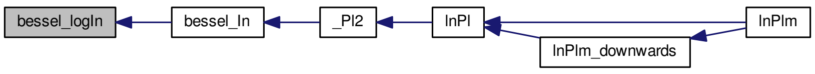
Return values

In	$\log I_n(x)$

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.10 bessel_logIn_half()

```
double bessel_logIn_half (
             int n,
             double x )
```

Compute $\log I_{n+1/2}(x)$.

Compute logarithm of modified Bessel function of the first kind for half-integer order $I_{n+1/2}(x)$.

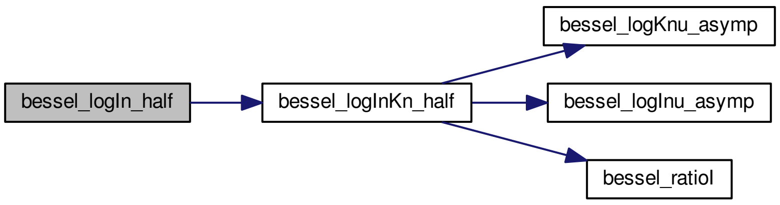
Parameters

in	*n*	order
in	*x*	argument

Return values

logI	$\log I_{n+1/2}(x)$

Here is the call graph for this function:



5.1.2.11 bessel_logInKn_half()

```
void bessel_logInKn_half (
             int n,
             const double x,
             double * logIn_p,
             double * logKn_p )
```

Compute modified Bessel functions of first and second kind for half-integer orders.

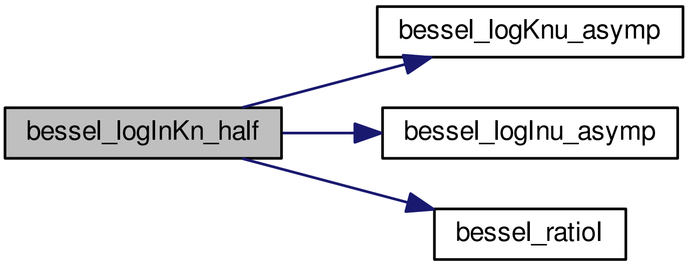
This function computes the logarithm of the modified Bessel functions $I_{n+1/2}(x)$ and $K_{n+1/2}(x)$. The results are saved in logIn_p and logKn_p.

If logIn_p or logKn_p is NULL, the variable is not referenced.

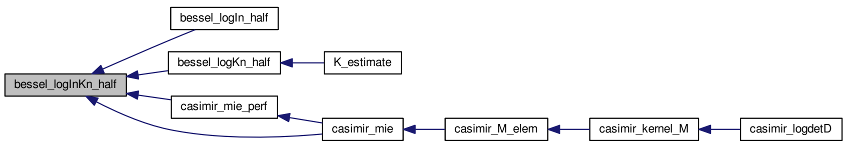
Parameters

in	*n*	order
in	*x*	argument
out	*logIn←_p*	pointer for $\log I_{n+1/2}(x)$
out	*logKn←_p*	pointer for $\log K_{n+1/2}(x)$

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.12 bessel_logInu_asymp()

```
double bessel_logInu_asymp (
            double nu,
            double x )
```

Compute modified Bessel function $I_\nu(x)$ using asymptotic expansion.

For $n \geq 100$ the asymptotic expansion is accurate.

See also `https://dlmf.nist.gov/10.41#ii`.

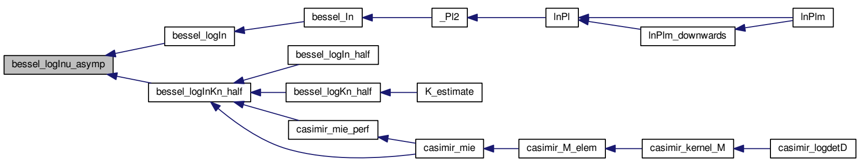
Parameters

`in`	*nu*	order
`in`	*x*	argument

Return values

logI	$\log I_\nu(x)$

Here is the caller graph for this function:



5.1.2.13 bessel_logInu_series()

```
double bessel_logInu_series (
              double nu,
              double x )
```

Compute modified Bessel functions $I_\nu(x)$ using series expansion.

The modified Bessel function is computed using the series expansion

$$I_\nu(x) = \sum_{m=0}^\infty \frac{1}{m!\Gamma(1+m+\nu)} \left(\frac{x}{2}\right)^{2m+\nu}.$$

The functions succeeds for orders up to $\nu \le 100000$ when $x \le 10\sqrt{\nu}$. For larger values of x the function might return NAN.

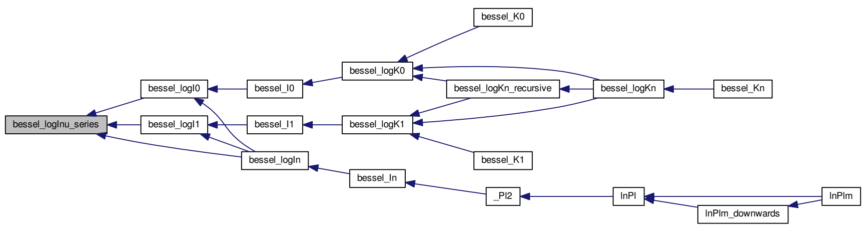
Parameters

in	*nu*	order
in	*x*	argument

Return values

Inu	$I_\nu(x)$ if successful or NAN otherwise

Here is the caller graph for this function:



5.1.2.14 bessel_logK0()

```
double bessel_logK0 (
            double x )
```

Logarithm of modified Bessel function $K_0(x)$.

- For small arguments $0 < x < 10^{-8}$, the limiting form

  $$K_0(x) \approx -\log(x/2) - \gamma$$

  for $x \to 0$ where γ denotes the Euler-Mascheroni constant is used.
- For large arguments $x \geq 800$, the Hankel expansion

  $$K_0(x) \approx \sqrt{\frac{\pi}{2x}} e^{-x} \left(1 - k + \frac{9}{2}k^2 - \frac{225}{6}k^3 \right), \quad k = \frac{1}{8x}$$

  is used.
- For intermediate values, the range is partitioned into the two intervals $[10^{-8}, 2)$ and $(2, 800)$ and Chebyshev polynomial expansions are employed in each interval.

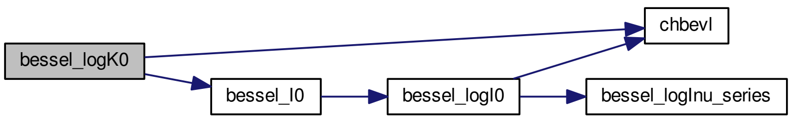
Parameters

in	*x*	argument

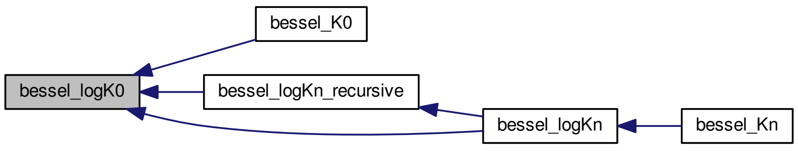
Return values

logK0	$\log K_0(x)$

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.15 bessel_logK1()

```
double bessel_logK1 (
              double x )
```

Logarithm of modified Bessel function $K_1(x)$.

- For small arguments $x < 10^{-8}$, the limiting form

  $$K_1(x) \approx 1/x$$

  for $x \to 0$ is used.
- For large arguments $x \geq 800$, the Hankel expansion

  $$K_1(x) \approx \sqrt{\frac{\pi}{2x}} e^{-x} \left(1 + 3k - \frac{15}{2}k^2 + \frac{315}{6}k^3\right), \quad k = \frac{1}{8x}$$

  is used.
- For intermediate values, the range is partitioned into the two intervals $[10^{-8}, 8)$ and $[8, 800)$ and Chebyshev polynomial expansions are employed in each interval.

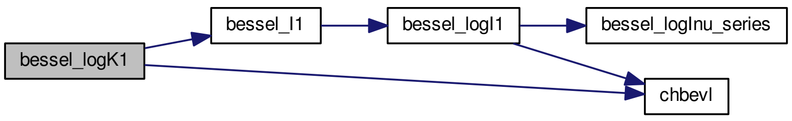
Parameters

in	*x*	argument

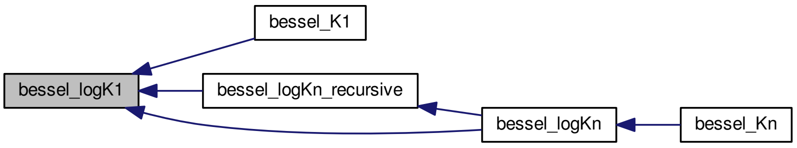
Return values

logK1	$\log K_1(x)$

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.16 bessel_logKn()

```
double bessel_logKn (
             int n,
             double x )
```

Logarithm of modified Bessel function $K_n(x)$ for integer order n.

- For $n = 0$ and $n = 1$ the function calls bessel_logK0 or bessel_logK1.
- For $n \geq 100$ an asymptotic expansion is used, see bessel_logKnu_asymp.
- Otherwise, the function is computed using a recursion relation, see bessel_logKn_recursive.

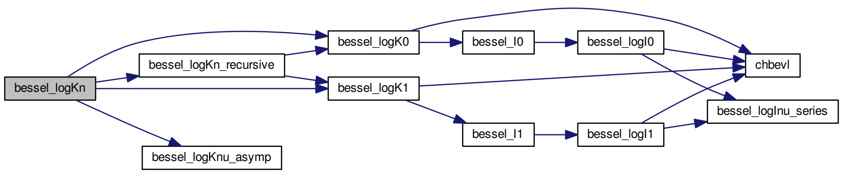
Parameters

in	*n*	order
in	*x*	argument

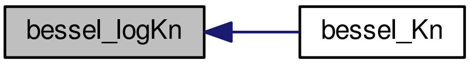
Return values

Kn	$\log K_n(x)$

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.17 bessel_logKn_half()

```
double bessel_logKn_half (
            int n,
            double x )
```

Compute $\log K_{n+1/2}(x)$.

Compute logarithm of modified Bessel function of the second kind $K_{n+1/2}(x)$.

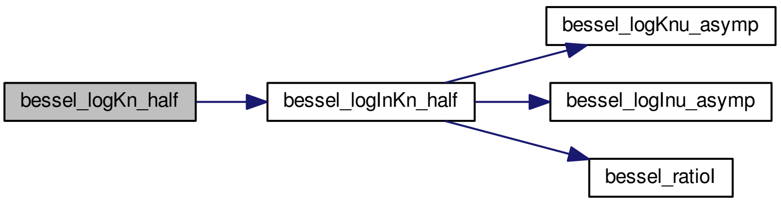
Parameters

in	*n*	order
in	*x*	argument

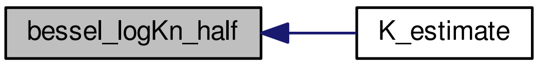
Return values

logK	$K_{n+1/2}(x)$

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.18 bessel_logKn_recursive()

```
double bessel_logKn_recursive (
              int n,
              double x )
```

Logarithm of modified Bessel functions $K_n(x)$.

The Bessel function $K_n(x)$ for integer order n is computed using the recurrence relation

$$K_{j+1}(x) = K_{j-1}(x) + \frac{2j}{x} K_j(x)$$

in upwards direction. The Bessel functions $K_0(x)$ and $K_1(x)$ are computed using bessel_logK0 and bessel_logK1.

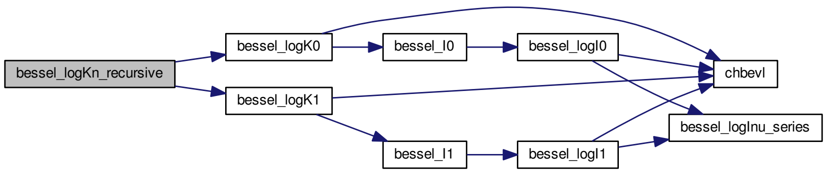
Parameters

in	*n*	order
in	*x*	argument

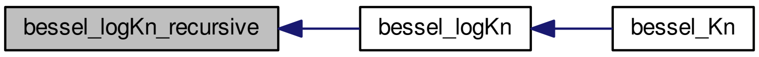
Return values

logKn	$K_n(x)$

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.19 bessel_logKnu_asymp()

```
double bessel_logKnu_asymp (
             double nu,
             double x )
```

Compute modified Bessel function $K_\nu(x)$ using asymptotic expansion.

For $n \geq 100$ the asymptotic expansion is accurate.

See also https://dlmf.nist.gov/10.41#ii.

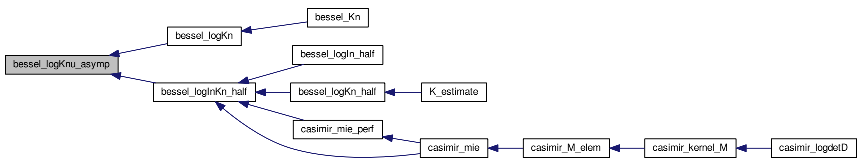
Parameters

in	*nu*	order
in	*x*	argument

Return values

logI	$\log I_\nu(x)$

Here is the caller graph for this function:



5.1.2.20 bessel_ratioI()

```
double bessel_ratioI (
             double nu,
             double x )
```

Calculate $I_\nu(x)/I_{\nu+1}(x)$.

Compute the ratio of the modified Bessel functions of the first kind $I_\nu(x)/I_{\nu+1}(x)$ using a continued fraction, see `https://dlmf.nist.gov/10.33`.

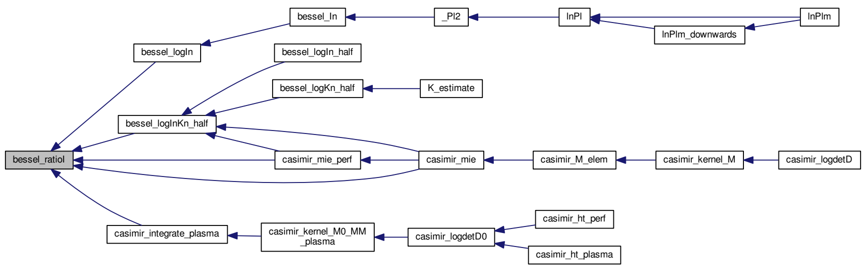
Parameters

nu	order
x	argument

Return values

ratio	$I_\nu(x)/I_{\nu+1}(x)$

Here is the caller graph for this function:



5.1.2.21 chbevl()

```
static double chbevl (
             double x,
             double array[],
             int n )  [static]
```

Evaluate Chebyshev series.

Evaluates the series

$$y = \sum_{i=0}^{N-1}{}' \operatorname{coef}[i] \cdot T_i(x/2)$$

of Chebyshev polynomials T_i at argument $x/2$. The prime indicates that the term for $i = 0$ has to be weighted by a factor $1/2$.

Coefficients are stored in reverse order, i.e. the zero order term is last in the array. Note: n is the number of coefficients, not the order.

If coefficients are for the interval a to b, x must have been transformed to $x \to 2(2x-b-a)/(b-a)$ before entering the routine. This maps x from (a,b) to $(-1,1)$, over which the Chebyshev polynomials are defined.

If the coefficients are for the inverted interval, in which (a,b) is mapped to $(1/b,1/a)$, the transformation required is $x \to 2(2ab/x-b-a)/(b-a)$. If b is infinity, this becomes $x \to 4a/x-1$.

Speed: Taking advantage of the recurrence properties of the Chebyshev polynomials, the routine requires one more addition per loop than evaluating a nested polynomial of the same degree.

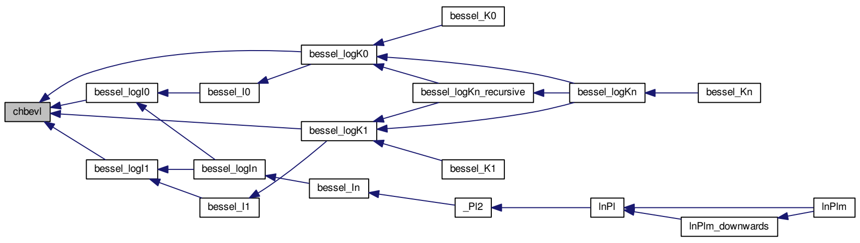
Parameters

`in`	*x*	Chebyshev series is evaluated at this point
`in`	*array*	Chebyshev coefficients
`in`	*n*	number of Chebyshev coefficients, number of elements of array

Return values

eval	Chebychev series evaluated at x

Here is the caller graph for this function:



5.1.3 Variable Documentation

5.1.3.1 I0_coeffs

```
double I0_coeffs[]  [static]
```

Initial value:

```
=
{
    -7.23318048787475395456E-18,
    -4.83050448594418207126E-18,
     4.46562142029675999901E-17,
     3.46122286769746109310E-17,
    -2.82762398051658348494E-16,
    -3.42548561967721913462E-16,
     1.77256013305652638360E-15,
     3.81168066935262242075E-15,
    -9.55484669882830764870E-15,
    -4.15056934728722208663E-14,
     1.54008621752140982691E-14,
     3.85277838274214270114E-13,
     7.18012445138366623367E-13,
    -1.79417853150680611778E-12,
    -1.32158118404477131188E-11,
    -3.14991652796324136454E-11,
     1.18891471078464383424E-11,
     4.94060238822496958910E-10,
     3.39623202570838634515E-9,
     2.26666899049817806459E-8,
     2.04891858946906374183E-7,
     2.89137052083475648297E-6,
     6.88975834691682398426E-5,
     3.36911647825569408990E-3,
     8.04490411014108831608E-1
}
```

Chebyshev coefficients for $\exp(-x)\sqrt{x}I_0(x)$ in the inverted interval $[8, \infty]$.

$$\lim_{x\to\infty} \exp(-x)\sqrt{x}I_0(x) = 1/\sqrt{2\pi}.$$

5.1.3.2 I1_coeffs

```
double I1_coeffs[ ]  [static]
```

Initial value:

```
=
{
     7.51729631084210481353E-18,
     4.41434832307170791151E-18,
    -4.65030536848935832153E-17,
    -3.20952592199342395980E-17,
     2.96262899764595013876E-16,
     3.30820231092092828324E-16,
    -1.88035477551078244854E-15,
    -3.81440307243700780478E-15,
     1.04202769841288027642E-14,
     4.27244001671195135429E-14,
    -2.10154184277266431302E-14,
    -4.08355111109219731823E-13,
    -7.19855177624590851209E-13,
     2.03562854414708950722E-12,
     1.41258074366137813316E-11,
     3.25260358301548823856E-11,
    -1.89749581235054123450E-11,
    -5.58974346219658380687E-10,
    -3.83538038596423702205E-9,
    -2.63146884688951950684E-8,
    -2.51223623787020892529E-7,
    -3.88256480887769039346E-6,
    -1.10588938762623716291E-4,
    -9.76109749136146840777E-3,
     7.78576235018280120474E-1
}
```

Chebyshev coefficients for $\exp(-x)\sqrt{x}I_1(x)$ in the inverted interval $[8, \infty]$.

$$\lim_{x\to\infty} \exp(-x)\sqrt{x}I_1(x) = 1/\sqrt{2\pi}.$$

5.1.3.3 K0_coeffsA

```
double K0_coeffsA[ ]  [static]
```

Initial value:

```
=
{
    1.37446543561352307156E-16,
    4.25981614279661018399E-14,
    1.03496952576338420167E-11,
    1.90451637722020886025E-9,
    2.53479107902614945675E-7,
    2.28621210311945178607E-5,
    1.26461541144692592338E-3,
    3.59799365153615016266E-2,
    3.44289899924628486886E-1,
    -5.35327393233902768720E-1
}
```

Chebyshev coefficients for $K_0(x)+\log(x/2)I_0(x)$ in the interval $[0, 2]$. The odd order coefficients are all zero; only the even order coefficients are listed.

$$\lim_{x\to 0} (K_0(x) + \log(x/2)I_0(x)) = -\gamma.$$

5.1.3.4 K0_coeffsB

```
double K0_coeffsB[]  [static]
```

Initial value:

```
= {
     5.30043377268626276149E-18,
    -1.64758043015242134646E-17,
     5.21039150503902756861E-17,
    -1.67823109680541210385E-16,
     5.51205597852431940784E-16,
    -1.84859337734377901440E-15,
     6.34007647740507060557E-15,
    -2.22751332699166985548E-14,
     8.03289077536357521100E-14,
    -2.98009692317273043925E-13,
     1.14034058820847496303E-12,
    -4.51459788337394416547E-12,
     1.85594911495471785253E-11,
    -7.95748924447710747776E-11,
     3.57739728140030116597E-10,
    -1.69753450938905987466E-9,
     8.57403401741422608519E-9,
    -4.66048989768794782956E-8,
     2.76681363944501510342E-7,
    -1.83175552271911948767E-6,
     1.39498137188764993662E-5,
    -1.28495495816278026384E-4,
     1.56988388573005337491E-3,
    -3.14481013119645005427E-2,
     2.44030308206595545468E0
}
```

Chebyshev coefficients for $\exp(x)\sqrt{x}K_0(x)$ in the inverted interval $[2, \infty]$.

$$\lim_{x \to \infty} \exp(x)\sqrt{x}K_0(x) = \sqrt{\pi/2}.$$

5.1.3.5 K1_coeffsA

```
double K1_coeffsA[]  [static]
```

Initial value:

```
=
{
    -7.02386347938628759343E-18,
    -2.42744985051936593393E-15,
    -6.66690169419932900609E-13,
    -1.41148839263352776110E-10,
    -2.21338763073472585583E-8,
    -2.43340614156596823496E-6,
    -1.73028895751305206302E-4,
    -6.97572385963986435018E-3,
    -1.22611180822657148235E-1,
    -3.53155960776544875667E-1,
     1.52530022733894777053E0
}
```

Chebyshev coefficients for $x\,(K_1(x) - \log(x/2)I_1(x))$ in the interval $[0, 2]$.

$$\lim_{x \to 0} x\,(K_1(x) - \log(x/2)I_1(x)) = 1.$$

5.1.3.6 K1_coeffsB

```
double K1_coeffsB[] [static]
```

Initial value:

```
=
{
    -5.7567444836650171575 5E-18,
     1.79405087314755922667E-17,
    -5.68946255844285935196E-17,
     1.83809354436663880070E-16,
    -6.05704724837331885336E-16,
     2.03870316562433424052E-15,
    -7.01983709041831346144E-15,
     2.47715442448130437068E-14,
    -8.97670518232499435011E-14,
     3.34841966607842919884E-13,
    -1.28917396095102890680E-12,
     5.13963967348173025100E-12,
    -2.12996783842756842877E-11,
     9.21831518760500529508E-11,
    -4.19035475934189648750E-10,
     2.01504975519703286596E-9,
    -1.03457624656780970260E-8,
     5.74108412545004946722E-8,
    -3.50196060308781257119E-7,
     2.40648494783721712015E-6,
    -1.93619797416608296024E-5,
     1.95215518471351631108E-4,
    -2.85781685962277938680E-3,
     1.03923736576817238437E-1,
     2.72062619048444266945E0
}
```

Chebyshev coefficients for $\exp(x)\sqrt{x}K_1(x)$ in the interval $[2,\infty]$.

$$\lim_{x\to\infty}\exp(x)\sqrt{x}K_1(x)=\sqrt{\pi/2}.$$

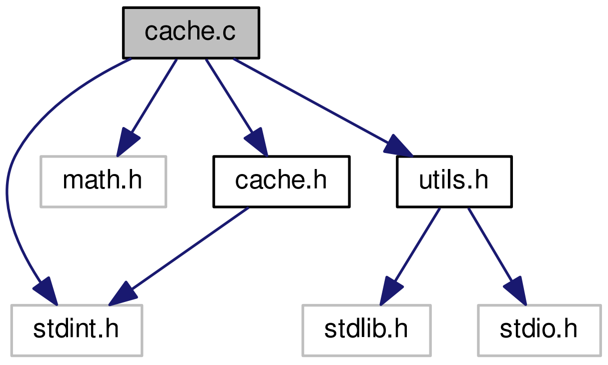
5.2 cache.c File Reference

implementation of a simple cache using a hash table

```
#include <stdint.h>
#include <math.h>
#include "utils.h"
```

```
#include "cache.h"
```

Include dependency graph for cache.c:



Functions

- cache_t ∗ cache_new (int entries, double filling)

  Create a new cache.
- void cache_free (cache_t ∗cache)

  Free cache instance.
- void cache_insert (cache_t ∗cache, uint64_t key, double value)

  Insert element into cache.
- double cache_lookup (cache_t ∗cache, uint64_t key)

  Find element corresponding to key key.

5.2.1 Detailed Description

implementation of a simple cache using a hash table

Author

Michael Hartmann michael.hartmann@physik.uni-augsburg.de

Date

December, 2018

5.2.2 Function Documentation

5.2.2.1 cache_free()

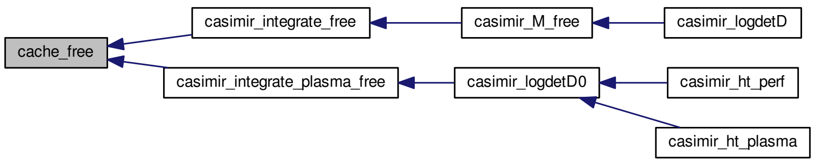
```
void cache_free (
            cache_t ∗ cache )
```

Free cache instance.

Parameters

cache	cache instance

Here is the caller graph for this function:



5.2.2.2 cache_insert()

```
void cache_insert (
            cache_t * cache,
            uint64_t key,
            double value )
```

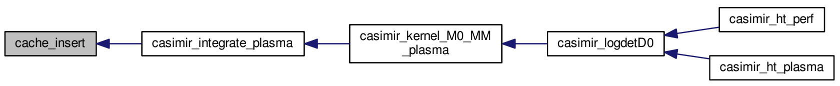
Insert element into cache.

Insert the element value with key key to the cache.

Parameters

cache	cache instance
key	key
value	value

Here is the caller graph for this function:



5.2.2.3 cache_lookup()

```
double cache_lookup (
            cache_t * cache,
            uint64_t key )
```

Find element corresponding to key key.

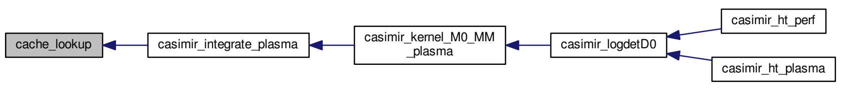
Parameters

cache	cache instance
key	key

Return values

element	if found
NAN	otherwise

Here is the caller graph for this function:



5.2.2.4 cache_new()

```
cache_t* cache_new (
            int entries,
            double filling )
```

Create a new cache.

Create a new cache instance. This cache is quite specific. You specifiy the maximum number of entries and a filling level. The cache is implemented as a hash map that maps keys (uint64_t) to doubles.

If the cache cannot contain more elements, the oldest entry will be thrown away, similar to a FIFO. There is no logic to detect collisions. If there is a collision, the old value will be overwritten. You can specifiy a filling level ($0 <$ filling < 1).

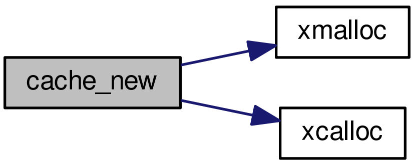
Parameters

entries	maximum number of entries the cache can store
filling	filling level

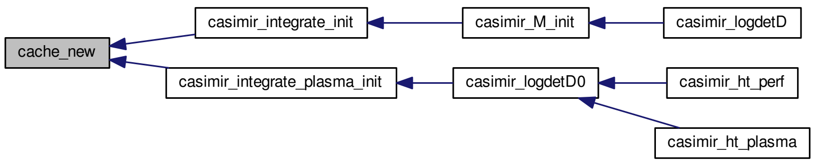
Return values

cache	cache_t instance

Here is the call graph for this function:



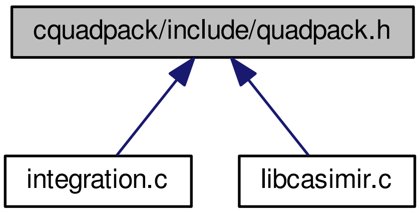
Here is the caller graph for this function:



5.3 cquadpack/include/quadpack.h File Reference

library for numerical integration of one-dimensional functions

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



Macros

- #define GK_7_15 1
- #define GK_10_21 2
- #define GK_15_31 3
- #define GK_20_41 4
- #define GK_25_51 5
- #define GK_30_61 6

Functions

- double dqagi (double f(double, void ∗), double bound, int inf, double epsabs, double epsrel, double ∗abserr, int ∗neval, int ∗ier, void ∗user_data)

    Integration over (semi-) infinite intervals.
- double dqags (double f(double, void ∗), double a, double b, double epsabs, double epsrel, double ∗abserr, int ∗neval, int ∗ier, void ∗user_data)

    Integration over finite intervals.
- double dqage (double f(double, void ∗), double a, double b, double epsabs, double epsrel, int irule, double ∗abserr, int ∗neval, int ∗ier, int ∗last, void ∗user_data)

5.3.1 Detailed Description

library for numerical integration of one-dimensional functions

Date

January, 2019

5.3.2 Macro Definition Documentation

5.3.2.1 GK_10_21

```
#define GK_10_21 2
```

Gauss-Kronrod 10-21 rule

5.3.2.2 GK_15_31

```
#define GK_15_31 3
```

Gauss-Kronrod 15-31 rule

5.3.2.3 GK_20_41

```
#define GK_20_41 4
```

Gauss-Kronrod 20-41 rule

5.3.2.4 GK_25_51

```
#define GK_25_51 5
```

Gauss-Kronrod 25-51 rule

5.3.2.5 GK_30_61

```
#define GK_30_61 6
```

Gauss-Kronrod 30-61 rule

5.3.2.6 GK_7_15

```
#define GK_7_15 1
```

Gauss-Kronrod 7-15 rule

5.3.3 Function Documentation

5.3.3.1 dqage()

```
double dqage (
            double  fdouble, void *,
            double a,
            double b,
            double epsabs,
            double epsrel,
            int irule,
            double * abserr,
            int * neval,
            int * ier,
            int * last,
            void * user_data )
```

Approximation to definite integral

Allows user's choice of Gauss-Kronrod integration rule.

error messages:

- ier=1: Maximum number of subdivisions allowed has been achieved. It is advised to analyze the integrand in order to determine the integration difficulties.
- ier=2: The occurrence of roundoff error is detected, which prevents the requested tolerance from being achieved.
- ier=3: Extremely bad integrand behaviour occurs at some points of the integration interval.
- ier=6: The input is invalid.

Parameters

in	*f*	double precision function to be integrated
in	*a*	lower limit of integration
in	*b*	upper limit of integration

Parameters

in	*epsabs*	absolute accuracy requested
in	*epsrel*	relative accuracy requested
in	*epsrel*	relative accuracy requested
in	*irule*	integration rule to be used (GK_7_15, GK_7_15, GK_10_21, GK_15_31, GK_20_41, GK_25_51, or GK_30_61)
out	*abserr*	estimate of the modulus of the absolute error, which should equal or exceed abs(I-result)
out	*neval*	number of integrand evaluations
out	*ier*	error message; ier=0 for normal and reliable termination, otherwise ier>0
out	*last*	number of subintervals actually produced in the subdivision process
out	*user_data*	pointer that will be passed as second argument to integrand function f

Return values

result	approximation to the integral

5.3.3.2 dqagi()

```
double dqagi (
             double  fdouble, void *,
             double bound,
             int inf,
             double epsabs,
             double epsrel,
             double * abserr,
             int * neval,
             int * ier,
             void * user_data )
```

Integration over (semi-) infinite intervals.

Adaptive integration routine which handles functions to be integrated between -infinity to +infinity, or between either of those limits and some finite, real boundary.

The adaptive strategy compares results of integration over the interval with the sum of results obtained from integration of bisected interval. Since error estimates are available from each regional integration, the interval with the largest error is bisected and new results are computed. This bisection process is continued until the error is less than the prescribed limit or convergence failure is determined.

Note that bisection, in the sense used above, refers to bisection of the transformed interval.

error messages:

- ier=0: Normal and reliable termination of the routine. It is assumed that the requested accuracy has been achieved.
- ier=1: Maximum number of subdivisions allowed has been achieved. It is advised to analyze the integrand in order to determine the integration difficulties.

- ier=2: The occurrence of roundoff error is detected, which prevents the requested tolerance from being achieved. The error may be under-estimated.
- ier=3: Extremely bad integrand behaviour occurs at some points of the integration interval.
- ier=4: The algorithm does not converge. Roundoff error is detected in the extrapolation table. It is assumed that the requested tolerance cannot be achieved, and that the returned result is the best which can be obtained.
- ier=5: The integral is probably divergent, or slowly convergent. It must be noted that divergence can occur with any other value of ier.
- ier=6: The input is invalid.

Parameters

`in`	*f*	double precision function to be integrated
`in`	*bound*	optional finite bound on integral
`in`	*inf*	specifies range of integration as follows: • inf=-1: range is from -infinity to bound, • inf=+1: range is from bound to +infinity, • inf=+2: range is from -infinity to +infinity, (bound is ignored in this case)
`in`	*epsabs*	absolute accuracy requested
`in`	*epsrel*	relative accuracy requested
`out`	*abserr*	estimate of the modulus of the absolute error, which should equal or exceed abs(I-result)
`out`	*neval*	number of integrand evaluations
`out`	*ier*	error message; ier=0 for normal and reliable termination, otherwise ier>0
`out`	*user_data*	pointer that will be passed as second argument to integrand function f

Return values

result	approximation to the integral

5.3.3.3 dqags()

```
double dqags (
             double  fdouble, void *,
             double a,
             double b,
             double epsabs,
             double epsrel,
             double * abserr,
             int * neval,
             int * ier,
             void * user_data )
```

Integration over finite intervals.

Adaptive integration routine which handles functions to be integrated between two finite bounds.

The adaptive strategy compares results of integration over the given interval with the sum of results obtained from integration over a bisected interval. Since error estimates are available from each regional integration, the region with the largest error is bisected and new results are computed. This bisection process is continued until the error is less than the prescribed limit or convergence failure is determined.

error messages:

- ier=1 Maximum number of subdivisions allowed has been achieved. It is advised to analyze the integrand in order to determine the integration difficulties.
- ier=2: The occurrence of roundoff error is detected, which prevents the requested tolerance from being achieved. The error may be under-estimated.
- ier=3: Extremely bad integrand behaviour occurs at some points of the integration interval.
- ier=4: The algorithm does not converge. Roundoff error is detected in the extrapolation table. It is presumed that the requested tolerance cannot be achieved, and that the returned result is the best which can be obtained.
- ier=5: The integral is probably divergent, or slowly convergent. It must be noted that divergence can occur with any other value of ier.
- ier=6: The input is invalid.

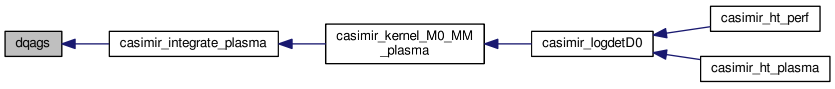
Parameters

`in`	*f*	double precision function to be integrated
`in`	*a*	lower limit of integration
`in`	*b*	upper limit of integration
`in`	*epsabs*	absolute accuracy requested
`in`	*epsrel*	relative accuracy requested
`out`	*abserr*	estimate of the modulus of the absolute error, which should equal or exceed abs(I-result)
`out`	*neval*	number of integrand evaluations
`out`	*ier*	error message; ier=0 for normal and reliable termination, otherwise ier>0
`out`	*user_data*	pointer that will be passed as second argument to integrand function f

Return values

result	approximation to the integral

Here is the caller graph for this function:

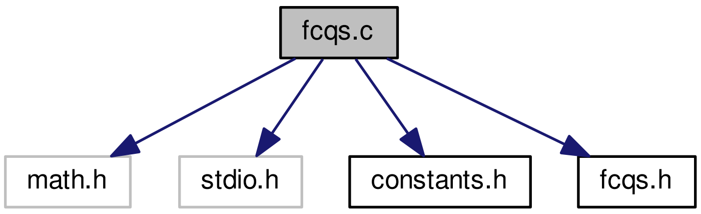


5.4 fcqs.c File Reference

exponentially convergent Fourier-Chebshev quadrature scheme (experimental)

```
#include <math.h>
#include <stdio.h>
#include "constants.h"
#include "fcqs.h"
```

Include dependency graph for fcqs.c:



Macros

- #define MMIN 5
- #define **MMAX** 2560

Functions

- static double cot2 (double x)

  Squared cotangent.
- static double wi_semiinf (double ti, double L, double N)

  Weights for quadrature scheme (semiinfinite interval)
- static double wi_finite (double ti, double N)

  Weights for quadrature scheme (infinite interval)
- double fcqs_semiinf (double f(double, void ∗), void ∗args, double ∗epsrel, int ∗neval, double L, int ∗ier)

  Integrate function $f(x)$ over interval $[0, \infty)$.
- double fcqs_finite (double f(double, void ∗), void ∗args, double a, double b, double ∗epsrel, int ∗neval, int ∗ier)

  Integrate function $f(x)$ over interval $[a, b]$.

5.4.1 Detailed Description

exponentially convergent Fourier-Chebshev quadrature scheme (experimental)

Author

Michael Hartmann michael.hartmann@physik.uni-augsburg.de

Date

December, 2018

5.4.2 Macro Definition Documentation

5.4.2.1 MMIN

```
#define MMIN 5
```

MMIN and MMAX must be chosen in a way that there exists a positive integer k such that MMAX = MMIN $*$ 2$**$k.

5.4.3 Function Documentation

5.4.3.1 cot2()

```
static double cot2 (
             double x )  [static]
```

Squared cotangent.

Compute square of cotangent of x, i.e. $(\cos x/\sin x)^2$

Parameters

in	*x*	argument

Return values

cot2	$\cot^2(x)$

5.4.3.2 fcqs_finite()

```
double fcqs_finite (
             double  fdouble, void *,
             void * args,
             double a,
             double b,
             double * epsrel,
             int * neval,
             int * ier )
```

Integrate function $f(x)$ over interval $[a, b]$.

This method uses an adaptive exponentially convergent Fourier-Chebshev quadrature to compute the integral over the interval $[a, b]$. The method approximately doubles the number of nodes until the desired accuracy is achieved.

Values of ier after integration:

- ier=0: evaluation successful
- ier=1: relative accuracy epsrel must be positive
- ier=2: integrand returned NAN
- ier=3: integrand returned +inf or -inf
- ier=4: could not achieve desired accuracy

Parameters

`in`	*f*	integrand
`in`	*args*	pointer given to f when called
`in`	*a*	left border of integration
`in`	*b*	right border of integration
`in,out`	*epsrel*	on begin desired accuracy, afterwards achieved accuracy
`out`	*neval*	number of evaluations of integrand (may be set to NULL)
`out`	*ier*	exit code

Return values

integral	numerical value of integral

5.4.3.3 fcqs_semiinf()

```
double fcqs_semiinf (
             double  fdouble, void *,
             void * args,
             double * epsrel,
             int * neval,
             double L,
             int * ier )
```

Integrate function $f(x)$ over interval $[0, \infty)$.

This method uses an adaptive exponentially convergent Fourier-Chebshev quadrature to compute the integral over the interval $[0, \infty)$. The method approximately doubles the number of nodes until the desired accuracy is achieved.

Values of ier after integration:

- ier=0: evaluation successful
- ier=1: relative accuracy epsrel must be positive
- ier=2: integrand returned NAN
- ier=3: integrand returned +inf or -inf
- ier=4: could not achieve desired accuracy

Parameters

`in`	*f*	integrand
`in`	*args*	pointer given to f when called
`in,out`	*epsrel*	on begin desired accuracy, afterwards achieved accuracy
`in`	*neval*	number of evaluations of integrand (may be set to NULL)
`in`	*L*	boosting parameter
`out`	*ier*	exit code

Return values

integral	numerical value of integral

5.4.3.4 wi_finite()

```
static double wi_finite (
             double ti,
             double N )  [static]
```

Weights for quadrature scheme (infinite interval)

The weights correspond to (3.1e) of [1]. Here we have used that $\cos(j\pi) = (-1)^j$.

References:

- [1] Boyd, Exponentially Convergent Fourier-Chebychev Quadrature Schemes on Bounded and Infinite Intervals, Journal of Scientific Computing, Vol. 2, No. 2 (1987)

Parameters

`in`	*ti*	node
`in`	*N*	order / number of points

Return values

wi	weight

5.4.3.5 wi_semiinf()

```
static double wi_semiinf (
             double ti,
             double L,
             double N )  [static]
```

Weights for quadrature scheme (semiinfinite interval)

The weights correspond to (3.2e) of [1]. Here we have used that $\cos(j\pi) = (-1)^j$.

References:

- [1] Boyd, Exponentially Convergent Fourier-Chebychev Quadrature Schemes on Bounded and Infinite Intervals, Journal of Scientific Computing, Vol. 2, No. 2 (1987)

Parameters

`in`	*ti*	node
`in`	*L*	boosting parameter
`in`	*N*	order / number of points

Return values

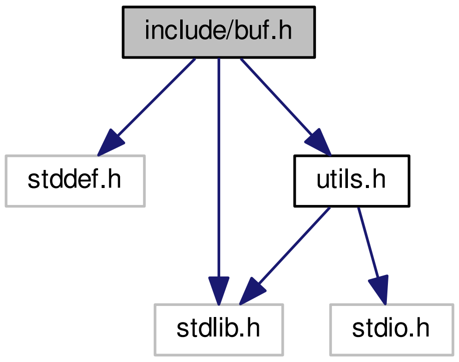
wi	weight

5.5 include/buf.h File Reference

growable memory buffers for C99

```
#include <stddef.h>
#include <stdlib.h>
#include "utils.h"
```

Include dependency graph for buf.h:



Data Structures

- struct buf

Macros

- #define **BUF_INIT_CAPACITY** 32
- #define **buf_ptr**(v) ((struct buf ∗)((char ∗)(v) - offsetof(struct buf, buffer)))
- #define buf_free(v)
- #define buf_size(v) ((v) ? buf_ptr((v))->size : 0)
- #define buf_capacity(v) ((v) ? buf_ptr((v))->capacity : 0)
- #define buf_push(v, e)
- #define buf_pop(v) ((v)[--buf_ptr(v)->size])
- #define buf_grow(v, n) ((v) = buf_grow1((v), sizeof(∗(v)), n))
- #define buf_trunc(v, n) ((v) = buf_grow1((v), sizeof(∗(v)), n - buf_capacity(v)))
- #define buf_clear(v) ((v) ? (buf_ptr((v))->size = 0) : 0)

Functions

- static void ∗ **buf_grow1** (void ∗v, size_t esize, ptrdiff_t n)

5.5.1 Detailed Description

growable memory buffers for C99

Author

Christopher Wellons

Date

September, 2018 This is free and unencumbered software released into the public domain.

Original version from `https://github.com/skeeto/growable-buf`.

Note: buf_push(), buf_grow(), buf_trunc(), and buf_free() may change the buffer pointer, and any previously-taken pointers should be considered invalidated.

Example usage:

```
float *values = 0;
for (size_t i = 0; i < 25; i++)
    buf_push(values, rand() / (float)RAND_MAX);
for (size_t i = 0; i < buf_size(values); i++)
    printf("values[%zu] = %f\n", i, values[i]);
buf_free(values);
```

5.5.2 Macro Definition Documentation

5.5.2.1 buf_capacity

```
#define buf_capacity(
              v ) ((v) ? buf_ptr((v))->capacity : 0)
```

return the total capacity of the buffer (size_t)

5.5.2.2 buf_clear

```
#define buf_clear(
              v ) ((v) ?  (buf_ptr((v))->size = 0) :  0)
```

set buffer size to 0 (for push/pop)

5.5.2.3 buf_free

```
#define buf_free(
              v )
```

Value:

```
do { \
        if (v) { \
            free(buf_ptr((v))); \
            (v) = 0; \
        } \
    } while (0)
```

destroy and free the buffer

5.5.2.4 buf_grow

```
#define buf_grow(
              v,
              n ) ((v) = buf_grow1((v), sizeof(*(v)), n))
```

increase buffer capactity by (ptrdiff_t) N elements

5.5.2.5 buf_pop

```
#define buf_pop(
              v ) ((v)[--buf_ptr(v)->size])
```

remove and return an element E from the end

5.5.2.6 buf_push

```
#define buf_push(
              v,
              e )
```

Value:

```
do { \
        if (buf_capacity((v)) == buf_size((v))) { \
            (v) = buf_grow1(v, sizeof(*(v)), \
                            !buf_capacity((v)) ? \
                              BUF_INIT_CAPACITY : \
                              buf_capacity((v))); \
        } \
        (v)[buf_ptr((v))->size++] = (e); \
    } while (0)
```

append an element E to the end

5.5.2.7 buf_size

```
#define buf_size(
              v ) ((v) ? buf_ptr((v))->size : 0)
```

return the number of elements in the buffer (size_t)

5.5.2.8 buf_trunc

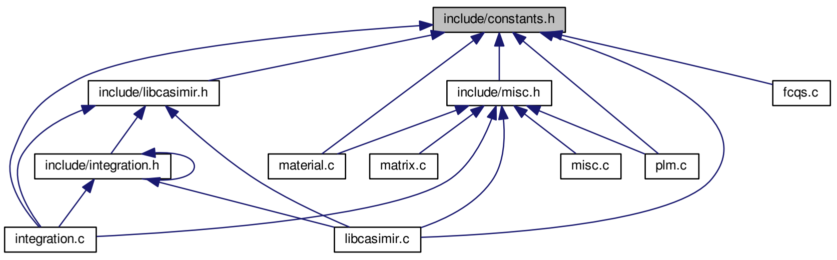
```
#define buf_trunc(
              v,
              n ) ((v) = buf_grow1((v), sizeof(*(v)), n - buf_capacity(v)))
```

set buffer capactity to exactly (ptrdiff_t) N elements

5.6 include/constants.h File Reference

define macros and constants

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



Macros

- #define MIN(a, b) (((a)<(b))?(a):(b))
- #define MAX(a, b) ((((a))>((b)))?((a)):((b)))
- #define SGN(val) ((0 < (val)) - ((val) < 0))
- #define pow_2(x) ((x)*(x))
- #define M_PI 3.14159265358979323846
- #define M_LOG2 0.6931471805599453
- #define M_LOGPI 1.1447298858494002
- #define CASIMIR_hbar 1.0545718e-34
- #define CASIMIR_hbar_eV 6.582119514e-16
- #define CASIMIR_kB 1.38064852e-23
- #define CASIMIR_c 299792458.

Typedefs

- typedef signed char sign_t

5.6.1 Detailed Description

define macros and constants

Author

Michael Hartmann michael.hartmann@physik.uni-augsburg.de

Date

December, 2018

5.6.2 Macro Definition Documentation

5.6.2.1 CASIMIR_c

```
#define CASIMIR_c 299792458.
```

speed of light c in vacuum [m/s]

5.6.2.2 CASIMIR_hbar

```
#define CASIMIR_hbar 1.0545718e-34
```

reduced Planck constant $\hbar$ [m² kg / s]

5.6.2.3 CASIMIR_hbar_eV

```
#define CASIMIR_hbar_eV 6.582119514e-16
```

reduced Planck constant $\hbar$ [eV s/rad]

5.6.2.4 CASIMIR_kB

```
#define CASIMIR_kB 1.38064852e-23
```

Boltzman constant k_B [m² kg / (K s²)]

5.6.2.5 M_LOG2

```
#define M_LOG2 0.6931471805599453
```

$\log(2)$

5.6.2.6 M_LOGPI

```
#define M_LOGPI 1.1447298858494002
```

$\log(\pi)$

5.6.2.7 M_PI

```
#define M_PI 3.14159265358979323846
```

value for $\pi = 3.141592...$

5.6.2.8 MAX

```
#define MAX(
              a,
              b ) ((((a))>((b)))?((a)):((b)))
```

macro to get maximum of two numbers

5.6.2.9 MIN

```
#define MIN(
              a,
              b ) (((a)<(b))?(a):(b))
```

macro to get minimum of two numbers

5.6.2.10 pow_2

```
#define pow_2(
              x ) ((x)*(x))
```

compute x^2

5.6.2.11 SGN

```
#define SGN(
              val ) ((0 < (val)) - ((val) < 0))
```

macro to get sign of numbers

5.6.3 Typedef Documentation

5.6.3.1 sign_t

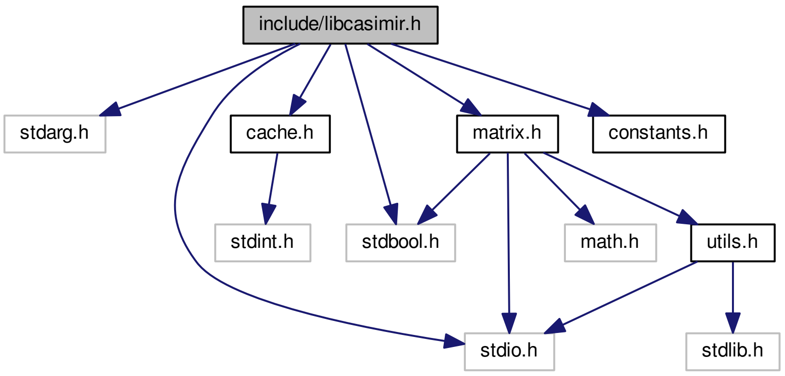
```
typedef signed char sign_t
```

define sign_t as a signed char, because "char can be either signed or unsigned depending on the implementation"

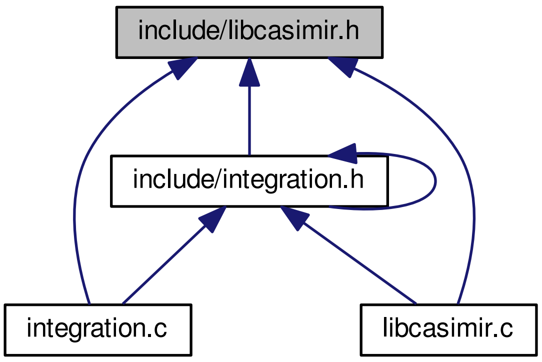
5.7 include/libcasimir.h File Reference

```
#include <stdarg.h>
#include <stdbool.h>
#include <stdio.h>
#include "cache.h"
#include "matrix.h"
#include "constants.h"
```

Include dependency graph for libcasimir.h:



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



Data Structures

- struct casimir
- struct integration_t
- struct integration_plasma_t
- struct casimir_M_t

Macros

- #define CASIMIR_MINIMUM_LDIM 20
- #define CASIMIR_FACTOR_LDIM 5
- #define CASIMIR_EPSREL 1e-8
- #define CASIMIR_CACHE_ELEMS 1000000

Typedefs

- typedef struct casimir casimir_t

Enumerations

- enum polarization_t { **TE**, **TM** }

Functions

- void casimir_build (FILE ∗stream, const char ∗prefix)

  Print information on build to stream.
- void casimir_info (casimir_t ∗self, FILE ∗stream, const char ∗prefix)

  Print object information to stream.
- double casimir_lnLambda (int l1, int l2, int m)

  Calculate logarithm $\Lambda^{(m)}_{\ell_1\ell_2}$.
- casimir_t ∗ casimir_init (double R, double L)

  Create a new Casimir object.
- void casimir_free (casimir_t ∗self)

  Free memory for Casimir object.
- double casimir_epsilonm1_plate (casimir_t ∗self, double xi_)

  Evaluate dielectric function of the plate.
- double casimir_epsilonm1_sphere (casimir_t ∗self, double xi_)

  Evaluate dielectric function of the sphere.
- void casimir_set_epsilonm1 (casimir_t ∗self, double(∗epsilonm1)(double xi_, void ∗userdata), void ∗userdata)

  Set dielectric function for plate and sphere.
- void casimir_set_epsilonm1_plate (casimir_t ∗self, double(∗epsilonm1)(double xi_, void ∗userdata), void ∗userdata)

  Set dielectric function of plate.
- void casimir_set_epsilonm1_sphere (casimir_t ∗self, double(∗epsilonm1)(double xi_, void ∗userdata), void ∗userdata)

  Set dielectric function of sphere.
- int casimir_get_ldim (casimir_t ∗self)

  Get dimension of vector space.
- int casimir_set_ldim (casimir_t ∗self, int ldim)

Set dimension of vector space.
- detalg_t casimir_get_detalg (casimir_t ∗self)

  Get algorithm to calculate determinant.
- int casimir_set_detalg (casimir_t ∗self, detalg_t detalg)

  Set algorithm to calculate deterimant.
- double casimir_get_epsrel (casimir_t ∗self)

  Get relative error for numerical integration.
- int casimir_set_epsrel (casimir_t ∗self, double epsrel)

  Set relative error for numerical integration.
- void casimir_mie (casimir_t ∗self, double xi_, int l, double ∗lna, double ∗lnb)

  Return logarithm of Mie coefficients a_ℓ, b_ℓ for arbitrary metals.
- void casimir_mie_perf (casimir_t ∗self, double xi_, int l, double ∗lna, double ∗lnb)

  Calculate Mie coefficients a_ℓ, b_ℓ for perfect reflectors.
- double casimir_kernel_M (int i, int j, void ∗args_)

  Kernel of round-trip matrix.
- casimir_M_t ∗ casimir_M_init (casimir_t ∗self, int m, double xi_)

  Initialize casimir_M_t object.
- double casimir_M_elem (casimir_M_t ∗self, int l1, int l2, char p1, char p2)

  Compute matrix elements of round-trip operator.
- void casimir_M_free (casimir_M_t ∗self)

  Free casimir_M_t object.
- double casimir_logdetD (casimir_t ∗self, double xi_, int m)

  Compute $\log\det\mathcal{D}^{(m)}\left(\frac{\xi\mathcal{L}}{c}\right)$.
- void casimir_fresnel (casimir_t ∗self, double xi_, double k, double ∗r_TE, double ∗r_TM)

  Calculate Fresnel coefficients r_TE and r_TM for arbitrary metals.
- int casimir_estimate_lminmax (casimir_t ∗self, int m, size_t ∗lmin_p, size_t ∗lmax_p)

  Estimate ℓ_min and ℓ_max.
- double casimir_epsilonm1_perf (__attribute__((unused)) double xi_, __attribute__((unused)) void ∗userdata)

  Dielectric function for perfect reflectors.
- double casimir_epsilonm1_drude (double xi_, void ∗userdata)

  Dielectric function for Drude reflectors.
- double casimir_ht_drude (casimir_t ∗casimir)

  Compute high-temperature limit for Drude metals.
- double casimir_ht_perf (casimir_t ∗casimir, double eps)

  Compute free energy in the high-temperature limit for perfect reflectors.
- double casimir_ht_plasma (casimir_t ∗casimir, double omegap_, double eps)

  Compute free energy in the high-temperature limit for plasma model.
- double casimir_kernel_M0_EE (int i, int j, void ∗args)

  Kernel for EE block.
- double casimir_kernel_M0_MM (int i, int j, void ∗args)

  Kernel for MM block.
- double casimir_kernel_M0_MM_plasma (int i, int j, void ∗args_)

  Kernel for MM block (plasma model)
- void casimir_logdetD0 (casimir_t ∗self, int m, double omegap, double ∗EE, double ∗MM, double ∗MM_↩
plasma)

  Compute $\log\det\mathcal{D}^{(m)}(\xi=0)$ for EE and/or MM contribution.

5.7.1 Detailed Description

Author

Michael Hartmann michael.hartmann@physik.uni-augsburg.de

Date

December, 2017

5.7.2 Macro Definition Documentation

5.7.2.1 CASIMIR_CACHE_ELEMS

```
#define CASIMIR_CACHE_ELEMS 1000000
```

default number of elems of the cache for I integrals

5.7.2.2 CASIMIR_EPSREL

```
#define CASIMIR_EPSREL 1e-8
```

default relative error for integration

5.7.2.3 CASIMIR_FACTOR_LDIM

```
#define CASIMIR_FACTOR_LDIM 5
```

by default: lmax=ceil(5/LbyR)

5.7.2.4 CASIMIR_MINIMUM_LDIM

```
#define CASIMIR_MINIMUM_LDIM 20
```

minimum value for lmax

5.7.3 Typedef Documentation

5.7.3.1 casimir_t

```
typedef struct casimir casimir_t
```

The Casimir object. This structure stores all essential information about temperature, geometry and the reflection properties of the mirrors.

Do not modify the attributes of the structure yourself!

5.7.4 Enumeration Type Documentation

5.7.4.1 polarization_t

```
enum polarization_t
```

typoe for polarization: either TE or TM.

5.7.5 Function Documentation

5.7.5.1 casimir_build()

```
void casimir_build (
            FILE * stream,
            const char * prefix )
```

Print information on build to stream.

The information contains compiler, build time, git head and git branch if available. If prefix is not NULL, the string prefix will added in front of each line.

Parameters

stream	output stream
prefix	prefix of each line or NULL

5.7.5.2 casimir_epsilonm1_drude()

```
double casimir_epsilonm1_drude (
            double xi,
            void * userdata )
```

Dielectric function for Drude reflectors.

Dielectric function for Drude

$$\epsilon(\xi) - 1 = \frac{\omega_\mathrm{P}^2}{\xi(\xi+\gamma)}$$

The parameters ω_P and γ must be provided by userdata:

- userdata[0] = ω_P in rad/s
- userdata[1] = γ in rad/s

Parameters

`in`	*xi*	frequency in rad/s
`in`	*userdata*	userdata

Return values

epsilon	epsilon(xi)

5.7.5.3 casimir_epsilonm1_perf()

```
double casimir_epsilonm1_perf (
            __attribute__((unused)) double xi_,
            __attribute__((unused)) void * userdata )
```

Dielectric function for perfect reflectors.

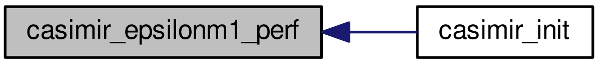
Parameters

`in`	*xi_*	ignored
`in`	*userdata*	ignored

Return values

inf	$\epsilon(\xi) = \infty$

Here is the caller graph for this function:



5.7.5.4 casimir_epsilonm1_plate()

```
double casimir_epsilonm1_plate (
            casimir_t * self,
            double xi_ )
```

Evaluate dielectric function of the plate.

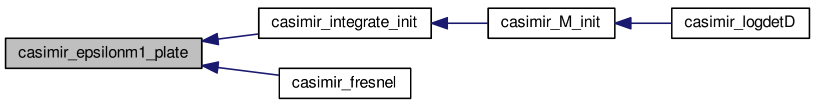
Parameters

in	*self*	Casimir object
in	*xi←_*	$\xi\mathcal{L}/c$

Return values

epsm1	$\epsilon(\mathrm{i}\xi)$

Here is the caller graph for this function:



5.7.5.5 casimir_epsilonm1_sphere()

```
double casimir_epsilonm1_sphere (
            casimir_t * self,
            double xi_ )
```

Evaluate dielectric function of the sphere.

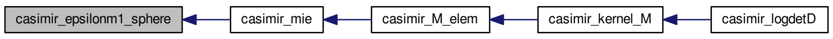
Parameters

in	*self*	Casimir object
in	*xi_*	$\xi\mathcal{L}/c$

Return values

epsm1	$\epsilon(\mathrm{i}\xi)$

Here is the caller graph for this function:



5.7.5.6 casimir_estimate_lminmax()

```
int casimir_estimate_lminmax (
              casimir_t * self,
              int m,
              size_t * lmin_p,
              size_t * lmax_p )
```

Estimate ℓ_min and ℓ_max.

Estimate the vector space: The main contributions comes from the vicinity $\ell_1 = \ell_2 = X$ and only depend on geometry, L/R, and the quantum number m. This function calculates X using the formula in the high-temperature limit and calculates $\ell_\mathrm{min}, \ell_\mathrm{max}$.

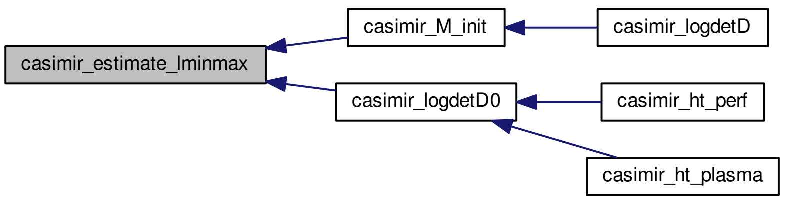
Parameters

in	*self*	Casimir object
in	*m*	quantum number
out	*lmin_p*	minimum value of ℓ
out	*lmax_p*	maximum value of ℓ

Return values

l	approximately the value of ℓ where $\mathcal{M}^m_{\ell\ell}$ is maximal

Here is the caller graph for this function:



5.7.5.7 casimir_free()

```
void casimir_free (
            casimir_t * self )
```

Free memory for Casimir object.

Free allocated memory for the Casimir object self.

Parameters

in,out	*self*	Casimir object

5.7.5.8 casimir_fresnel()

```
void casimir_fresnel (
            casimir_t * self,
            double xi_,
            double k_,
            double * r_TE,
            double * r_TM )
```

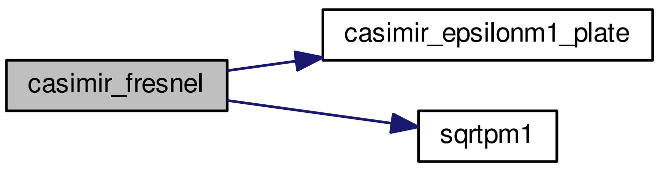
Calculate Fresnel coefficients $r_{\rm TE}$ and $r_{\rm TM}$ for arbitrary metals.

This function calculates the Fresnel coefficients $r_p = r_p(i\xi, k)$ for $p = \rm TE, TM$.

Parameters

in	*self*	Casimir object
in	*xi_*	$\xi\mathcal{L}/c$
in	*k_*	$k\mathcal{L}$
in,out	*r_TE*	Fresnel coefficient for TE mode
in,out	*r_TM*	Fresnel coefficient for TM mode

Here is the call graph for this function:



5.7.5.9 casimir_get_detalg()

```
detalg_t casimir_get_detalg (
             casimir_t * self )
```

Get algorithm to calculate determinant.

Parameters

in	*self*	Casimir object

Return values

detalg	

5.7.5.10 casimir_get_epsrel()

```
double casimir_get_epsrel (
             casimir_t * self )
```

Get relative error for numerical integration.

See casimir_set_epsrel.

Return values

epsrel	relative error

5.7.5.11 casimir_get_ldim()

```
int casimir_get_ldim (
            casimir_t * self )
```

Get dimension of vector space.

See casimir_set_ldim.

Parameters

in,out	*self*	Casimir object

Return values

ldim	dimension of vector space

5.7.5.12 casimir_ht_drude()

```
double casimir_ht_drude (
            casimir_t * casimir )
```

Compute high-temperature limit for Drude metals.

For Drude metals the Fresnel coefficients become $r_{\mathrm{TM}} = 1$, $r_{\mathrm{TE}} = 0$ for $\xi \to 0$, i.e. only the EE polarization block needs to be considered.

For Drude the free energy for $\xi = 0$ can be computed analytically. We use Eq. (8) from Ref. [1] to compute the contribution.

References:

- [1] Bimonte, Emig, "Exact results for classical Casimir interactions: Dirichlet and Drude model in the sphere-sphere and sphere-plane geometry", Phys. Rev. Lett. 109 (2012), https://doi.org/10.1103/↩PhysRevLett.109.160403

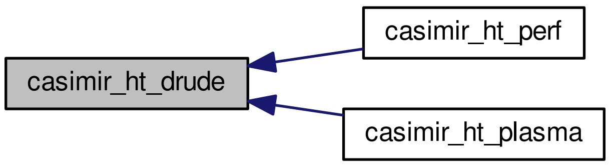
Parameters

in	*casimir*	Casimir object

Return values

F	free energy in units of $k_{\mathrm{B}}T$

Here is the caller graph for this function:



5.7.5.13 casimir_ht_perf()

```
double casimir_ht_perf (
             casimir_t * casimir,
             double eps )
```

Compute free energy in the high-temperature limit for perfect reflectors.

For perfect reflectors the Fresnel coefficients become $r_\mathrm{TM} = 1$, $r_\mathrm{TE} = -1$ in the limit $\xi \to 0$, and only the polarization blocks EE and MM need to be considered.

The contribution for EE, i.e. Drude, can be computed analytically, see casimir_ht_drude. For the MM block we numerically compute the determinants up to $m = M$ until

$$\frac{\log\det\mathcal{D}^{(M)}(0)}{\sum_{m=0}^{M}{}' \log\det\mathcal{D}^{(m)}(0)} < \epsilon\,.$$

We use Ref. [1] to compute the contribution for $m = 0$.

References:

- [1] Bimonte, Classical Casimir interaction of perfectly conducting sphere and plate (2017), https://arxiv.org/abs/1701.06461

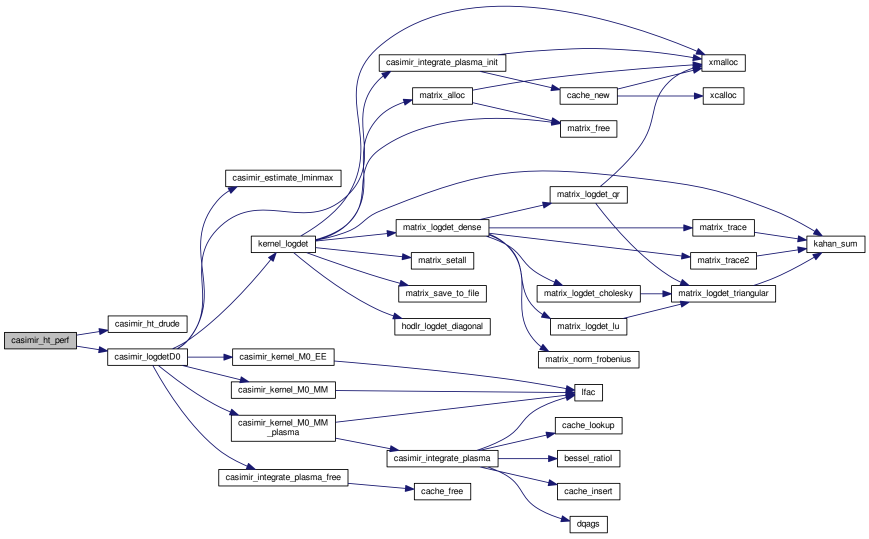
Parameters

in	*casimir*	Casimir object
in	*eps*	ϵ abort criterion

Return values

energy	free energy in units of $k_\mathrm{B}T$

Here is the call graph for this function:



5.7.5.14 casimir_ht_plasma()

```
double casimir_ht_plasma (
            casimir_t * casimir,
            double omegap,
            double eps )
```

Compute free energy in the high-temperature limit for plasma model.

The abort criterion eps is the same as in casimir_ht_perf.

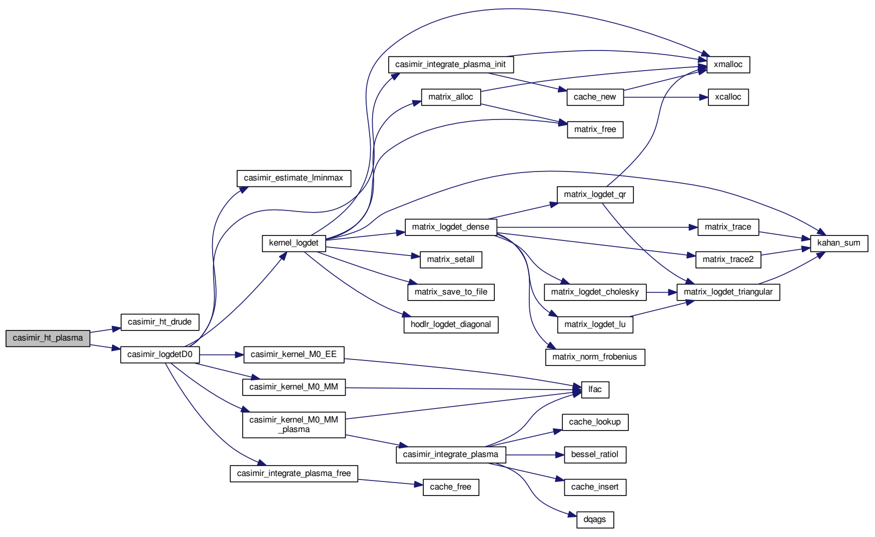
Parameters

in	*casimir*	Casimir object
in	*omegap*	plasma frequency in rad/s
in	*eps*	abort criterion

Return values

F	free energy in units of $k_\mathrm{B}T$

Here is the call graph for this function:



5.7.5.15 casimir_info()

```
void casimir_info (
            casimir_t * self,
            FILE * stream,
            const char * prefix )
```

Print object information to stream.

Print information about the object self to stream.

Parameters

self	Casimir object
stream	where to print the string
prefix	if prefix != NULL: start every line with the string contained in prefix

5.7.5.16 casimir_init()

```
casimir_t* casimir_init (
            double R,
            double L )
```

Create a new Casimir object.

This function will initialize a Casimir object. By default the dielectric function corresponds to perfect reflectors, i.e. $\epsilon(\xi) = \infty$.

By default, the value of ℓ_dim is chosen by:

$$\ell_\mathrm{dim} = \mathrm{ceil}\left(\max\left(\mathrm{CASIMIR\_MINIMUM\_LDIM}, \mathrm{CASIMIR\_FACTOR\_LDIM} \cdot \frac{R}{L}\right)\right)$$

Restrictions: $L/R > 0$

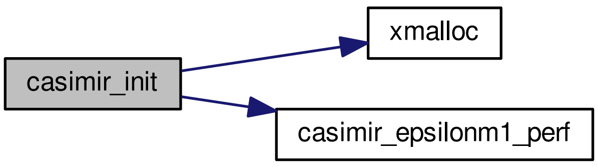
Parameters

in	*R*	radius of sphere in m
in	*L*	smallest separation between sphere and plate in m

Return values

object	Casimir object if successful
NULL	if an error occured

Here is the call graph for this function:



5.7.5.17 casimir_kernel_M()

```
double casimir_kernel_M (
            int i,
            int j,
            void * args_ )
```

Kernel of round-trip matrix.

This function returns the matrix elements of the round-trip operator $\mathcal{M}^{(m)}$.

The round-trip matrix is a $2\ell_\mathrm{dim} \times 2\ell_\mathrm{dim}$ matrix, the matrix elements start at 0, i.e. $0 \le i, j < 2\ell_\mathrm{dim}$.

This function is intended to be passed as a callback to kernel_logdet. If you want to compute matrix elements of the round-trip operator, it is probably simpler to use casimir_M_elem.

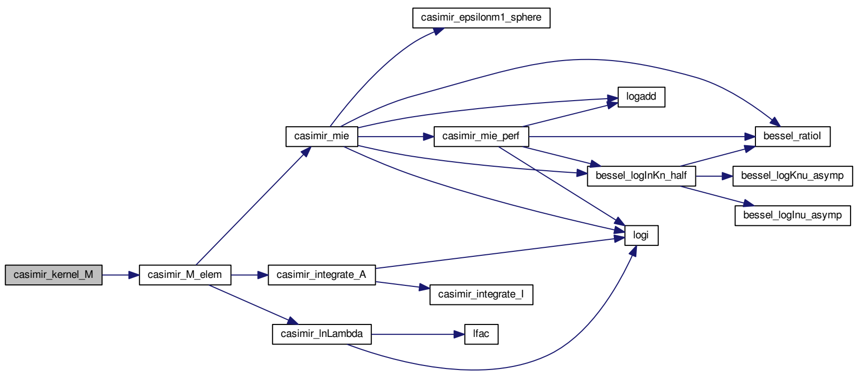
Parameters

in	*i*	row
in	*j*	column
in	*args←_*	casimir_M_t object, see casimir_M_init

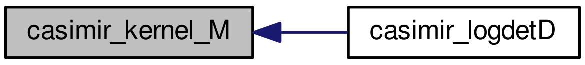
Return values

Mij	$\mathcal{M}_{ij}^{(m)}(\xi)$

Here is the call graph for this function:



Here is the caller graph for this function:



5.7.5.18 casimir_kernel_M0_EE()

```
double casimir_kernel_M0_EE (
            int i,
            int j,
            void * args_ )
```

Kernel for EE block.

Function that returns matrix elements of the round-trip matrix $\mathcal{M}$ for $\xi = 0$ and polarization $p_1 = p_2 = \mathrm{E}$.

See also casimir_logdetD0.

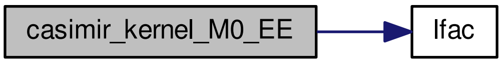
Parameters

in	*i*	row (starting from 0)
in	*j*	column (starting from 0)
in	*args*← _	pointer to casimir_M_t object

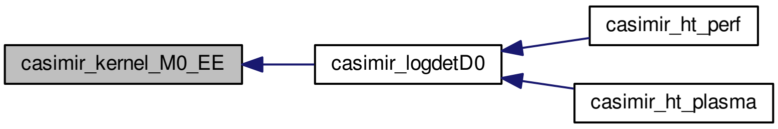
Return values

Mij	matrix element

Here is the call graph for this function:



Here is the caller graph for this function:



5.7.5.19 casimir_kernel_M0_MM()

```
double casimir_kernel_M0_MM (
            int i,
            int j,
            void * args_ )
```

Kernel for MM block.

Function that returns matrix elements of round-trip matrix $\mathcal{M}$ for $\xi = 0$ and polarization $p_1 = p_2 = \mathrm{M}$.

See also casimir_logdetD0.

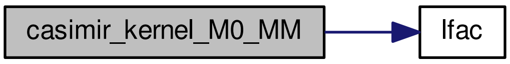
Parameters

in	*i*	row (starting from 0)
in	*j*	column (starting from 0)
in	*args_*	pointer to casimir_M_t object

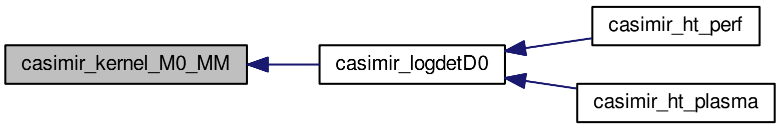
Return values

Mij	matrix element

Here is the call graph for this function:



Here is the caller graph for this function:



5.7.5.20 casimir_kernel_M0_MM_plasma()

```
double casimir_kernel_M0_MM_plasma (
            int i,
            int j,
            void * args_ )
```

Kernel for MM block (plasma model)

Function that returns matrix elements of round-trip matrix $\mathcal{M}$ for $\xi = 0$ and polarization $p_1 = p_2 = \mathrm{M}$ (plasma model).

See also casimir_logdetD0.

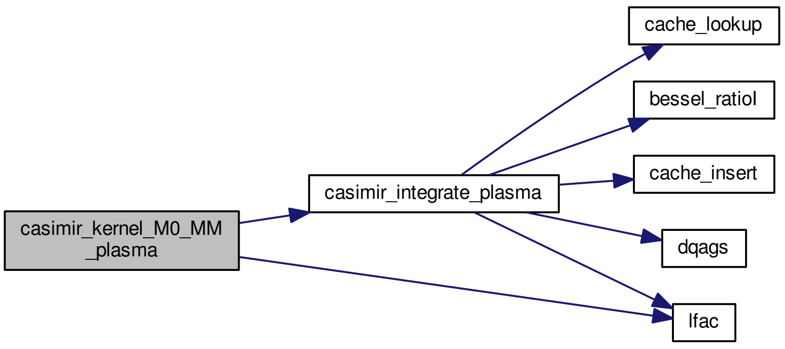
Parameters

in	*i*	row (starting from 0)
in	*j*	column (starting from 0)
in	*args←↩ _*	pointer to casimir_M_t object

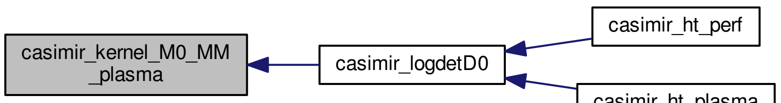
Return values

Mij	matrix element

Here is the call graph for this function:



Here is the caller graph for this function:



5.7.5.21 casimir_lnLambda()

```
double casimir_lnLambda (
              int l1,
              int l2,
              int m )
```

Calculate logarithm $\Lambda_{\ell_1 \ell_2}^{(m)}$.

This function returns the logarithm of $\Lambda_{\ell_1 \ell_2}^{(m)}$ for ℓ_1, ℓ_2, m.

$$\Lambda_{\ell_1,\ell_2}^{(m)} = \frac{2N_{\ell_1,m}N_{\ell_2,m}}{\sqrt{\ell_1(\ell_1+1)\ell_2(\ell_2+1)}}$$

Symmetries: $\Lambda_{\ell_1,\ell_2}^{(m)} = \Lambda_{\ell_2,\ell_1}^{(m)}$

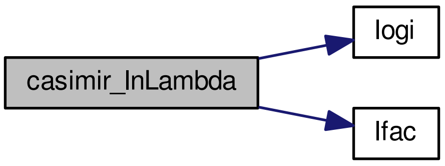
Parameters

`in`	*l1*	l1>0
`in`	*l2*	l2>0
`in`	*m*	m <= l1 and m <= l2

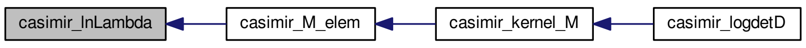
Return values

lnLambda	$\log \Lambda_{\ell_1,\ell_2}^{(m)}$

Here is the call graph for this function:



Here is the caller graph for this function:



5.7.5.22 casimir_logdetD()

```
double casimir_logdetD (
            casimir_t * self,
            double xi_,
            int m )
```

Compute $\log \det \mathcal{D}^{(m)} \left(\frac{\xi\mathcal{L}}{c}\right)$.

This function computes the logarithm of the determinant of the scattering matrix for the frequency $\xi\mathcal{L}/c$ and quantum number m.

For $\xi = 0$ see casimir_logdetD0.

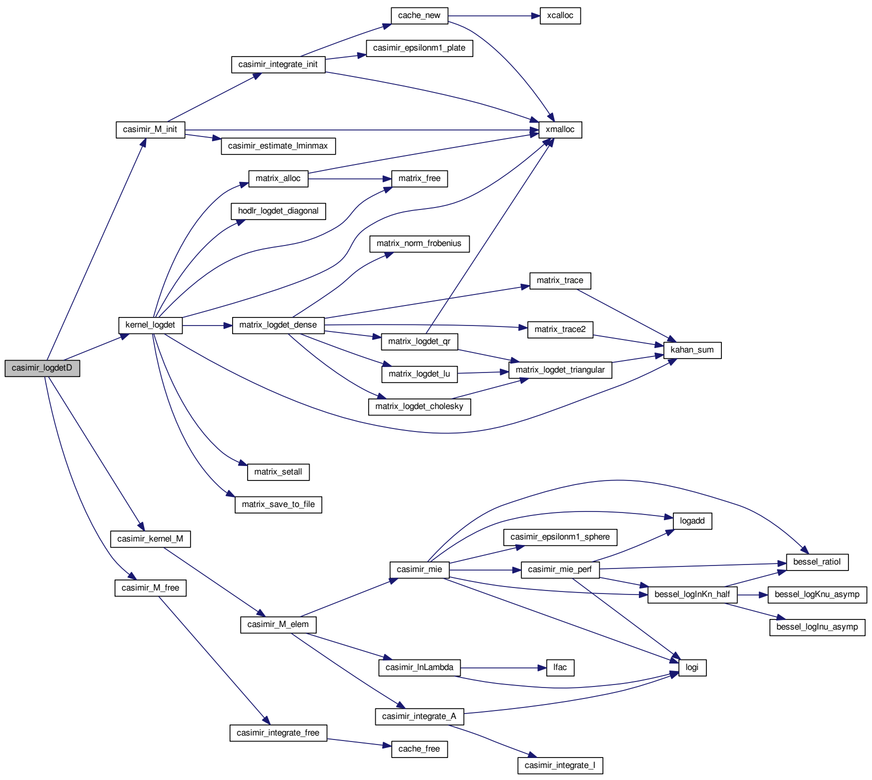
Parameters

self	Casimir object
*xi_*	$\xi\mathcal{L}/c > 0$
m	quantum number m

Return values

logdetD	

Here is the call graph for this function:



5.7.5.23 casimir_logdetD0()

```
void casimir_logdetD0 (
            casimir_t * self,
            int m,
            double omegap,
            double * EE,
            double * MM,
            double * MM_plasma )
```

Compute $\log\det\mathcal{D}^{(m)}(\xi=0)$ for EE and/or MM contribution.

Compute numerically for a given value of m the contribution of the polarization block EE and/or MM. If EE, MM or MM_plasma is NULL, the value will not be computed.

For Drude metals there exists an analytical formula to compute logdetD, see casimir_ht_drude.

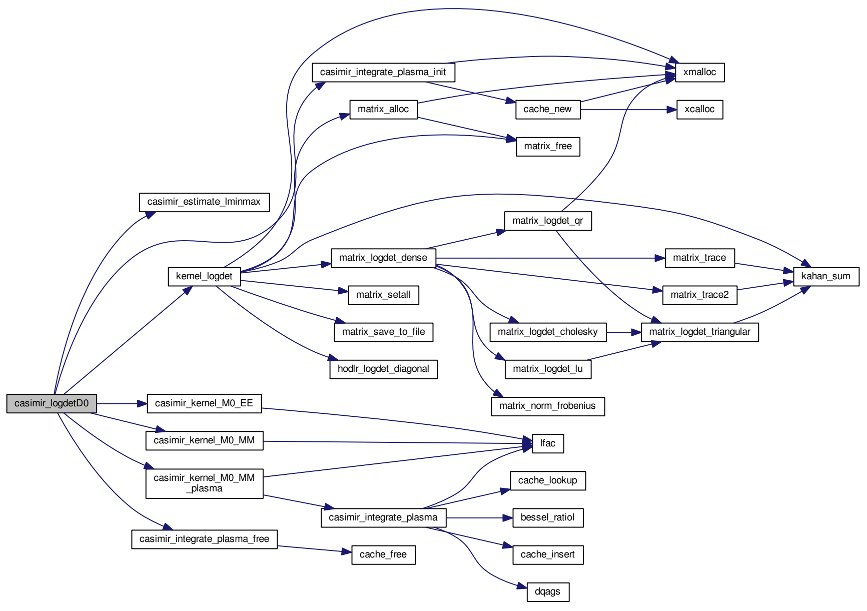
For perfect reflectors see also casimir_ht_perf.

For the Plasma model see also casimir_ht_plasma.

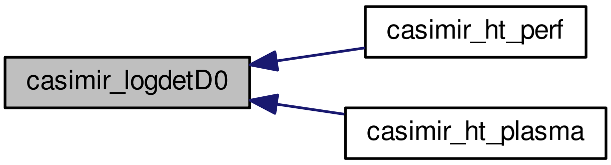
Parameters

in	*self*	Casimir object
in	*m*	quantum number m
in	*omegap*	plasma frequency in rad/s (only used to compute MM_plasma)
out	*EE*	pointer to store contribution for EE block
out	*MM*	pointer to store contribution for MM block
out	*MM_plasma*	pointer to store contribution for MM block (Plasma model)

Here is the call graph for this function:



Here is the caller graph for this function:



5.7.5.24 casimir_M_elem()

```
double casimir_M_elem (
            casimir_M_t * self,
            int l1,
            int l2,
            char p1,
            char p2 )
```

Compute matrix elements of round-trip operator.

This function computes matrix elements of the round-trip operator.

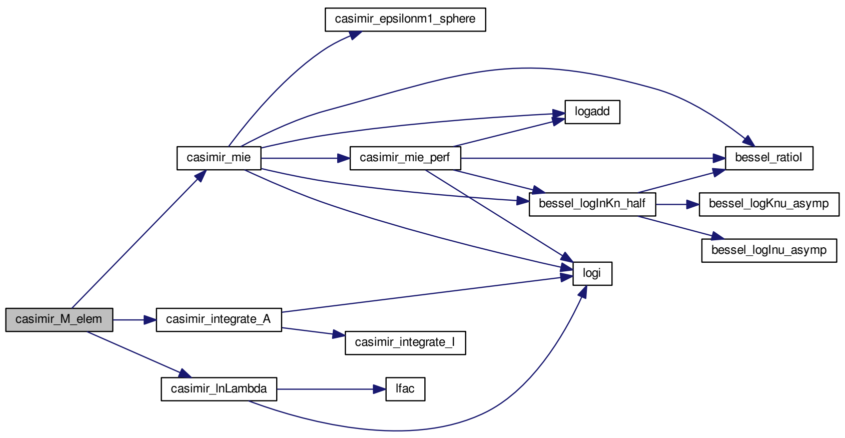
Parameters

in	*self*	casimir_M_t object, see casimir_M_init
in	*l1*	angular momentum ℓ_1
in	*l2*	angular momentum ℓ_2
in	*p1*	polarization p_1 (E or M)
in	*p2*	polarization p_2 (E or M)

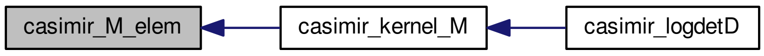
Return values

elem	$\mathcal{M}^{(m)}_{\ell_1,\ell_2}(p_1,p_2)$

Here is the call graph for this function:



Here is the caller graph for this function:



5.7.5.25 casimir_M_free()

```
void casimir_M_free (
            casimir_M_t * self )
```

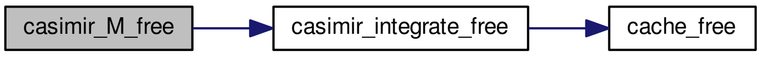
Free casimir_M_t object.

Frees memory allocated by casimir_M_init.

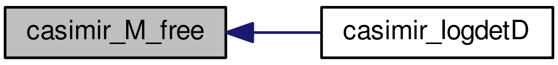
Parameters

in,out	*self*	casimir_M_t object

Here is the call graph for this function:



Here is the caller graph for this function:



5.7.5.26 casimir_M_init()

```
casimir_M_t* casimir_M_init (
            casimir_t * casimir,
            int m,
            double xi_ )
```

Initialize casimir_M_t object.

This object contains all information necessary to compute the matrix elements of the round-trip operator $\mathcal{M}^{(m)}(\xi)$. It also contains a cache for the Mie coefficients.

The returned object can be given to casimir_kernel_M to compute the matrix elements of the round-trip operator.

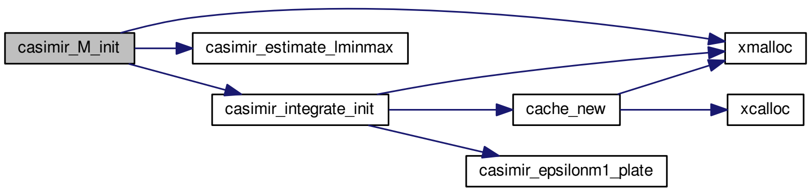
Parameters

in	*casimir*	Casimir object
in	*m*	azimuthal quantum number m
in	*xi_*	$\xi\mathcal{L}/c$

Return values

obj	casimir_M_t object that can be given to casimir_kernel_M

Here is the call graph for this function:



Here is the caller graph for this function:

casimir_M_init
casimir_logdetD

5.7.5.27 casimir_mie()

```
void casimir_mie (
            casimir_t * self,
            double xi_,
            int l,
            double * lna,
            double * lnb )
```

Return logarithm of Mie coefficients a_ℓ, b_ℓ for arbitrary metals.

For $\omega_\mathrm{P} = \infty$ the Mie coefficient for perfect reflectors are returned (see casimir_mie_perf).

lna and lnb must be valid pointers.

For generic metals, we calculate the Mie coefficients a_ℓ und b_ℓ using the expressions taken from [1]. Ref. [1] is the erratum to [2]. Please note that the equations (3.30) and (3.31) in [3] are wrong. The formulas are corrected in [4].

Note: If sla ≈ slb or slc ≈ sld, there is a loss of significance when calculating sla-slb or slc-sld.

The signs are given by $\mathrm{sgn}(a_\ell) = (-1)^\ell$, $\mathrm{sgn}(b_\ell) = (-1)^{\ell+1}$.

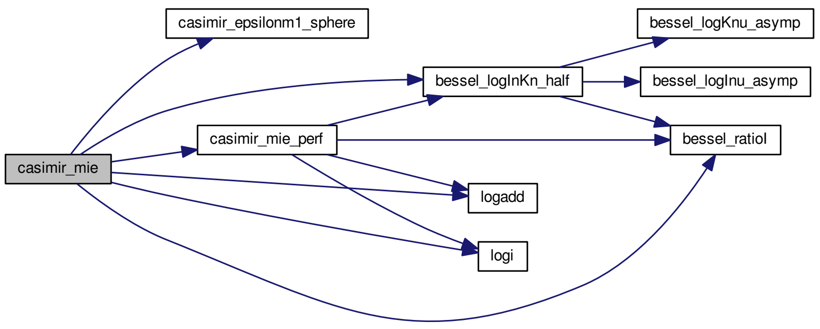
References:

- [1] Erratum: Thermal Casimir effect for Drude metals in the plane-sphere geometry, Canaguier-Durand, Neto, Lambrecht, Reynaud (2010) `http://journals.aps.org/pra/abstract/10.1103/PhysRevA.83.039905`
- [2] Thermal Casimir effect for Drude metals in the plane-sphere geometry, Canaguier-Durand, Neto, Lambrecht, Reynaud (2010), `http://journals.aps.org/pra/abstract/10.1103/PhysRevA.82.012511`
- [3] Negative Casimir entropies in the plane-sphere geometry, Hartmann, 2014
- [4] Casimir effect in the plane-sphere geometry: Beyond the proximity force approximation, Hartmann, 2018

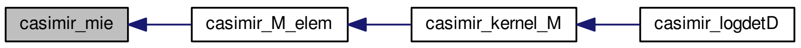
Parameters

`in,out`	*self*	Casimir object
`in`	*xi←_*	$\xi\mathcal{L}/c$
`in`	*l*	angular momentum ℓ
`out`	*lna*	logarithm of Mie coefficient a_ℓ
`out`	*lnb*	logarithm of Mie coefficient b_ℓ

Here is the call graph for this function:



Here is the caller graph for this function:



5.7.5.28 casimir_mie_perf()

```
void casimir_mie_perf (
             casimir_t * self,
             double xi_,
             int l,
             double * lna,
             double * lnb )
```

Calculate Mie coefficients a_ℓ, b_ℓ for perfect reflectors.

This function calculates the logarithms of the Mie coefficients $a_\ell(i\chi)$ and $b_\ell(i\chi)$ for perfect reflectors. The Mie coefficients are evaluated at the argument $\chi = \xi R/c$.

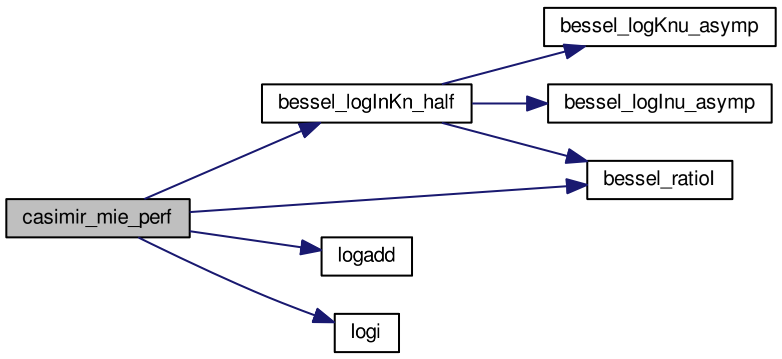
The signs are given by $\mathrm{sgn}(a_\ell) = (-1)^\ell$, $\mathrm{sgn}(b_\ell) = (-1)^{\ell+1}$.

lna and lnb must be valid pointers and must not be NULL.

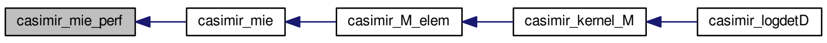
Parameters

`in,out`	*self*	Casimir object
`in`	*xi←↩ _*	$\xi\mathcal{L}/c > 0$
`in`	*l*	angular momentum $\ell > 0$
`out`	*lna*	logarithm of $\|a_\ell\|$
`out`	*lnb*	logarithm of $\|b_\ell\|$

Here is the call graph for this function:



Here is the caller graph for this function:



5.7.5.29 casimir_set_detalg()

```
int casimir_set_detalg (
            casimir_t * self,
            detalg_t detalg )
```

Set algorithm to calculate deterimant.

The algorithm is given by detalg. Usually you don't want to change the algorithm to compute the determinant.

detalg may be: DETALG_HODLR or DETALG_LU, DETALG_QR, DETALG_CHOLESKY.

If successul, the function returns 1. If the algorithm is not supported because of missing LAPACK support, 0 is returned.

Parameters

in,out	*self*	Casimir object
in	*detalg*	algorithm to compute determinant

Return values

success	1 if successful, 0 if not successful

5.7.5.30 casimir_set_epsilonm1()

```
void casimir_set_epsilonm1 (
            casimir_t * self,
            double(*)(double xi_, void *userdata) epsilonm1,
            void * userdata )
```

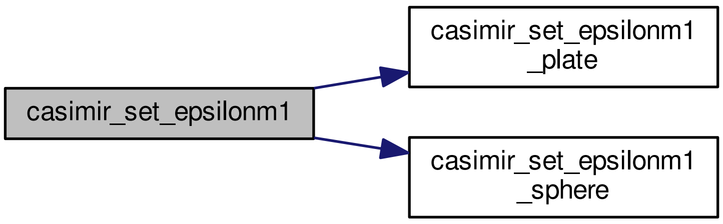
Set dielectric function for plate and sphere.

See also casimir_set_epsilonm1_plate and casimir_set_epsilonm1_sphere.

Parameters

in,out	*self*	Casimir object
in	*epsilonm1*	callback to the function that calculates $\epsilon(\mathrm{i}\xi) - 1$
in	*userdata*	arbitrary pointer to data that is passwd to epsilonm1 whenever the function is called

Here is the call graph for this function:



5.7.5.31 casimir_set_epsilonm1_plate()

```
void casimir_set_epsilonm1_plate (
            casimir_t * self,
            double(*)(double xi_, void *userdata) epsilonm1,
            void * userdata )
```

Set dielectric function of plate.

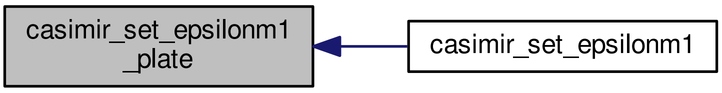
The Fresnel coefficient r_p depend on the dielectric function $\epsilon(\mathrm{i}\xi)$. By default, perfect reflectors with a dielectric function $\epsilon(\mathrm{i}\xi) = \infty$ are used.

However, you can also specify an arbitrary function for $\epsilon(\mathrm{i}\xi)$. userdata is an arbitrary pointer that will be given to the callback function.

Parameters

`in,out`	*self*	Casimir object
`in`	*epsilonm1*	callback to the function that calculates $\epsilon(\mathrm{i}\xi) - 1$
`in`	*userdata*	arbitrary pointer to data that is passwd to epsilonm1 whenever the function is called

Here is the caller graph for this function:



5.7.5.32 casimir_set_epsilonm1_sphere()

```
void casimir_set_epsilonm1_sphere (
            casimir_t * self,
            double(*)(double xi_, void *userdata) epsilonm1,
            void * userdata )
```

Set dielectric function of sphere.

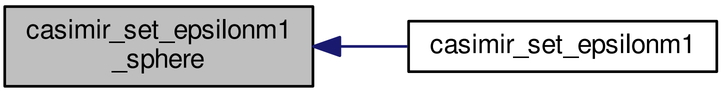
The Mie coefficient a_ℓ, b_ℓ depend on the dielectric function $\epsilon(\mathrm{i}\xi)$. By default, perfect reflectors with a dielectric function $\epsilon(\mathrm{i}\xi) = \infty$ are used.

However, you can also specify an arbitrary function for $\epsilon(\mathrm{i}\xi)$. userdata is an arbitrary pointer that will be given to the callback function.

Parameters

`in,out`	*self*	Casimir object
`in`	*epsilonm1*	callback to the function that calculates $\epsilon(\mathrm{i}\xi) - 1$
`in`	*userdata*	arbitrary pointer to data that is passwd to epsilonm1 whenever the function is called

Here is the caller graph for this function:



5.7.5.33 casimir_set_epsrel()

```
int casimir_set_epsrel (
            casimir_t * self,
            double epsrel )
```

Set relative error for numerical integration.

Set relative error for numerical integration.

Parameters

in	*self*	Casimir object
in	*epsrel*	relative error

Return values

0	if an error occured
1	on success

5.7.5.34 casimir_set_ldim()

```
int casimir_set_ldim (
            casimir_t * self,
            int ldim )
```

Set dimension of vector space.

The round trip matrices are infinite. For a numerical evaluation the dimension has to be truncated to a finite value. The accuracy of the result depends on the truncation of the vector space. ldim determines the dimension in the angular momentum ℓ that is used.

Parameters

`in,out`	*self*	Casimir object
`in`	*ldim*	dimension in angular momentum ℓ

Return values

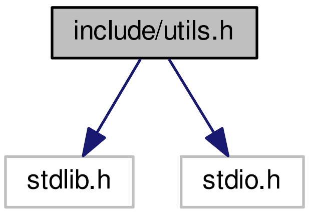
1	if successful
0	if ldim < 1

5.8 include/utils.h File Reference

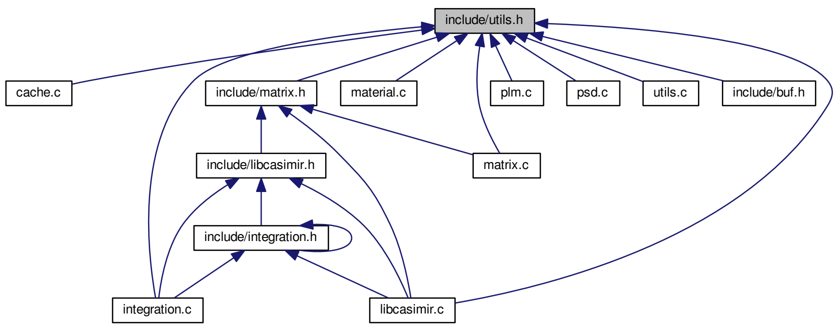
wrappers for malloc, calloc and realloc, assert-like macros, now()-function

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

Include dependency graph for utils.h:



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



Macros

- #define COMPILER "unknown"
- #define TERMINATE(cond, ...) if(cond) { fprintf(stderr, "Fatal error: "); fprintf(stderr, __VA_ARGS__↩
); fprintf(stderr, " (in %s, %s:%d)\n", __func__, __FILE__, __LINE__); abort(); }
- #define WARN(cond, ...) if(cond) { fprintf(stderr, "Warning: "); fprintf(stderr, __VA_ARGS__); fprintf(stderr, " (in %s, %s:%d)\n", __func__, __FILE__, __LINE__); }
- #define xfree(p) do { free(p); p = NULL; } while(0)

Functions

- double now (void)

  Seconds since 01/01/1970.
- void time_as_string (char ∗s, size_t len)

  Write time into string.
- void ∗ xmalloc (size_t size)

  Wrapper for malloc.
- void ∗ xrealloc (void ∗p, size_t size)

  Wrapper for realloc.
- void ∗ xcalloc (size_t nmemb, size_t size)

  Wrapper for calloc.
- void disable_buffering (void)

  Disable buffering to stderr and stdout.
- void strrep (char ∗s, const char a, const char b)

  Replace character by different character in string.
- void strim (char ∗str)

  Remove whitespace at beginng and end of string.

5.8.1 Detailed Description

wrappers for malloc, calloc and realloc, assert-like macros, now()-function

Author

Michael Hartmann `michael.hartmann@physik.uni-augsburg.de`

Date

July, 2017

5.8.2 Macro Definition Documentation

5.8.2.1 COMPILER

```
#define COMPILER "unknown"
```

name of compile

5.8.2.2 TERMINATE

```
#define TERMINATE(
              cond,
              ...  ) if(cond) { fprintf(stderr, "Fatal error:  "); fprintf(stderr, __VA_ARGS_↩
_); fprintf(stderr, " (in %s, %s:%d)\n", __func__, __FILE__, __LINE__); abort(); }
```

Macro similar to assert that prints a warning to stderr and aborts

5.8.2.3 WARN

```
#define WARN(
              cond,
              ...  ) if(cond) { fprintf(stderr, "Warning:  "); fprintf(stderr, __VA_ARGS__);
fprintf(stderr, " (in %s, %s:%d)\n", __func__, __FILE__, __LINE__); }
```

macro similar to assert that prints a warning to stderr

5.8.2.4 xfree

```
#define xfree(
              p ) do { free(p); p = NULL; } while(0)
```

macro for free that sets pointer p to NULL after freeing memory

5.8.3 Function Documentation

5.8.3.1 disable_buffering()

```
void disable_buffering (
             void  )
```

Disable buffering to stderr and stdout.

5.8.3.2 now()

```
double now (
             void  )
```

Seconds since 01/01/1970.

This function returns the seconds since 1st Jan 1970 in µs precision.

Return values

time	seconds since 1st Jan 1970

5.8.3.3 strim()

```
void strim (
            char * str )
```

Remove whitespace at beginng and end of string.

If str is NULL the function doesn't do anything. Otherwise, trailing whitespace and whitespace at the beginning of the string are removed.

Parameters

str	string

5.8.3.4 strrep()

```
void strrep (
            char * s,
            const char a,
            const char b )
```

Replace character by different character in string.

Replace occurence of a by b in the string s.

Parameters

`in,out`	*s*	string, terminated by \0
`in`	*a*	character to replace
`in`	*b*	substitute

5.8.3.5 time_as_string()

```
void time_as_string (
            char * s,
            size_t len )
```

Write time into string.

Write current time in a human readable format into string s. The output is similar to "Aug 30 2018 14:37:35".

Parameters

s	string
len	maximum length of array s

5.8.3.6 xcalloc()

```
void* xcalloc (
             size_t nmemb,
             size_t size )
```

Wrapper for calloc.

This function is a wrapper for calloc. If calloc fails TERMINATE is called.

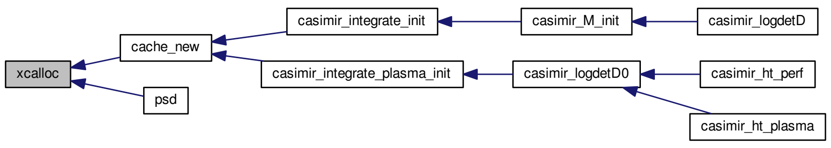
Parameters

nmemb	number of elements
size	size of each element

Return values

ptr	pointer to memory

Here is the caller graph for this function:



5.8.3.7 xmalloc()

```
void* xmalloc (
             size_t size )
```

Wrapper for malloc.

This function is a wrapper for malloc. If malloc fails TERMINATE is called.

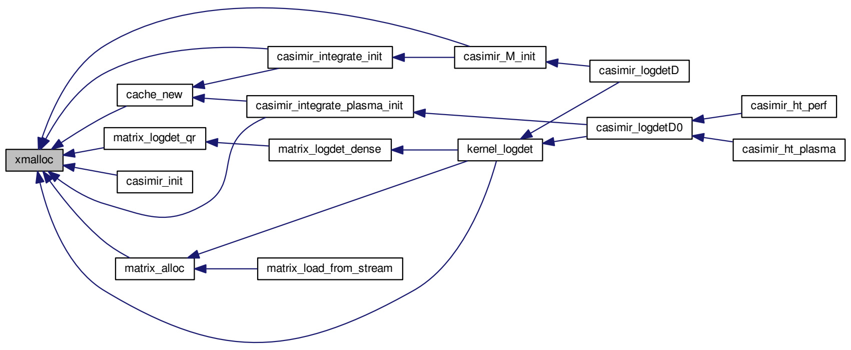
Parameters

size	size of bytes to allocate

Return values

ptr	pointer to memory

Here is the caller graph for this function:



5.8.3.8 xrealloc()

```
void* xrealloc (
             void * p,
             size_t size )
```

Wrapper for realloc.

This function is a wrapper for realloc. If realloc fails TERMINATE is called.

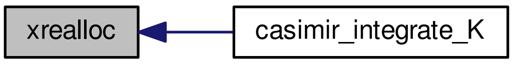
Parameters

p	ptr to old memory
size	size

Return values

newptr	pointer to new memory

Here is the caller graph for this function:

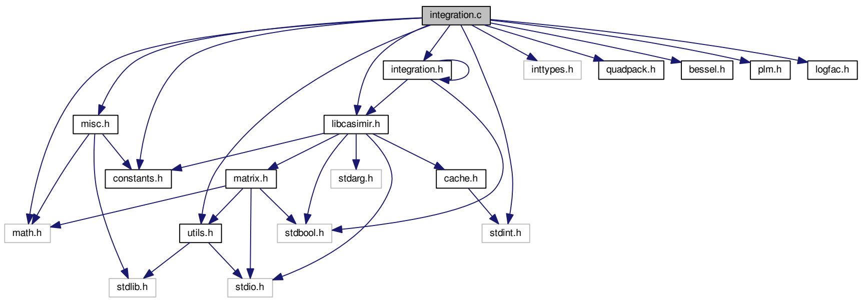


5.9 integration.c File Reference

Perform integration for arbitrary materials.

```
#include <math.h>
#include <stdint.h>
#include <inttypes.h>
#include "quadpack.h"
#include "constants.h"
#include "bessel.h"
#include "plm.h"
#include "utils.h"
#include "misc.h"
#include "libcasimir.h"
#include "logfac.h"
#include "integration.h"
```

Include dependency graph for integration.c:



Data Structures

- struct integrand_t
- struct integrand_plasma_t

Macros

- #define **f**(x) _f((x), nu, m, alpha)

Functions

- static uint64_t **hash** (uint64_t l1, uint64_t l2, uint64_t p)
- static double **_f** (double x, int nu, int m, double alpha)
- double K_estimate (int nu, int m, double alpha, double eps, double ∗a, double ∗b, double ∗approx)

    Estimate position and width of peak.
- static double **K_integrand** (double x, void ∗args_)
- static double **_casimir_integrate_K** (integration_t ∗self, int nu, polarization_t p, sign_t ∗sign)
- double casimir_integrate_K (integration_t ∗self, int nu, polarization_t p, sign_t ∗sign)

    Compute integral $\mathcal{K}_{\nu,p}^{(m)}(\alpha)$.
- static double **_alpha** (double p, double n, double nu)
- static double **_casimir_integrate_I** (integration_t ∗self, int l1, int l2, polarization_t p_, sign_t ∗sign)
- double casimir_integrate_I (integration_t ∗self, int l1, int l2, polarization_t p, sign_t ∗sign)

    Compute integral $\mathcal{I}_{\ell_1,\ell_2,p}^{(m)}(\alpha)$.
- integration_t ∗ casimir_integrate_init (casimir_t ∗casimir, double xi_, int m, double epsrel)

    Initialize integration.
- void casimir_integrate_free (integration_t ∗integration)

    Free integration object.
- double casimir_integrate_A (integration_t ∗self, int l1, int l2, polarization_t p, sign_t ∗sign)
- double casimir_integrate_B (integration_t ∗self, int l1, int l2, polarization_t p, sign_t ∗sign)
- double casimir_integrate_C (integration_t ∗self, int l1, int l2, polarization_t p, sign_t ∗sign)
- double casimir_integrate_D (integration_t ∗self, int l1, int l2, polarization_t p, sign_t ∗sign)

    Compute integral $D_{\ell_1,\ell_2,p}^{(m)}(\xi)$.
- integration_plasma_t ∗ casimir_integrate_plasma_init (casimir_t ∗casimir, double omegap, double epsrel)

    Initialize integration object for plasma high temperature limit ($\xi = 0$)
- static double **_integrand_plasma** (double t, void ∗args_)
- double casimir_integrate_plasma (integration_plasma_t ∗self, int l1, int l2, int m, double ∗ratio1, double ∗ratio2)

    Compute integral for plasma high temperatures.
- void casimir_integrate_plasma_free (integration_plasma_t ∗self)

    Free plasma integration object.

5.9.1 Detailed Description

Perform integration for arbitrary materials.

Author

Michael Hartmann michael.hartmann@physik.uni-augsburg.de

Date

December, 2018

5.9.2 Function Documentation

5.9.2.1 casimir_integrate_A()

```
double casimir_integrate_A (
            integration_t * self,
            int l1,
            int l2,
            polarization_t p,
            sign_t * sign )
```

Compute integral $A^{(m)}_{\ell_1,\ell_2,p}(\xi)$

Compute the integral

$$A^{(m)}_{\ell_1,\ell_2,p}(\xi) = \frac{m^2\xi}{c}\int_0^\infty \mathrm{d}k \frac{r_p}{k\kappa} e^{-2\kappa\mathcal{L}} P_{\ell_1}^m\left(\frac{\kappa c}{\xi}\right) P_{\ell_2}^m\left(\frac{\kappa c}{\xi}\right)$$

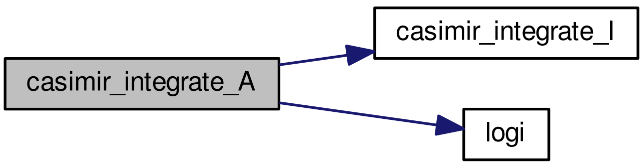
Parameters

in	*self*	integration object
in	*l1*	parameter
in	*l2*	parameter
in	*p*	polarization; either TE or TM
out	*sign*	sign of integral $\mathrm{sgn}\left(A^{(m)}_{\ell_1,\ell_2,p}(\xi)\right)$

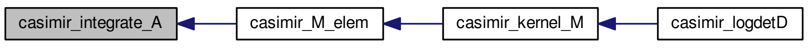
Return values

logA	$\log\left\vert A^{(m)}_{\ell_1,\ell_2,p}(\xi)\right\vert$

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.2.2 casimir_integrate_B()

```
double casimir_integrate_B (
            integration_t * self,
            int l1,
            int l2,
            polarization_t p,
            sign_t * sign )
```

Compute integral $B^{(m)}_{\ell_1,\ell_2,p}(\xi)$

Compute the integral

$$B^{(m)}_{\ell_1,\ell_2,p}(\xi) = \frac{c^3}{\xi^3}\int_0^\infty \mathrm{d}k \frac{k^3}{\kappa} r_p e^{-2\kappa\mathcal{L}} P^{m\prime}_{\ell_1}\left(\frac{\kappa c}{\xi}\right) P^{m\prime}_{\ell_2}\left(\frac{\kappa c}{\xi}\right)$$

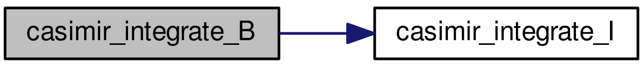
Parameters

in	*self*	integration object
in	*l1*	parameter
in	*l2*	parameter
in	*p*	polarization; either TE or TM
out	*sign*	sign of integral $\operatorname{sgn}\left(B^{(m)}_{\ell_1,\ell_2,p}(\xi)\right)$

Return values

logB	$\log\left\|B^{(m)}_{\ell_1,\ell_2,p}(\xi)\right\|$

Here is the call graph for this function:



$$\begin{tikzcd}\boxed{\text{casimir\_integrate\_B}} \arrow[r] & \boxed{\text{casimir\_integrate\_I}}\end{tikzcd}$$

5.9.2.3 casimir_integrate_C()

```
double casimir_integrate_C (
            integration_t * self,
            int l1,
            int l2,
            polarization_t p,
            sign_t * sign )
```

Compute integral $C^{(m)}_{\ell_1,\ell_2,p}(\xi)$

Compute the integral

$$C^{(m)}_{\ell_1,\ell_2,p}(\xi) = \frac{mc}{\xi}\int_0^\infty \mathrm{d}k \frac{k}{\kappa} r_p e^{-2\kappa\mathcal{L}} P^m_{\ell_1}\left(\frac{\kappa c}{\xi}\right) P^{m\prime}_{\ell_2}\left(\frac{\kappa c}{\xi}\right)$$

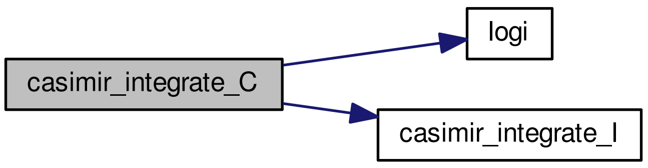
Parameters

`in`	*self*	integration object
`in`	*l1*	parameter
`in`	*l2*	parameter
`in`	*p*	polarization; either TE or TM
`out`	*sign*	sign of integral $\mathrm{sgn}\left(C^{(m)}_{\ell_1,\ell_2,p}(\xi)\right)$

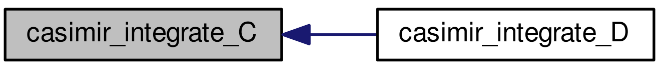
Return values

logC	$\log\left\|C^{(m)}_{\ell_1,\ell_2,p}(\xi)\right\|$

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.2.4 casimir_integrate_D()

```
double casimir_integrate_D (
            integration_t * self,
            int l1,
            int l2,
            polarization_t p,
            sign_t * sign )
```

Compute integral $D^{(m)}_{\ell_1,\ell_2,p}(\xi)$.

Compute

$$D^{(m)}_{\ell_1,\ell_2,p}(\xi) = C^{(m)}_{\ell_2,\ell_2,1}(\xi)$$

This function calls casimir_integrate_C.

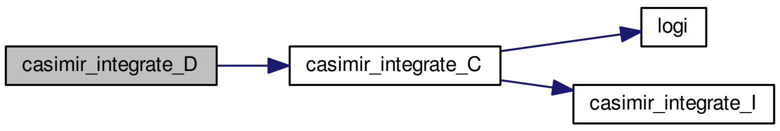
Parameters

in	*self*	integration object
in	*l1*	parameter
in	*l2*	parameter
in	*p*	polarization; either TE or TM
out	*sign*	sign of integral $\operatorname{sgn}\left(D^{(m)}_{\ell_1,\ell_2,p}(\xi)\right)$

Return values

logD	$\log\left\|D^{(m)}_{\ell_1,\ell_2,p}(\xi)\right\|$

Here is the call graph for this function:



5.9.2.5 casimir_integrate_free()

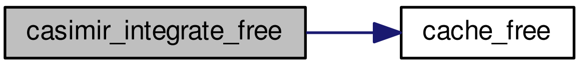
```
void casimir_integrate_free (
            integration_t * integration )
```

Free integration object.

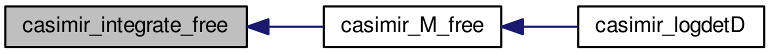
Parameters

`in,out`	*integration*	integration object

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.2.6 casimir_integrate_I()

```
double casimir_integrate_I (
            integration_t * self,
            int l1,
            int l2,
            polarization_t p,
            sign_t * sign )
```

Compute integral $\mathcal{I}^{(m)}_{\ell_1,\ell_2,p}(\alpha)$.

Compute the integral

$$\mathcal{I}^{(m)}_{\ell_1,\ell_2,p}(\alpha) = \int_0^\infty \mathrm{d}x\, r_p \frac{e^{-\alpha x}}{x^2-1} P^m_{\ell_1}(x) P^m_{\ell_2}(x)$$

This function returns the sign of the integral and its logarithmic value.

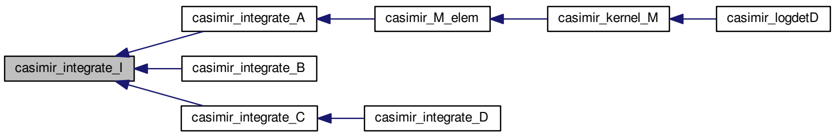
Parameters

`in`	*self*	integration object
`in`	*l1*	parameter
`in`	*l2*	parameter
`in`	*p*	polarization; either TE or TM
`out`	*sign*	sign of integral $\mathrm{sgn}\left(\mathcal{I}^{(m)}_{\ell_1,\ell_2,p}(\alpha)\right)$

Return values

logI	$\log\left\|\mathcal{I}^{(m)}_{\ell_1,\ell_2,p}(\alpha)\right\|$

Here is the caller graph for this function:



5.9.2.7 casimir_integrate_init()

```
integration_t* casimir_integrate_init (
             casimir_t * casimir,
             double xi_,
             int m,
             double epsrel )
```

Initialize integration.

The aspect ratio L/R and the dielectric function of the metals $\epsilon(i\xi)$ are taken from the casimir object. The integration is performed to a relative accuracy of epsrel.

This function returns an object in order to compute the actual integrals. The memory of this object has to be freed after use by a call to casimir_integrate_free.

The computation is sped up using caches. The number of elements of the cache for the K integrals are proportional to ldim, the elements for the I integrals are fixed. This value can be changed using the environmental variable CASIMIR_CACHE_ELEMS.

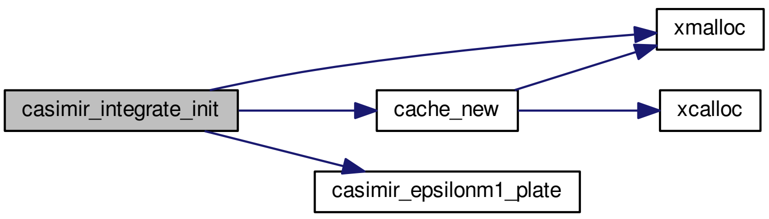
Parameters

in	*casimir*	Casimir object
in	*xi_*	$\xi\mathcal{L}/c$
in	*m*	magnetic quantum number
in	*epsrel*	relative accuracy of integration

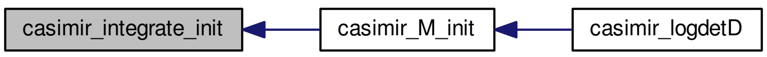
Return values

integration	object

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.2.8 casimir_integrate_K()

```
double casimir_integrate_K (
             integration_t * self,
             int nu,
             polarization_t p,
             sign_t * sign )
```

Compute integral $\mathcal{K}_{\nu,p}^{(m)}(\alpha)$.

This function solves for $m > 0$ the integral

$$\mathcal{K}_{\nu,p}^{(m)}(\alpha) = \int_1^\infty \mathrm{d}x\, r_p \frac{e^{-\alpha x}}{x^2-1} P_\nu^{2m}(x)$$

and for $m = 0$ the integral

$$\mathcal{K}_{\nu,p}^{(0)}(\alpha) = \int_1^\infty \mathrm{d}x\, r_p e^{-\alpha x} P_\nu^2(x)\,.$$

The function returns the logarithm of the value of the integral and its sign.

The projection of the wavevector onto the xy-plane is given by $k = \frac{\xi}{c}\sqrt{x^2-1}$ and $\alpha = 2\xi\mathcal{L}/c$.

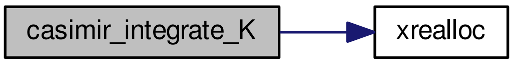
Parameters

in	*self*	integration object
in	*nu*	parameter
in	*p*	polarization, either TE or TM
out	*sign*	sign of $\mathcal{K}_{\nu,p}^{(m)}(\alpha)$

Return values

logK	$\log\left\lvert\mathcal{K}_{\nu,p}^{(m)}(\alpha)\right\rvert$

Here is the call graph for this function:



5.9.2.9 casimir_integrate_plasma()

```
double casimir_integrate_plasma (
              integration_plasma_t * self,
              int l1,
              int l2,
              int m,
              double * ratio1,
              double * ratio2 )
```

Compute integral for plasma high temperatures.

Compute the integral

$$\int_0^\infty \mathrm{d}x\, x^{\ell_1+\ell_2}\mathrm{e}^{-x} r_\mathrm{TE}$$

where

$$r_\mathrm{TE} = \frac{\sqrt{x^2+\beta^2}-x}{\sqrt{x^2+\beta^2}+x}$$

and $\beta = 2\omega_\mathrm{P}(L+R)/c$.

- If ratio1 is not NULL, ratio1 will be set to $I_{\ell_1-1/2}(\alpha)/I_{\ell_1+1/2}(\alpha)$ where $\alpha = 2\xi(L+R)/c$.
- If ratio2 is not NULL, ratio2 will be set to $I_{\ell_2-1/2}(\alpha)/I_{\ell_2+1/2}(\alpha)$ where $\alpha = 2\xi(L+R)/c$.

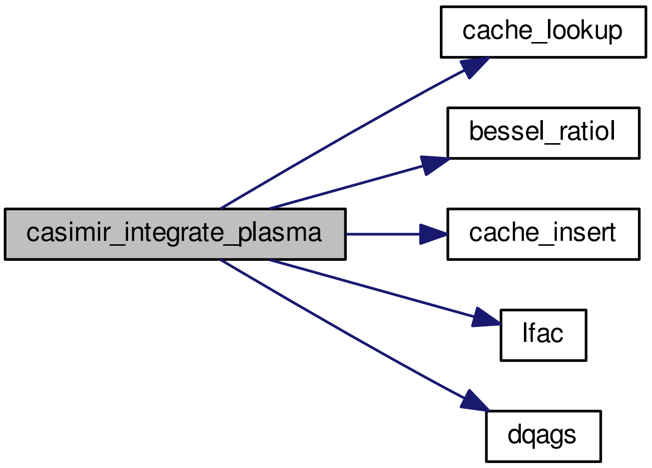
Parameters

in	*self*	plasma integration object
in	*l1*	ℓ_1
in	*l2*	ℓ_2
in	*m*	m
out	*ratio1*	
out	*ratio2*	

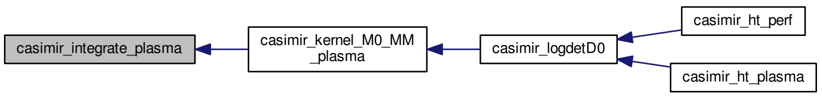
Return values

I	value of integral

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.2.10 casimir_integrate_plasma_free()

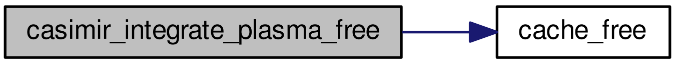
```
void casimir_integrate_plasma_free (
            integration_plasma_t * self )
```

Free plasma integration object.

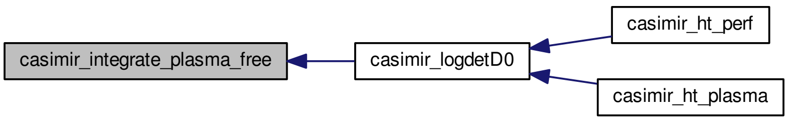
Parameters

in,out	*self*	plasma integration object

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.2.11 casimir_integrate_plasma_init()

```
integration_plasma_t* casimir_integrate_plasma_init (
             casimir_t * casimir,
             double omegap,
             double epsrel )
```

Initialize integration object for plasma high temperature limit ($\xi = 0$)

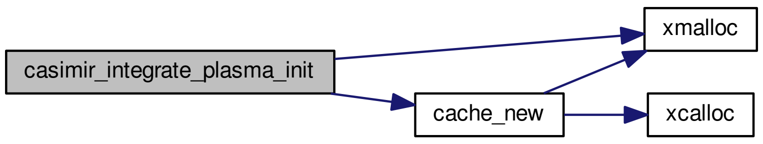
Parameters

in	*casimir*	Casimir object
in	*omegap*	plasma frequency in rad/s
in	*epsrel*	relative error for integration

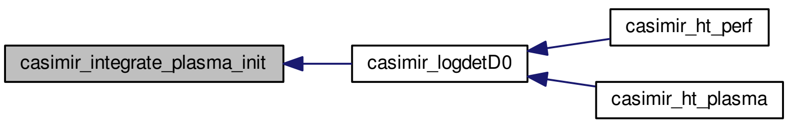
Return values

self	plasma integration object

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.2.12 K_estimate()

```
double K_estimate (
              int nu,
              int m,
              double alpha,
              double eps,
              double * a,
              double * b,
              double * approx )
```

Estimate position and width of peak.

We want to estimate the position and the width of the peak of the integrand for $m > 0$

$$\int_1^\infty \mathrm{d}x\, r_p \frac{e^{-\alpha x}}{x^2-1} P_\nu^{2m}(x) = \int_1^\infty \mathrm{d}x\, r_p g(x) = \int_1^\infty \mathrm{d}x\, r_p e^{-f(x)}$$

and for $m = 0$

$$\int_1^\infty \mathrm{d}x\, r_p e^{-\alpha x} P_\nu^2(x) = \int_1^\infty \mathrm{d}x\, r_p g(x) = \int_1^\infty \mathrm{d}x\, r_p e^{-f(x)}$$

with ($m > 0$)

$$f(x) = \alpha x - \log P_\nu^{2m}(x) + \log(x^2-1),$$

and ($m = 0$)

$$f(x) = \alpha x - \log P_\nu^2(x)\,.$$

We will assume that the Fresnel coefficient r_p varies slowly with respect to the width of the peak and set it to 1.

We find the maximum of $f(x)$ using Newton's method on $f'(x)$. With the maximum x_max and the second derivative at the maximum $f''(x_\mathrm{max})$, we estimate the width of the peak and the value of the integral using Laplace's method:

$$\int_1^\infty \mathrm{d}x\, e^{-f(x)} \approx \sqrt{\frac{2\pi}{-f''(x_\mathrm{max})}} e^{-f(x_\mathrm{max})}$$

The left border a and the right border b are determined by eps, such that

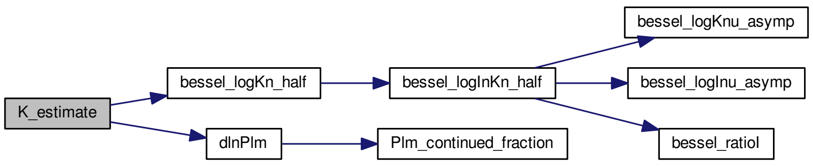
$$e^{-f(a)} \approx e^{-f(b)} \approx \epsilon e^{-f(x_\mathrm{max})}\,.$$

However, a cannot be smaller than 1.

Parameters

in	*nu*	parameter ν
in	*m*	parameter m
in	*alpha*	α
in	*eps*	ϵ
out	*a*	left border
out	*b*	right border
out	*approx*	logarithm of estimated value of integral

Here is the call graph for this function:



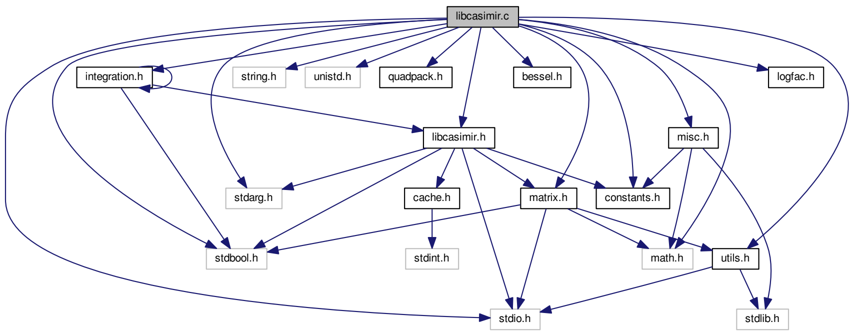
5.10 libcasimir.c File Reference

library to calculate the free Casimir energy in the plane-sphere geometry

```
#include <math.h>
#include <stdarg.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "quadpack.h"
#include "constants.h"
#include "bessel.h"
#include "integration.h"
#include "libcasimir.h"
#include "matrix.h"
#include "logfac.h"
#include "misc.h"
```

```
#include "utils.h"
```

Include dependency graph for libcasimir.c:



Functions

various functions

- double casimir_lnLambda (int l1, int l2, int m)
  Calculate logarithm $\Lambda_{\ell_1 \ell_2}^{(m)}$.
- int casimir_estimate_lminmax (casimir_t ∗self, int m, size_t ∗lmin_p, size_t ∗lmax_p)
  Estimate $\ell_{\min}$ and $\ell_{\max}$.

Dielectric functions

- double casimir_epsilonm1_plate (casimir_t ∗self, double xi_)
  Evaluate dielectric function of the plate.
- double casimir_epsilonm1_sphere (casimir_t ∗self, double xi_)
  Evaluate dielectric function of the sphere.
- double casimir_epsilonm1_perf (__attribute__((unused)) double xi_, __attribute__((unused)) void ∗userdata)
  Dielectric function for perfect reflectors.
- double casimir_epsilonm1_drude (double xi, void ∗userdata)
  Dielectric function for Drude reflectors.

initialization and setting parameters

- casimir_t ∗ casimir_init (double R, double L)
  Create a new Casimir object.
- void casimir_free (casimir_t ∗self)
  Free memory for Casimir object.
- void casimir_build (FILE ∗stream, const char ∗prefix)
  Print information on build to stream.
- void casimir_info (casimir_t ∗self, FILE ∗stream, const char ∗prefix)
  Print object information to stream.
- int casimir_set_epsrel (casimir_t ∗self, double epsrel)
  Set relative error for numerical integration.
- double casimir_get_epsrel (casimir_t ∗self)
  Get relative error for numerical integration.
- void casimir_set_epsilonm1 (casimir_t ∗self, double(∗epsilonm1)(double xi_, void ∗userdata), void ∗userdata)
  Set dielectric function for plate and sphere.

- void casimir_set_epsilonm1_plate (casimir_t ∗self, double(∗epsilonm1)(double xi_, void ∗userdata), void ∗userdata)

    Set dielectric function of plate.
- void casimir_set_epsilonm1_sphere (casimir_t ∗self, double(∗epsilonm1)(double xi_, void ∗userdata), void ∗userdata)

    Set dielectric function of sphere.
- int casimir_set_detalg (casimir_t ∗self, detalg_t detalg)

    Set algorithm to calculate deterimant.
- detalg_t casimir_get_detalg (casimir_t ∗self)

    Get algorithm to calculate determinant.
- int casimir_set_ldim (casimir_t ∗self, int ldim)

    Set dimension of vector space.
- int casimir_get_ldim (casimir_t ∗self)

    Get dimension of vector space.

Mie and Fresnell coefficients

- void casimir_mie_perf (casimir_t ∗self, double xi_, int l, double ∗lna, double ∗lnb)

    Calculate Mie coefficients a_ℓ, b_ℓ for perfect reflectors.
- void casimir_mie (casimir_t ∗self, double xi_, int l, double ∗lna, double ∗lnb)

    Return logarithm of Mie coefficients a_ℓ, b_ℓ for arbitrary metals.
- void casimir_fresnel (casimir_t ∗self, double xi_, double k_, double ∗r_TE, double ∗r_TM)

    Calculate Fresnel coefficients r_{TE} and r_{TM} for arbitrary metals.

Kernels

- casimir_M_t ∗ casimir_M_init (casimir_t ∗casimir, int m, double xi_)

    Initialize casimir_M_t object.
- double casimir_kernel_M (int i, int j, void ∗args_)

    Kernel of round-trip matrix.
- double casimir_M_elem (casimir_M_t ∗self, int l1, int l2, char p1, char p2)

    Compute matrix elements of round-trip operator.
- void casimir_M_free (casimir_M_t ∗self)

    Free casimir_M_t object.
- double casimir_kernel_M0_EE (int i, int j, void ∗args_)

    Kernel for EE block.
- double casimir_kernel_M0_MM_plasma (int i, int j, void ∗args_)

    Kernel for MM block (plasma model)
- double casimir_kernel_M0_MM (int i, int j, void ∗args_)

    Kernel for MM block.

Compute determinants

- double casimir_logdetD (casimir_t ∗self, double xi_, int m)

    Compute $\log\det\mathcal{D}^{(m)}\left(\frac{\xi\mathcal{L}}{c}\right)$.
- void casimir_logdetD0 (casimir_t ∗self, int m, double omegap, double ∗EE, double ∗MM, double ∗MM_←plasma)

    Compute $\log\det\mathcal{D}^{(m)}(\xi=0)$ for EE and/or MM contribution.

high-temperature limit

- double casimir_ht_drude (casimir_t ∗casimir)

    Compute high-temperature limit for Drude metals.
- double casimir_ht_perf (casimir_t ∗casimir, double eps)

    Compute free energy in the high-temperature limit for perfect reflectors.
- double casimir_ht_plasma (casimir_t ∗casimir, double omegap, double eps)

    Compute free energy in the high-temperature limit for plasma model.

5.10.1 Detailed Description

library to calculate the free Casimir energy in the plane-sphere geometry

Author

Michael Hartmann michael.hartmann@physik.uni-augsburg.de

Date

December, 2017

5.10.2 Function Documentation

5.10.2.1 casimir_build()

```
void casimir_build (
            FILE * stream,
            const char * prefix )
```

Print information on build to stream.

The information contains compiler, build time, git head and git branch if available. If prefix is not NULL, the string prefix will added in front of each line.

Parameters

stream	output stream
prefix	prefix of each line or NULL

5.10.2.2 casimir_epsilonm1_drude()

```
double casimir_epsilonm1_drude (
            double xi,
            void * userdata )
```

Dielectric function for Drude reflectors.

Dielectric function for Drude

$$\epsilon(\xi) - 1 = \frac{\omega_\mathrm{P}^2}{\xi(\xi+\gamma)}$$

The parameters ω_P and γ must be provided by userdata:

- userdata[0] = ω_P in rad/s
- userdata[1] = γ in rad/s

Parameters

in	*xi*	frequency in rad/s
in	*userdata*	userdata

Return values

epsilon	epsilon(xi)

5.10.2.3 casimir_epsilonm1_perf()

```
double casimir_epsilonm1_perf (
             __attribute__((unused)) double xi_,
             __attribute__((unused)) void * userdata )
```

Dielectric function for perfect reflectors.

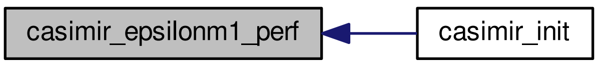
Parameters

in	*xi_*	ignored
in	*userdata*	ignored

Return values

inf	$\epsilon(\xi) = \infty$

Here is the caller graph for this function:



5.10.2.4 casimir_epsilonm1_plate()

```
double casimir_epsilonm1_plate (
             casimir_t * self,
             double xi_ )
```

Evaluate dielectric function of the plate.

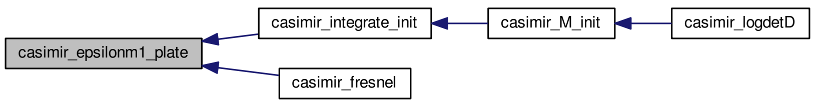
Parameters

in	*self*	Casimir object
in	*xi←_*	$\xi\mathcal{L}/c$

Return values

epsm1	$\epsilon(\mathrm{i}\xi)$

Here is the caller graph for this function:



5.10.2.5 casimir_epsilonm1_sphere()

```
double casimir_epsilonm1_sphere (
            casimir_t * self,
            double xi_ )
```

Evaluate dielectric function of the sphere.

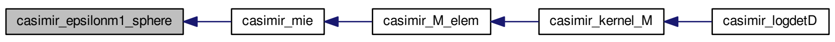
Parameters

in	*self*	Casimir object
in	*xi←_*	$\xi\mathcal{L}/c$

Return values

epsm1	$\epsilon(\mathrm{i}\xi)$

Here is the caller graph for this function:



5.10.2.6 casimir_estimate_lminmax()

```
int casimir_estimate_lminmax (
             casimir_t * self,
             int m,
             size_t * lmin_p,
             size_t * lmax_p )
```

Estimate $\ell_{\rm min}$ and $\ell_{\rm max}$.

Estimate the vector space: The main contributions comes from the vicinity $\ell_1 = \ell_2 = X$ and only depend on geometry, L/R, and the quantum number m. This function calculates X using the formula in the high-temperature limit and calculates $\ell_{\rm min}, \ell_{\rm max}$.

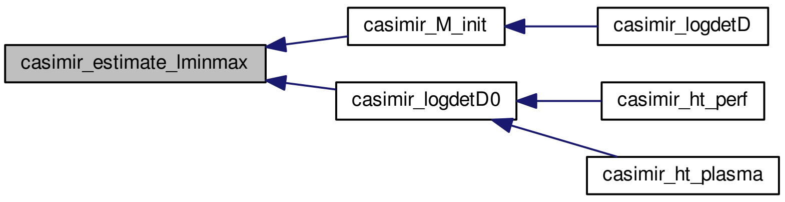
Parameters

in	*self*	Casimir object
in	*m*	quantum number
out	*lmin←↩ _p*	minimum value of ℓ
out	*lmax←↩ _p*	maximum value of ℓ

Return values

l	approximately the value of ℓ where $\mathcal{M}^m_{\ell\ell}$ is maximal

Here is the caller graph for this function:



5.10.2.7 casimir_free()

```
void casimir_free (
             casimir_t * self )
```

Free memory for Casimir object.

Free allocated memory for the Casimir object self.

Parameters

in,out	*self*	Casimir object

5.10.2.8 casimir_fresnel()

```
void casimir_fresnel (
            casimir_t * self,
            double xi_,
            double k_,
            double * r_TE,
            double * r_TM )
```

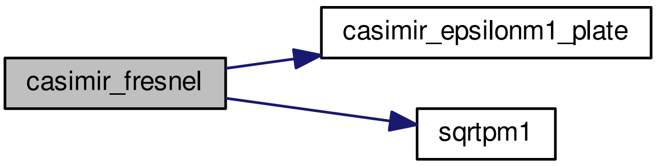
Calculate Fresnel coefficients r_{TE} and r_{TM} for arbitrary metals.

This function calculates the Fresnel coefficients $r_p = r_p(i\xi, k)$ for $p = \mathrm{TE}, \mathrm{TM}$.

Parameters

in	*self*	Casimir object
in	*xi_*	$\xi\mathcal{L}/c$
in	*k_*	$k\mathcal{L}$
in,out	*r_TE*	Fresnel coefficient for TE mode
in,out	*r_TM*	Fresnel coefficient for TM mode

Here is the call graph for this function:



5.10.2.9 casimir_get_detalg()

```
detalg_t casimir_get_detalg (
            casimir_t * self )
```

Get algorithm to calculate determinant.

Parameters

`in`	*self*	Casimir object

Return values

detalg	

5.10.2.10 casimir_get_epsrel()

```
double casimir_get_epsrel (
            casimir_t * self )
```

Get relative error for numerical integration.

See casimir_set_epsrel.

Return values

epsrel	relative error

5.10.2.11 casimir_get_ldim()

```
int casimir_get_ldim (
            casimir_t * self )
```

Get dimension of vector space.

See casimir_set_ldim.

Parameters

`in,out`	*self*	Casimir object

Return values

ldim	dimension of vector space

5.10.2.12 casimir_ht_drude()

```
double casimir_ht_drude (
            casimir_t * casimir )
```

Compute high-temperature limit for Drude metals.

For Drude metals the Fresnel coefficients become $r_{\rm TM}=1$, $r_{\rm TE}=0$ for $\xi\to0$, i.e. only the EE polarization block needs to be considered.

For Drude the free energy for $\xi=0$ can be computed analytically. We use Eq. (8) from Ref. [1] to compute the contribution.

References:

- [1] Bimonte, Emig, "Exact results for classical Casimir interactions: Dirichlet and Drude model in the sphere-sphere and sphere-plane geometry", Phys. Rev. Lett. 109 (2012), https://doi.org/10.1103/↩ PhysRevLett.109.160403

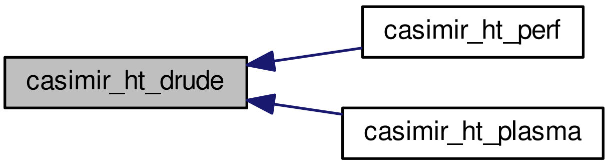
Parameters

in	*casimir*	Casimir object

Return values

F	free energy in units of $k_\mathrm{B}T$

Here is the caller graph for this function:



5.10.2.13 casimir_ht_perf()

```
double casimir_ht_perf (
            casimir_t * casimir,
            double eps )
```

Compute free energy in the high-temperature limit for perfect reflectors.

For perfect reflectors the Fresnel coefficients become $r_{\rm TM}=1$, $r_{\rm TE}=-1$ in the limit $\xi\to0$, and only the polarization blocks EE and MM need to be considered.

The contribution for EE, i.e. Drude, can be computed analytically, see casimir_ht_drude. For the MM block we numerically compute the determinants up to $m = M$ until

$$\frac{\log\det\mathcal{D}^{(M)}(0)}{\sum_{m=0}^{M}{}' \log\det\mathcal{D}^{(m)}(0)} < \epsilon\,.$$

We use Ref. [1] to compute the contribution for $m = 0$.

References:

- [1] Bimonte, Classical Casimir interaction of perfectly conducting sphere and plate (2017), `https://arxiv.org/abs/1701.06461`

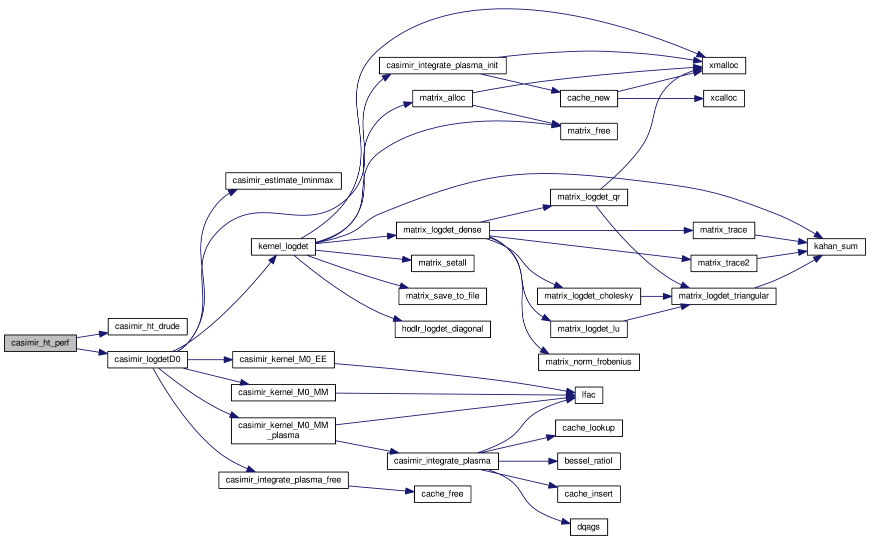
Parameters

in	*casimir*	Casimir object
in	*eps*	ϵ abort criterion

Return values

energy	free energy in units of $k_\mathrm{B}T$

Here is the call graph for this function:



5.10.2.14 casimir_ht_plasma()

```
double casimir_ht_plasma (
            casimir_t * casimir,
            double omegap,
            double eps )
```

Compute free energy in the high-temperature limit for plasma model.

The abort criterion eps is the same as in casimir_ht_perf.

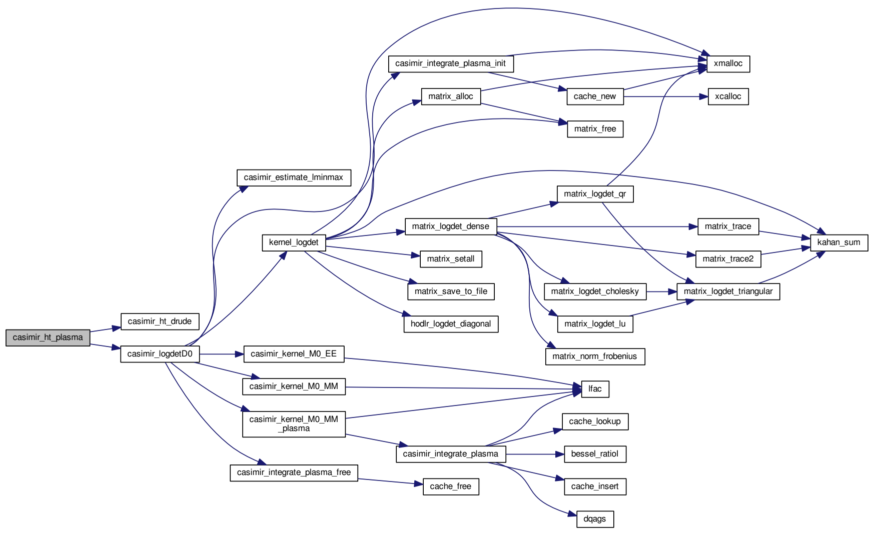
Parameters

in	*casimir*	Casimir object
in	*omegap*	plasma frequency in rad/s
in	*eps*	abort criterion

Return values

F	free energy in units of $k_\mathrm{B}T$

Here is the call graph for this function:



5.10.2.15 casimir_info()

```
void casimir_info (
            casimir_t * self,
            FILE * stream,
            const char * prefix )
```

Print object information to stream.

Print information about the object self to stream.

Parameters

self	Casimir object
stream	where to print the string
prefix	if prefix != NULL: start every line with the string contained in prefix

5.10.2.16 casimir_init()

```
casimir_t* casimir_init (
            double R,
            double L )
```

Create a new Casimir object.

This function will initialize a Casimir object. By default the dielectric function corresponds to perfect reflectors, i.e. $\epsilon(\xi) = \infty$.

By default, the value of ℓ_dim is chosen by:

$$\ell_\mathrm{dim} = \mathrm{ceil}\left(\max\left(\mathrm{CASIMIR\_MINIMUM\_LDIM}, \mathrm{CASIMIR\_FACTOR\_LDIM} \cdot \frac{R}{L}\right)\right)$$

Restrictions: $L/R > 0$

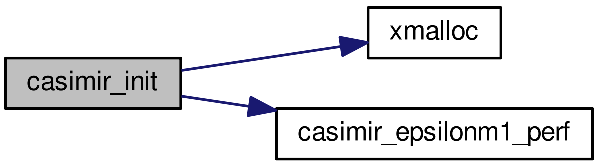
Parameters

in	*R*	radius of sphere in m
in	*L*	smallest separation between sphere and plate in m

Return values

object	Casimir object if successful
NULL	if an error occured

Here is the call graph for this function:



5.10.2.17 casimir_kernel_M()

```
double casimir_kernel_M (
            int i,
            int j,
            void * args_ )
```

Kernel of round-trip matrix.

This function returns the matrix elements of the round-trip operator $\mathcal{M}^{(m)}$.

The round-trip matrix is a $2\ell_\mathrm{dim} \times 2\ell_\mathrm{dim}$ matrix, the matrix elements start at 0, i.e. $0 \leq i, j < 2\ell_\mathrm{dim}$.

This function is intended to be passed as a callback to kernel_logdet. If you want to compute matrix elements of the round-trip operator, it is probably simpler to use casimir_M_elem.

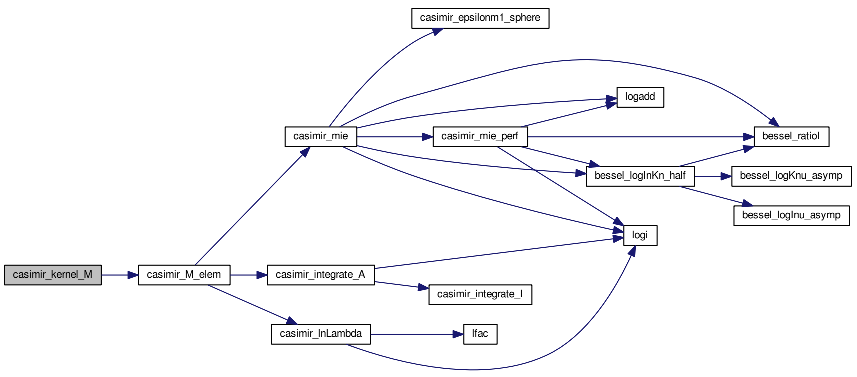
Parameters

in	*i*	row
in	*j*	column
in	*args_*	casimir_M_t object, see casimir_M_init

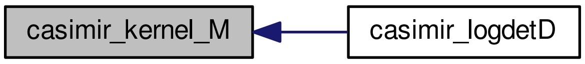
Return values

Mij	$\mathcal{M}^{(m)}_{ij}(\xi)$

Here is the call graph for this function:



Here is the caller graph for this function:



5.10.2.18 casimir_kernel_M0_EE()

```
double casimir_kernel_M0_EE (
            int i,
            int j,
            void * args_ )
```

Kernel for EE block.

Function that returns matrix elements of the round-trip matrix $\mathcal{M}$ for $\xi = 0$ and polarization $p_1 = p_2 = \mathrm{E}$.

See also casimir_logdetD0.

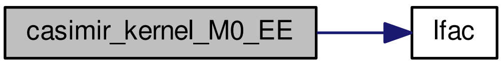
Parameters

in	*i*	row (starting from 0)
in	*j*	column (starting from 0)
in	*args← _*	pointer to casimir_M_t object

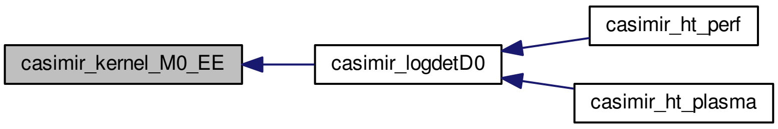
Return values

Mij	matrix element

Here is the call graph for this function:



Here is the caller graph for this function:



5.10.2.19 casimir_kernel_M0_MM()

```
double casimir_kernel_M0_MM (
            int i,
            int j,
            void * args_ )
```

Kernel for MM block.

Function that returns matrix elements of round-trip matrix $\mathcal{M}$ for $\xi = 0$ and polarization $p_1 = p_2 = \mathrm{M}$.

See also casimir_logdetD0.

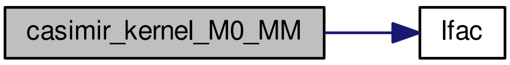
Parameters

in	*i*	row (starting from 0)
in	*j*	column (starting from 0)
in	*args*↩ _	pointer to casimir_M_t object

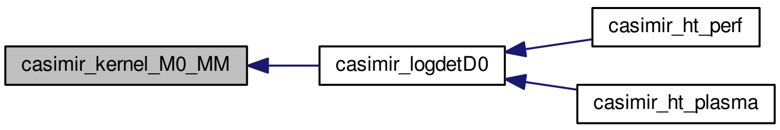
Return values

Mij	matrix element

Here is the call graph for this function:



Here is the caller graph for this function:



5.10.2.20 casimir_kernel_M0_MM_plasma()

```
double casimir_kernel_M0_MM_plasma (
              int i,
              int j,
              void * args_ )
```

Kernel for MM block (plasma model)

Function that returns matrix elements of round-trip matrix $\mathcal{M}$ for $\xi = 0$ and polarization $p_1 = p_2 = \mathrm{M}$ (plasma model).

See also casimir_logdetD0.

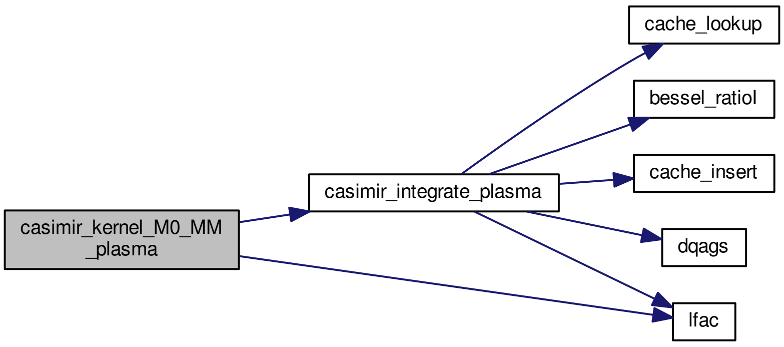
Parameters

in	*i*	row (starting from 0)
in	*j*	column (starting from 0)
in	*args←_*	pointer to casimir_M_t object

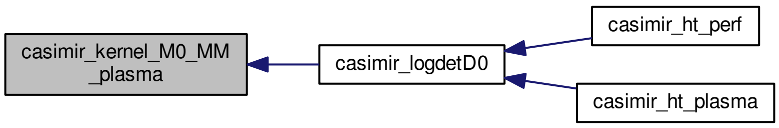
Return values

Mij	matrix element

Here is the call graph for this function:



Here is the caller graph for this function:



5.10.2.21 casimir_lnLambda()

```
double casimir_lnLambda (
            int l1,
            int l2,
            int m )
```

Calculate logarithm $\Lambda^{(m)}_{\ell_1 \ell_2}$.

This function returns the logarithm of $\Lambda^{(m)}_{\ell_1 \ell_2}$ for ℓ_1, ℓ_2, m.

$$\Lambda^{(m)}_{\ell_1,\ell_2} = \frac{2N_{\ell_1,m}N_{\ell_2,m}}{\sqrt{\ell_1(\ell_1+1)\ell_2(\ell_2+1)}}$$

Symmetries: $\Lambda^{(m)}_{\ell_1,\ell_2} = \Lambda^{(m)}_{\ell_2,\ell_1}$

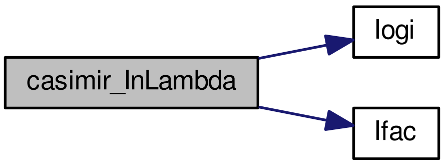
Parameters

in	*l1*	l1>0
in	*l2*	l2>0
in	*m*	m <= l1 and m <= l2

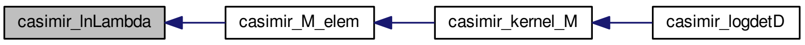
Return values

lnLambda	$\log \Lambda_{\ell_1,\ell_2}^{(m)}$

Here is the call graph for this function:



Here is the caller graph for this function:



5.10.2.22 casimir_logdetD()

```
double casimir_logdetD (
             casimir_t * self,
             double xi_,
             int m )
```

Compute $\log\det\mathcal{D}^{(m)}\left(\frac{\xi\mathcal{L}}{c}\right)$.

This function computes the logarithm of the determinant of the scattering matrix for the frequency $\xi\mathcal{L}/c$ and quantum number m.

For $\xi = 0$ see casimir_logdetD0.

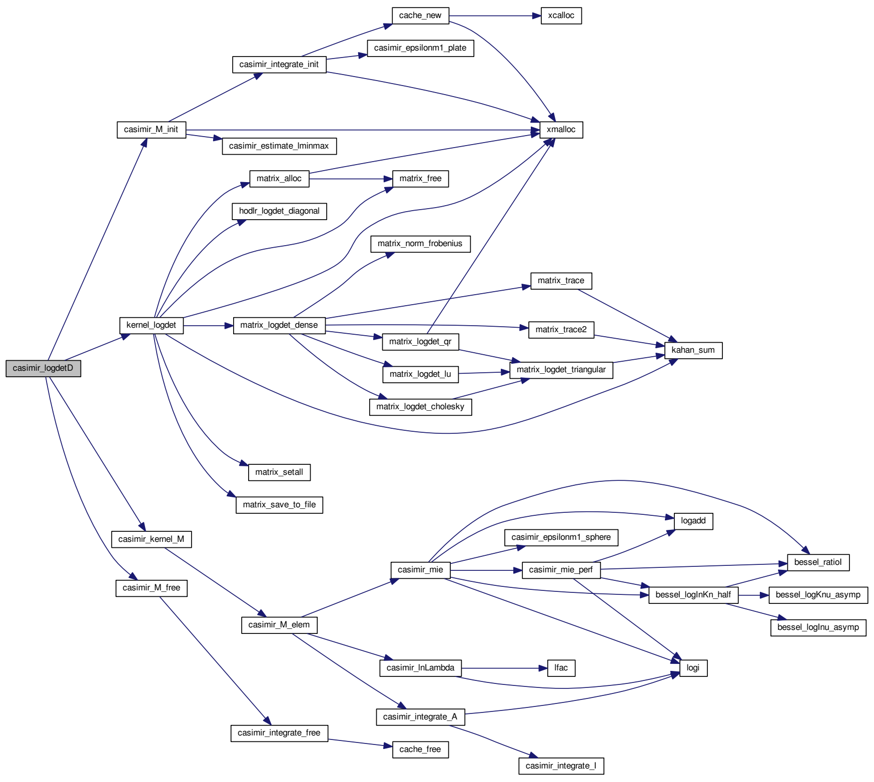
Parameters

self	Casimir object
xi↩_	$\xi\mathcal{L}/c > 0$
m	quantum number m

Return values

logdetD	

Here is the call graph for this function:



5.10.2.23 casimir_logdetD0()

```
void casimir_logdetD0 (
            casimir_t * self,
            int m,
            double omegap,
            double * EE,
            double * MM,
            double * MM_plasma )
```

Compute $\log\det\mathcal{D}^{(m)}(\xi=0)$ for EE and/or MM contribution.

Compute numerically for a given value of m the contribution of the polarization block EE and/or MM. If EE, MM or MM_plasma is NULL, the value will not be computed.

For Drude metals there exists an analytical formula to compute logdetD, see casimir_ht_drude.

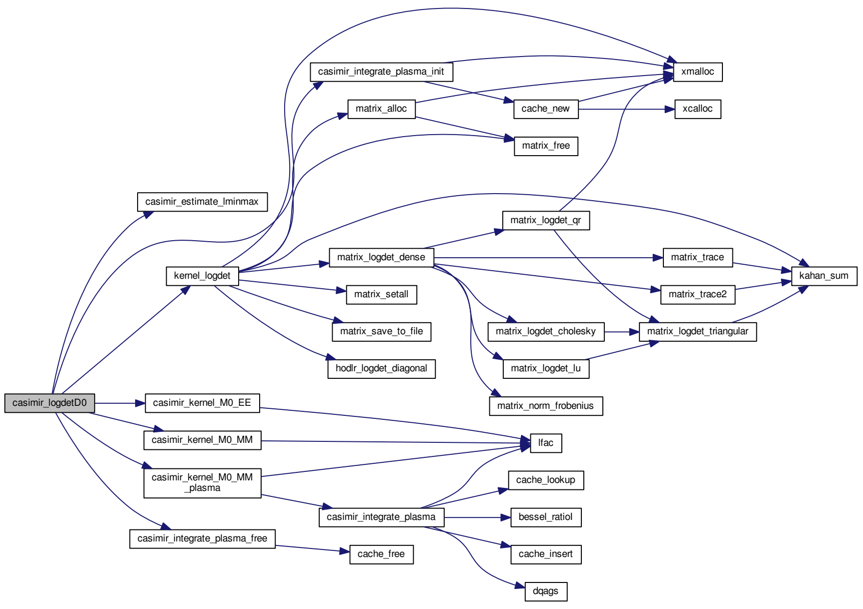
For perfect reflectors see also casimir_ht_perf.

For the Plasma model see also casimir_ht_plasma.

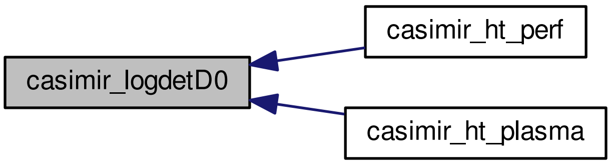
Parameters

in	*self*	Casimir object
in	*m*	quantum number m
in	*omegap*	plasma frequency in rad/s (only used to compute MM_plasma)
out	*EE*	pointer to store contribution for EE block
out	*MM*	pointer to store contribution for MM block
out	*MM_plasma*	pointer to store contribution for MM block (Plasma model)

Here is the call graph for this function:



Here is the caller graph for this function:



5.10.2.24 casimir_M_elem()

```
double casimir_M_elem (
             casimir_M_t * self,
             int l1,
             int l2,
             char p1,
             char p2 )
```

Compute matrix elements of round-trip operator.

This function computes matrix elements of the round-trip operator.

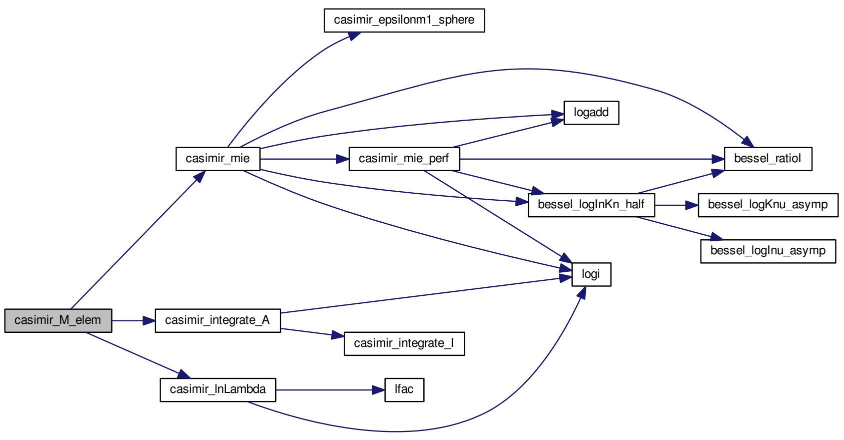
Parameters

in	*self*	casimir_M_t object, see casimir_M_init
in	*l1*	angular momentum ℓ_1
in	*l2*	angular momentum ℓ_2
in	*p1*	polarization p_1 (E or M)
in	*p2*	polarization p_2 (E or M)

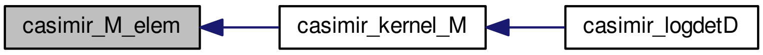
Return values

elem	$\mathcal{M}^{(m)}_{\ell_1,\ell_2}(p_1,p_2)$

Here is the call graph for this function:



Here is the caller graph for this function:



5.10.2.25 casimir_M_free()

```
void casimir_M_free (
            casimir_M_t * self )
```

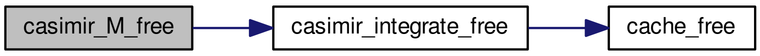
Free casimir_M_t object.

Frees memory allocated by casimir_M_init.

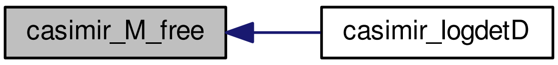
Parameters

in,out	*self*	casimir_M_t object

Here is the call graph for this function:



Here is the caller graph for this function:



5.10.2.26 casimir_M_init()

```
casimir_M_t* casimir_M_init (
             casimir_t * casimir,
             int m,
             double xi_ )
```

Initialize casimir_M_t object.

This object contains all information necessary to compute the matrix elements of the round-trip operator $\mathcal{M}^{(m)}(\xi)$. It also contains a cache for the Mie coefficients.

The returned object can be given to casimir_kernel_M to compute the matrix elements of the round-trip operator.

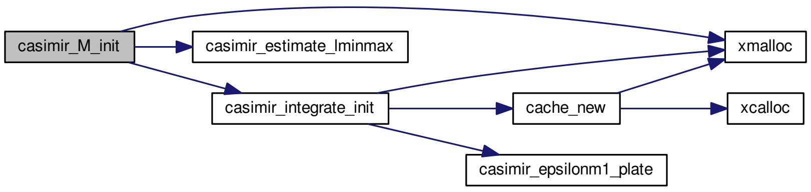
Parameters

in	*casimir*	Casimir object
in	*m*	azimuthal quantum number m
in	*xi_*	$\xi\mathcal{L}/c$

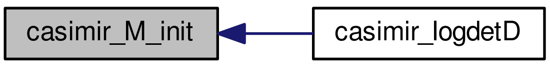
Return values

obj	casimir_M_t object that can be given to casimir_kernel_M

Here is the call graph for this function:



Here is the caller graph for this function:



5.10.2.27 casimir_mie()

```
void casimir_mie (
            casimir_t * self,
            double xi_,
            int l,
            double * lna,
            double * lnb )
```

Return logarithm of Mie coefficients a_ℓ, b_ℓ for arbitrary metals.

For $\omega_\mathrm{P} = \infty$ the Mie coefficient for perfect reflectors are returned (see casimir_mie_perf).

lna and lnb must be valid pointers.

For generic metals, we calculate the Mie coefficients a_ℓ und b_ℓ using the expressions taken from [1]. Ref. [1] is the erratum to [2]. Please note that the equations (3.30) and (3.31) in [3] are wrong. The formulas are corrected in [4].

Note: If sla $\approx$ slb or slc $\approx$ sld, there is a loss of significance when calculating sla-slb or slc-sld.

The signs are given by $\mathrm{sgn}(a_\ell) = (-1)^\ell$, $\mathrm{sgn}(b_\ell) = (-1)^{\ell+1}$.

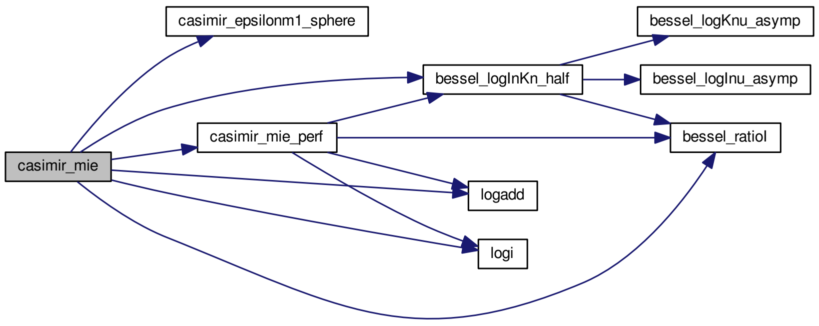
References:

- [1] Erratum: Thermal Casimir effect for Drude metals in the plane-sphere geometry, Canaguier-Durand, Neto, Lambrecht, Reynaud (2010) `http://journals.aps.org/pra/abstract/10.1103/PhysRevA.83.039905`
- [2] Thermal Casimir effect for Drude metals in the plane-sphere geometry, Canaguier-Durand, Neto, Lambrecht, Reynaud (2010), `http://journals.aps.org/pra/abstract/10.1103/PhysRevA.82.012511`
- [3] Negative Casimir entropies in the plane-sphere geometry, Hartmann, 2014
- [4] Casimir effect in the plane-sphere geometry: Beyond the proximity force approximation, Hartmann, 2018

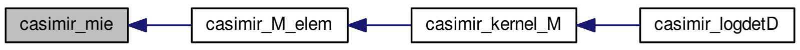
Parameters

`in,out`	*self*	Casimir object
`in`	*xi_*	$\xi\mathcal{L}/c$
`in`	*l*	angular momentum ℓ
`out`	*lna*	logarithm of Mie coefficient a_ℓ
`out`	*lnb*	logarithm of Mie coefficient b_ℓ

Here is the call graph for this function:



Here is the caller graph for this function:



5.10.2.28 casimir_mie_perf()

```
void casimir_mie_perf (
             casimir_t * self,
             double xi_,
             int l,
             double * lna,
             double * lnb )
```

Calculate Mie coefficients a_ℓ, b_ℓ for perfect reflectors.

This function calculates the logarithms of the Mie coefficients $a_\ell(i\chi)$ and $b_\ell(i\chi)$ for perfect reflectors. The Mie coefficients are evaluated at the argument $\chi = \xi R/c$.

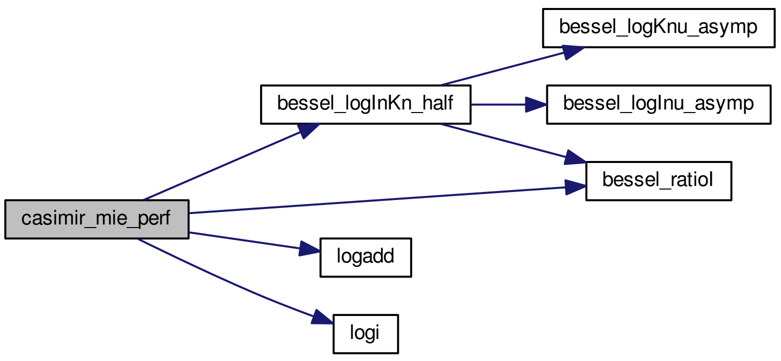
The signs are given by $\mathrm{sgn}(a_\ell) = (-1)^\ell$, $\mathrm{sgn}(b_\ell) = (-1)^{\ell+1}$.

lna and lnb must be valid pointers and must not be NULL.

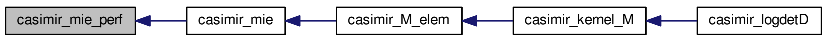
Parameters

in,out	*self*	Casimir object
in	*xi_*	$\xi\mathcal{L}/c > 0$
in	*l*	angular momentum $\ell > 0$
out	*lna*	logarithm of $\lvert a_\ell \rvert$
out	*lnb*	logarithm of $\lvert b_\ell \rvert$

Here is the call graph for this function:



Here is the caller graph for this function:



5.10.2.29 casimir_set_detalg()

```
int casimir_set_detalg (
            casimir_t * self,
            detalg_t detalg )
```

Set algorithm to calculate deterimant.

The algorithm is given by detalg. Usually you don't want to change the algorithm to compute the determinant.

detalg may be: DETALG_HODLR or DETALG_LU, DETALG_QR, DETALG_CHOLESKY.

If successul, the function returns 1. If the algorithm is not supported because of missing LAPACK support, 0 is returned.

Parameters

`in,out`	*self*	Casimir object
`in`	*detalg*	algorithm to compute determinant

Return values

success	1 if successful, 0 if not successful

5.10.2.30 casimir_set_epsilonm1()

```
void casimir_set_epsilonm1 (
            casimir_t * self,
            double(*)(double xi_, void *userdata) epsilonm1,
            void * userdata )
```

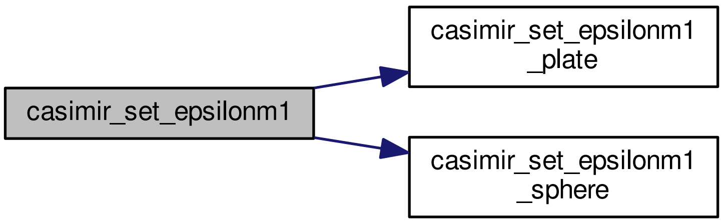
Set dielectric function for plate and sphere.

See also casimir_set_epsilonm1_plate and casimir_set_epsilonm1_sphere.

Parameters

in,out	*self*	Casimir object
in	*epsilonm1*	callback to the function that calculates $\epsilon(\mathrm{i}\xi)-1$
in	*userdata*	arbitrary pointer to data that is passwd to epsilonm1 whenever the function is called

Here is the call graph for this function:



5.10.2.31 casimir_set_epsilonm1_plate()

```
void casimir_set_epsilonm1_plate (
            casimir_t * self,
            double(*)(double xi_, void *userdata) epsilonm1,
            void * userdata )
```

Set dielectric function of plate.

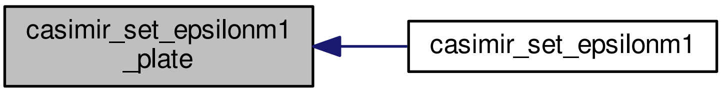
The Fresnel coefficient r_p depend on the dielectric function $\epsilon(\mathrm{i}\xi)$. By default, perfect reflectors with a dielectric function $\epsilon(\mathrm{i}\xi)=\infty$ are used.

However, you can also specify an arbitrary function for $\epsilon(\mathrm{i}\xi)$. userdata is an arbitrary pointer that will be given to the callback function.

Parameters

in,out	*self*	Casimir object
in	*epsilonm1*	callback to the function that calculates $\epsilon(\mathrm{i}\xi) - 1$
in	*userdata*	arbitrary pointer to data that is passwd to epsilonm1 whenever the function is called

Here is the caller graph for this function:



5.10.2.32 casimir_set_epsilonm1_sphere()

```
void casimir_set_epsilonm1_sphere (
            casimir_t * self,
            double(*)(double xi_, void *userdata) epsilonm1,
            void * userdata )
```

Set dielectric function of sphere.

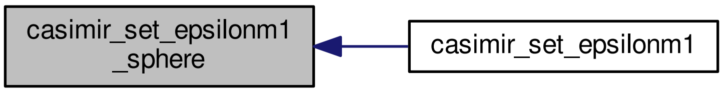
The Mie coefficient a_ℓ, b_ℓ depend on the dielectric function $\epsilon(\mathrm{i}\xi)$. By default, perfect reflectors with a dielectric function $\epsilon(\mathrm{i}\xi) = \infty$ are used.

However, you can also specify an arbitrary function for $\epsilon(\mathrm{i}\xi)$. userdata is an arbitrary pointer that will be given to the callback function.

Parameters

in,out	*self*	Casimir object
in	*epsilonm1*	callback to the function that calculates $\epsilon(\mathrm{i}\xi) - 1$
in	*userdata*	arbitrary pointer to data that is passwd to epsilonm1 whenever the function is called

Here is the caller graph for this function:



5.10.2.33 casimir_set_epsrel()

```
int casimir_set_epsrel (
            casimir_t * self,
            double epsrel )
```

Set relative error for numerical integration.

Set relative error for numerical integration.

Parameters

in	*self*	Casimir object
in	*epsrel*	relative error

Return values

0	if an error occured
1	on success

5.10.2.34 casimir_set_ldim()

```
int casimir_set_ldim (
            casimir_t * self,
            int ldim )
```

Set dimension of vector space.

The round trip matrices are infinite. For a numerical evaluation the dimension has to be truncated to a finite value. The accuracy of the result depends on the truncation of the vector space. ldim determines the dimension in the angular momentum ℓ that is used.

Parameters

`in,out`	*self*	Casimir object
`in`	*ldim*	dimension in angular momentum ℓ

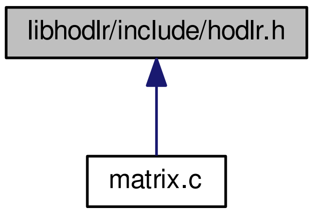
Return values

1	if successful
0	if ldim $<$ 1

5.11 libhodlr/include/hodlr.h File Reference

C wrapper for HODLR library.

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



Functions

- EXTERNC double hodlr_logdet_diagonal (int dim, double(∗callback)(int, int, void ∗), void ∗args, double ∗diagonal, unsigned int nLeaf, double tolerance, int is_symmetric)

  Calculate $\log\det(1-M)$ *using HODLR approach.*
- EXTERNC double hodlr_logdet (int dim, double(∗callback)(int, int, void ∗), void ∗args, unsigned int nLeaf, double tolerance, int is_symmetric)

  Calculate log(det(Id-M)) using HODLR approach.

5.11.1 Detailed Description

C wrapper for HODLR library.

Date

January, 2019

5.11.2 Function Documentation

5.11.2.1 hodlr_logdet()

```
EXTERNC double hodlr_logdet (
            int dim,
            double(*)(int, int, void *) callback,
            void * args,
            unsigned int nLeaf,
            double tolerance,
            int is_symmetric )
```

Calculate log(det(Id-M)) using HODLR approach.

Parameters

dim	dimension of matrix M
callback	function that returns matrix elements of M
args	pointer that is passed as third argument to callback
nLeaf	nLeaf is the dimension of the smallest block at the leaf level
tolerance	requested accuracy of result
is_symmetric	matrix is symmetric (1) or not symmetric (0)

Return values

logdet	$\log\det(1-M)$

5.11.2.2 hodlr_logdet_diagonal()

```
EXTERNC double hodlr_logdet_diagonal (
            int dim,
            double(*)(int, int, void *) callback,
            void * args,
            double * diagonal,
            unsigned int nLeaf,
            double tolerance,
            int is_symmetric )
```

Calculate $\log\det(1-M)$ using HODLR approach.

Parameters

dim	dimension of matrix M
callback	function that returns matrix elements of M
args	pointer that is passed as third argument to callback
diagonal	array with the diagonal elements of M

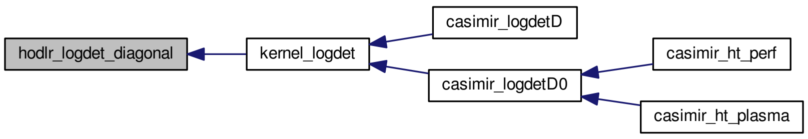
Parameters

nLeaf	nLeaf is the dimension of the smallest block at the leaf level
tolerance	requested accuracy of result
is_symmetric	matrix is symmetric (1) or not symmetric (0)

Return values

logdet	$\log\det(1-M)$

Here is the caller graph for this function:

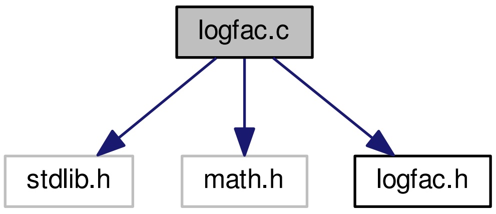


5.12 logfac.c File Reference

computation of logarithm and factorial for integer arguments; created by logfac.py

```
#include <stdlib.h>
#include <math.h>
#include "logfac.h"
```

Include dependency graph for logfac.c:



Functions

- double logi (unsigned int n)

  Calculate $\log(n)$ *for integer* n.
- double lfac (unsigned int n)

  Calculate $\log(n!) = \log(\Gamma(n+1))$.
- double lfac2 (unsigned int n)

  Calculate $\log(n!!)$.

Variables

- static double lookup_logi [ ]
- static double lookup_lfac [ ]
- const size_t **__lookup_logi_elems** = sizeof(lookup_logi)/sizeof(lookup_logi[0])
- const size_t **__lookup_lfac_elems** = sizeof(lookup_lfac)/sizeof(lookup_lfac[0])

5.12.1 Detailed Description

computation of logarithm and factorial for integer arguments; created by logfac.py

Author

Michael Hartmann `michael.hartmann@physik.uni-augsburg.de`

Date

January, 2019

5.12.2 Function Documentation

5.12.2.1 lfac()

```
double lfac (
            unsigned int n )
```

Calculate $\log(n!) = \log(\Gamma(n+1))$.

This function computes the logarithm of the factorial $n!$. This function uses a lookup table for $n \leq 1024$

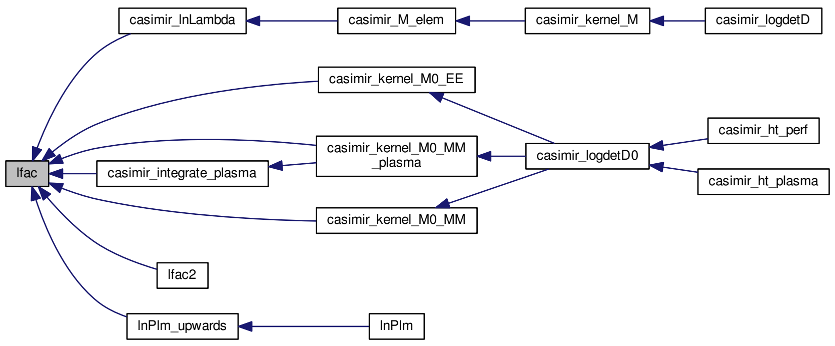
Parameters

in	*n*	integer

Return values

lfac	$\log(n!)$

Here is the caller graph for this function:



5.12.2.2 lfac2()

```
double lfac2 (
              unsigned int n )
```

Calculate $\log(n!!)$.

This function computes the logarithm of the double factorial $n!!$.

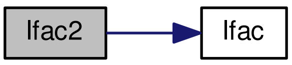
Parameters

in	*n*	argument

Return values

lfac2	$n!!$

Here is the call graph for this function:



5.12.2.3 logi()

```
double logi (
            unsigned int n )
```

Calculate $\log(n)$ for integer n.

This function uses a lookup table to avoid calling log() for $n \leq 65536$

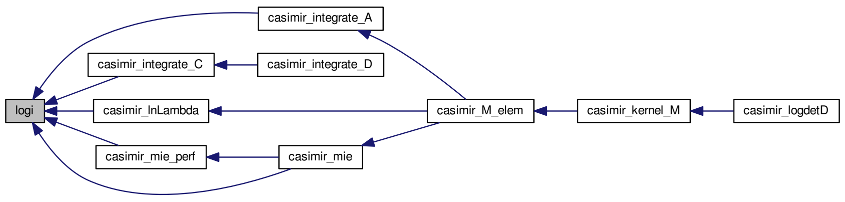
Parameters

in	*n*	integer

Return values

logn	$\log(n)$

Here is the caller graph for this function:



5.12.3 Variable Documentation

5.12.3.1 lookup_lfac

```
double lookup_lfac[ ]  [static]
```

lookup table for $n!$, see lfac

5.12.3.2 lookup_logi

```
double lookup_logi[ ]  [static]
```

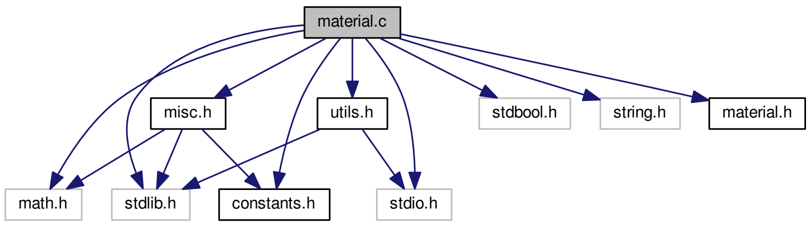
lookup table for $\log(n)$, see logi

5.13 material.c File Reference

support for arbitrary dielectric functions

```
#include <math.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "constants.h"
#include "material.h"
#include "utils.h"
#include "misc.h"
```

Include dependency graph for material.c:



Functions

- static bool _parse (const char ∗line, const char ∗key, const char separator, double ∗value)

  Helper function to parse strings.
- material_t ∗ material_init (const char ∗filename, double calL)

  Initialize material.
- void material_get_extrapolation (material_t ∗material, double ∗omegap_low, double ∗gamma_low, double ∗omegap_high, double ∗gamma_high)

  Get extrapolation parameters.
- void material_free (material_t ∗material)

  Free material object.
- void material_info (material_t ∗material, FILE ∗stream, const char ∗prefix)

  Print information about object to stream.
- double material_epsilonm1 (double xi, void ∗args)

  Dielectric function for material.

5.13.1 Detailed Description

support for arbitrary dielectric functions

Author

Michael Hartmann michael.hartmann@physik.uni-augsburg.de

Date

January, 2019

5.13.2 Function Documentation

5.13.2.1 _parse()

```
static bool _parse (
            const char * line,
            const char * key,
            const char separator,
            double * value )  [static]
```

Helper function to parse strings.

Parse a string in the form of "key separator value" where key and value represent floating numbers. If key or separator is not found, false is returned. If the string is matched successfully, value is set.

Parameters

in	*line*	string to parse
in	*key*	key
in	*separator*	separator
out	*value*	numerical value of the string "value"

Return values

true	parsing successful
false	parsing not successful

5.13.2.2 material_epsilonm1()

```
double material_epsilonm1 (
            double xi,
            void * args )
```

Dielectric function for material.

Return the dielectric function $\epsilon(\mathrm{i}\xi)-1$ for the material. For frequencies greater (smaller) than the maximum (minimum) tabulated frequency, an extrapolation using a Drude model is used. For the tabulated values linear interpolation is used.

Parameters

in	*xi*	frequency in rad/s
in	*args*	material (must be of type material_t $*$)

5.13.2.3 material_free()

```
void material_free (
            material_t * material )
```

Free material object.

Parameters

material	material object

5.13.2.4 material_get_extrapolation()

```
void material_get_extrapolation (
            material_t * material,
            double * omegap_low,
            double * gamma_low,
            double * omegap_high,
            double * gamma_high )
```

Get extrapolation parameters.

For frequencies where there is no tabulated data available, the value of the dielectric function will be extrapolated assuming Drude behaviour:

$$\epsilon(\mathrm{i}\xi) = 1 + \frac{\omega_\mathrm{P}^2}{\xi(\xi+\gamma)}$$

The parameters for the plasma frequency ω_P and the relaxation frequency γ for $\xi > \xi_\mathrm{max}$ and $\xi < \xi_\mathrm{min}$ will be stored into omegap_high, gamma_high, and omegap_low, gamma_low. If a pointer is NULL, the memory is not referenced.

Parameters

in	*material*	material object
out	*omegap_low*	plasma frequency for high-frequency extrapolation (in rad/s)
out	*gamma_low*	relaxation frequency for high-frequency extrapolation (in rad/s)
out	*omegap_high*	plasma frequency for low-frequency extrapolation (in rad/s)
out	*gamma_high*	relaxation frequency for low-frequency extrapolation (in rad/s)

5.13.2.5 material_info()

```
void material_info (
            material_t * material,
            FILE * stream,
            const char * prefix )
```

Print information about object to stream.

Print information (filename, number of points, ξ_min, ξ_max, ...) to stream. If prefix is not NULL, each line will start with the string given in prefix.

Parameters

in	*material*	material object
in	*stream*	output stream (e.g. stdout)
in	*prefix*	prefix for each line or NULL

5.13.2.6 material_init()

```
material_t* material_init (
            const char * filename,
            double calL )
```

Initialize material.

The material properties are read from the file given by filename.

This function temporarily overwrites the value of LC_NUMERIC in the environment. LC_NUMERIC is restored before returning from the function.

Be aware that this function does not check every corner case, so it is dangerous to read untrusted files.

Parameters

in	*filename*	path to material specification
in	*calL*	$L+R$, separation between plane and center of sphere

Return values

material	if successful
NULL	if file cannot be read or is in wrong format

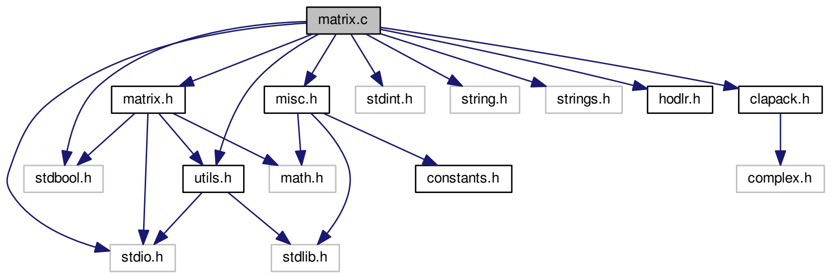
5.14 matrix.c File Reference

Matrix functions.

```
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <strings.h>
#include <hodlr.h>
```

```
#include "matrix.h"
#include "misc.h"
#include "utils.h"
#include "clapack.h"
```

Include dependency graph for matrix.c:



Functions

- double kernel_logdet (int dim, double(∗kernel)(int, int, void ∗), void ∗args, int symmetric, detalg_t detalg)

  Compute $\log\det(1 - A)$.
- matrix_t ∗ matrix_alloc (const size_t dim)

  Create new matrix object.
- void matrix_free (matrix_t ∗A)

  Free matrix.
- int matrix_save_to_stream (matrix_t ∗A, FILE ∗stream)

  Save matrix to stream.
- int matrix_save_to_file (matrix_t ∗A, const char ∗filename)

  Save matrix to file.
- matrix_t ∗ matrix_load_from_stream (FILE ∗stream)

  Load matrix from stream.
- matrix_t ∗ matrix_load_from_file (const char ∗filename)

  Load matrix from file.
- void matrix_setall (matrix_t ∗A, double z)

  Set all matrix elements to value z.
- double matrix_trace (matrix_t ∗A)

  Calculate trace of matrix.
- double matrix_trace2 (matrix_t ∗A)

  Calculate trace of A^2.
- double matrix_norm_frobenius (matrix_t ∗A)

  Calculate Frobenius norm of A.
- double matrix_logdet_triangular (matrix_t ∗A)

  Calculate $\log\det A$ *for triangular matrix* A.
- double matrix_logdet_dense (matrix_t ∗A, double z, detalg_t detalg)

  Calculate $\log\det(1 + zA)$ *for matrix* A.
- double matrix_logdet_lu (matrix_t ∗A)

  Calculate $\log\det A$ *using LU decomposition.*
- double matrix_logdet_cholesky (matrix_t ∗A, char uplo)

  Calculate $\log\det A$ *using Cholesky decomposition.*
- double matrix_logdet_qr (matrix_t ∗A)

  Calculate $\log\det A$ *using QR decomposition.*

5.14.1 Detailed Description

Matrix functions.

Author

Michael Hartmann michael.hartmann@physik.uni-augsburg.de

Date

January, 2019

5.14.2 Function Documentation

5.14.2.1 kernel_logdet()

```
double kernel_logdet (
             int dim,
             double(*)(int, int, void *) kernel,
             void * args,
             int symmetric,
             detalg_t detalg )
```

Compute $\log\det(1-A)$.

This function computes $\log\det(1-A)$ using either the HODLR approach or LU decomposition. The matrix A is given as a callback function. This callback accepts two integers, the row and the column of the matrix entry (starting from 0), and a pointer to args. The callback returns the corresponding matrix element.

If the matrix elements of A are small, i.e., if the modulus of the trace is smaller than 1e-8, the trace will be used as an approximation to prevent a loss of significance. If the modulus of the trace is larger than the modulus of the value computed using HODLR, the trace approximation is returned.

If the determinant is not computed using the HODLR approach, all matrix elements have to be computed. In this case the matrix A is written to the filesystem if the environment variable CASIMIR_DUMP is set. If the variable is set, the matrix will be stored in the path given by CASIMIR_DUMP as a two-dimensional numpy array (npy). This option might be useful for debugging. Also note that if detalg is CHOLESKY, only the upper half of the matrix will be initialized.

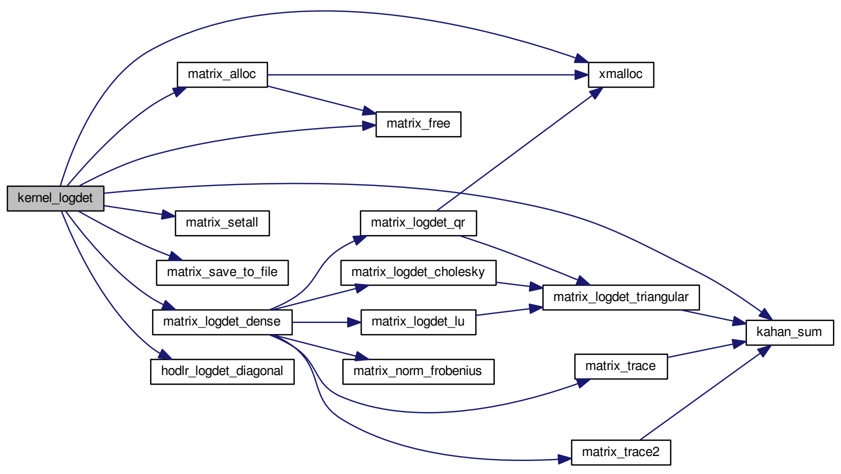
Parameters

in	*dim*	dimension of matrix
in	*kernel*	callback function that returns matrix elements of A
in	*args*	pointer given to callback function kernel
in	*symmetric*	bool indicating whether matrix is symmetric
in	*detalg*	algorithm (DETALG_HODLR, DETALG_LU, DETALG_QR, DETALG_CHOLESKY)

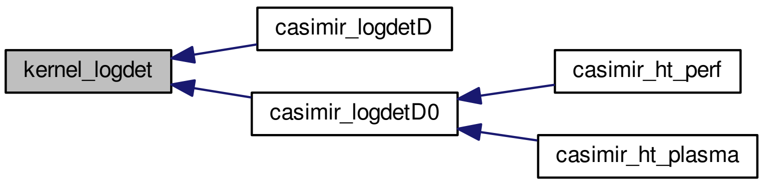
Return values

logdet	$\log\det(1-A)$

Here is the call graph for this function:



Here is the caller graph for this function:



5.14.2.2 matrix_alloc()

```
matrix_t* matrix_alloc (
            const size_t dim )
```

Create new matrix object.

Create a new square matrix with dimension dim x dim. The matrix will not be initialized.

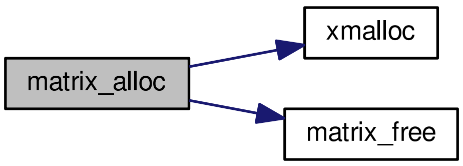
Parameters

in	*dim*	dimension of square matrix

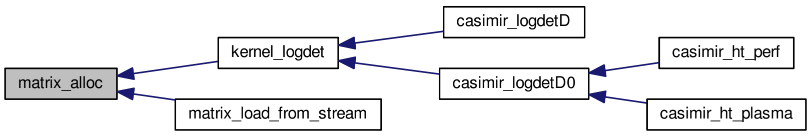
Return values

A	matrix

Here is the call graph for this function:



Here is the caller graph for this function:



5.14.2.3 matrix_free()

```
void matrix_free (
            matrix_t * A )
```

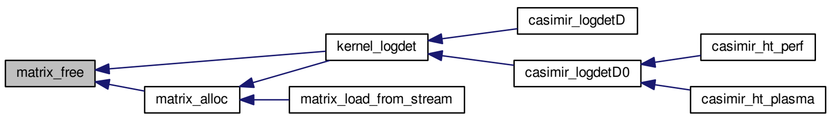
Free matrix.

This function frees the memory allocated for the matrix A.

Parameters

in,out	*A*	matrix

Here is the caller graph for this function:



5.14.2.4 matrix_load_from_file()

```
matrix_t* matrix_load_from_file (
             const char * filename )
```

Load matrix from file.

Load matrix matrix from file filename. See matrix_load_from_stream for more information.

Parameters

in	*filename*	filename of output file

Return values

A	matrix if successful
NULL	if an error occured

5.14.2.5 matrix_load_from_stream()

```
matrix_t* matrix_load_from_stream (
             FILE * stream )
```

Load matrix from stream.

This function loads a matrix from a given stream. The input must be in .npy format. The input matrix must be a square matrix.

The function will rudimentary parse the description string and abort if an error occures. Do not use this function on untrusted data.

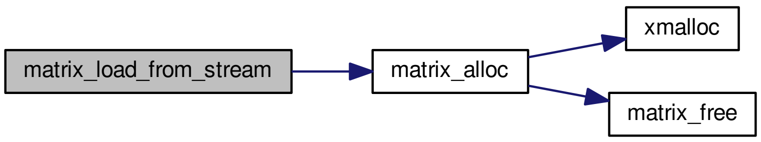
Parameters

in	*stream*	stream

Return values

A	matrix if successful
NULL	if an error occured

Here is the call graph for this function:



5.14.2.6 matrix_logdet_cholesky()

```
double matrix_logdet_cholesky (
             matrix_t * A,
             char uplo )
```

Calculate $\log\det A$ using Cholesky decomposition.

Calculate Cholesky decomposition of A and use matrix_logdet_triangular to calculate $\log\det A$.

Only the lower part of the matrix (uplo=L) or the upper part of the matrix (uplo=U) are used.

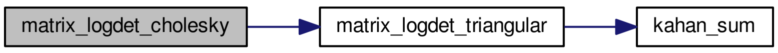
Parameters

`in,out`	*A*	matrix
`in`	*uplo*	L or U

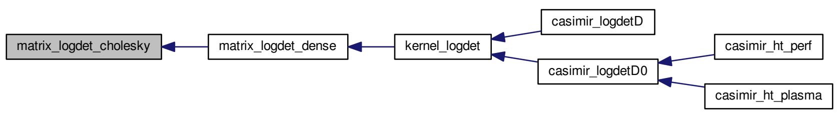
Return values

logdet	$\log\det A$

Here is the call graph for this function:



Here is the caller graph for this function:



5.14.2.7 matrix_logdet_dense()

```
double matrix_logdet_dense (
             matrix_t * A,
             double z,
             detalg_t detalg )
```

Calculate $\log\det(1+zA)$ for matrix A.

Compute $\log\det(1+zA)$ using LAPACK. The algorithm is chosen by detalg and may be DETALG_QR, DETAL↩ G_LU or DETALG_CHOLESKY.

If the Frobenius norm of zA is smaller than 1, the function tries to approximate $\log\det A$ using a Mercator series (if possible) to reduce the complexity for an $N \times N$ matrix A from $\mathcal{O}(N^3)$ to $\mathcal{O}(N^2)$.

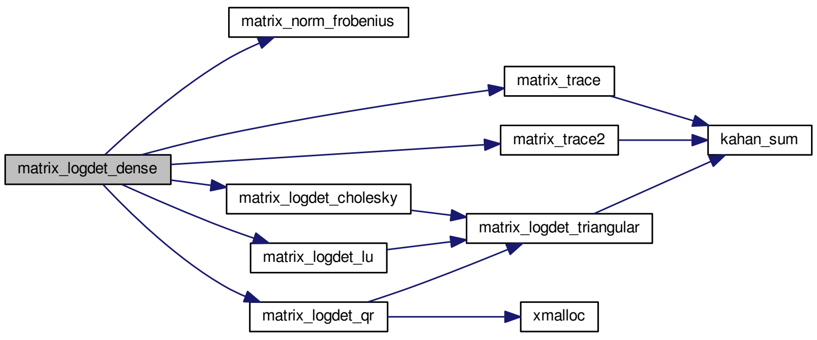
Parameters

in,out	*A*	matrix; will be overwritten.
in	*z*	factor z
in	*detalg*	algorithm to use (cholesky, lu or qr)

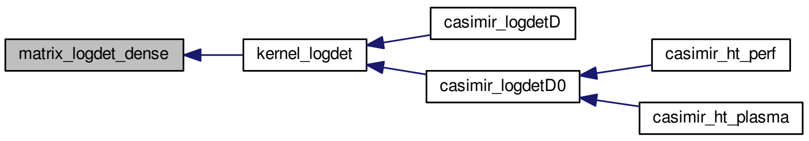
Return values

logdet	$\log\det(1+zA)$

Here is the call graph for this function:



Here is the caller graph for this function:



5.14.2.8 matrix_logdet_lu()

```
double matrix_logdet_lu (
            matrix_t * A )
```

Calculate $\log\det A$ using LU decomposition.

Calculate LU decomposition of A and use matrix_logdet_triangular to calculate $\log\det A$.

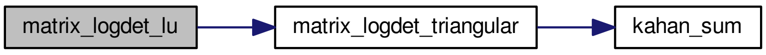
Parameters

in,out	*A*	matrix

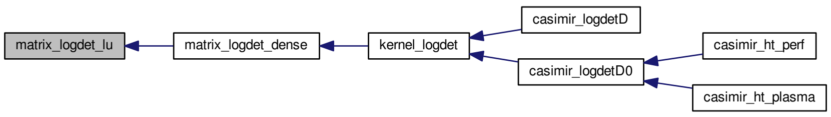
Return values

logdet	$\log\det A$

Here is the call graph for this function:



Here is the caller graph for this function:



5.14.2.9 matrix_logdet_qr()

```
double matrix_logdet_qr (
            matrix_t * A )
```

Calculate $\log\det A$ using QR decomposition.

Calculate QR decomposition of A and use matrix_logdet_triangular to calculate $\log\det A$.

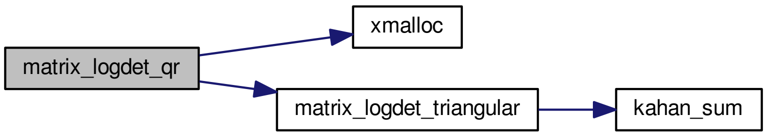
Parameters

`in,out`	*A*	matrix

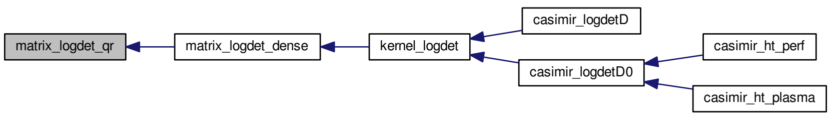
Return values

logdet	$\log\det A$

Here is the call graph for this function:



Here is the caller graph for this function:



5.14.2.10 matrix_logdet_triangular()

```
double matrix_logdet_triangular (
            matrix_t * A )
```

Calculate $\log\det A$ for triangular matrix A.

This function calculates the logarithm of the determinant of the matrix A assuming A is upper or lower triangular:

$$\log\det A = \log\prod_j A_{jj} = \sum_j \log A_{jj}$$

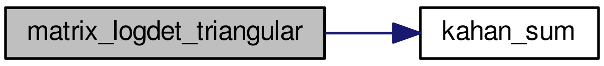
Parameters

in	*A*	triangular matrix

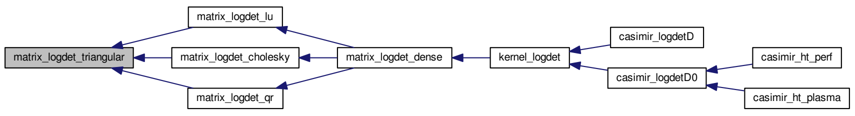
Return values

logdet	$\log\det A$

Here is the call graph for this function:



Here is the caller graph for this function:



5.14.2.11 matrix_norm_frobenius()

```
double matrix_norm_frobenius (
            matrix_t * A )
```

Calculate Frobenius norm of A.

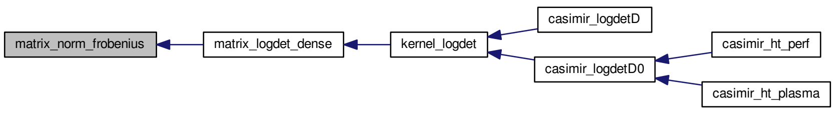
Parameters

in	*A*	matrix

Return values

|← *A*|	Frobenius norm of A

Here is the caller graph for this function:



5.14.2.12 matrix_save_to_file()

```
int matrix_save_to_file (
            matrix_t * A,
            const char * filename )
```

Save matrix to file.

Save matrix A to file filename. See matrix_save_to_stream for more information.

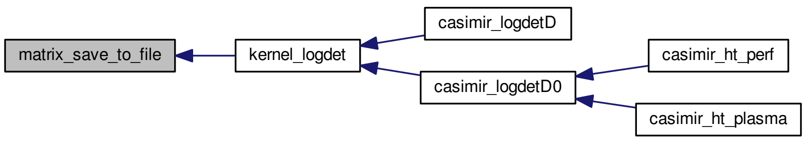
Parameters

in	*A*	matrix
in	*filename*	filename of output file

Return values

0	

Here is the caller graph for this function:



5.14.2.13 matrix_save_to_stream()

```
int matrix_save_to_stream (
            matrix_t * A,
            FILE * stream )
```

Save matrix to stream.

This function saves the matrix A to the stream given by stream. The output is in the numpy .npy format.

Parameters

in	*A*	matrix
in	*stream*	stream

Return values

0	

5.14.2.14 matrix_setall()

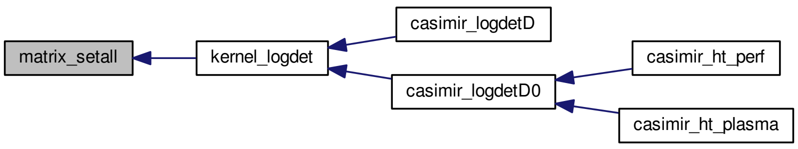
```
void matrix_setall (
             matrix_t * A,
             double z )
```

Set all matrix elements to value z.

Parameters

in,out	*A*	matrix
in	*z*	value

Here is the caller graph for this function:



5.14.2.15 matrix_trace()

```
double matrix_trace (
             matrix_t * A )
```

Calculate trace of matrix.

This function uses Kahan sumation (see kahan_sum) to reduce rounding errors.

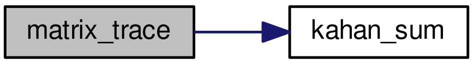
Parameters

in	*A*	matrix

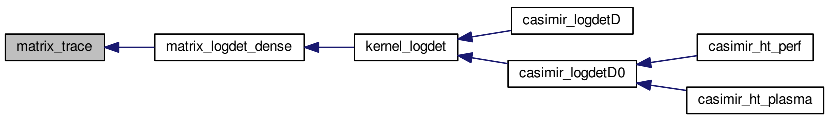
Return values

trace	trace of A

Here is the call graph for this function:



Here is the caller graph for this function:



5.14.2.16 matrix_trace2()

```
double matrix_trace2 (
            matrix_t * A )
```

Calculate trace of A^2.

This function uses Kahan sumation (see kahan_sum) to reduce rounding errors.

The function needs $\mathcal{O}(N^2)$ operation for an $N \times N$ matrix.

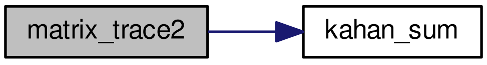
Parameters

in	*A*	matrix

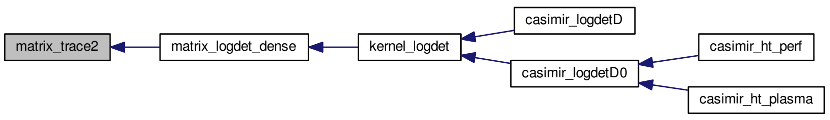
Return values

trace	$\text{tr}\left(A^2\right)$

Here is the call graph for this function:



Here is the caller graph for this function:

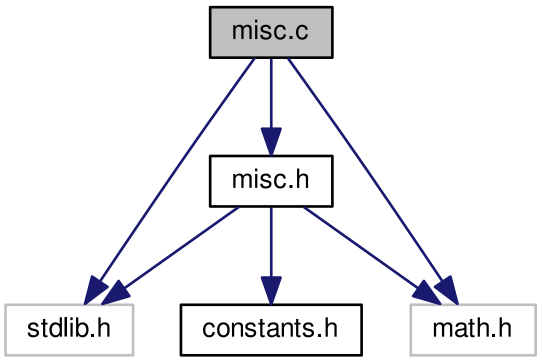


5.15 misc.c File Reference

various mathematical functions

```
#include <stdlib.h>
#include <math.h>
#include "misc.h"
```

Include dependency graph for misc.c:



Functions

- double kahan_sum (double input[ ], size_t N)

  Compute sum of array elements.
- double sqrtpm1 (double x)

  Compute $\sqrt{1+x}-1.$
- double logadd (const double log_a, const double log_b)

  Add two numbers given by their logarithms.
- double logadd_ms (log_t list[ ], const int N, sign_t ∗sign)

  Add N numbers given by their logarithms.

5.15.1 Detailed Description

various mathematical functions

Author

Michael Hartmann `michael.hartmann@physik.uni-augsburg.de`

Date

July, 2017

5.15.2 Function Documentation

5.15.2.1 kahan_sum()

```
double kahan_sum (
            double input[ ],
            size_t N )
```

Compute sum of array elements.

This function calculates the sum of the elements of the array input. This function uses the Kahan summation algorithm to reduce numerical error.

The algorithm is taken from Wikipedia, see `https://en.wikipedia.org/wiki/Kahan_summation←_algorithm`.

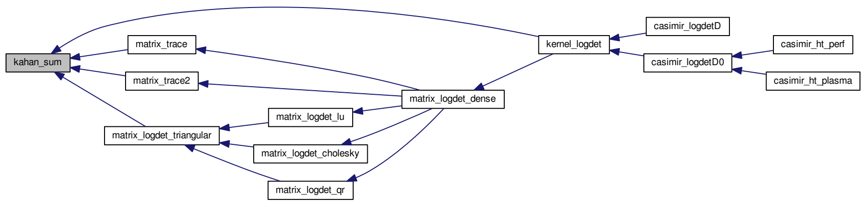
Parameters

in	*input*	array
in	*N*	length of array

Returns

sum sum of array elements

Here is the caller graph for this function:



5.15.2.2 logadd()

```
double logadd (
             const double log_a,
             const double log_b )
```

Add two numbers given by their logarithms.

Both numbers are assumed to be nonnegative.

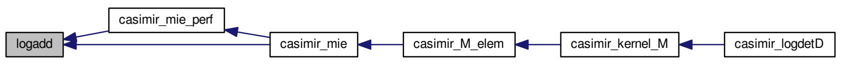
Parameters

in	*log_a*	number
in	*log_b*	number

Returns

log_sum $\log\left[\exp(\log\_a)+\exp\left(\log\_b\right)\right]$

Here is the caller graph for this function:



5.15.2.3 logadd_ms()

```
double logadd_ms (
            log_t list[],
            const int N,
            sign_t * sign )
```

Add N numbers given by their logarithms.

The logarithm and the sign of the N numbers are given by list. The numbers of elements of list must be N, the sign of the result will be stored in sign.

Parameters

in	*list*	list of numbers given by logarithm and sign
in	*N*	number of elements of list
out	*sign*	sign of the result

Returns

logsum log(sum_i list_i)

5.15.2.4 sqrtpm1()

```
double sqrtpm1 (
            double x )
```

Compute $\sqrt{1+x}-1$.

If x is small, $\sqrt{1+x}\approx 1$ and a loss of significance occurs when calculating $\sqrt{1+x}-1$.

For this reason we compute

$$\sqrt{1+x}-1=\frac{x}{\sqrt{1+x}+1}$$

to avoid a loss of significance if x is small.

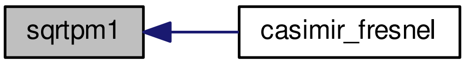
Parameters

in	*x*	

Return values

sqrt(1+x)-1	

Here is the caller graph for this function:

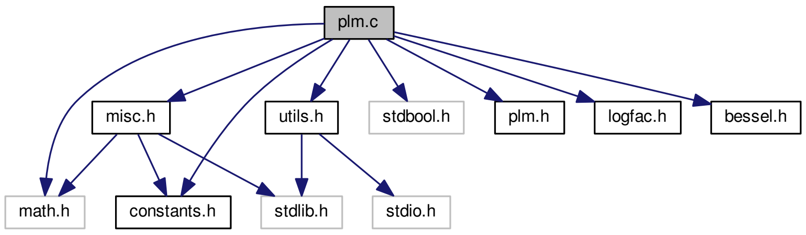


5.16 plm.c File Reference

computation of Legendre and associated Legendre polynomials

```
#include <math.h>
#include <stdbool.h>
#include "constants.h"
#include "plm.h"
#include "logfac.h"
#include "misc.h"
#include "bessel.h"
#include "utils.h"
```

Include dependency graph for plm.c:



Functions

- double lnPlm (int l, int m, double x)

  Associated Legendre polynomials for argument $x > 1$.
- double lnPlm_upwards (int l, int m, double x)

  Associated Legendre polynomials using upwards recurrence relation.
- static double _Pl1 (int l, double x, double sinhxi)

  Compute Legendre polynomial $\log P_l(x)$ for large x.
- static double _fn (int n, double hn[13])
- static double _Pl2 (int l, double x)

  Compute Legendre polynomial $\log P_l(x)$ for small x.
- static double _Pl3 (int l, double x)

  Compute Legendre polynomial $\log P_l(x)$ using recurrence relation.

- double lnPl (int l, double x)

  Compute Legendre polynomial $\log P_l(x)$.
- double Plm_continued_fraction (const long l, const long m, const double x)

  Calculate fraction $P_l^{m-1}(x)/P_l^m(x)$.
- double lnPlm_downwards (int l, int m, double x)

  Compute associated Legendre polynomials using downwards recurrence relation.
- double dlnPlm (int l, int m, double x, double ∗d2lnPlm)

  Compute 1st and 2nd logarithmic derivative of associated Legendre polynomial.

5.16.1 Detailed Description

computation of Legendre and associated Legendre polynomials

Author

Michael Hartmann `michael.hartmann@physik.uni-augsburg.de`

Date

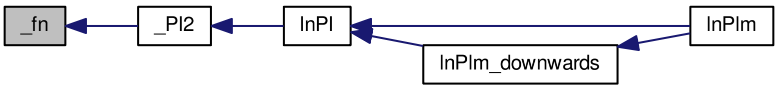
January, 2019

5.16.2 Function Documentation

5.16.2.1 _fn()

```
static double _fn (
            int n,
            double hn[13] )  [static]
```

see equations (3.27)-(3.31) Here is the caller graph for this function:



5.16.2.2 _Pl1()

```
static double _Pl1 (
            int l,
            double x,
            double sinhxi )  [static]
```

Compute Legendre polynomial $\log P_l(x)$ for large x.

Evaluation of $\log P_l(x)$ for $x \geq 1$ using an asymptotic expansion provided that

$$(l+1)\sqrt{(x+1)(x-1)} \geq 25.$$

$\mathcal{O}(1)$ computation of Legendre polynomials and Gauss-Legendre nodes and weights for parallel computing, section 3.2.

See lnPl.

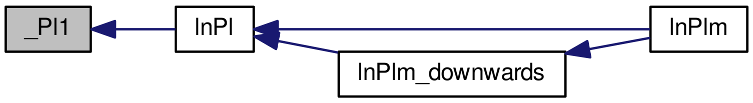
Parameters

in	*l*	degree
in	*x*	argument
in	*sinhxi*	$\sinh\xi = \sqrt{(x+1)(x-1)}$

Return values

log←Pl	$\log P_l(x)$

Here is the caller graph for this function:



5.16.2.3 _Pl2()

```
static double _Pl2 (
            int l,
            double x )  [static]
```

Compute Legendre polynomial $\log P_l(x)$ for small x.

Evaluation of $\log P_l(x)$ for ≥ 1 using an asymptotic expansion provided that

$$(l+1)\sqrt{(x+1)(x-1)} < 25.$$

$\mathcal{O}(1)$ computation of Legendre polynomials and Gauss-Legendre nodes and weights for parallel computing, section 3.3.

See lnPl.

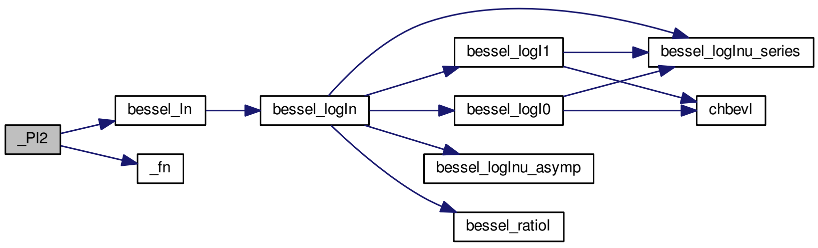
Parameters

in	*l*	
in	*x*	

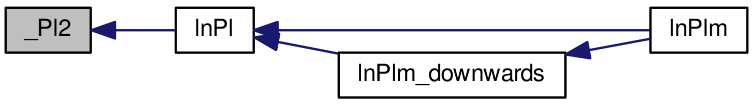
Return values

log←Pl	$\log P_l(x)$

Here is the call graph for this function:



Here is the caller graph for this function:



5.16.2.4 _Pl3()

```
static double _Pl3 (
             int l,
             double x )  [static]
```

Compute Legendre polynomial $\log P_l(x)$ using recurrence relation.

Evaluation of $\log P_l(x)$ for $x \geq 1$ using the recurrence relation

$$(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x).$$

See lnPl.

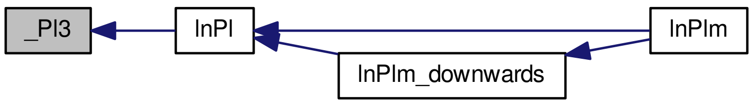
Parameters

`in`	*l*	order
`in`	*x*	argument

Return values

log↩ Pl	$\log P_l(x)$

Here is the caller graph for this function:



5.16.2.5 dlnPlm()

```
double dlnPlm (
              int l,
              int m,
              double x,
              double * d2lnPlm )
```

Compute 1st and 2nd logarithmic derivative of associated Legendre polynomial.

Compute $\frac{\mathrm{d}}{\mathrm{d}x}\log P_l^m(x)$ and $\frac{\mathrm{d}^2}{\mathrm{d}x^2}\log P_l^m(x)$.

If d2lnPlm is NULL, the 2nd logarithmic derivative will not be computed.

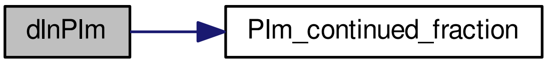
Parameters

`in`	*l*	degree
`in`	*m*	order
`in`	*x*	argument
`out`	*d2lnPlm*	2nd logarithmic derivative of $P_l^m(x)$

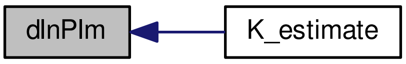
Return values

dlnPlm	first logarithmic derivative of $P_l^m(x)$

Here is the call graph for this function:



Here is the caller graph for this function:



5.16.2.6 lnPl()

```
double lnPl (
            int l,
            double x )
```

Compute Legendre polynomial $\log P_l(x)$.

Evaluation of $\log P_l(x)$ for $x \geq 1$.

For $l < 100$ a recurrence relation is used (see _Pl3), otherwise asymptotic expansions are used (see _Pl1 and _Pl2).

The function returns $\log P_l(x)$.

Reference:

- Bogaert, Michiels, Fostier, O(1) Computation of Legendre Polynomials and Gauss–Legendre Nodes and Weights for Parallel Computing, SIAM J. Sci. Comput. 3, 34 (2012)

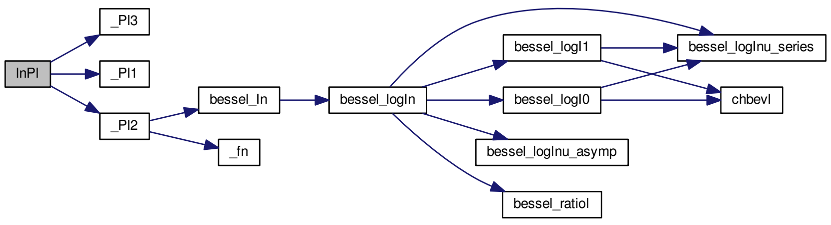
Parameters

in	*l*	degree
in	*x*	argument

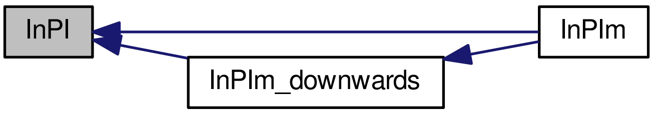
Return values

log↩ Pl	$\log P_l(x)$

Here is the call graph for this function:



Here is the caller graph for this function:



5.16.2.7 lnPlm()

```
double lnPlm (
              int l,
              int m,
              double x )
```

Associated Legendre polynomials for argument $x > 1$.

This function calculates associated Legendre functions for $m \geq 0$ and $x > 1$.

The associated Legendre polynomials for $x > 1$ are defined as follows (see references)

$$P_l^m(x) = (x^2-1)^{m/2}\frac{\mathrm{d}}{\mathrm{d}x^m}P_l(x).$$

Note that in contrast to the common choice in physics, we omit the Condon-Shortly phase $(-1)^m$, and interchange the factors x^2 and 1 in the first bracket after the equal sign. With this definition the associated Legendre polynomials are real and positive functions.

For $l-m \le 200$ we use an upwards recurrence relation in m, see lnPlm_upwards, otherwise we use a downwards recurrence relation in m, see lnPlm_downwards .

References:

- DLMF, §14.7.11, `http://dlmf.nist.gov/14.7#E11`
- Zhang, Jin, Computation of Special Functions, 1996

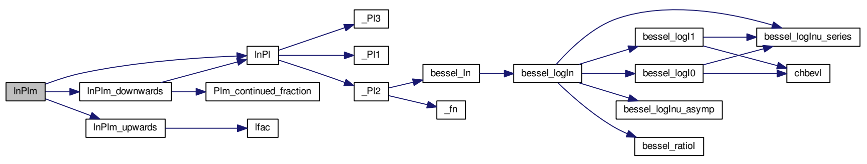
Parameters

`in`	*l*	degree
`in`	*m*	order
`in`	*x*	argument

Return values

logPlm	$\log P_l^m(x)$

Here is the call graph for this function:



5.16.2.8 lnPlm_downwards()

```
double lnPlm_downwards (
              int l,
              int m,
              double x )
```

Compute associated Legendre polynomials using downwards recurrence relation.

First, the fraction $P_l^m(x)/P_l^{m-1}(x)$ is computed using Plm_continued_fraction. Then the downwards recurrence relation `http://dlmf.nist.gov/14.10.E6` is used from $P_l^m(x)$ to $P_l^0(x)$. Together with $P_l(x)$ (see lnPl) one can compute $P_l^m(x)$.

This routine is efficient if $l \gg m$.

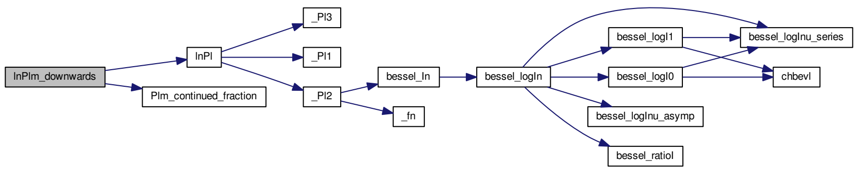
Parameters

`in`	*l*	degree
`in`	*m*	order
`in`	*x*	argument

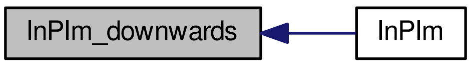
Return values

logPlm	$\log P_l^m(x)$

Here is the call graph for this function:



Here is the caller graph for this function:



5.16.2.9 lnPlm_upwards()

```
double lnPlm_upwards (
              int l,
              int m,
              double x )
```

Associated Legendre polynomials using upwards recurrence relation.

The values of $P_l^m(x)$ is computed using the recurrence relation http://dlmf.nist.gov/14.10.E3 in upwards direction starting from

$$P_m^m(x) = \frac{(2m)!}{2^m m!}(x^2-1)^{m/2}$$

(http://dlmf.nist.gov/14.7.E15).

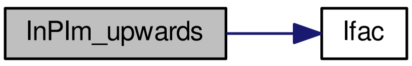
Parameters

in	*l*	degree
in	*m*	order
in	*x*	argument

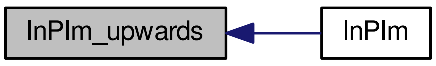
Return values

logPlm	$\log P_l^m(x)$

Here is the call graph for this function:



Here is the caller graph for this function:



5.16.2.10 Plm_continued_fraction()

```
double Plm_continued_fraction (
             const long l,
             const long m,
             const double x )
```

Calculate fraction $P_l^{m-1}(x)/P_l^m(x)$.

The fraction is computed using a continued fraction, see http://dlmf.nist.gov/14.14.E1 .

To evaluate the continued fraction, we use http://dlmf.nist.gov/1.12#E5 and http://dlmf.↩
nist.gov/1.12#E6.

See also Numerical Recipes in C, chapter 5.2, Evaluation of Continued Fractions.

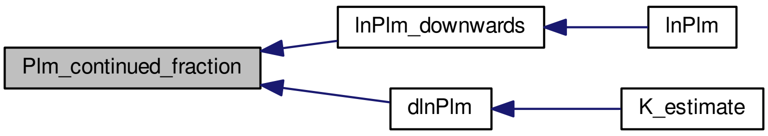
Parameters

`in`	*l*	degree
`in`	*m*	order
`in`	*x*	argument

Return values

ratio	$P_l^{m-1}(x)/P_l^m(x)$

Here is the caller graph for this function:

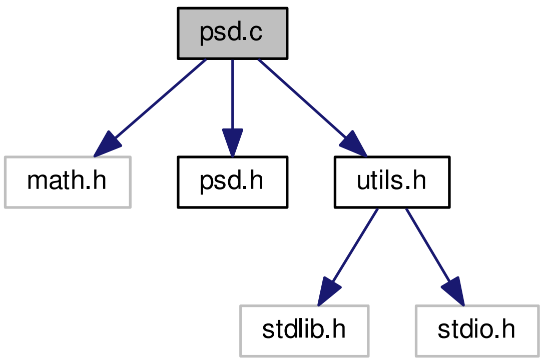


5.17 psd.c File Reference

expansion coefficients and poles for Pade spectrum decomposition

```
#include <math.h>
#include "psd.h"
#include "utils.h"
```

Include dependency graph for psd.c:



Functions

- int dstemr_ (char ∗jobz, char ∗range, int ∗n, double ∗d__, double ∗e, double ∗vl, double ∗vu, int ∗il, int ∗iu, int ∗m, double ∗w, double ∗z__, int ∗ldz, int ∗nzc, int ∗isuppz, int ∗tryrac, double ∗work, int ∗lwork, int ∗iwork, int ∗liwork, int ∗info)
- static double _eta (int N, double z)

  Compute expansion coefficients.
- int psd (int N, double xi[N], double eta[N])

  Compute poles ξ_j and expansion coefficients η_j for PSD.

5.17.1 Detailed Description

expansion coefficients and poles for Pade spectrum decomposition

Author

Michael Hartmann `michael.hartmann@physik.uni-augsburg.de`

Date

December, 2018

5.17.2 Function Documentation

5.17.2.1 _eta()

```
static double _eta (
             int N,
             double z )  [static]
```

Compute expansion coefficients.

Compute expansion coefficient η_j according to the paragraph around equations (12) and (13). See psd.

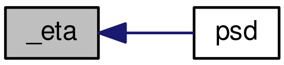
Parameters

in	*N*	order
in	*z*	$z = -\xi_j^2$

Return values

eta↩_j	η_j

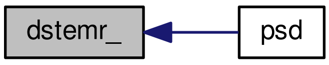
Here is the caller graph for this function:



5.17.2.2 dstemr_()

```
int dstemr_ (
             char * jobz,
             char * range,
             int * n,
             double * d__,
             double * e,
             double * vl,
             double * vu,
             int * il,
             int * iu,
             int * m,
             double * w,
             double * z__,
             int * ldz,
             int * nzc,
             int * isuppz,
             int * tryrac,
             double * work,
             int * lwork,
             int * iwork,
             int * liwork,
             int * info )
```

prototype for LAPACK routine Here is the caller graph for this function:



5.17.2.3 psd()

```
int psd (
             int N,
             double xi[N],
             double eta[N] )
```

Compute poles ξ_j and expansion coefficients η_j for PSD.

This function computes the poles ξ_j (at imaginary frequency) and the expansion coefficients η_j for the Pade spectrum decomposition of order N, see reference [1]. The poles are stored in the array xi, the coefficients are stored in the array eta.

References:

- Hu, Xu, Yan, J. Chem. Phys. 133, 101106 (2010)

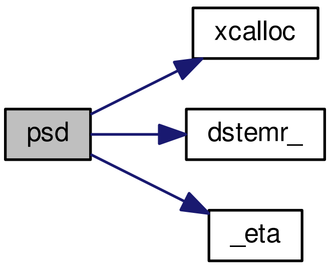
Parameters

in	*N*	order
out	*xi*	poles
out	*eta*	expansion coefficients

Return values

success	0 if successful

Here is the call graph for this function:



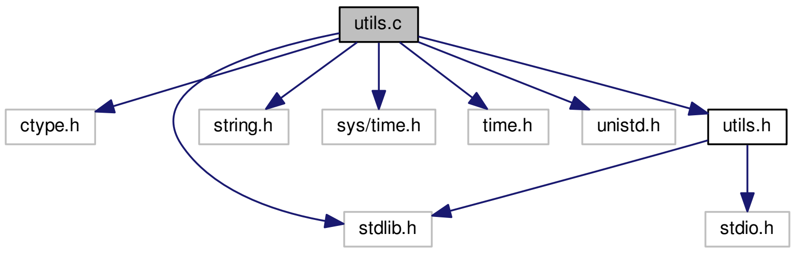
5.18 utils.c File Reference

wrappers for malloc, calloc realloc, and a few more useful functions

```
#include <ctype.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <sys/time.h>
#include <time.h>
#include <unistd.h>
#include "utils.h"
```

Include dependency graph for utils.c:



Functions

- void ∗ xmalloc (size_t size)

  Wrapper for malloc.
- void ∗ xcalloc (size_t nmemb, size_t size)

  Wrapper for calloc.
- void ∗ xrealloc (void ∗p, size_t size)

  Wrapper for realloc.
- double now (void)

  Seconds since 01/01/1970.
- void time_as_string (char ∗s, size_t len)

  Write time into string.
- void disable_buffering (void)

  Disable buffering to stderr and stdout.
- void strep (char ∗s, const char a, const char b)

  Replace character by different character in string.
- void strim (char ∗str)

  Remove whitespace at beginng and end of string.

5.18.1 Detailed Description

wrappers for malloc, calloc realloc, and a few more useful functions

Author

Michael Hartmann `michael.hartmann@physik.uni-augsburg.de`

Date

January, 2018

5.18.2 Function Documentation

5.18.2.1 disable_buffering()

```
void disable_buffering (
             void  )
```

Disable buffering to stderr and stdout.

5.18.2.2 now()

```
double now (
             void  )
```

Seconds since 01/01/1970.

This function returns the seconds since 1st Jan 1970 in µs precision.

Return values

time	seconds since 1st Jan 1970

5.18.2.3 strim()

```
void strim (
             char * str )
```

Remove whitespace at beginng and end of string.

If str is NULL the function doesn't do anything. Otherwise, trailing whitespace and whitespace at the beginning of the string are removed.

Parameters

str	string

5.18.2.4 strrep()

```
void strrep (
             char * s,
             const char a,
             const char b )
```

Replace character by different character in string.

Replace occurence of a by b in the string s.

Parameters

in,out	*s*	string, terminated by \0
in	*a*	character to replace
in	*b*	substitute

5.18.2.5 time_as_string()

```
void time_as_string (
             char * s,
             size_t len )
```

Write time into string.

Write current time in a human readable format into string s. The output is similar to "Aug 30 2018 14:37:35".

Parameters

s	string
len	maximum length of array s

5.18.2.6 xcalloc()

```
void* xcalloc (
             size_t nmemb,
             size_t size )
```

Wrapper for calloc.

This function is a wrapper for calloc. If calloc fails TERMINATE is called.

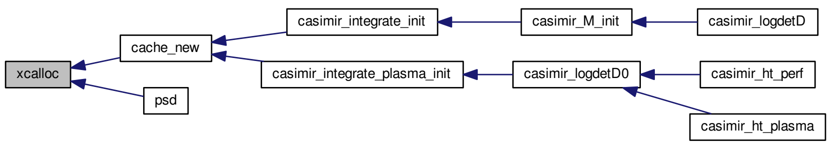
Parameters

nmemb	number of elements
size	size of each element

Return values

ptr	pointer to memory

Here is the caller graph for this function:



5.18.2.7 xmalloc()

```
void* xmalloc (
            size_t size )
```

Wrapper for malloc.

This function is a wrapper for malloc. If malloc fails TERMINATE is called.

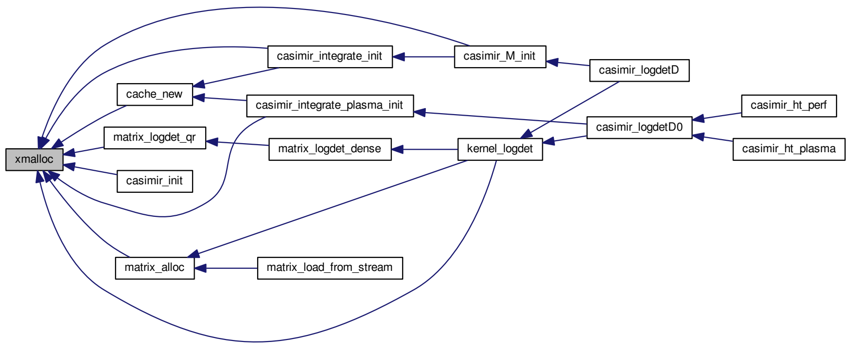
Parameters

size	size of bytes to allocate

Return values

ptr	pointer to memory

Here is the caller graph for this function:



5.18.2.8 xrealloc()

```
void* xrealloc (
            void * p,
            size_t size )
```

Wrapper for realloc.

This function is a wrapper for realloc. If realloc fails TERMINATE is called.

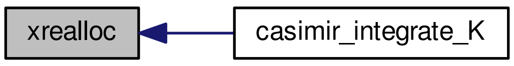
Parameters

p	ptr to old memory
size	size

Return values

newptr	pointer to new memory

Here is the caller graph for this function:



Index