

libcasimir

v0.4.2

Generated by Doxygen 1.8.13

Contents

1	libcasimir	1
1.1	Overview	1
1.2	Files	1
2	Data Structure Index	3
2.1	Data Structures	3
3	File Index	5
3.1	File List	5
4	Data Structure Documentation	7
4.1	argparse Struct Reference	7
4.1.1	Detailed Description	8
4.2	argparse_option Struct Reference	8
4.2.1	Detailed Description	8
4.3	buf Struct Reference	9
4.3.1	Field Documentation	9
4.3.1.1	buffer	9
4.3.1.2	capacity	9
4.3.1.3	size	9
4.4	cache_t Struct Reference	9
4.5	casimir Struct Reference	10
4.5.1	Detailed Description	10
4.5.2	Field Documentation	10
4.5.2.1	call	10

4.5.2.2	detalg	11
4.5.2.3	epsrel	11
4.5.2.4	L	11
4.5.2.5	LbyR	11
4.5.2.6	ldim	11
4.5.2.7	R	11
4.5.2.8	y	11
4.6	casimir_M_t Struct Reference	12
4.7	casimir_mpi_t Struct Reference	13
4.8	casimir_task_t Struct Reference	13
4.9	integrand_plasma_t Struct Reference	14
4.10	integrand_t Struct Reference	14
4.10.1	Detailed Description	14
4.11	integration_plasma_t Struct Reference	15
4.12	integration_t Struct Reference	15
4.12.1	Detailed Description	16
4.13	kernel_args_t Struct Reference	16
4.14	log_t Struct Reference	16
4.14.1	Detailed Description	16
4.14.2	Field Documentation	17
4.14.2.1	s	17
4.14.2.2	v	17
4.15	material_t Struct Reference	17
4.15.1	Detailed Description	17
4.15.2	Field Documentation	17
4.15.2.1	call	18
4.15.2.2	epsm1	18
4.15.2.3	filename	18
4.15.2.4	gamma_high	18
4.15.2.5	gamma_low	18
4.15.2.6	omegap_high	18
4.15.2.7	omegap_low	18
4.15.2.8	points	18
4.15.2.9	xi	19
4.15.2.10	xi_max	19
4.15.2.11	xi_min	19
4.16	matrix_t Struct Reference	19
4.16.1	Detailed Description	19
4.16.2	Field Documentation	19
4.16.2.1	dim	20
4.16.2.2	dim2	20
4.16.2.3	lda	20
4.16.2.4	M	20

5 File Documentation	21
5.1 <code>bessel.c</code> File Reference	21
5.1.1 Detailed Description	23
5.1.2 Function Documentation	23
5.1.2.1 <code>bessel_I0()</code>	23
5.1.2.2 <code>bessel_I1()</code>	24
5.1.2.3 <code>bessel_In()</code>	25
5.1.2.4 <code>bessel_K0()</code>	26
5.1.2.5 <code>bessel_K1()</code>	26
5.1.2.6 <code>bessel_Kn()</code>	27
5.1.2.7 <code>bessel_logI0()</code>	28
5.1.2.8 <code>bessel_logI1()</code>	29
5.1.2.9 <code>bessel_logIn()</code>	30
5.1.2.10 <code>bessel_logIn_half()</code>	31
5.1.2.11 <code>bessel_logInKn_half()</code>	32
5.1.2.12 <code>bessel_logInu_asymp()</code>	33
5.1.2.13 <code>bessel_logInu_series()</code>	34
5.1.2.14 <code>bessel_logK0()</code>	35
5.1.2.15 <code>bessel_logK1()</code>	36
5.1.2.16 <code>bessel_logKn()</code>	37
5.1.2.17 <code>bessel_logKn_half()</code>	38
5.1.2.18 <code>bessel_logKn_recursive()</code>	39
5.1.2.19 <code>bessel_logKnu_asymp()</code>	40
5.1.2.20 <code>bessel_ratiol()</code>	40
5.1.2.21 <code>chbevI()</code>	41
5.1.3 Variable Documentation	42
5.1.3.1 <code>I0_coeffs</code>	42
5.1.3.2 <code>I1_coeffs</code>	43
5.1.3.3 <code>K0_coeffsA</code>	43
5.1.3.4 <code>K0_coeffsB</code>	44

5.1.3.5	K1_coeffsA	44
5.1.3.6	K1_coeffsB	45
5.2	cache.c File Reference	45
5.2.1	Detailed Description	46
5.2.2	Function Documentation	46
5.2.2.1	cache_free()	46
5.2.2.2	cache_insert()	47
5.2.2.3	cache_lookup()	47
5.2.2.4	cache_new()	48
5.3	cquadpack/include/quadpack.h File Reference	49
5.3.1	Detailed Description	50
5.3.2	Macro Definition Documentation	50
5.3.2.1	GK_10_21	50
5.3.2.2	GK_15_31	50
5.3.2.3	GK_20_41	50
5.3.2.4	GK_25_51	50
5.3.2.5	GK_30_61	51
5.3.2.6	GK_7_15	51
5.3.3	Function Documentation	51
5.3.3.1	dqage()	51
5.3.3.2	dqagi()	52
5.3.3.3	dqags()	53
5.4	fcqs.c File Reference	54
5.4.1	Detailed Description	55
5.4.2	Macro Definition Documentation	56
5.4.2.1	MMIN	56
5.4.3	Function Documentation	56
5.4.3.1	cot2()	56
5.4.3.2	fcqs_finite()	56
5.4.3.3	fcqs_semiinf()	57

5.4.3.4	wi_finite()	58
5.4.3.5	wi_semiinf()	58
5.5	include/buf.h File Reference	59
5.5.1	Detailed Description	60
5.5.2	Macro Definition Documentation	60
5.5.2.1	buf_capacity	60
5.5.2.2	buf_clear	61
5.5.2.3	buf_free	61
5.5.2.4	buf_grow	61
5.5.2.5	buf_pop	61
5.5.2.6	buf_push	61
5.5.2.7	buf_size	62
5.5.2.8	buf_trunc	62
5.6	include/constants.h File Reference	62
5.6.1	Detailed Description	63
5.6.2	Macro Definition Documentation	63
5.6.2.1	CASIMIR_c	63
5.6.2.2	CASIMIR_hbar	63
5.6.2.3	CASIMIR_hbar_eV	63
5.6.2.4	CASIMIR_kB	63
5.6.2.5	M_LOG2	63
5.6.2.6	M_LOGPI	64
5.6.2.7	M_PI	64
5.6.2.8	MAX	64
5.6.2.9	MIN	64
5.6.2.10	pow_2	64
5.6.2.11	SGN	64
5.6.3	Typedef Documentation	64
5.6.3.1	sign_t	65
5.7	include/libcasimir.h File Reference	65

5.7.1	Detailed Description	68
5.7.2	Macro Definition Documentation	68
5.7.2.1	CASIMIR_CACHE_ELEMS	68
5.7.2.2	CASIMIR_EPSREL	68
5.7.2.3	CASIMIR_FACTOR_LDIM	68
5.7.2.4	CASIMIR_MINIMUM_LDIM	68
5.7.3	Typedef Documentation	68
5.7.3.1	casimir_t	69
5.7.4	Enumeration Type Documentation	69
5.7.4.1	polarization_t	69
5.7.5	Function Documentation	69
5.7.5.1	casimir_build()	69
5.7.5.2	casimir_epsilonm1_drude()	69
5.7.5.3	casimir_epsilonm1_perf()	70
5.7.5.4	casimir_epsilonm1_plate()	71
5.7.5.5	casimir_epsilonm1_sphere()	71
5.7.5.6	casimir_estimate_lminmax()	72
5.7.5.7	casimir_free()	73
5.7.5.8	casimir_fresnel()	73
5.7.5.9	casimir_get_detalg()	74
5.7.5.10	casimir_get_epsrel()	74
5.7.5.11	casimir_get_ldim()	75
5.7.5.12	casimir_ht_drude()	75
5.7.5.13	casimir_ht_perf()	76
5.7.5.14	casimir_ht_plasma()	77
5.7.5.15	casimir_info()	78
5.7.5.16	casimir_init()	78
5.7.5.17	casimir_kernel_M()	79
5.7.5.18	casimir_kernel_M0_EE()	80
5.7.5.19	casimir_kernel_M0_MM()	81

5.7.5.20	<code>casimir_kernel_M0_MM_plasma()</code>	82
5.7.5.21	<code>casimir_lnLambda()</code>	83
5.7.5.22	<code>casimir_logdetD()</code>	84
5.7.5.23	<code>casimir_logdetD0()</code>	86
5.7.5.24	<code>casimir_M_elem()</code>	87
5.7.5.25	<code>casimir_M_free()</code>	88
5.7.5.26	<code>casimir_M_init()</code>	89
5.7.5.27	<code>casimir_mie()</code>	90
5.7.5.28	<code>casimir_mie_perf()</code>	91
5.7.5.29	<code>casimir_set_detalg()</code>	92
5.7.5.30	<code>casimir_set_epsilonm1()</code>	93
5.7.5.31	<code>casimir_set_epsilonm1_plate()</code>	93
5.7.5.32	<code>casimir_set_epsilonm1_sphere()</code>	94
5.7.5.33	<code>casimir_set_epsrel()</code>	95
5.7.5.34	<code>casimir_set_ldim()</code>	95
5.8	include/utils.h File Reference	96
5.8.1	Detailed Description	97
5.8.2	Macro Definition Documentation	97
5.8.2.1	COMPILER	97
5.8.2.2	TERMINATE	98
5.8.2.3	WARN	98
5.8.2.4	xfree	98
5.8.3	Function Documentation	98
5.8.3.1	<code>disable_buffering()</code>	98
5.8.3.2	<code>now()</code>	98
5.8.3.3	<code>strim()</code>	99
5.8.3.4	<code>strrep()</code>	99
5.8.3.5	<code>time_as_string()</code>	99
5.8.3.6	<code>xcalloc()</code>	100
5.8.3.7	<code>xmalloc()</code>	100

5.8.3.8	<code>xrealloc()</code>	101
5.9	integration.c File Reference	102
5.9.1	Detailed Description	103
5.9.2	Function Documentation	103
5.9.2.1	<code>casimir_integrate_A()</code>	104
5.9.2.2	<code>casimir_integrate_B()</code>	105
5.9.2.3	<code>casimir_integrate_C()</code>	105
5.9.2.4	<code>casimir_integrate_D()</code>	107
5.9.2.5	<code>casimir_integrate_free()</code>	107
5.9.2.6	<code>casimir_integrate_I()</code>	108
5.9.2.7	<code>casimir_integrate_init()</code>	109
5.9.2.8	<code>casimir_integrate_K()</code>	110
5.9.2.9	<code>casimir_integrate_plasma()</code>	111
5.9.2.10	<code>casimir_integrate_plasma_free()</code>	112
5.9.2.11	<code>casimir_integrate_plasma_init()</code>	113
5.9.2.12	<code>K_estimate()</code>	114
5.10	libcasimir.c File Reference	115
5.10.1	Detailed Description	118
5.10.2	Function Documentation	118
5.10.2.1	<code>casimir_build()</code>	118
5.10.2.2	<code>casimir_epsilonm1_drude()</code>	118
5.10.2.3	<code>casimir_epsilonm1_perf()</code>	119
5.10.2.4	<code>casimir_epsilonm1_plate()</code>	119
5.10.2.5	<code>casimir_epsilonm1_sphere()</code>	120
5.10.2.6	<code>casimir_estimate_lminmax()</code>	121
5.10.2.7	<code>casimir_free()</code>	121
5.10.2.8	<code>casimir_fresnel()</code>	122
5.10.2.9	<code>casimir_get_detalg()</code>	122
5.10.2.10	<code>casimir_get_epsrel()</code>	123
5.10.2.11	<code>casimir_get_ldim()</code>	123

5.10.2.12 casimir_ht_drude()	123
5.10.2.13 casimir_ht_perf()	124
5.10.2.14 casimir_ht_plasma()	125
5.10.2.15 casimir_info()	126
5.10.2.16 casimir_init()	127
5.10.2.17 casimir_kernel_M()	128
5.10.2.18 casimir_kernel_M0_EE()	129
5.10.2.19 casimir_kernel_M0_MM()	130
5.10.2.20 casimir_kernel_M0_MM_plasma()	131
5.10.2.21 casimir_lnLambda()	132
5.10.2.22 casimir_logdetD()	133
5.10.2.23 casimir_logdetD0()	134
5.10.2.24 casimir_M_elem()	136
5.10.2.25 casimir_M_free()	137
5.10.2.26 casimir_M_init()	138
5.10.2.27 casimir_mie()	139
5.10.2.28 casimir_mie_perf()	140
5.10.2.29 casimir_set_detalg()	141
5.10.2.30 casimir_set_epsilonm1()	142
5.10.2.31 casimir_set_epsilonm1_plate()	142
5.10.2.32 casimir_set_epsilonm1_sphere()	143
5.10.2.33 casimir_set_epsrel()	144
5.10.2.34 casimir_set_ldim()	144
5.11 libhodlr/include/hodlr.h File Reference	145
5.11.1 Detailed Description	145
5.11.2 Function Documentation	146
5.11.2.1 hodlr_logdet()	146
5.11.2.2 hodlr_logdet_diagonal()	146
5.12 logfac.c File Reference	147
5.12.1 Detailed Description	148

5.12.2	Function Documentation	148
5.12.2.1	lfac()	148
5.12.2.2	lfac2()	149
5.12.2.3	logi()	150
5.12.3	Variable Documentation	150
5.12.3.1	lookup_lfac	150
5.12.3.2	lookup_logi	150
5.13	material.c File Reference	151
5.13.1	Detailed Description	151
5.13.2	Function Documentation	152
5.13.2.1	_parse()	152
5.13.2.2	material_epsilonm1()	152
5.13.2.3	material_free()	153
5.13.2.4	material_get_extrapolation()	153
5.13.2.5	material_info()	153
5.13.2.6	material_init()	154
5.14	matrix.c File Reference	154
5.14.1	Detailed Description	156
5.14.2	Function Documentation	156
5.14.2.1	kernel_logdet()	156
5.14.2.2	matrix_alloc()	157
5.14.2.3	matrix_free()	158
5.14.2.4	matrix_load_from_file()	159
5.14.2.5	matrix_load_from_stream()	159
5.14.2.6	matrix_logdet_cholesky()	160
5.14.2.7	matrix_logdet_dense()	161
5.14.2.8	matrix_logdet_lu()	162
5.14.2.9	matrix_logdet_qr()	163
5.14.2.10	matrix_logdet_triangular()	164
5.14.2.11	matrix_norm_frobenius()	165

5.14.2.12	matrix_save_to_file()	166
5.14.2.13	matrix_save_to_stream()	166
5.14.2.14	matrix_setall()	167
5.14.2.15	matrix_trace()	167
5.14.2.16	matrix_trace2()	168
5.15	misc.c File Reference	169
5.15.1	Detailed Description	170
5.15.2	Function Documentation	170
5.15.2.1	kahan_sum()	170
5.15.2.2	logadd()	171
5.15.2.3	logadd_ms()	172
5.15.2.4	sqrtpm1()	172
5.16	plm.c File Reference	173
5.16.1	Detailed Description	174
5.16.2	Function Documentation	174
5.16.2.1	_fn()	174
5.16.2.2	_PI1()	175
5.16.2.3	_PI2()	175
5.16.2.4	_PI3()	176
5.16.2.5	dlnPlm()	177
5.16.2.6	lnPI()	178
5.16.2.7	lnPlm()	179
5.16.2.8	lnPlm_downwards()	180
5.16.2.9	lnPlm_upwards()	181
5.16.2.10	Plm_continued_fraction()	182
5.17	psd.c File Reference	183
5.17.1	Detailed Description	184
5.17.2	Function Documentation	184
5.17.2.1	_eta()	184
5.17.2.2	dstemr_()	185
5.17.2.3	psd()	186
5.18	utils.c File Reference	186
5.18.1	Detailed Description	187
5.18.2	Function Documentation	188
5.18.2.1	disable_buffering()	188
5.18.2.2	now()	188
5.18.2.3	strim()	188
5.18.2.4	strep()	188
5.18.2.5	time_as_string()	189
5.18.2.6	xalloc()	189
5.18.2.7	xmalloc()	190
5.18.2.8	xrealloc()	191

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
<code>casimir.c</code>	command line interface to libcasimir (see also user manual)
<code>casimir_logdetD.c</code>	command line interface to compute determinants of the scattering matrix (see also user manual)
<code>cylinder.cpp</code>	command line interface to compute Casimir interaction in the plane-cylinder geometry (see also user manual)
<code>cquadpack/src/*.c</code>	integration routines (CQUADPACK), see cquadpack/include/quadpack.h
<code>libhodlr/src/hodlr.cpp</code>	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 adaptive convergent Fourier-Chebyshev 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:

argparse	7
argparse_option	8
buf	9
cache_t	9
casimir	10
casimir_M_t	12
casimir_mpi_t	13
casimir_task_t	13
integrand_plasma_t	14
integrand_t	14
integration_plasma_t	15
integration_t	15
kernel_args_t	16
log_t	16
material_t	17
matrix_t	19

Chapter 3

File Index

3.1 File List

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

bessel.c	Computation of Bessel functions	21
cache.c	Implementation of a simple cache using a hash table	45
fcqs.c	Exponentially convergent Fourier-Chebyshev quadrature scheme (experimental)	54
integration.c	Perform integration for arbitrary materials	102
libcasimir.c	Library to calculate the free Casimir energy in the plane-sphere geometry	115
logfac.c	Computation of logarithm and factorial for integer arguments; created by logfac.py	147
material.c	Support for arbitrary dielectric functions	151
matrix.c	Matrix functions	154
misc.c	Various mathematical functions	169
plm.c	Computation of Legendre and associated Legendre polynomials	173
psd.c	Expansion coefficients and poles for Pade spectrum decomposition	183
utils.c	Wrappers for malloc, calloc realloc, and a few more useful functions	186
cquadpack/include/quadpack.h	Library for numerical integration of one-dimensional functions	49
include/argparse.h	??
include/bessel.h	??
include/buf.h	Growable memory buffers for C99	59
include/cache.h	??
include/casimir.h	??
include/clapack.h	??
include/constants.h	Define macros and constants	62

include/ cylinder.h	??
include/ fcqs.h	??
include/ integration.h	??
include/ libcasimir.h	65
include/ logfac.h	??
include/ material.h	??
include/ matrix.h	??
include/ misc.h	??
include/ plm.h	??
include/ psd.h	??
include/ utils.h	
Wrappers for malloc, calloc and realloc, assert-like macros, now() -function	96
libhodge/include/ hodge.h	
C wrapper for HODLR library	145

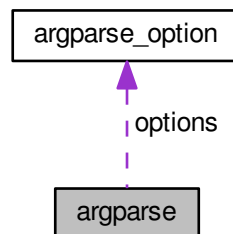
Chapter 4

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 ARGPARSE_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(* [epsilon1_plate](#))(double xi_, void *userdata)
- void * [userdata_plate](#)

dielectric function of the sphere

- double(* [epsilon1_sphere](#))(double xi_, void *userdata)
- void * [userdata_sphere](#)

accuracy and numerical parameters

- int [ldim](#)
- double [epsrel](#)
- detalg_t [deta1g](#)

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
```

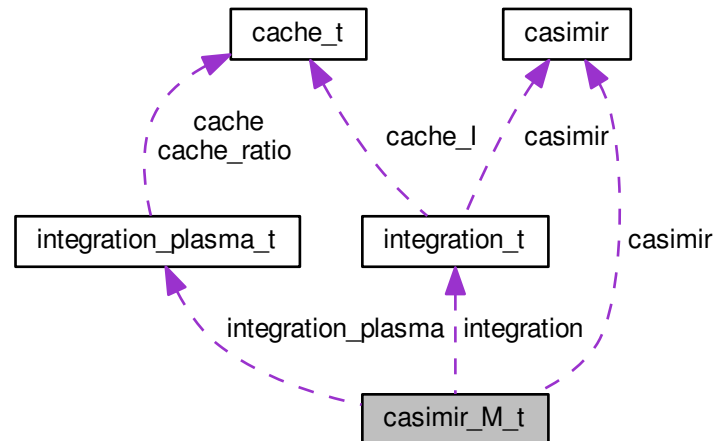
$\log(R/(R+L)/2)$

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

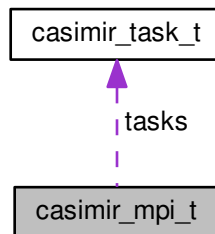
- `casimir_t * casimir`
- `int m`
- `int lmin`
- `int ldim`
- `integration_t * integration`
- `integration_plasma_t * integration_plasma`
- `double xi_`
- `double * al`
- `double * bl`

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

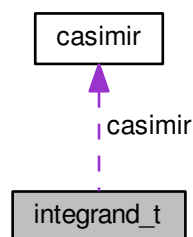
- int **nu**
- double **omegap**
- double **log_prefactor**

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

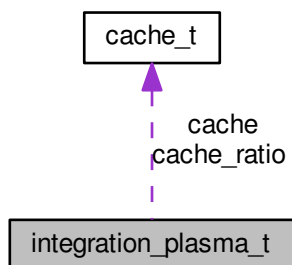
arguments for integrand in function K_integrand

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**

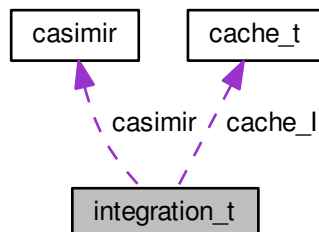
The documentation for this struct was generated from the following file:

- [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_l**
- 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, $\epsilon(i\xi) - 1$

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 high 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

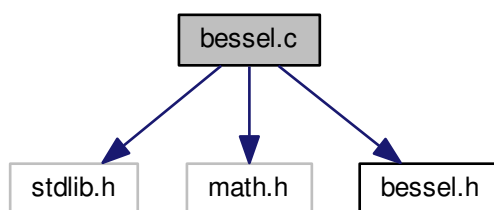
File Documentation

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_logI_nu_asymp](#) (double nu, double x)
Compute modified Bessel function $I_\nu(x)$ using asymptotic expansion.
- double [bessel_logK_nu_asymp](#) (double nu, double x)
Compute modified Bessel function $K_\nu(x)$ using asymptotic expansion.
- double [bessel_logI_nu_series](#) (double nu, double x)
Compute modified Bessel functions $I_\nu(x)$ using series expansion.

modified Bessel functions for half-integer orders

- void [bessel_logI_nKn_half](#) (int n, const double x, double *logI_n_p, double *logK_n_p)
Compute modified Bessel functions of first and second kind for half-integer orders.
- double [bessel_logI_n_half](#) (int n, double x)
Compute $\log I_{n+1/2}(x)$.
- double [bessel_logK_n_half](#) (int n, double x)
Compute $\log K_{n+1/2}(x)$.

modified Bessel functions for orders $\nu=0,1$

- 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](#).

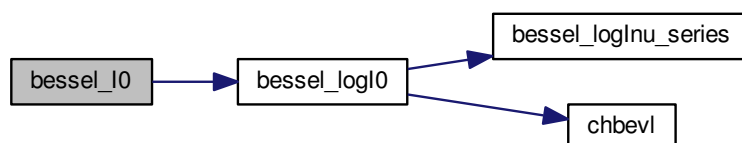
Parameters

in	x	argument
----	-----	----------

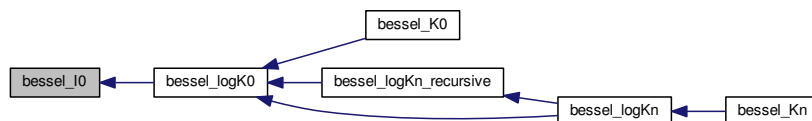
Return values

I_0	$I_0(x)$
-------	----------

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.2 `bessell_I1()`

```
double bessell_I1 (
    double x )
```

Modified Bessel function $I_1(x)$.

See [bessell_logI1](#).

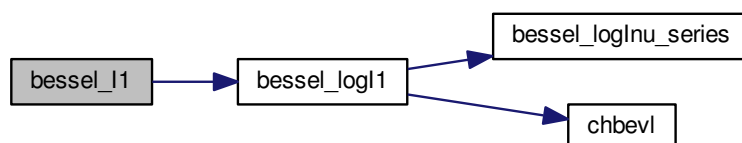
Parameters

in	x	argument
----	---	----------

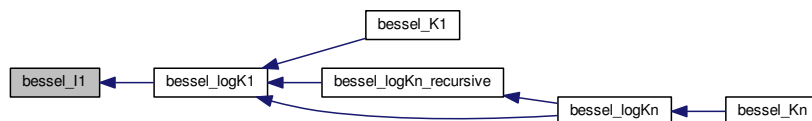
Return values

l1	$I_1(x)$
----	----------

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.3 `bessel_In()`

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

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

See [bessel_logln](#).

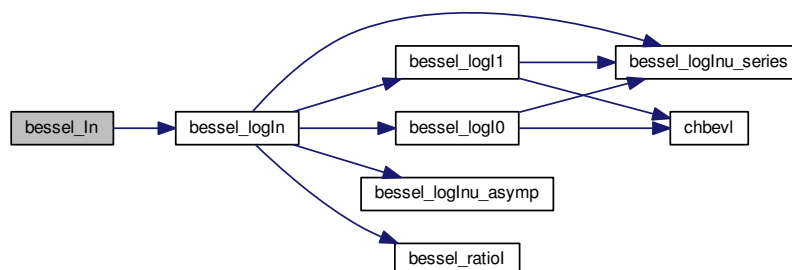
Parameters

in	n	order
in	x	argument

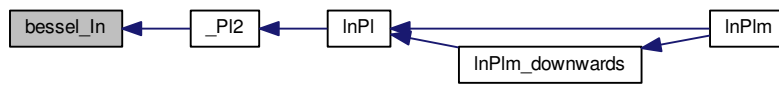
Return values

ln	$I_n(x)$
------	----------

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.4 bessell_K0()

```
double bessell_K0 (
    double x )
```

Modified Bessel function $K_0(x)$.

See [bessell_logK0](#).

Parameters

in	x	argument
----	-----	----------

Return values

$K0$	$K_0(x)$
------	----------

Here is the call graph for this function:



5.1.2.5 bessell_K1()

```
double bessell_K1 (
    double x )
```

Modified Bessel function $K_1(x)$.

See [bessell_logK1](#).

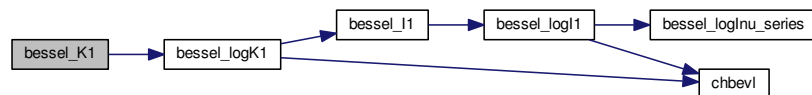
Parameters

in	x	argument
----	-----	----------

Return values

K_1	$K_1(x)$
-------	----------

Here is the call graph for this function:

5.1.2.6 `bessel_Kn()`

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

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

See [bessell_logKn](#).

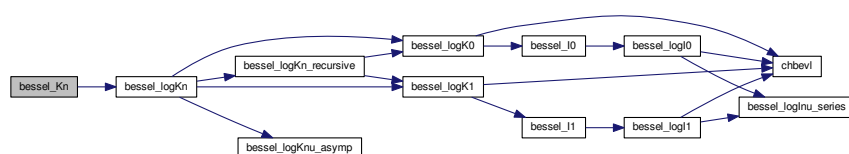
Parameters

in	n	order
in	x	argument

Return values

K_n	$K_n(x)$
-------	----------

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_logI0u_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.

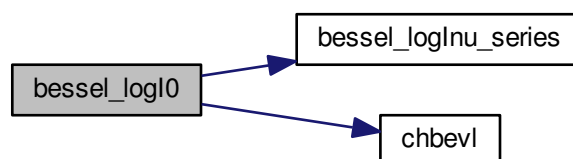
Parameters

in	x	argument
----	---	----------

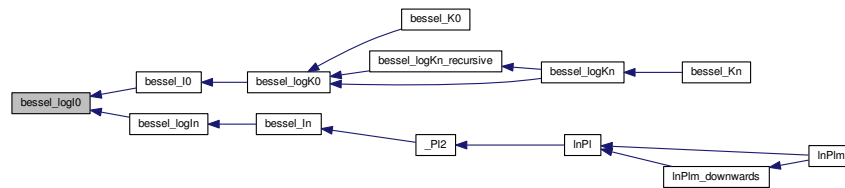
Return values

\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_logl1()`

```
double bessell_logl1 (
    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_loglnu_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.

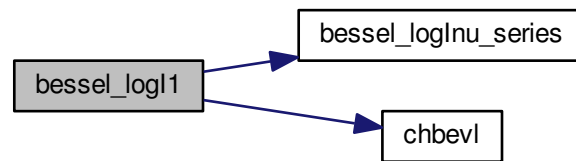
Parameters

in	x	argument
----	---	----------

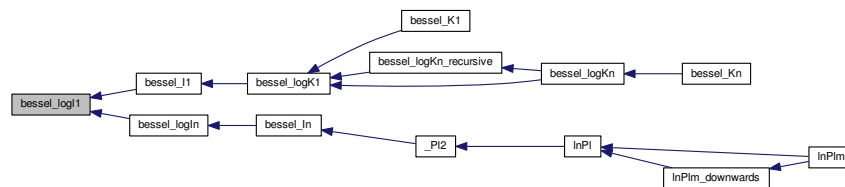
Return values

$\log \leftrightarrow$ l1	$\log I_1(x)$
------------------------------	---------------

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.9 `bessel_logln()`

```
double bessel_logln (
    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_logl0](#) or [bessel_logl1](#).
- For $n \geq 100$ an asymptotic expansion is used, see [bessel_loglnu_asymp](#).
- For $n < 100$ and $x < 5\sqrt{n}$ a series expansion is used, see [bessel_loglnu_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_logl0](#), [bessel_logl1](#), [bessel_loglnu_asymp](#), [bessel_loglnu_series](#), and [bessel_rat0l](#).

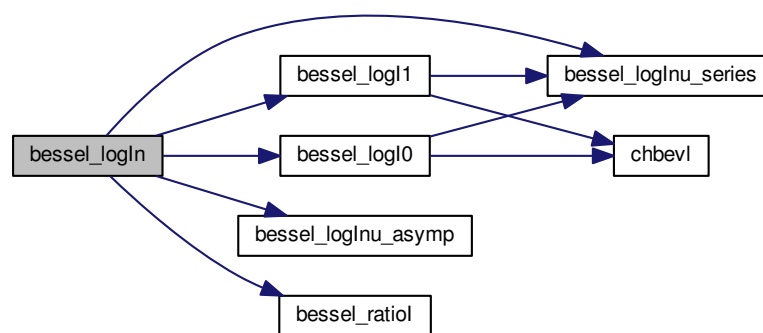
Parameters

in	n	order
in	x	argument

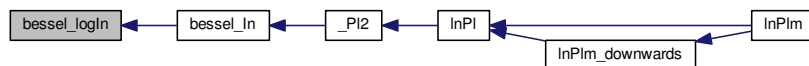
Return values

\ln	$\log I_n(x)$
-------	---------------

Here is the call graph for this function:



Here is the caller graph for this function:

5.1.2.10 `bessell_logln_half()`

```
double bessell_logln_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 $I_{n+1/2}(x)$.

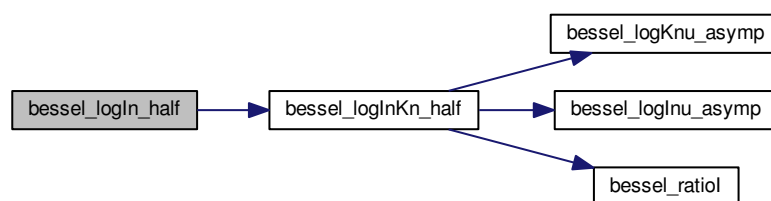
Parameters

in	n	order
in	x	argument

Return values

<i>logI</i>	$\log I_{n+1/2}(x)$
-------------	---------------------

Here is the call graph for this function:

**5.1.2.11 bessell_loglnKn_half()**

```

void bessell_loglnKn_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.

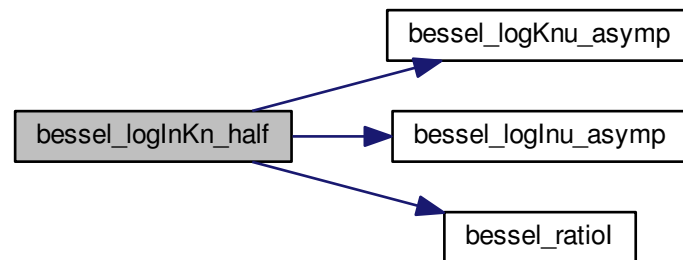
This function computes the logarithm of the modified Bessel functions $I_{n+1/2}(x)$ and $K_{n+1/2}(x)$. The results are saved in logIn_p and logKn_p.

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

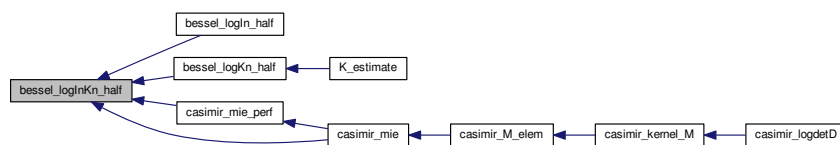
Parameters

in	n	order
in	x	argument
out	<i>logIn</i> ↔ <i>_p</i>	pointer for $\log I_{n+1/2}(x)$
out	<i>logKn</i> ↔ <i>_p</i>	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_loglnu_asymp()`

```
double bessel_loglnu_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>.

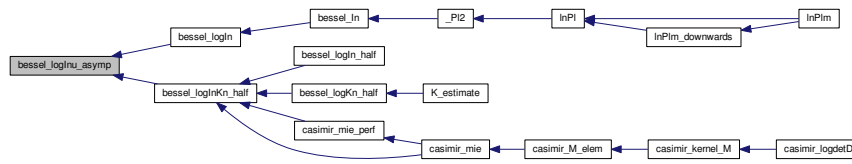
Parameters

in	<i>nu</i>	order
in	<i>x</i>	argument

Return values

<i>logI</i>	$\log I_\nu(x)$
-------------	-----------------

Here is the caller graph for this function:



5.1.2.13 bessell_loglnu_series()

```
double bessell_loglnu_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 $\nu \leq 100000$ when $x \leq 10\sqrt{\nu}$. For larger values of x the function might return NAN.

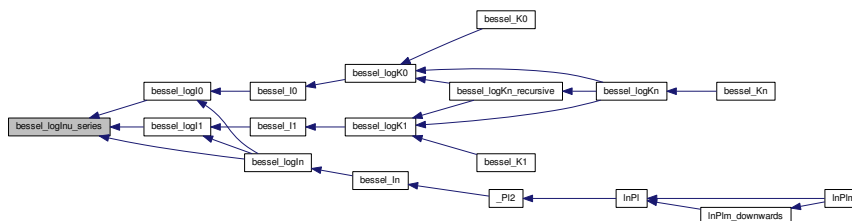
Parameters

in	<i>nu</i>	order
in	<i>x</i>	argument

Return values

<i>lnu</i>	$I_\nu(x)$ 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 $x \rightarrow 0$ 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.

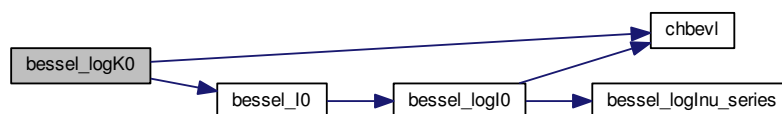
Parameters

in	x	argument
----	---	----------

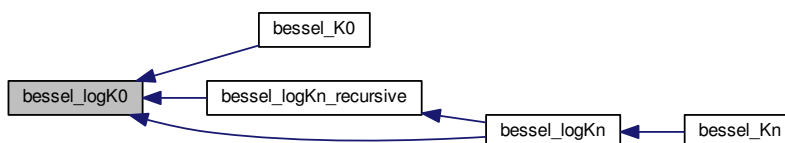
Return values

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 $x \rightarrow 0$ 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.

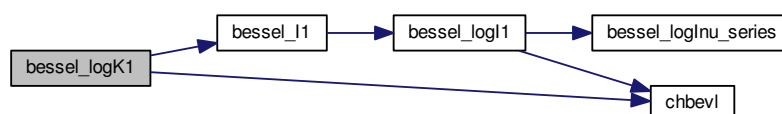
Parameters

in	x	argument
----	---	----------

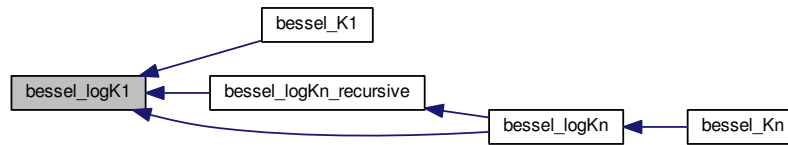
Return values

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 bessell_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_asyp](#).
- Otherwise, the function is computed using a recursion relation, see [bessel_logKn_recursive](#).

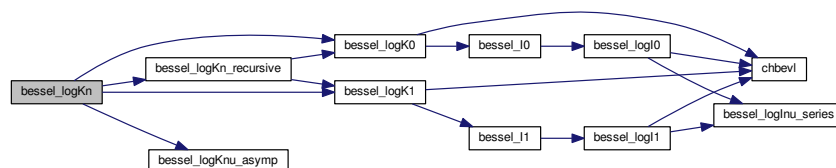
Parameters

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 bessell_logKn_half()

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

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

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

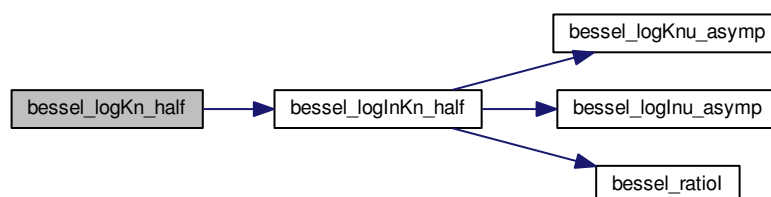
Parameters

in	n	order
in	x	argument

Return values

$\log K$	$K_{n+1/2}(x)$
----------	----------------

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.18 `bessell_logKn_recursive()`

```
double bessell_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 [bessell_logK0](#) and [bessell_logK1](#).

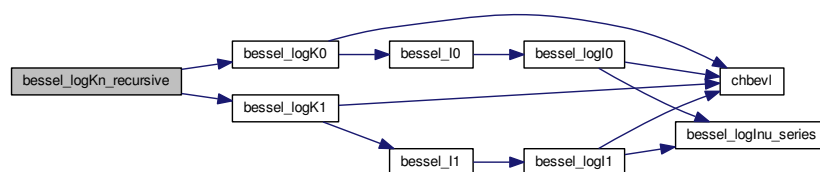
Parameters

in	n	order
in	x	argument

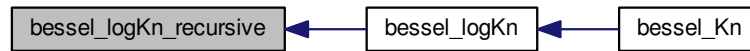
Return values

$\log K_n$	$K_n(x)$
------------	----------

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.19 bessell_logKnu_asymp()

```
double bessell_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>.

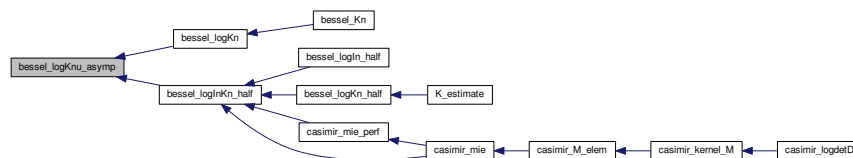
Parameters

in	<i>nu</i>	order
in	<i>x</i>	argument

Return values

<i>logI</i>	$\log I_\nu(x)$
-------------	-----------------

Here is the caller graph for this function:



5.1.2.20 bessell_ratioI()

```
double bessell_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>.

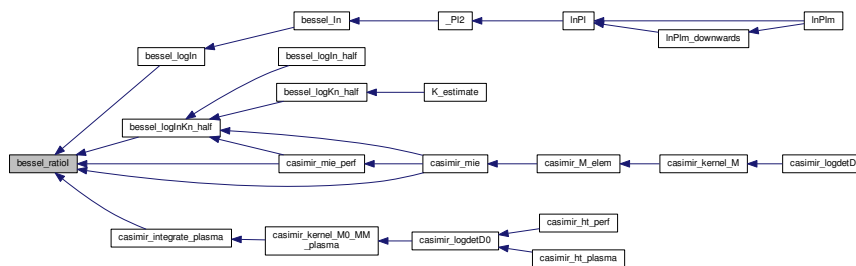
Parameters

<code>nu</code>	order
<code>x</code>	argument

Return values

<code>ratio</code>	$I_\nu(x)/I_{\nu+1}(x)$
--------------------	-------------------------

Here is the caller graph for this function:



5.1.2.21 `chbevI()`

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

Evaluate Chebyshev series.

Evaluates the series

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

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 $x \rightarrow 2(2x - b - a)/(b - a)$ 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 $x \rightarrow 2(2ab/x - b - a)/(b - a)$. If b is infinity, this becomes $x \rightarrow 4a/x - 1$.

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.

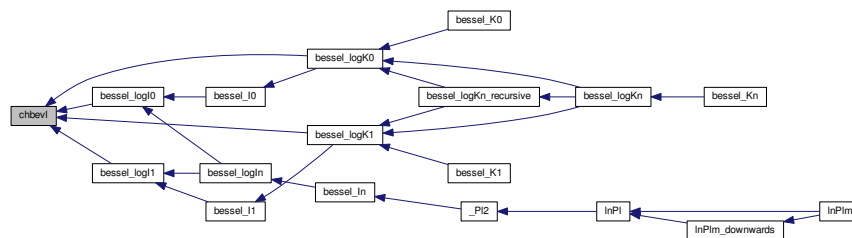
Parameters

in	x	Chebyshev series is evaluated at this point
in	<i>array</i>	Chebyshev coefficients
in	n	number of Chebyshev coefficients, number of elements of array

Return values

<i>eval</i>	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 $[8, \infty]$.

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

5.1.3.2 `l1_coeffs`

```
double l1_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 $[8, \infty]$.

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

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.

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

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 $[2, \infty]$.

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

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 $[0, 2]$.

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

5.1.3.6 K1_coeffsB

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

Initial value:

```
=
{
    -5.75674448366501715755E-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 $[2, \infty]$.

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

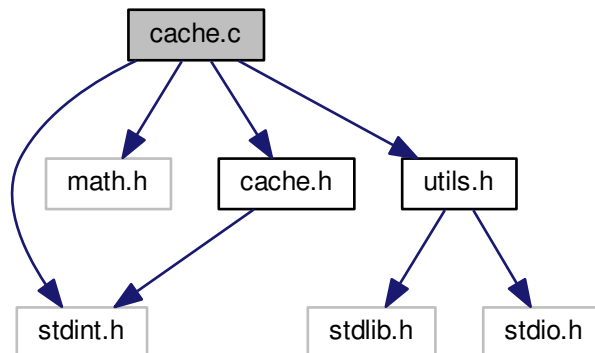
5.2 cache.c File Reference

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()`

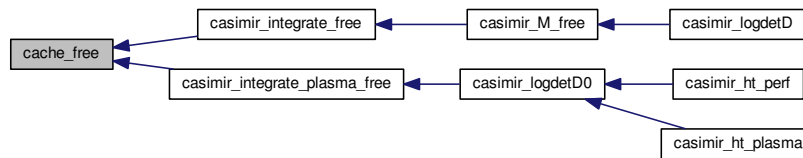
```
void cache_free (
    cache_t * cache )
```

Free cache instance.

Parameters

<i>cache</i>	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 )

```

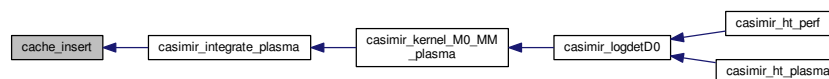
Insert element into cache.

Insert the element value with key key to the cache.

Parameters

<i>cache</i>	cache instance
<i>key</i>	key
<i>value</i>	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*.

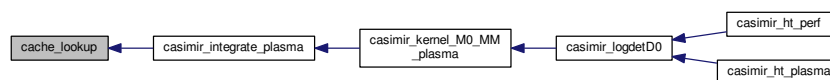
Parameters

<i>cache</i>	cache instance
<i>key</i>	key

Return values

<i>element</i>	if found
<i>NAN</i>	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 specify 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 specify a filling level ($0 < \text{filling} < 1$).

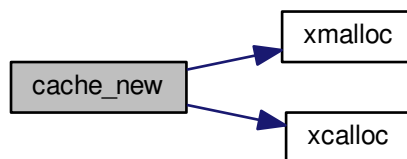
Parameters

<i>entries</i>	maximum number of entries the cache can store
<i>filling</i>	filling level

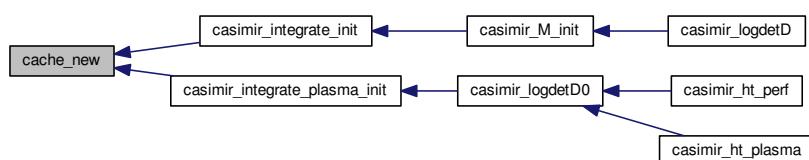
Return values

<i>cache</i>	<code>cache_t</code> instance
--------------	-------------------------------

Here is the call graph for this function:



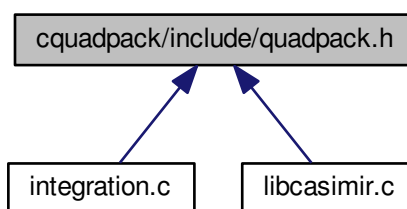
Here is the caller graph for this function:



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	<i>f</i>	double precision function to be integrated
in	<i>a</i>	lower limit of integration
in	<i>b</i>	upper limit of integration

Parameters

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

Return values

<i>result</i>	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	<i>f</i>	double precision function to be integrated
in	<i>bound</i>	optional finite bound on integral
in	<i>inf</i>	specifies range of integration as follows: <ul style="list-style-type: none"> • <code>inf=-1</code>: range is from -infinity to bound, • <code>inf=+1</code>: range is from bound to +infinity, • <code>inf=+2</code>: range is from -infinity to +infinity, (bound is ignored in this case)
in	<i>epsabs</i>	absolute accuracy requested
in	<i>epsrel</i>	relative accuracy requested
out	<i>abserr</i>	estimate of the modulus of the absolute error, which should equal or exceed <code>abs(1-result)</code>
out	<i>neval</i>	number of integrand evaluations
out	<i>ier</i>	error message; <code>ier=0</code> for normal and reliable termination, otherwise <code>ier>0</code>
out	<i>user_data</i>	pointer that will be passed as second argument to integrand function <code>f</code>

Return values

<i>result</i>	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.

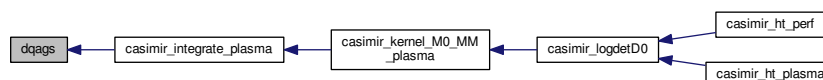
Parameters

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

Return values

<i>result</i>	approximation to the integral
---------------	-------------------------------

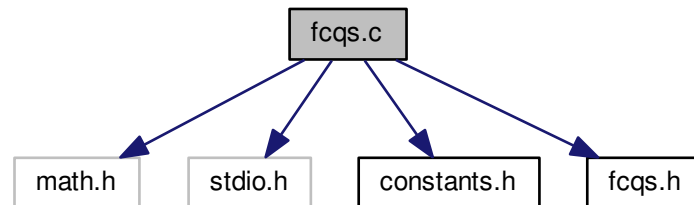
Here is the caller graph for this function:



5.4 fcqs.c File Reference

exponentially convergent Fourier-Chebyshev 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 `MMA` 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-Chebyshev 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

<i>cot2</i>	$\cot^2(x)$
-------------	-------------

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

Return values

<i>integral</i>	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-Chebyshev 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	<i>f</i>	integrand
in	<i>args</i>	pointer given to f when called
in, out	<i>epsrel</i>	on begin desired accuracy, afterwards achieved accuracy
in	<i>neval</i>	number of evaluations of integrand (may be set to NULL)
in	<i>L</i>	boosting parameter
out	<i>ier</i>	exit code

Return values

<i>integral</i>	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-Chebyshev Quadrature Schemes on Bounded and Infinite Intervals, Journal of Scientific Computing, Vol. 2, No. 2 (1987)

Parameters

in	<i>ti</i>	node
in	<i>N</i>	order / number of points

Return values

<i>wi</i>	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-Chebyshev 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

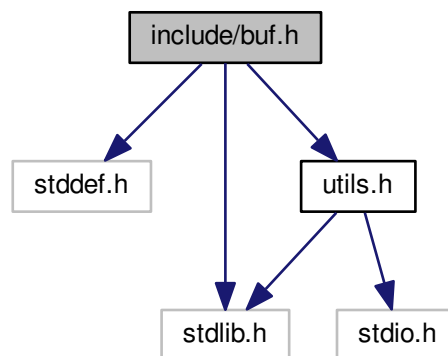
Return values

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 capacity 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

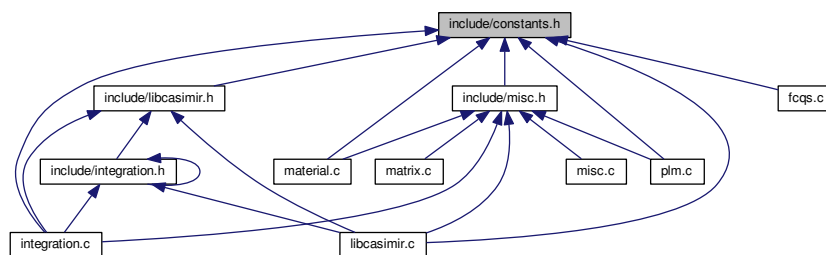
```
#define buf_trunc(  
    v,  
    n ) ((v) = buf_grow1((v), sizeof(*(v)), n - buf_capacity(v)))
```

set buffer capacity 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 $\pi = 3.141592\dots$

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

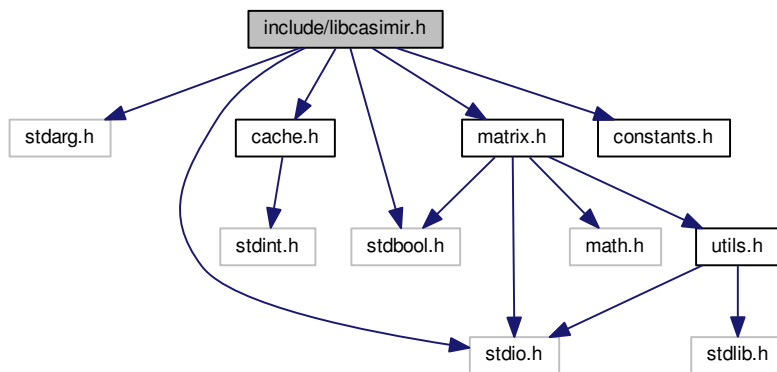
```
typedef signed char sign_t
```

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

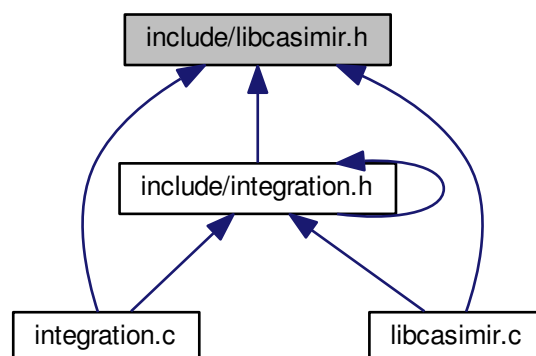
5.7 include/libcasimir.h File Reference

```
#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_InLambda](#) (int l1, int l2, int m)
Calculate logarithm $\Lambda_{\ell_1 \ell_2}^{(m)}$.
- [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 determinant.

 - `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 *rTE, double *rTM)`

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: $\text{Imax} = \text{ceil}(5/L_{\text{byR}})$

5.7.2.4 CASIMIR_MINIMUM_LDIM

```
#define CASIMIR_MINIMUM_LDIM 20
```

minimum value for Imax

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
```

type 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

<i>stream</i>	output stream
<i>prefix</i>	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

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

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	<i>xi</i>	frequency in rad/s
in	<i>userdata</i>	userdata

Return values

<i>epsilon</i>	<code>epsilon(xi)</code>
----------------	--------------------------

5.7.5.3 casimir_epsilonml_perf()

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

Dielectric function for perfect reflectors.

Parameters

in	<i>xi_</i>	ignored
in	<i>userdata</i>	ignored

Return values

<i>inf</i>	$\epsilon(\xi) = \infty$
------------	--------------------------

Here is the caller graph for this function:



5.7.5.4 casimir_epsilon_m1_plate()

```
double casimir_epsilon_m1_plate (
    casimir_t * self,
    double xi_ )
```

Evaluate dielectric function of the plate.

Parameters

in	<i>self</i>	Casimir object
in	$\xi \leftrightarrow$ —	$\xi \mathcal{L}/c$

Return values

<i>epsm1</i>	$\epsilon(i\xi)$
--------------	------------------

Here is the caller graph for this function:



5.7.5.5 casimir_epsilon_m1_sphere()

```
double casimir_epsilon_m1_sphere (
    casimir_t * self,
    double xi_ )
```

Evaluate dielectric function of the sphere.

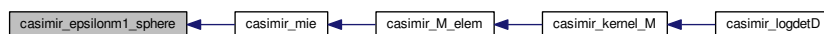
Parameters

in	<i>self</i>	Casimir object
in	$\xi \leftrightarrow$	$\xi \mathcal{L}/c$
	—	

Return values

<i>epsm1</i>	$\epsilon(i\xi)$
--------------	------------------

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 ℓ_{\min}, ℓ_{\max} .

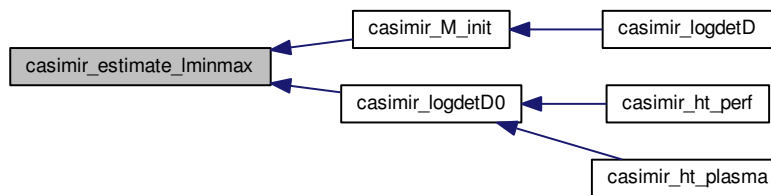
Parameters

in	<i>self</i>	Casimir object
in	m	quantum number
out	$lmin \leftrightarrow$ $_p$	minimum value of ℓ
out	$lmax \leftrightarrow$ $_p$	maximum value of ℓ

Return values

l	approximately the value of ℓ where $\mathcal{M}_{\ell\ell}^m$ 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	<i>self</i>	Casimir object
---------	-------------	----------------

5.7.5.8 casimir_fresnel()

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

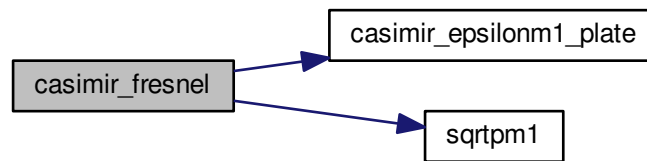
Calculate Fresnel coefficients r_{TE} and r_{TM} for arbitrary metals.

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

Parameters

in	<i>self</i>	Casimir object
in	<i>xi_</i>	$\xi\mathcal{L}/c$
in	<i>k_</i>	$k\mathcal{L}$
in, out	<i>r_TE</i>	Fresnel coefficient for TE mode
in, out	<i>r_TM</i>	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 $r_{\text{TM}} = 1$, $r_{\text{TE}} = 0$ for $\xi \rightarrow 0$, 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>

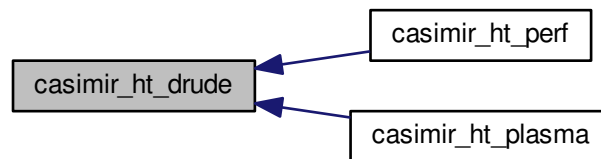
Parameters

in	casimir	Casimir object
----	---------	----------------

Return values

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

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 $r_{\text{TM}} = 1$, $r_{\text{TE}} = -1$ in the limit $\xi \rightarrow 0$, 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>

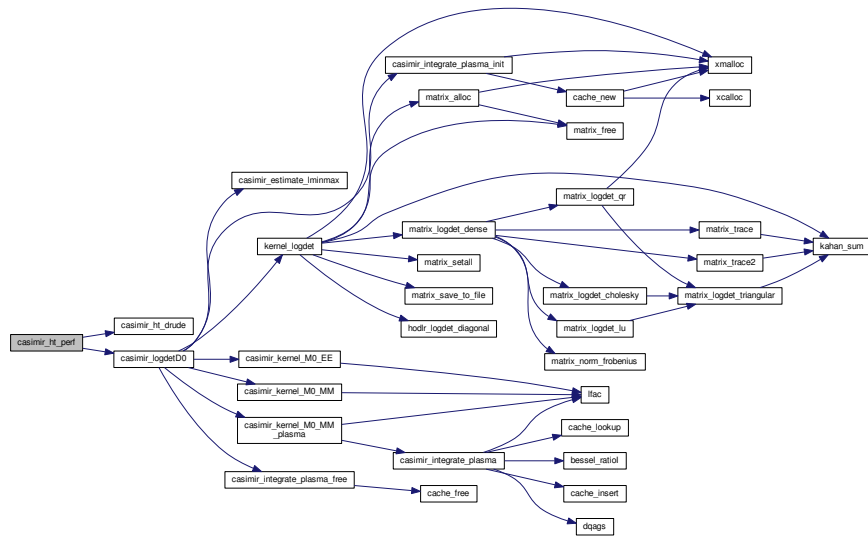
Parameters

in	<i>casimir</i>	Casimir object
in	<i>eps</i>	ϵ abort criterion

Return values

<i>energy</i>	free energy in units of $k_{\text{B}}T$
---------------	---

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	<i>casimir</i>	Casimir object
in	<i>omegap</i>	plasma frequency in rad/s
in	<i>eps</i>	abort criterion

Return values

<i>F</i>	free energy in units of $k_B T$
----------	---------------------------------

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:

$$\ell_{\text{dim}} = \text{ceil} \left(\max \left(\text{CASIMIR_MINIMUM_LDIM}, \text{CASIMIR_FACTOR_LDIM} \cdot \frac{R}{L} \right) \right)$$

Restrictions: $L/R > 0$

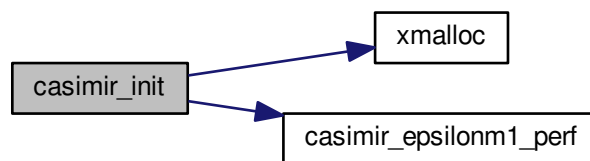
Parameters

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

Return values

<i>object</i>	Casimir object if successful
<i>NULL</i>	if an error occurred

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 $2\ell_{\text{dim}} \times 2\ell_{\text{dim}}$ matrix, the matrix elements start at 0, i.e. $0 \leq i, j < 2\ell_{\text{dim}}$.

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](#).

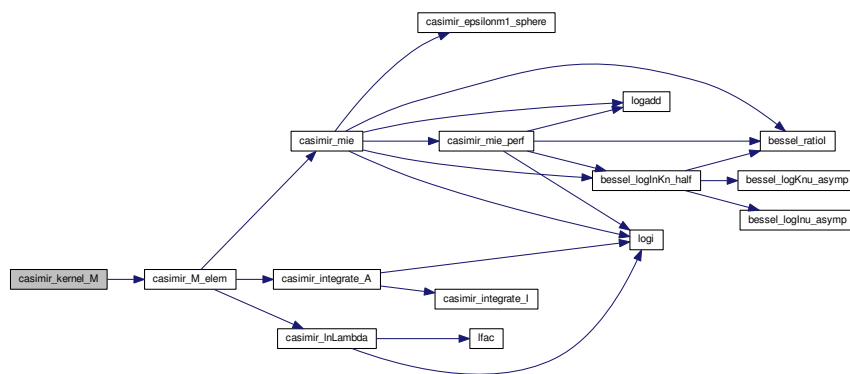
Parameters

in	i	row
in	j	column
in	$args_{\leftrightarrow}$	casimir_M_t object, see casimir_M_init
	—	

Return values

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

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 $p_1 = p_2 = \text{E}$.

See also [casimir_logdetD0](#).

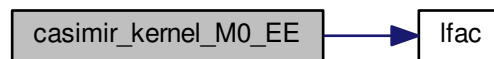
Parameters

in	<i>i</i>	row (starting from 0)
in	<i>j</i>	column (starting from 0)
in	<i>args</i> ↔ —	pointer to casimir_M_t object

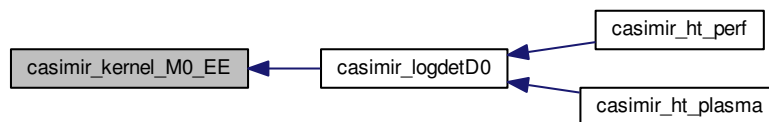
Return values

<i>M_{ij}</i>	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 $p_1 = p_2 = M$.

See also [casimir_logdetD0](#).

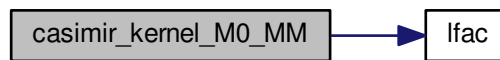
Parameters

in	i	row (starting from 0)
in	j	column (starting from 0)
in	$args_{\leftrightarrow}$	pointer to casimir_M_t object
	—	

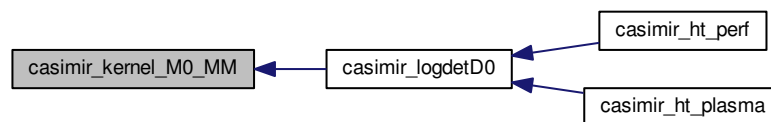
Return values

M_{ij}	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 $p_1 = p_2 = \text{M}$ (plasma model).

See also [casimir_logdetD0](#).

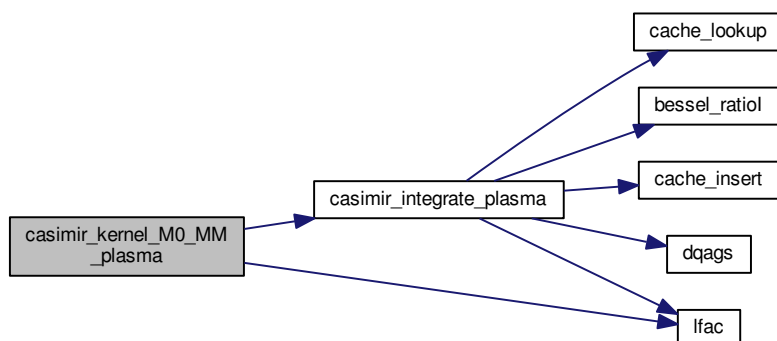
Parameters

in	i	row (starting from 0)
in	j	column (starting from 0)
in	$args$ ↔	pointer to casimir_M_t object
	—	

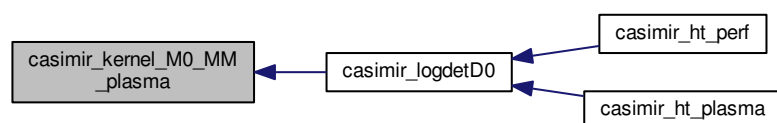
Return values

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

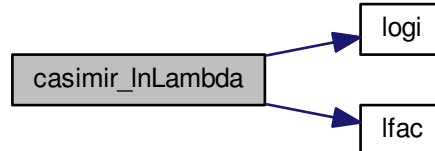
Parameters

in	<i>l1</i>	<i>l1</i> >0
in	<i>l2</i>	<i>l2</i> >0
in	<i>m</i>	<i>m</i> <= <i>l1</i> and <i>m</i> <= <i>l2</i>

Return values

<i>lnLambda</i>	$\log \Lambda_{\ell_1, \ell_2}^{(m)}$
-----------------	---------------------------------------

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](#).

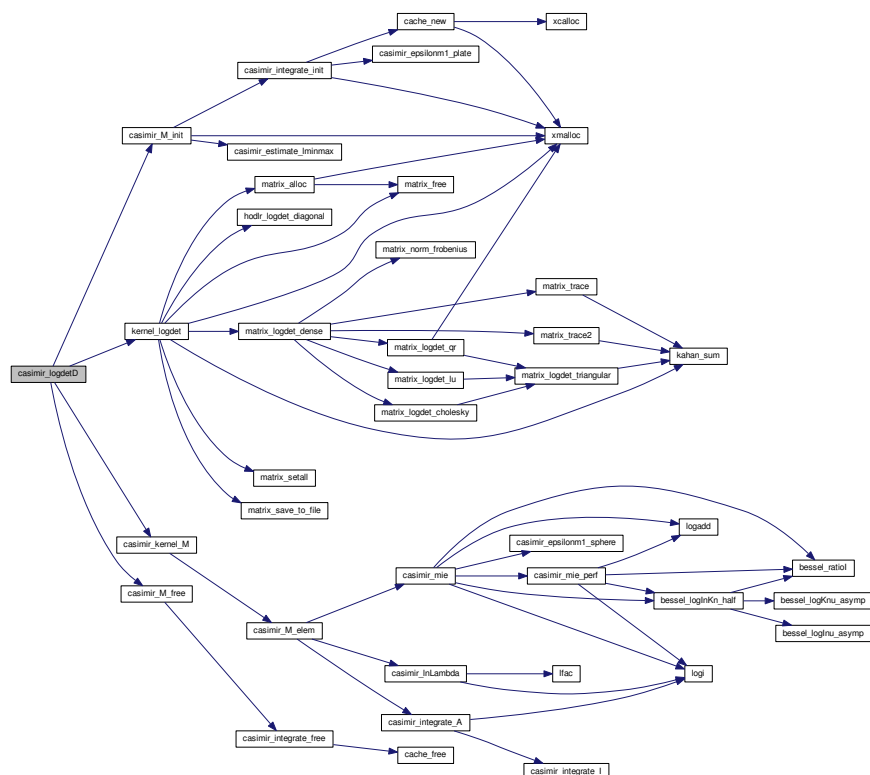
Parameters

<i>self</i>	Casimir object
$xi \mapsto$	$\xi \mathcal{L}/c > 0$
—	
m	quantum number m

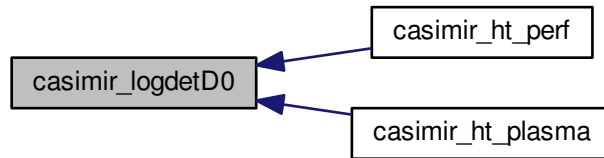
Return values

$\log \det D$	
---------------	--

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.

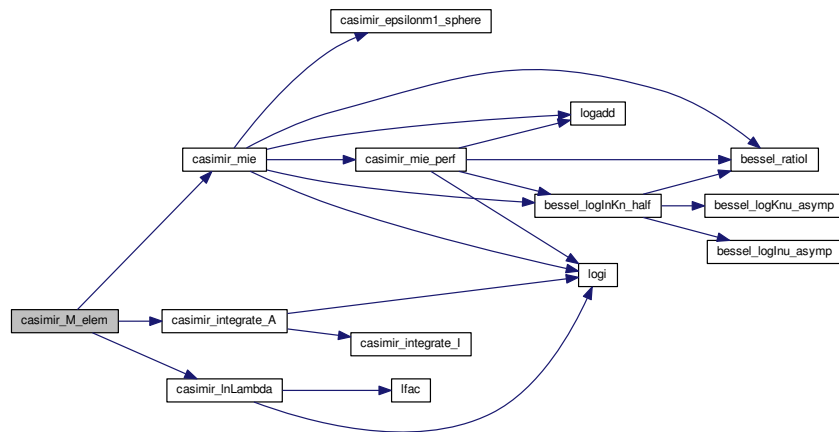
Parameters

in	<i>self</i>	<code>casimir_M_t</code> object, see <code>casimir_M_init</code>
in	<i>l1</i>	angular momentum ℓ_1
in	<i>l2</i>	angular momentum ℓ_2
in	<i>p1</i>	polarization p_1 (E or M)
in	<i>p2</i>	polarization p_2 (E or M)

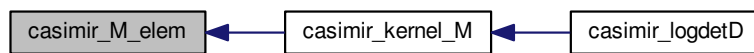
Return values

<i>elem</i>	$\mathcal{M}_{\ell_1, \ell_2}^{(m)}(p_1, p_2)$
-------------	--

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 )
```

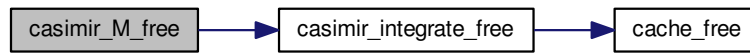
Free `casimir_M_t` object.

Frees memory allocated by `casimir_M_init`.

Parameters

<code>in, out</code>	<i>self</i>	<code>casimir_M_t</code> 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.

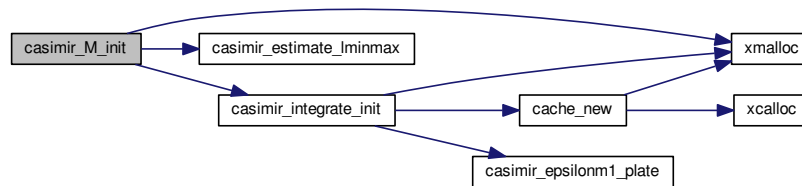
Parameters

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

Return values

<i>obj</i>	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:



5.7.5.27 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 $\omega_P = \infty$ 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 and 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 $\text{sgn}(a_\ell) = (-1)^\ell$, $\text{sgn}(b_\ell) = (-1)^{\ell+1}$.

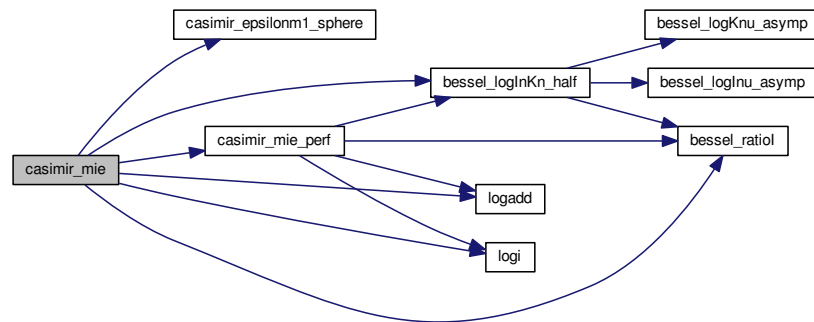
References:

- [1] Erratum: Thermal Casimir effect for Drude metals in the plane-sphere geometry, Canaguier-Durand, Neto, Lambrecht, Reynaud (2010) <http://journals.aps.org/prabstract/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/prabstract/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	<i>self</i>	Casimir object
in	$\chi \leftarrow$	$\xi \mathcal{L}/c$
in	ℓ	angular momentum ℓ
out	<i>lna</i>	logarithm of Mie coefficient a_ℓ
out	<i>lnb</i>	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 $\chi = \xi R/c$.

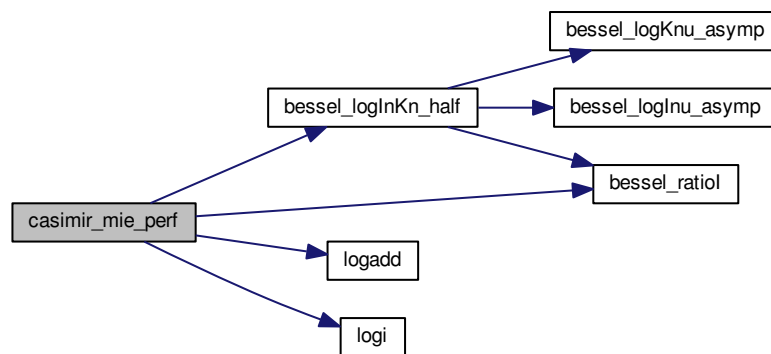
The signs are given by $\text{sgn}(a_\ell) = (-1)^\ell$, $\text{sgn}(b_\ell) = (-1)^{\ell+1}$.

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

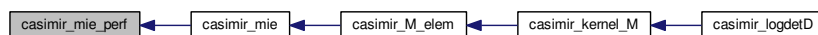
Parameters

in, out	<i>self</i>	Casimir object
in	$\xi \leftarrow \xi \mathcal{L}/c > 0$	
in	l	angular momentum $\ell > 0$
out	<i>lna</i>	logarithm of $ a_\ell $
out	<i>lnb</i>	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 determinant.

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

Parameters

in, out	<i>self</i>	Casimir object
in	<i>detalg</i>	algorithm to compute determinant

Return values

<i>success</i>	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 )
```

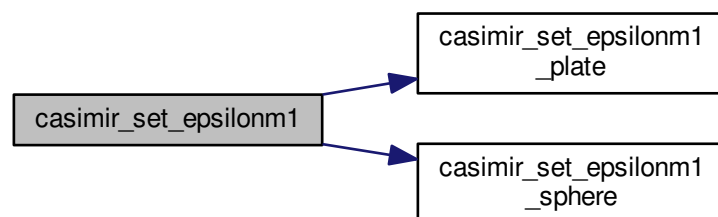
Set dielectric function for plate and sphere.

See also [casimir_set_epsilonm1_plate](#) and [casimir_set_epsilonm1_sphere](#).

Parameters

in, out	<i>self</i>	Casimir object
in	<i>epsilonm1</i>	callback to the function that calculates $\epsilon(i\xi) - 1$
in	<i>userdata</i>	arbitrary pointer to data that is passwd to <i>epsilonm1</i> 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.

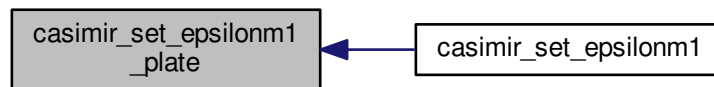
The Fresnel coefficient r_p depend on the dielectric function $\epsilon(i\xi)$. By default, perfect reflectors with a dielectric function $\epsilon(i\xi) = \infty$ are used.

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

Parameters

in, out	<i>self</i>	Casimir object
in	<i>epsilon</i> m1	callback to the function that calculates $\epsilon(i\xi) - 1$
in	<i>userdata</i>	arbitrary pointer to data that is passwd to <i>epsilon</i> m1 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.

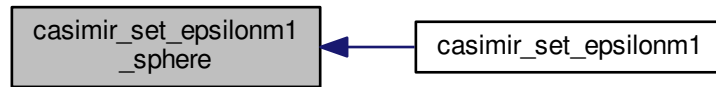
The Mie coefficient a_ℓ, b_ℓ depend on the dielectric function $\epsilon(i\xi)$. By default, perfect reflectors with a dielectric function $\epsilon(i\xi) = \infty$ are used.

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

Parameters

in, out	<i>self</i>	Casimir object
in	<i>epsilon</i> m1	callback to the function that calculates $\epsilon(i\xi) - 1$
in	<i>userdata</i>	arbitrary pointer to data that is passwd to <i>epsilon</i> m1 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	<i>self</i>	Casimir object
in	<i>epsrel</i>	relative error

Return values

0	if an error occurred
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

<code>in, out</code>	<i>self</i>	Casimir object
<code>in</code>	<i>ldim</i>	dimension in angular momentum ℓ

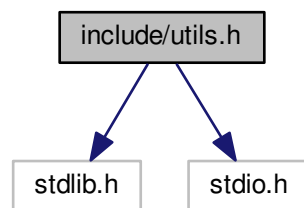
Return values

<code>1</code>	if successful
<code>0</code>	if <code>ldim < 1</code>

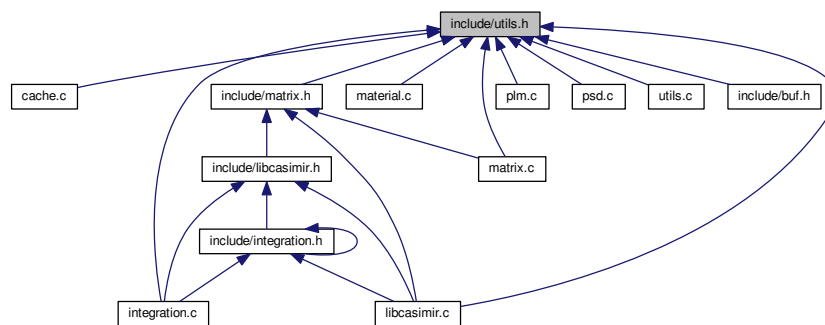
5.8 include/utils.h File Reference

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 μ s precision.

Return values

<i>time</i>	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

<i>str</i>	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

<i>in, out</i>	<i>s</i>	string, terminated by \0
<i>in</i>	<i>a</i>	character to replace
<i>in</i>	<i>b</i>	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

<i>s</i>	string
<i>len</i>	maximum length of array <i>s</i>

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.

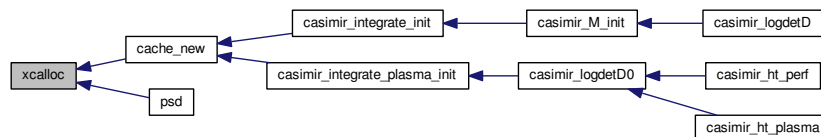
Parameters

<i>nmemb</i>	number of elements
<i>size</i>	size of each element

Return values

<i>ptr</i>	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.

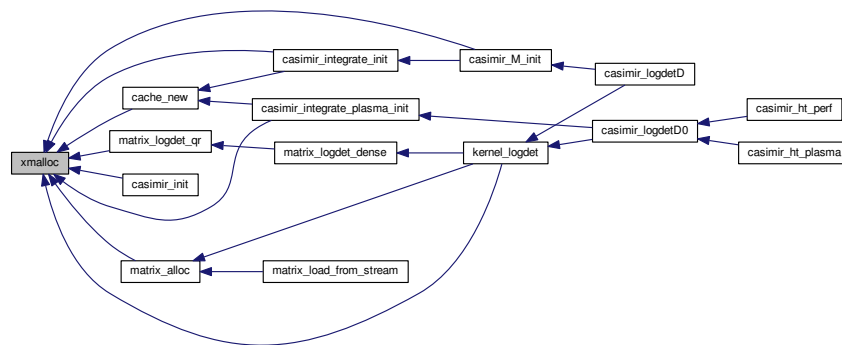
Parameters

<i>size</i>	size of bytes to allocate
-------------	---------------------------

Return values

<i>ptr</i>	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

<i>p</i>	ptr to old memory
<i>size</i>	size

Return values

<i>newptr</i>	pointer to new memory
---------------	-----------------------

Here is the caller graph for this function:



5.9 integration.c File Reference

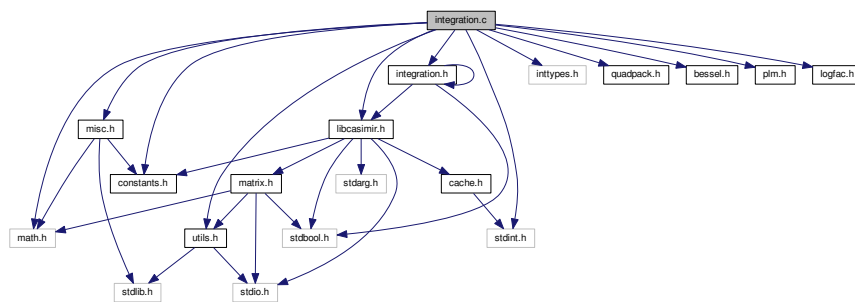
Perform integration for arbitrary materials.

```

#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 $A_{\ell_1, \ell_2, p}^{(m)}(\xi)$

Compute the integral

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

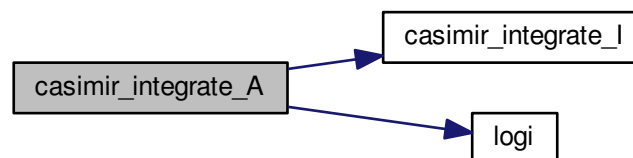
Parameters

in	<i>self</i>	integration object
in	<i>l1</i>	parameter
in	<i>l2</i>	parameter
in	<i>p</i>	polarization; either TE or TM
out	<i>sign</i>	sign of integral $\text{sgn} \left(A_{\ell_1, \ell_2, p}^{(m)}(\xi) \right)$

Return values

<i>logA</i>	$\log \left A_{\ell_1, \ell_2, p}^{(m)}(\xi) \right $
-------------	--

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 $B_{\ell_1, \ell_2, p}^{(m)}(\xi)$

Compute the integral

$$B_{\ell_1, \ell_2, p}^{(m)}(\xi) = \frac{c^3}{\xi^3} \int_0^\infty dk \frac{k^3}{\kappa} r_p e^{-2\kappa \mathcal{L}} P_{\ell_1}^{m'}\left(\frac{\kappa c}{\xi}\right) P_{\ell_2}^{m'}\left(\frac{\kappa c}{\xi}\right)$$

Parameters

in	<i>self</i>	integration object
in	<i>l1</i>	parameter
in	<i>l2</i>	parameter
in	<i>p</i>	polarization; either TE or TM
out	<i>sign</i>	sign of integral $\text{sgn}\left(B_{\ell_1, \ell_2, p}^{(m)}(\xi)\right)$

Return values

<i>logB</i>	$\log\left B_{\ell_1, \ell_2, p}^{(m)}(\xi)\right $
-------------	---

Here is the call graph for this function:



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 $C_{\ell_1, \ell_2, p}^{(m)}(\xi)$

Compute the integral

$$C_{\ell_1, \ell_2, p}^{(m)}(\xi) = \frac{mc}{\xi} \int_0^\infty dk \frac{k}{\kappa} r_p e^{-2\kappa \mathcal{L}} P_{\ell_1}^m \left(\frac{\kappa c}{\xi} \right) P_{\ell_2}^{m'} \left(\frac{\kappa c}{\xi} \right)$$

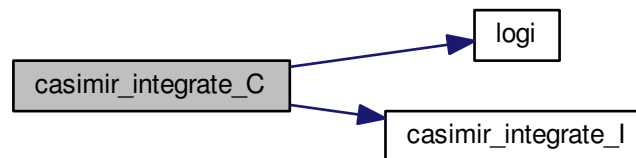
Parameters

in	<i>self</i>	integration object
in	<i>l1</i>	parameter
in	<i>l2</i>	parameter
in	<i>p</i>	polarization; either TE or TM
out	<i>sign</i>	sign of integral $\text{sgn} \left(C_{\ell_1, \ell_2, p}^{(m)}(\xi) \right)$

Return values

<i>logC</i>	$\log \left C_{\ell_1, \ell_2, p}^{(m)}(\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 $D_{\ell_1, \ell_2, p}^{(m)}(\xi)$.

Compute

$$D_{\ell_1, \ell_2, p}^{(m)}(\xi) = C_{\ell_2, \ell_2, 1}^{(m)}(\xi)$$

This function calls [casimir_integrate_C](#).

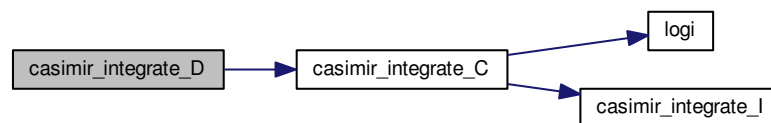
Parameters

in	<i>self</i>	integration object
in	<i>l1</i>	parameter
in	<i>l2</i>	parameter
in	<i>p</i>	polarization; either TE or TM
out	<i>sign</i>	sign of integral $\text{sgn} \left(D_{\ell_1, \ell_2, p}^{(m)}(\xi) \right)$

Return values

<i>logD</i>	$\log \left D_{\ell_1, \ell_2, p}^{(m)}(\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.

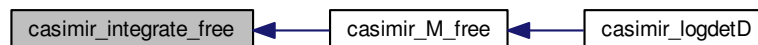
Parameters

in, out	<i>integration</i>	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 $\mathcal{I}_{\ell_1, \ell_2, p}^{(m)}(\alpha)$.

Compute the integral

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

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

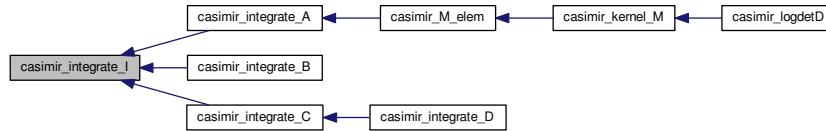
Parameters

in	<i>self</i>	integration object
in	<i>l1</i>	parameter
in	<i>l2</i>	parameter
in	<i>p</i>	polarization; either TE or TM
out	<i>sign</i>	sign of integral $\text{sgn}(\mathcal{I}_{\ell_1, \ell_2, p}^{(m)}(\alpha))$

Return values

<i>logl</i>	$\log \left \mathcal{I}_{\ell_1, \ell_2, p}^{(m)}(\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`.

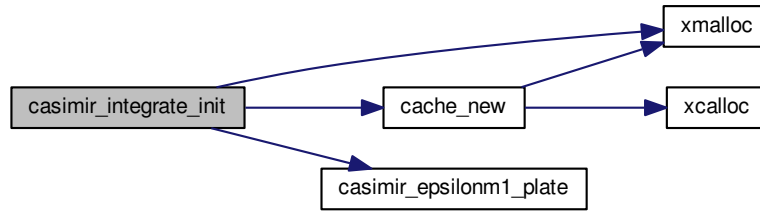
Parameters

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

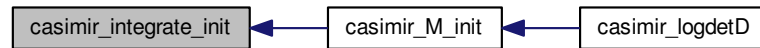
Return values

<i>integration</i>	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

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

and for $m = 0$ the integral

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

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 $\alpha = 2\xi\mathcal{L}/c$.

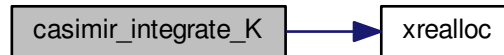
Parameters

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

Return values

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

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

$$\int_0^\infty dx x^{\ell_1+\ell_2} e^{-x} r_{\text{TE}}$$

where

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

and $\beta = 2\omega_P(L + R)/c$.

- 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 $\alpha = 2\xi(L + R)/c$.

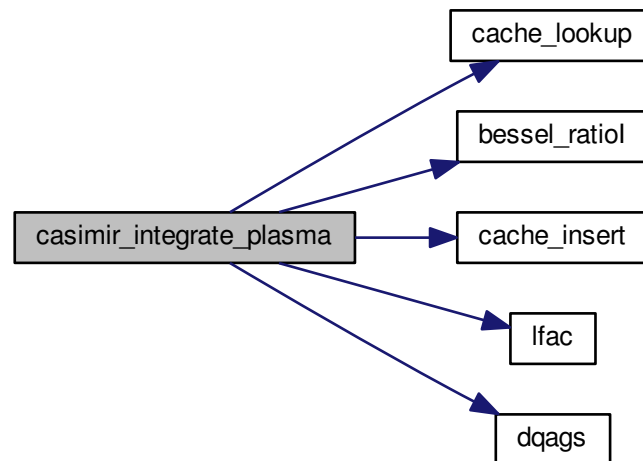
Parameters

in	<i>self</i>	plasma integration object
in	<i>l1</i>	ℓ_1
in	<i>l2</i>	ℓ_2
in	<i>m</i>	m
out	<i>ratio1</i>	
out	<i>ratio2</i>	

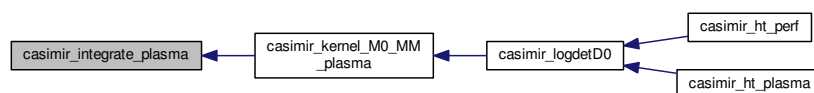
Return values

/	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()

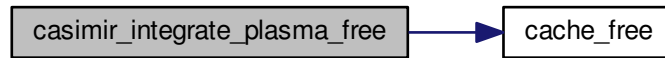
```
void casimir_integrate_plasma_free (
    integration_plasma_t * self )
```

Free plasma integration object.

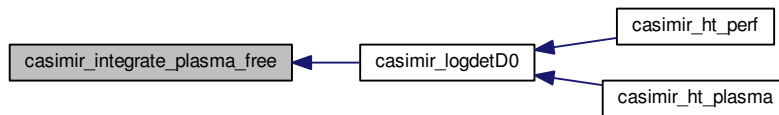
Parameters

in, out	<i>self</i>	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 ($\xi = 0$)

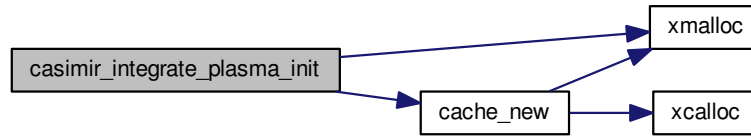
Parameters

in	<i>casimir</i>	Casimir object
in	<i>omegap</i>	plasma frequency in rad/s
in	<i>epsrel</i>	relative error for integration

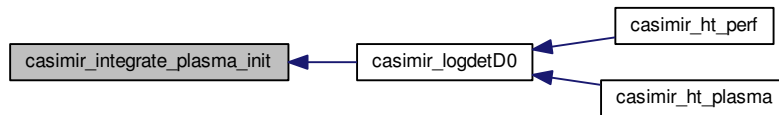
Return values

<i>self</i>	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 $m > 0$

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

and for $m = 0$

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

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 $f''(x_{\max})$, we estimate the width of the peak and the value of the integral using Laplace's method:

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

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

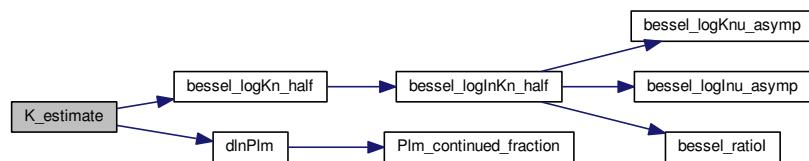
$$e^{-f(a)} \approx e^{-f(b)} \approx \epsilon e^{-f(x_{\max})}.$$

However, a cannot be smaller than 1.

Parameters

in	<i>nu</i>	parameter ν
in	<i>m</i>	parameter m
in	<i>alpha</i>	α
in	<i>eps</i>	ϵ
out	<i>a</i>	left border
out	<i>b</i>	right border
out	<i>approx</i>	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

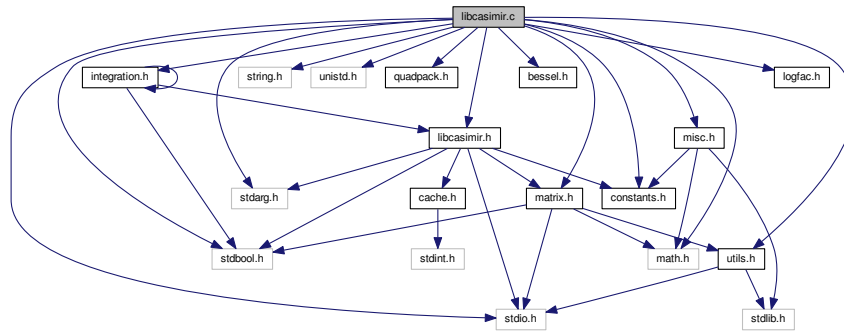
```

#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 ℓ_{\min} and ℓ_{\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 determinant.
- `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

<i>stream</i>	output stream
<i>prefix</i>	prefix of each line or NULL

5.10.2.2 casimir_epsilon_m1_drude()

```
double casimir_epsilon_m1_drude (
    double xi,
    void * userdata )
```

Dielectric function for Drude reflectors.

Dielectric function for Drude

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

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	<i>xi</i>	frequency in rad/s
in	<i>userdata</i>	userdata

Return values

<i>epsilon</i>	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.

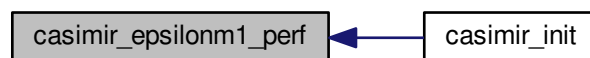
Parameters

in	<i>xi_</i>	ignored
in	<i>userdata</i>	ignored

Return values

<i>inf</i>	$\epsilon(\xi) = \infty$
------------	--------------------------

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	<i>self</i>	Casimir object
in	$\xi \leftarrow$	$\xi \mathcal{L}/c$
	—	

Return values

<i>epsm1</i>	$\epsilon(i\xi)$
--------------	------------------

Here is the caller graph for this function:

5.10.2.5 `casimir_epsilon_m1_sphere()`

```
double casimir_epsilon_m1_sphere (
    casimir_t * self,
    double xi_ )
```

Evaluate dielectric function of the sphere.

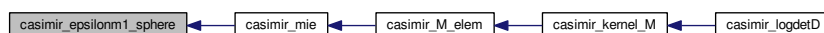
Parameters

in	<i>self</i>	Casimir object
in	$\xi \leftarrow$	$\xi \mathcal{L}/c$
	—	

Return values

<i>epsm1</i>	$\epsilon(i\xi)$
--------------	------------------

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 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 ℓ_{\min}, ℓ_{\max} .

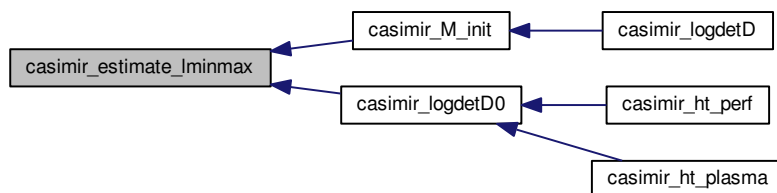
Parameters

in	<i>self</i>	Casimir object
in	<i>m</i>	quantum number
out	<i>lmin</i> _↔ <i>_p</i>	minimum value of ℓ
out	<i>lmax</i> _↔ <i>_p</i>	maximum value of ℓ

Return values

<i>l</i>	approximately the value of ℓ where $\mathcal{M}_{\ell\ell}^m$ 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	<i>self</i>	Casimir object
---------	-------------	----------------

5.10.2.8 casimir_fresnel()

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

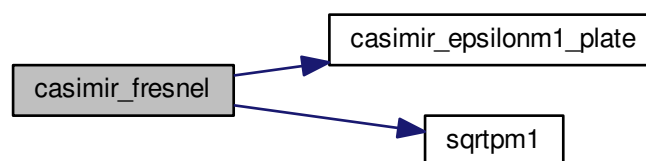
Calculate Fresnel coefficients r_{TE} and r_{TM} for arbitrary metals.

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

Parameters

in	<i>self</i>	Casimir object
in	ξ	$\xi\mathcal{L}/c$
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

<i>in</i>	<i>self</i>	Casimir object
-----------	-------------	----------------

Return values

<i>deta_{lg}</i>	
--------------------------	--

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

<i>epsrel</i>	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

<i>in, out</i>	<i>self</i>	Casimir object
----------------	-------------	----------------

Return values

<i>ldim</i>	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 $r_{\text{TM}} = 1$, $r_{\text{TE}} = 0$ for $\xi \rightarrow 0$, 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>

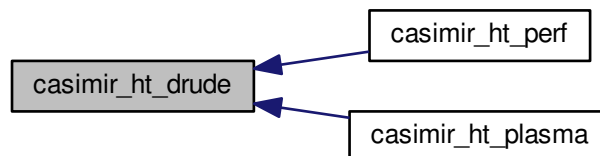
Parameters

in	<i>casimir</i>	Casimir object
----	----------------	----------------

Return values

<i>F</i>	free energy in units of $k_{\text{B}}T$
----------	---

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 $r_{\text{TM}} = 1$, $r_{\text{TE}} = -1$ in the limit $\xi \rightarrow 0$, 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>

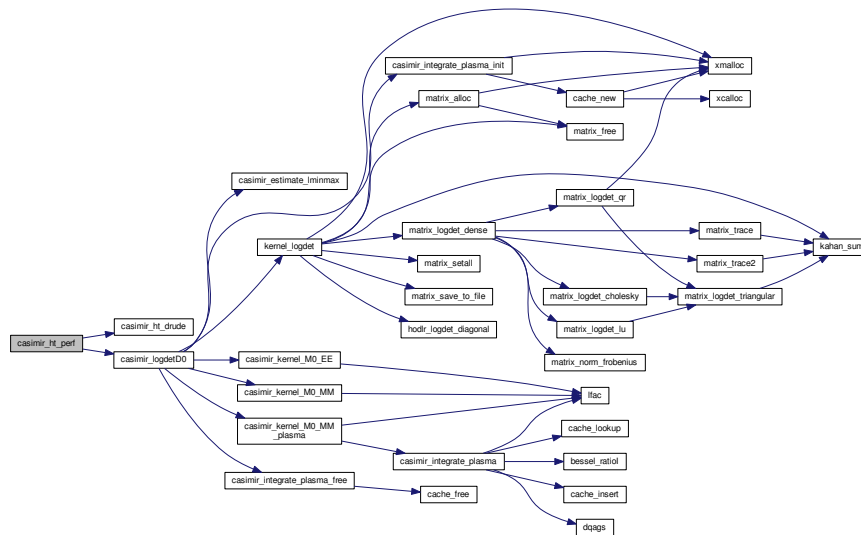
Parameters

in	<i>casimir</i>	Casimir object
in	<i>eps</i>	ϵ abort criterion

Return values

<i>energy</i>	free energy in units of $k_B T$
---------------	---------------------------------

Here is the call graph for this function:



5.10.2.14 casimir_ht_plasma()

```
double casimir_ht_plasma (
    casimir_t * casimir,
```


Parameters

<i>self</i>	Casimir object
<i>stream</i>	where to print the string
<i>prefix</i>	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:

$$\ell_{\text{dim}} = \text{ceil} \left(\max \left(\text{CASIMIR_MINIMUM_LDIM}, \text{CASIMIR_FACTOR_LDIM} \cdot \frac{R}{L} \right) \right)$$

Restrictions: $L/R > 0$

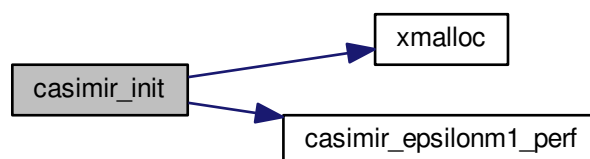
Parameters

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

Return values

<i>object</i>	Casimir object if successful
<i>NULL</i>	if an error occurred

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 $2\ell_{\text{dim}} \times 2\ell_{\text{dim}}$ matrix, the matrix elements start at 0, i.e. $0 \leq i, j < 2\ell_{\text{dim}}$.

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](#).

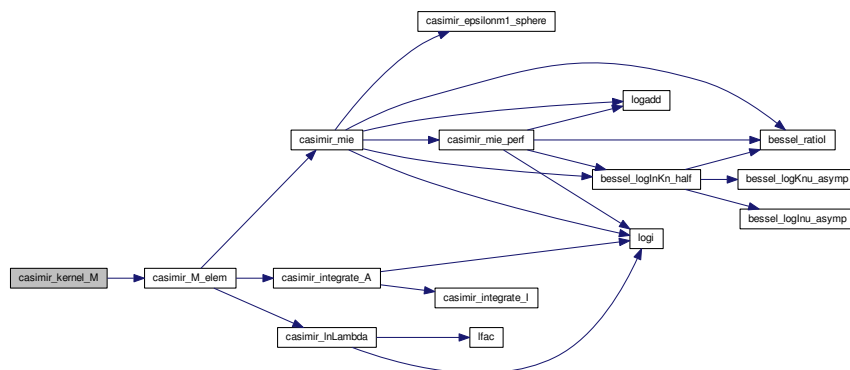
Parameters

in	<i>i</i>	row
in	<i>j</i>	column
in	<i>args_</i> ↔	casimir_M_t object, see casimir_M_init
	—	

Return values

<i>M_{ij}</i>	$\mathcal{M}_{ij}^{(m)}(\xi)$
-----------------------	-------------------------------

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 $p_1 = p_2 = E$.

See also [casimir_logdetD0](#).

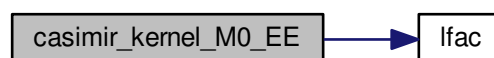
Parameters

in	<i>i</i>	row (starting from 0)
in	<i>j</i>	column (starting from 0)
in	<i>args_</i> ↔	pointer to casimir_M_t object
	—	

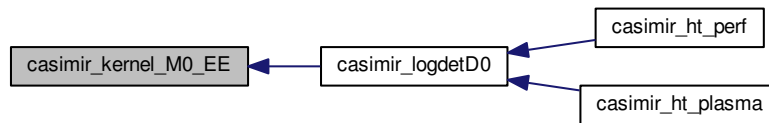
Return values

<i>M_{ij}</i>	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 $p_1 = p_2 = \text{M}$.

See also [casimir_logdetD0](#).

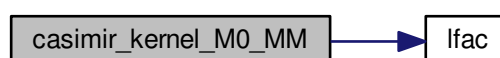
Parameters

in	<i>i</i>	row (starting from 0)
in	<i>j</i>	column (starting from 0)
in	<i>args_</i> ↔	pointer to casimir_M_t object
	—	

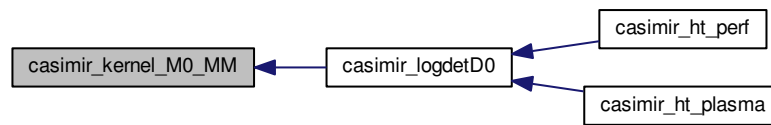
Return values

<i>Mij</i>	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 $p_1 = p_2 = \text{M}$ (plasma model).

See also [casimir_logdetD0](#).

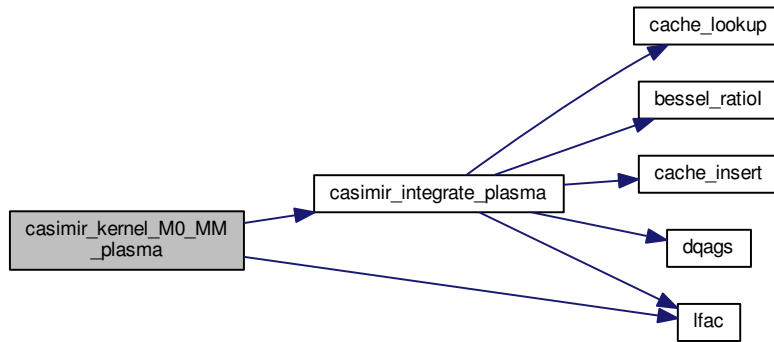
Parameters

in	<i>i</i>	row (starting from 0)
in	<i>j</i>	column (starting from 0)
in	<i>args</i> \leftrightarrow	pointer to casimir_M_t object
	—	

Return values

<i>M_{ij}</i>	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 $\Lambda_{\ell_1 \ell_2}^{(m)}$.

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

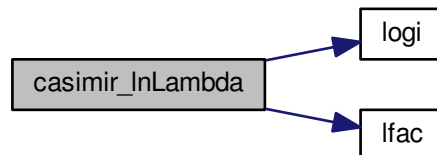
Parameters

in	<i>l1</i>	<i>l1</i> >0
in	<i>l2</i>	<i>l2</i> >0
in	<i>m</i>	<i>m</i> <= <i>l1</i> and <i>m</i> <= <i>l2</i>

Return values

<i>InLambda</i>	$\log \Lambda_{\ell_1, \ell_2}^{(m)}$
-----------------	---------------------------------------

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

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

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.

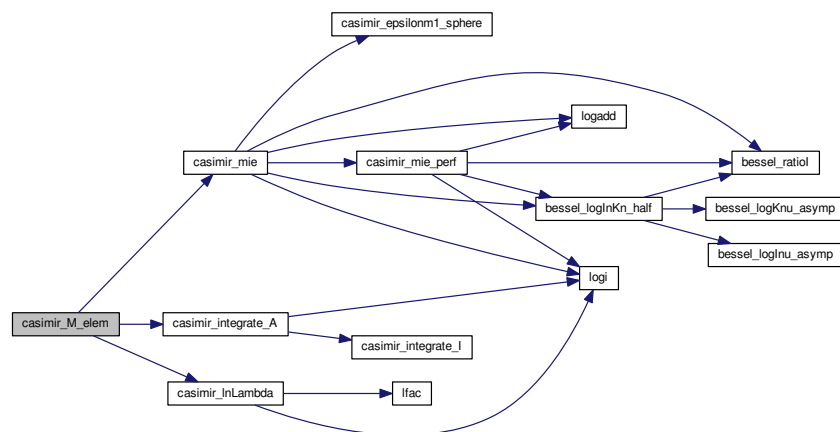
Parameters

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

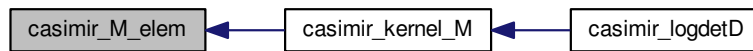
Return values

<i>elem</i>	$\mathcal{M}_{\ell_1, \ell_2}^{(m)}(p_1, p_2)$
-------------	--

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 )
```

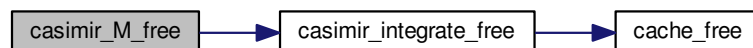
Free `casimir_M_t` object.

Frees memory allocated by `casimir_M_init`.

Parameters

<code>in, out</code>	<code>self</code>	<code>casimir_M_t</code> 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.

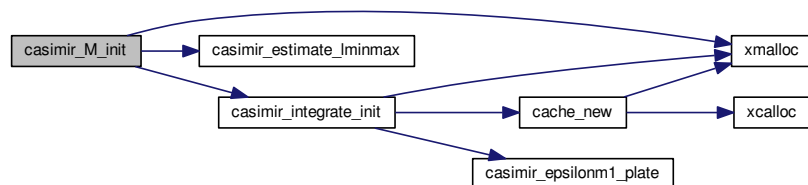
Parameters

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

Return values

<i>obj</i>	<code>casimir_M_t</code> object that can be given to <code>casimir_kernel_M</code>
------------	--

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 $\omega_P = \infty$ 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 and 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 $\text{sgn}(a_\ell) = (-1)^\ell$, $\text{sgn}(b_\ell) = (-1)^{\ell+1}$.

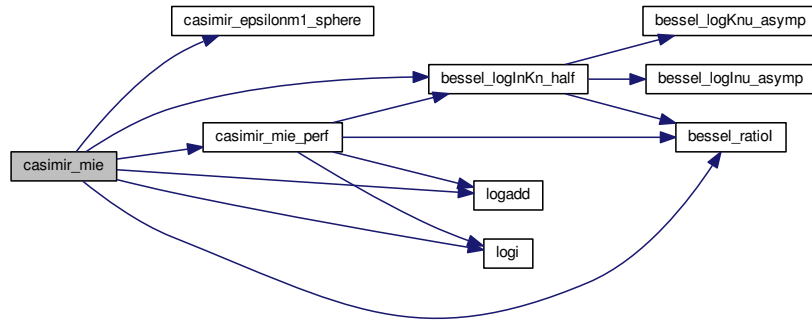
References:

- [1] Erratum: Thermal Casimir effect for Drude metals in the plane-sphere geometry, Canaguier-Durand, Neto, Lambrecht, Reynaud (2010) <http://journals.aps.org/pr/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/pr/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	$\xi \leftrightarrow -$	$\xi \mathcal{L}/c$
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 $\chi = \xi R/c$.

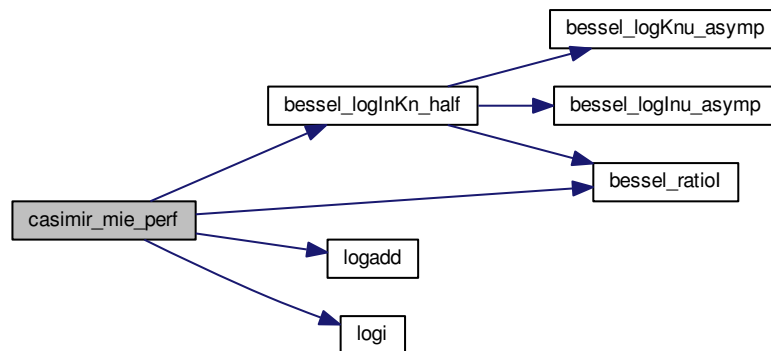
The signs are given by $\text{sgn}(a_\ell) = (-1)^\ell$, $\text{sgn}(b_\ell) = (-1)^{\ell+1}$.

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

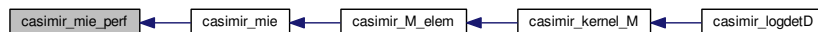
Parameters

in, out	self	Casimir object
in	$\xi \mapsto xi$	$\xi \mathcal{L}/c > 0$
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:



5.10.2.29 casimir_set_detalg()

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

Set algorithm to calculate determinant.

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

Parameters

in, out	<i>self</i>	Casimir object
in	<i>detalg</i>	algorithm to compute determinant

Return values

<i>success</i>	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 )
```

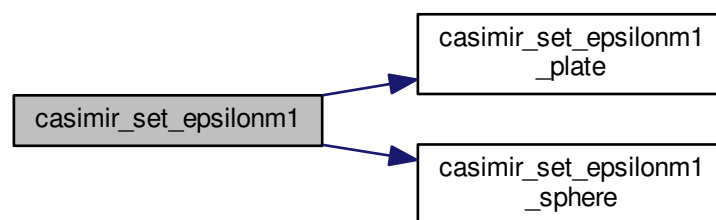
Set dielectric function for plate and sphere.

See also [casimir_set_epsilonm1_plate](#) and [casimir_set_epsilonm1_sphere](#).

Parameters

in, out	<i>self</i>	Casimir object
in	<i>epsilonm1</i>	callback to the function that calculates $\epsilon(i\xi) - 1$
in	<i>userdata</i>	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.

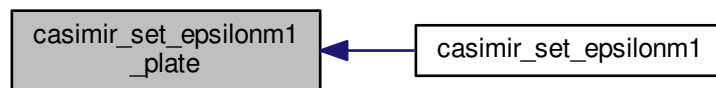
The Fresnel coefficient r_p depend on the dielectric function $\epsilon(i\xi)$. By default, perfect reflectors with a dielectric function $\epsilon(i\xi) = \infty$ are used.

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

Parameters

in, out	<i>self</i>	Casimir object
in	<i>epsilon_{m1}</i>	callback to the function that calculates $\epsilon(i\xi) - 1$
in	<i>userdata</i>	arbitrary pointer to data that is passed to <i>epsilon_{m1}</i> 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.

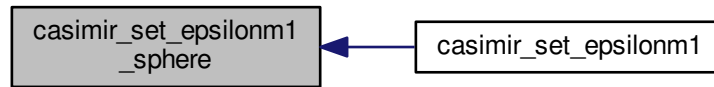
The Mie coefficient a_ℓ, b_ℓ depend on the dielectric function $\epsilon(i\xi)$. By default, perfect reflectors with a dielectric function $\epsilon(i\xi) = \infty$ are used.

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

Parameters

in, out	<i>self</i>	Casimir object
in	<i>epsilon_{m1}</i>	callback to the function that calculates $\epsilon(i\xi) - 1$
in	<i>userdata</i>	arbitrary pointer to data that is passed to <i>epsilon_{m1}</i> 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	<i>self</i>	Casimir object
in	<i>epsrel</i>	relative error

Return values

0	if an error occurred
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

<code>in, out</code>	<i>self</i>	Casimir object
<code>in</code>	<i>ldim</i>	dimension in angular momentum ℓ

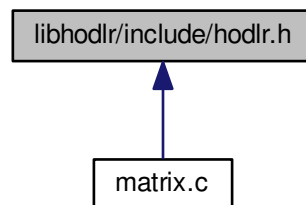
Return values

<code>1</code>	if successful
<code>0</code>	if <code>ldim < 1</code>

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 $\log(\det(I-M))$ 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 $\log(\det(\text{Id}-M))$ using HODLR approach.

See [hodlr_logdet_diagonal](#) for more information.

Parameters

<i>dim</i>	dimension of matrix M
<i>callback</i>	function that returns matrix elements of M
<i>args</i>	pointer that is passed as third argument to callback
<i>nLeaf</i>	nLeaf is the dimension of the smallest block at the leaf level
<i>tolerance</i>	requested accuracy of result
<i>sym_spd</i>	specifiy whether matrix is generic (0), symmetric (1) or spd (2)

Return values

<i>logdet</i>	$\log \det(1 - M)$
---------------	--------------------

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.

Compute $\log \det(1-A)$ for a matrix A of dimension given by dim. The diagonal elements of A are given by diagonal which is an array of dim elements. Arbitrary matrix elements A_{ij} are given by the `callback(i,j,args)`. The requested numerical precision is given by tolerance. returns the matrix elements of A.

nLeaf is the size (number of rows of the matrix) of the smallest block at the leaf level. The number of levels in the tree is given by $n_levels = \log_2(\text{dim}/n\text{Leaf})$.

Values for sym_psd:

- 0: generic matrix
- 1: matrix is symmetric
- 2: matrix is symmetric and positive definite

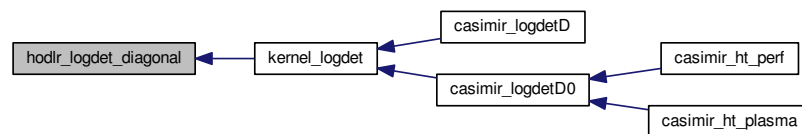
Parameters

<i>dim</i>	dimension of matrix M
<i>callback</i>	function that returns matrix elements of M
<i>args</i>	pointer that is passed as third argument to callback
<i>diagonal</i>	array with the diagonal elements of M
<i>nLeaf</i>	nLeaf is the dimension of the smallest block at the leaf level
<i>tolerance</i>	requested accuracy of result
<i>sym_spd</i>	specifiy whether matrix is generic (0), symmetric (1) or spd (2)

Return values

<i>logdet</i>	$\log \det(1 - M)$
---------------	--------------------

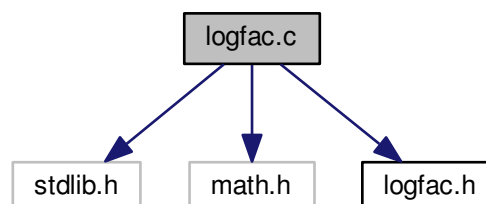
Here is the caller graph for this function:



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 $n \leq 1024$

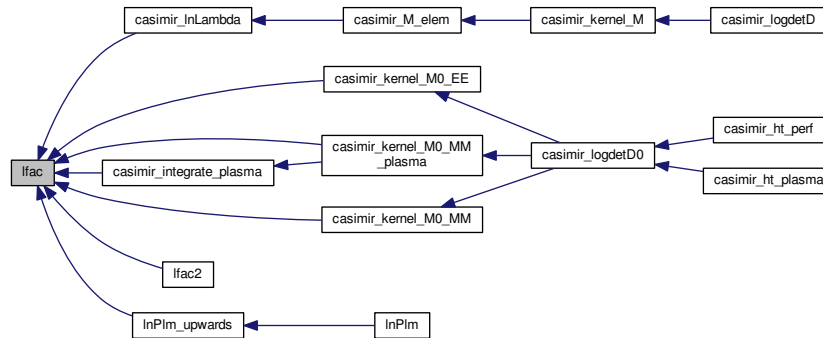
Parameters

in	n	integer
----	-----	---------

Return values

<i>lfac</i>	$\log(n!)$
-------------	------------

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	<i>n</i>	argument
----	----------	----------

Return values

<i>lfac2</i>	$n!!$
--------------	-------

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 $n \leq 65536$

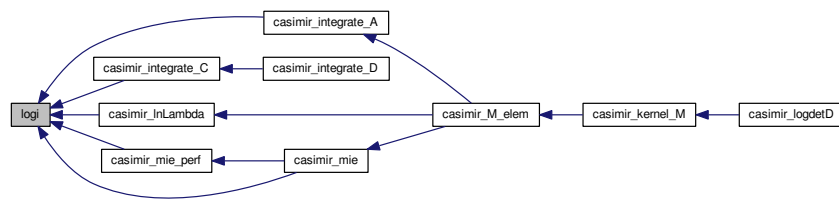
Parameters

in	n	integer
----	-----	---------

Return values

$\log n$	$\log(n)$
----------	-----------

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]
```

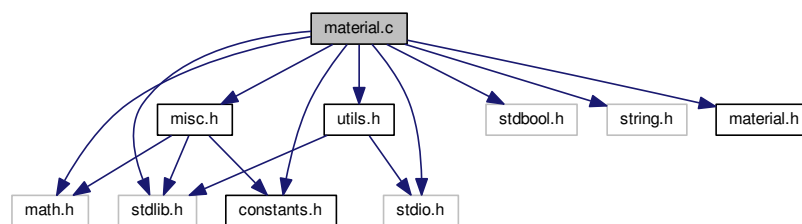
lookup table for $\log(n)$, see [logi](#)

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	<i>line</i>	string to parse
in	<i>key</i>	key
in	<i>separator</i>	separator
out	<i>value</i>	numerical value of the string "value"

Return values

<i>true</i>	parsing successful
<i>false</i>	parsing not successful

5.13.2.2 `material_epsilonm1()`

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

Dielectric function for material.

Return the dielectric function $\epsilon(i\xi) - 1$ 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	<i>xi</i>	frequency in rad/s
in	<i>args</i>	material (must be of type <code>material_t *</code>)

5.13.2.3 material_free()

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

Free material object.

Parameters

<i>material</i>	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:

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

The parameters for the plasma frequency ω_p and the relaxation frequency γ for $\xi > \xi_{\max}$ and $\xi < \xi_{\min}$ 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	<i>material</i>	material object
out	<i>omegap_low</i>	plasma frequency for high-frequency extrapolation (in rad/s)
out	<i>gamma_low</i>	relaxation frequency for high-frequency extrapolation (in rad/s)
out	<i>omegap_high</i>	plasma frequency for low-frequency extrapolation (in rad/s)
out	<i>gamma_high</i>	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	<i>material</i>	material object
in	<i>stream</i>	output stream (e.g. stdout)
in	<i>prefix</i>	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	<i>filename</i>	path to material specification
in	<i>calL</i>	$L + R$, separation between plane and center of sphere

Return values

<i>material</i>	if successful
<i>NULL</i>	if file cannot be read or is in wrong format

5.14 matrix.c File Reference

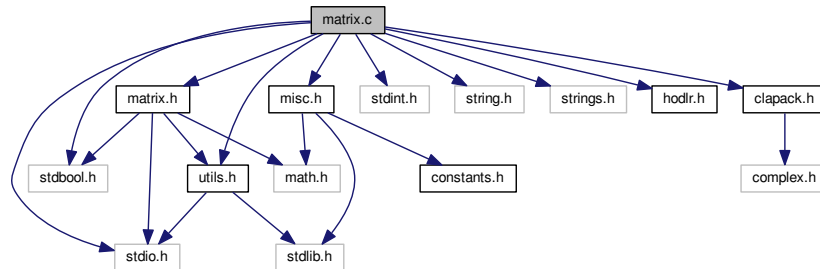
Matrix functions.

```
#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 sym_spd, 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 sym_spd,
    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 (np). This option might be useful for debugging. Also note that if detalg is CHOLESKY, only the upper half of the matrix will be initialized.

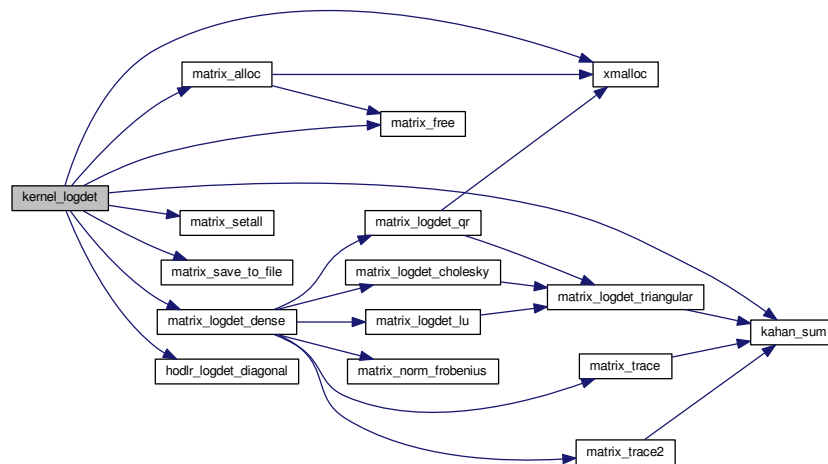
Parameters

in	<i>dim</i>	dimension of matrix
in	<i>kernel</i>	callback function that returns matrix elements of A
in	<i>args</i>	pointer given to callback function kernel
in	<i>sym_spd</i>	matrix is generic (0), symmetric (1), symmetric positive definite (2)
in	<i>detalg</i>	algorithm (DETALG_HODLR, DETALG_LU, DETALG_QR, DETALG_CHOLESKY)

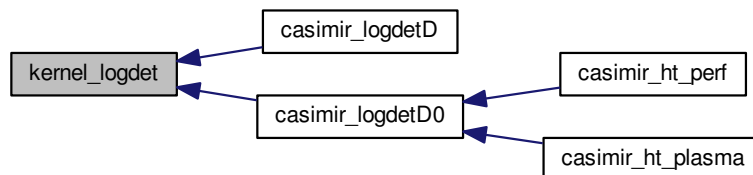
Return values

<i>logdet</i>	$\log \det(1 - A)$
---------------	--------------------

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.

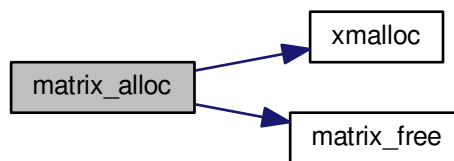
Parameters

in	<i>dim</i>	dimension of square matrix
----	------------	----------------------------

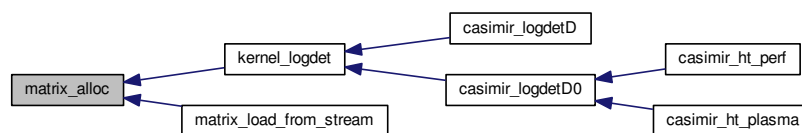
Return values

<i>A</i>	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 )
```

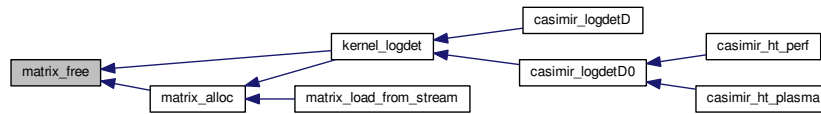
Free matrix.

This function frees the memory allocated for the matrix A.

Parameters

in, out	<i>A</i>	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	<i>filename</i>	filename of output file
----	-----------------	-------------------------

Return values

<i>A</i>	matrix if successful
<i>NULL</i>	if an error occurred

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 occurs. Do not use this function on untrusted data.

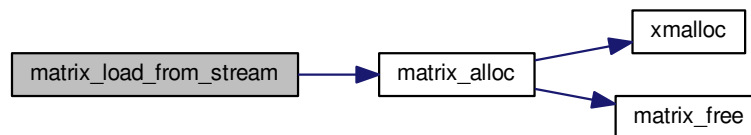
Parameters

in	<i>stream</i>	stream
----	---------------	--------

Return values

<i>A</i>	matrix if successful
<i>NULL</i>	if an error occurred

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.

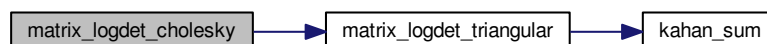
Parameters

in, out	<i>A</i>	matrix
in	<i>uplo</i>	L or U

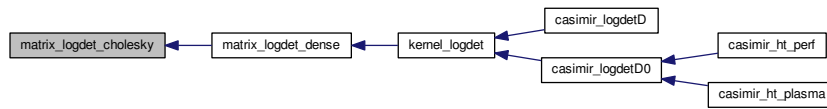
Return values

<i>logdet</i>	$\log \det A$
---------------	---------------

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, DETALG_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 $\mathcal{O}(N^2)$.

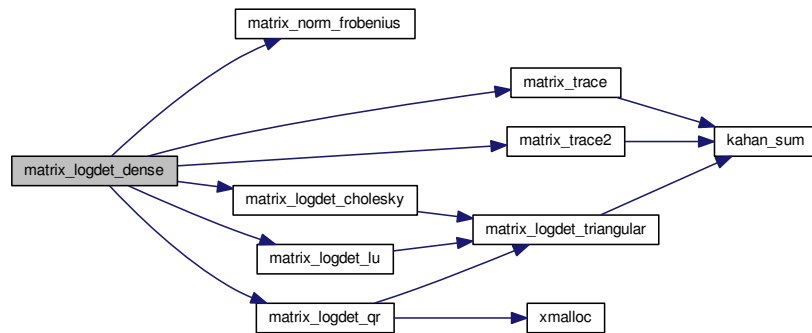
Parameters

in, out	A	matrix; will be overwritten.
in	z	factor z
in	<i>detalg</i>	algorithm to use (cholesky, lu or qr)

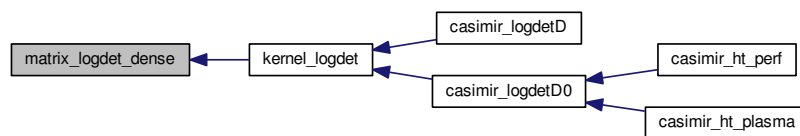
Return values

<i>logdet</i>	$\log \det(1 + zA)$
---------------	---------------------

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 $\log \det A$.

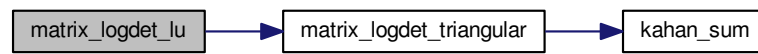
Parameters

in, out	A	matrix
---------	-----	--------

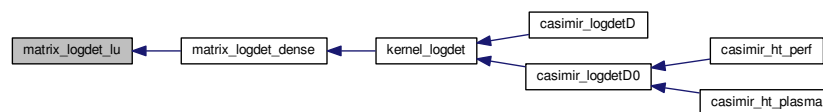
Return values

<i>logdet</i>	$\log \det A$
---------------	---------------

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 $\log \det A$.

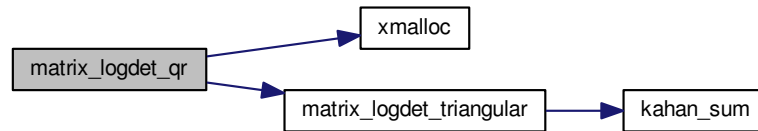
Parameters

<code>in, out</code>	<code>A</code>	matrix
----------------------	----------------	--------

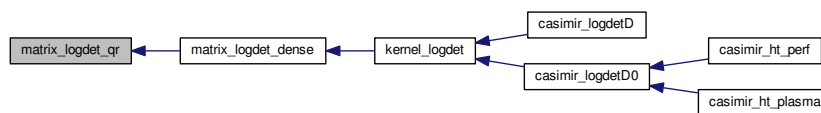
Return values

<code>logdet</code>	$\log \det A$
---------------------	---------------

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:

$$\log \det A = \log \prod_j A_{jj} = \sum_j \log A_{jj}$$

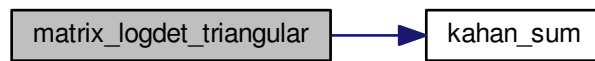
Parameters

in	A	triangular matrix
----	-----	-------------------

Return values

<i>logdet</i>	$\log \det A$
---------------	---------------

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 .

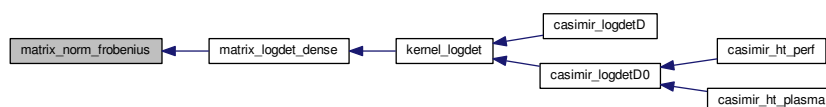
Parameters

in	A	matrix
----	-----	--------

Return values

$\left \begin{smallmatrix} \leftarrow \\ A \end{smallmatrix} \right $	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.

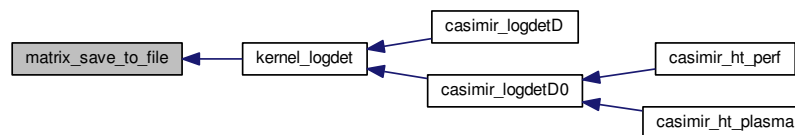
Parameters

in	A	matrix
in	<code>filename</code>	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 .npz format.

Parameters

in	A	matrix
in	<code>stream</code>	stream

Return values

0	
---	--

5.14.2.14 matrix_setall()

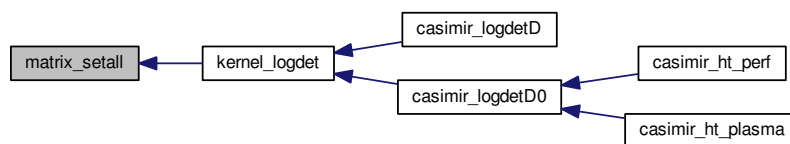
```
void matrix_setall (
    matrix_t * A,
    double z )
```

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 summation (see [kahan_sum](#)) to reduce rounding errors.

Parameters

in	A	matrix
----	-----	--------

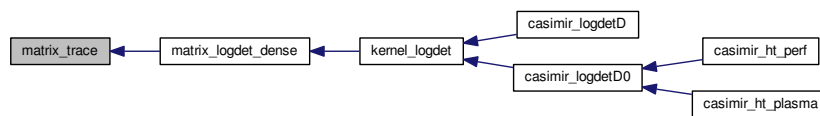
Return values

<i>trace</i>	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 summation (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
----	-----	--------

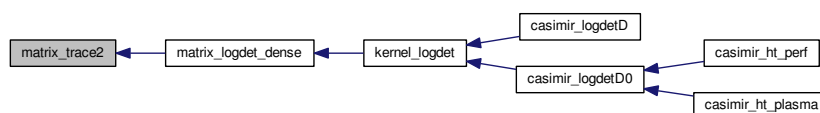
Return values

<i>trace</i>	$\text{tr}(A^2)$
--------------	------------------

Here is the call graph for this function:



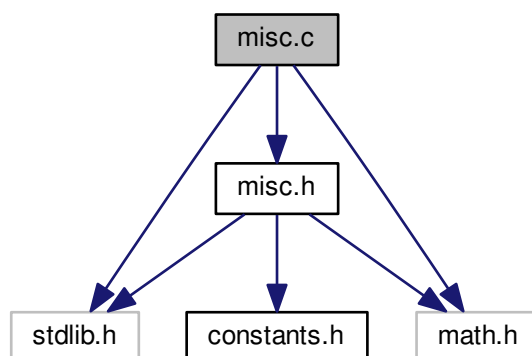
Here is the caller graph for this function:



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.

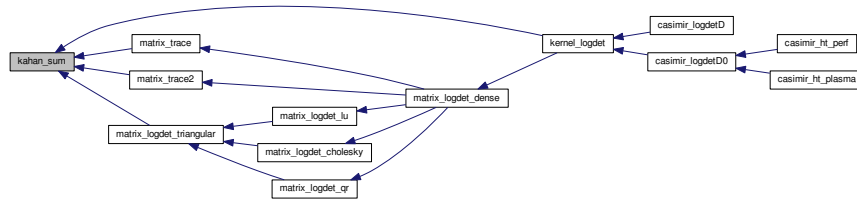
Parameters

in	<i>input</i>	array
in	<i>N</i>	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	$\log \leftrightarrow$ _a	number
in	$\log \leftrightarrow$ _b	number

Returns

log_sum $\log [\exp(\log_a) + \exp(\log_b)]$

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	<i>list</i>	list of numbers given by logarithm and sign
in	<i>N</i>	number of elements of list
out	<i>sign</i>	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.

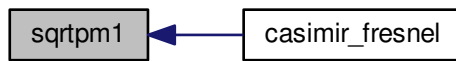
Parameters

in	<i>x</i>	
----	----------	--

Return values

$\text{sqrt}(1+x)-1$	
----------------------	--

Here is the caller graph for this function:

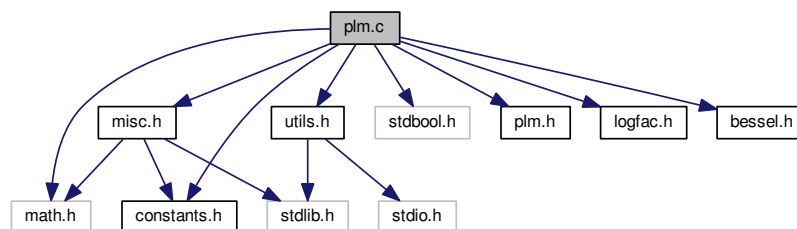


5.16 plm.c File Reference

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 `_P1` (int l, double x, double sinhi)
Compute Legendre polynomial $\log P_l(x)$ for large x .
- static double `_fn` (int n, double hn[13])
- static double `_P12` (int l, double x)
Compute Legendre polynomial $\log P_l(x)$ for small x .
- static double `_P13` (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

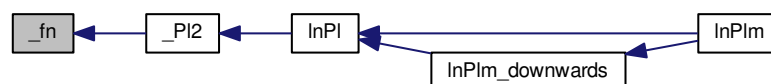
January, 2019

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](#).

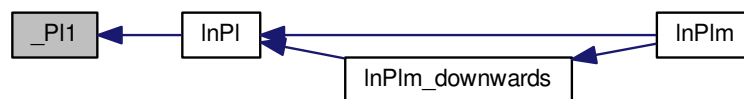
Parameters

in	l	degree
in	x	argument
in	\sinhxi	$\sinh \xi = \sqrt{(x+1)(x-1)}$

Return values

$\log \leftarrow$ 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 .

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](#).

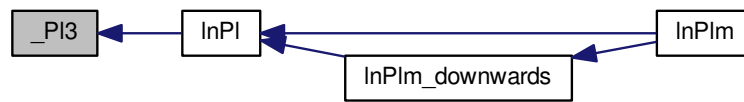
Parameters

in	<i>l</i>	order
in	<i>x</i>	argument

Return values

$\log \leftarrow$ <i>Pl</i>	$\log P_l(x)$
--------------------------------	---------------

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 $\frac{d}{dx} \log P_l^m(x)$ and $\frac{d^2}{dx^2} \log P_l^m(x)$.

If d2lnPlm is NULL, the 2nd logarithmic derivative will not be computed.

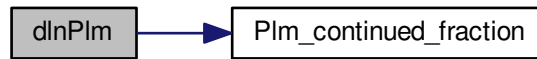
Parameters

in	<i>l</i>	degree
in	<i>m</i>	order
in	<i>x</i>	argument
out	<i>d2lnPlm</i>	2nd logarithmic derivative of $P_l^m(x)$

Return values

<code>dlnPlm</code>	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 [_PI3](#)), otherwise asymptotic expansions are used (see [_PI1](#) and [_PI2](#)).

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)

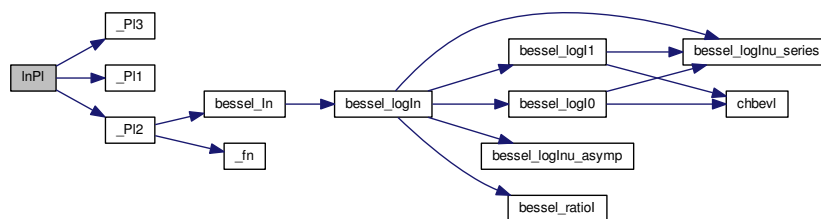
Parameters

in	<i>l</i>	degree
in	<i>x</i>	argument

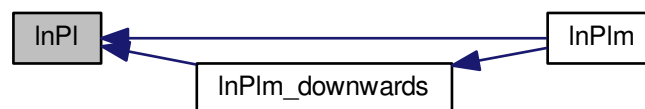
Return values

$\log \leftarrow \begin{smallmatrix} PI \\ \end{smallmatrix}$	$\log P_l(x)$
---	---------------

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 $x > 1$.

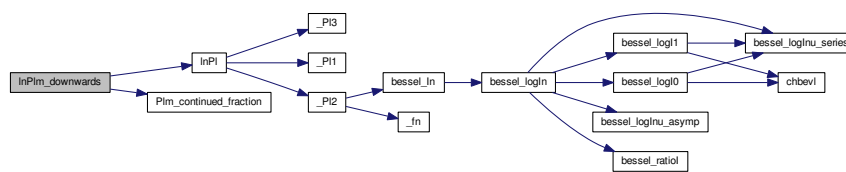
Parameters

in	<i>l</i>	degree
in	<i>m</i>	order
in	<i>x</i>	argument

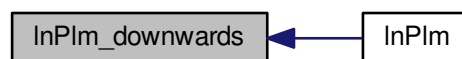
Return values

<i>logPlm</i>	$\log P_l^m(x)$
---------------	-----------------

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>).

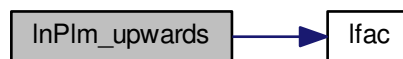
Parameters

in	l	degree
in	m	order
in	x	argument

Return values

$\log Plm$	$\log P_l^m(x)$
------------	-----------------

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.

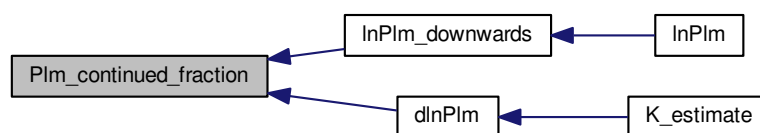
Parameters

in	l	degree
in	m	order
in	x	argument

Return values

<i>ratio</i>	$P_l^{m-1}(x)/P_l^m(x)$
--------------	-------------------------

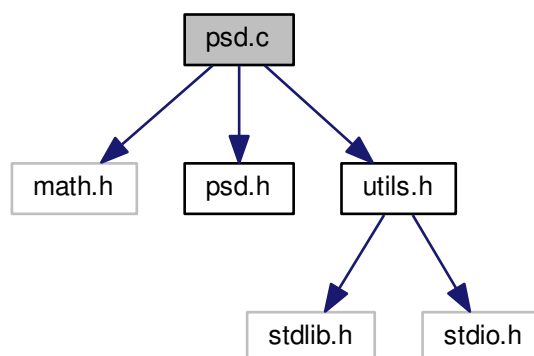
Here is the caller graph for this function:



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

η_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)

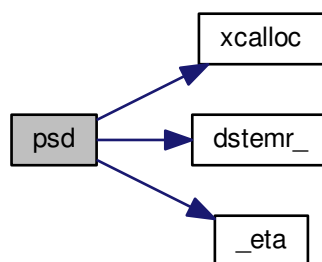
Parameters

in	N	order
out	ξ_i	poles
out	η_i	expansion coefficients

Return values

<i>success</i>	0 if successful
----------------	-----------------

Here is the call graph for this function:



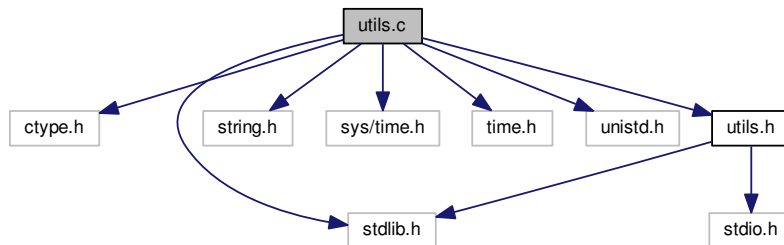
5.18 utils.c File Reference

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 [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.

5.18.1 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 μ s precision.

Return values

<i>time</i>	seconds since 1st Jan 1970
-------------	----------------------------

5.18.2.3 `strim()`

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

Remove whitespace at beginning 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

<i>str</i>	string
------------	--------

5.18.2.4 `strrep()`

```
void strrep (
    char * s,
```

```
const char a,  
const char b )
```

Replace character by different character in string.

Replace occurrence of *a* by *b* in the string *s*.

Parameters

<i>in, out</i>	<i>s</i>	string, terminated by <code>\0</code>
<i>in</i>	<i>a</i>	character to replace
<i>in</i>	<i>b</i>	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

<i>s</i>	string
<i>len</i>	maximum length of array <i>s</i>

5.18.2.6 `xalloc()`

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

Wrapper for `calloc`.

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

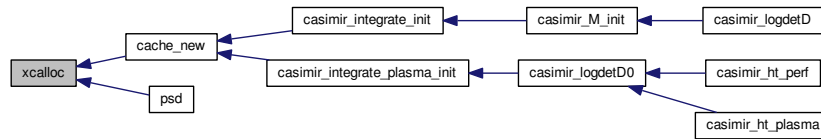
Parameters

<i>nmemb</i>	number of elements
<i>size</i>	size of each element

Return values

<i>ptr</i>	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.

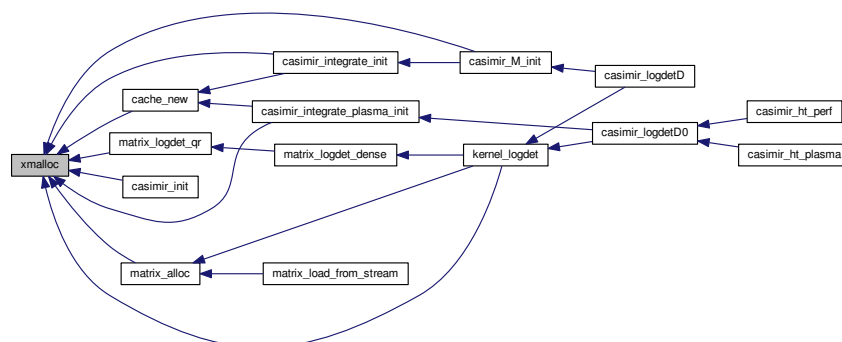
Parameters

<i>size</i>	size of bytes to allocate
-------------	---------------------------

Return values

<i>ptr</i>	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

<i>p</i>	ptr to old memory
<i>size</i>	size

Return values

<i>newptr</i>	pointer to new memory
---------------	-----------------------

Here is the caller graph for this function:



Index

- [_PI1](#)
 - [plm.c, 174](#)
 - [_PI2](#)
 - [plm.c, 175](#)
 - [_PI3](#)
 - [plm.c, 176](#)
 - [_eta](#)
 - [psd.c, 184](#)
 - [_fn](#)
 - [plm.c, 174](#)
 - [_parse](#)
 - [material.c, 152](#)
- [argparse, 7](#)
- [argparse_option, 8](#)
- [bessel.c, 21](#)
 - [bessel_I0, 23](#)
 - [bessel_I1, 24](#)
 - [bessel_In, 25](#)
 - [bessel_K0, 26](#)
 - [bessel_K1, 26](#)
 - [bessel_Kn, 27](#)
 - [bessel_logI0, 28](#)
 - [bessel_logI1, 29](#)
 - [bessel_logIn, 30](#)
 - [bessel_logIn_half, 31](#)
 - [bessel_logInKn_half, 32](#)
 - [bessel_logInu_asymp, 33](#)
 - [bessel_logInu_series, 34](#)
 - [bessel_logK0, 34](#)
 - [bessel_logK1, 36](#)
 - [bessel_logKn, 37](#)
 - [bessel_logKn_half, 38](#)
 - [bessel_logKn_recursive, 39](#)
 - [bessel_logKnu_asymp, 40](#)
 - [bessel_ratiol, 40](#)
 - [chbevl, 41](#)
 - [I0_coefs, 42](#)
 - [I1_coefs, 43](#)
 - [K0_coefsA, 43](#)
 - [K0_coefsB, 43](#)
 - [K1_coefsA, 44](#)
 - [K1_coefsB, 44](#)
- [bessel_I0](#)
 - [bessel.c, 23](#)
- [bessel_I1](#)
 - [bessel.c, 24](#)
- [bessel_In](#)
 - [bessel.c, 25](#)
- [bessel_K0](#)
 - [bessel.c, 26](#)
- [bessel_K1](#)
 - [bessel.c, 26](#)
- [bessel_Kn](#)
 - [bessel.c, 27](#)
- [bessel_logI0](#)
 - [bessel.c, 28](#)
- [bessel_logI1](#)
 - [bessel.c, 29](#)
- [bessel_logIn](#)
 - [bessel.c, 30](#)
- [bessel_logIn_half](#)
 - [bessel.c, 31](#)
- [bessel_logInKn_half](#)
 - [bessel.c, 32](#)
- [bessel_logInu_asymp](#)
 - [bessel.c, 33](#)
- [bessel_logInu_series](#)
 - [bessel.c, 34](#)
- [bessel_logK0](#)
 - [bessel.c, 34](#)
- [bessel_logK1](#)
 - [bessel.c, 36](#)
- [bessel_logKn](#)
 - [bessel.c, 37](#)
- [bessel_logKn_half](#)
 - [bessel.c, 38](#)
- [bessel_logKn_recursive](#)
 - [bessel.c, 39](#)
- [bessel_logKnu_asymp](#)
 - [bessel.c, 40](#)
- [bessel_ratiol](#)
 - [bessel.c, 40](#)

- [buf, 9](#)
- [buffer, 9](#)
- [capacity, 9](#)
- [size, 9](#)
- [buf.h](#)
- [buf_capacity, 60](#)
- [buf_clear, 60](#)
- [buf_free, 61](#)
- [buf_grow, 61](#)
- [buf_pop, 61](#)
- [buf_push, 61](#)
- [buf_size, 61](#)
- [buf_trunc, 62](#)
- [buf_capacity](#)
- [buf.h, 60](#)

- buf_clear
 - buf.h, 60
- buf_free
 - buf.h, 61
- buf_grow
 - buf.h, 61
- buf_pop
 - buf.h, 61
- buf_push
 - buf.h, 61
- buf_size
 - buf.h, 61
- buf_trunc
 - buf.h, 62
- buffer
 - buf, 9
- CASIMIR_CACHE_ELEMS
 - libcasimir.h, 68
- CASIMIR_EPSREL
 - libcasimir.h, 68
- CASIMIR_FACTOR_LDIM
 - libcasimir.h, 68
- CASIMIR_MINIMUM_LDIM
 - libcasimir.h, 68
- CASIMIR_c
 - constants.h, 63
- CASIMIR_hbar
 - constants.h, 63
- CASIMIR_hbar_eV
 - constants.h, 63
- CASIMIR_kB
 - constants.h, 63
- COMPILER
 - utils.h, 97
- cache.c, 45
 - cache_free, 46
 - cache_insert, 47
 - cache_lookup, 47
 - cache_new, 48
- cache_free
 - cache.c, 46
- cache_insert
 - cache.c, 47
- cache_lookup
 - cache.c, 47
- cache_new
 - cache.c, 48
- cache_t, 9
- call
 - casimir, 10
 - material_t, 17
- capacity
 - buf, 9
- casimir, 10
 - call, 10
 - detalg, 10
 - epsrel, 11
 - L, 11
 - LbyR, 11
 - ldim, 11
 - R, 11
 - y, 11
- casimir_M_elem
 - libcasimir.c, 135
 - libcasimir.h, 87
- casimir_M_free
 - libcasimir.c, 137
 - libcasimir.h, 88
- casimir_M_init
 - libcasimir.c, 137
 - libcasimir.h, 89
- casimir_M_t, 12
- casimir_build
 - libcasimir.c, 118
 - libcasimir.h, 69
- casimir_epsilonm1_drude
 - libcasimir.c, 118
 - libcasimir.h, 69
- casimir_epsilonm1_perf
 - libcasimir.c, 119
 - libcasimir.h, 70
- casimir_epsilonm1_plate
 - libcasimir.c, 119
 - libcasimir.h, 71
- casimir_epsilonm1_sphere
 - libcasimir.c, 120
 - libcasimir.h, 71
- casimir_estimate_lminmax
 - libcasimir.c, 120
 - libcasimir.h, 72
- casimir_free
 - libcasimir.c, 121
 - libcasimir.h, 73
- casimir_fresnel
 - libcasimir.c, 122
 - libcasimir.h, 73
- casimir_get_detalg
 - libcasimir.c, 122
 - libcasimir.h, 74
- casimir_get_epsrel
 - libcasimir.c, 123
 - libcasimir.h, 74
- casimir_get_ldim
 - libcasimir.c, 123
 - libcasimir.h, 74
- casimir_ht_drude
 - libcasimir.c, 123
 - libcasimir.h, 75
- casimir_ht_perf
 - libcasimir.c, 124
 - libcasimir.h, 76
- casimir_ht_plasma
 - libcasimir.c, 125
 - libcasimir.h, 77
- casimir_info
 - libcasimir.c, 126

- libcasimir.h, 78
- casimir_init
 - libcasimir.c, 127
 - libcasimir.h, 78
- casimir_integrate_A
 - integration.c, 103
- casimir_integrate_B
 - integration.c, 104
- casimir_integrate_C
 - integration.c, 105
- casimir_integrate_D
 - integration.c, 106
- casimir_integrate_free
 - integration.c, 107
- casimir_integrate_I
 - integration.c, 108
- casimir_integrate_init
 - integration.c, 109
- casimir_integrate_K
 - integration.c, 110
- casimir_integrate_plasma
 - integration.c, 111
- casimir_integrate_plasma_free
 - integration.c, 112
- casimir_integrate_plasma_init
 - integration.c, 113
- casimir_kernel_M0_EE
 - libcasimir.c, 129
 - libcasimir.h, 80
- casimir_kernel_M0_MM_plasma
 - libcasimir.c, 131
 - libcasimir.h, 82
- casimir_kernel_M0_MM
 - libcasimir.c, 130
 - libcasimir.h, 81
- casimir_kernel_M
 - libcasimir.c, 128
 - libcasimir.h, 79
- casimir_InLambda
 - libcasimir.c, 132
 - libcasimir.h, 83
- casimir_logdetD0
 - libcasimir.c, 134
 - libcasimir.h, 85
- casimir_logdetD
 - libcasimir.c, 133
 - libcasimir.h, 84
- casimir_mie
 - libcasimir.c, 138
 - libcasimir.h, 90
- casimir_mie_perf
 - libcasimir.c, 140
 - libcasimir.h, 91
- casimir_mpi_t, 13
- casimir_set_detalg
 - libcasimir.c, 141
 - libcasimir.h, 92
- casimir_set_epsilonm1
 - libcasimir.c, 142
 - libcasimir.h, 93
- casimir_set_epsilonm1_plate
 - libcasimir.c, 142
 - libcasimir.h, 93
- casimir_set_epsilonm1_sphere
 - libcasimir.c, 143
 - libcasimir.h, 94
- casimir_set_epsrel
 - libcasimir.c, 144
 - libcasimir.h, 95
- casimir_set_ldim
 - libcasimir.c, 144
 - libcasimir.h, 95
- casimir_t
 - libcasimir.h, 68
- casimir_task_t, 13
- chbevl
 - bessel.c, 41
- constants.h
 - CASIMIR_c, 63
 - CASIMIR_hbar, 63
 - CASIMIR_hbar_eV, 63
 - CASIMIR_kB, 63
 - M_LOG2, 63
 - M_LOGPI, 63
 - M_PI, 64
 - MAX, 64
 - MIN, 64
 - pow_2, 64
 - SGN, 64
 - sign_t, 64
- cot2
 - fcqs.c, 56
- cquadpack/include/quadpack.h, 49
- detalg
 - casimir, 10
- dim
 - matrix_t, 19
- dim2
 - matrix_t, 20
- disable_buffering
 - utils.c, 188
 - utils.h, 98
- dlnPlm
 - plm.c, 177
- dqage
 - quadpack.h, 51
- dqagi
 - quadpack.h, 52
- dqags
 - quadpack.h, 53
- dstemr_
 - psd.c, 185
- epsm1
 - material_t, 18
- epsrel

- casimir, 11
- fcqs.c, 54
 - cot2, 56
 - fcqs_finite, 56
 - fcqs_semiinf, 57
 - MMIN, 56
 - wi_finite, 58
 - wi_semiinf, 58
- fcqs_finite
 - fcqs.c, 56
- fcqs_semiinf
 - fcqs.c, 57
- filename
 - material_t, 18
- GK_10_21
 - quadpack.h, 50
- GK_15_31
 - quadpack.h, 50
- GK_20_41
 - quadpack.h, 50
- GK_25_51
 - quadpack.h, 50
- GK_30_61
 - quadpack.h, 50
- GK_7_15
 - quadpack.h, 51
- gamma_high
 - material_t, 18
- gamma_low
 - material_t, 18
- hodlr.h
 - hodlr_logdet, 146
 - hodlr_logdet_diagonal, 146
- hodlr_logdet
 - hodlr.h, 146
- hodlr_logdet_diagonal
 - hodlr.h, 146
- l0_coeffs
 - bessel.c, 42
- l1_coeffs
 - bessel.c, 43
- include/buf.h, 59
- include/constants.h, 62
- include/libcasimir.h, 65
- include/utls.h, 96
- integrand_plasma_t, 14
- integrand_t, 14
- integration.c, 102
 - casimir_integrate_A, 103
 - casimir_integrate_B, 104
 - casimir_integrate_C, 105
 - casimir_integrate_D, 106
 - casimir_integrate_free, 107
 - casimir_integrate_I, 108
 - casimir_integrate_init, 109
 - casimir_integrate_K, 110
 - casimir_integrate_plasma, 111
 - casimir_integrate_plasma_free, 112
 - casimir_integrate_plasma_init, 113
 - K_estimate, 114
- integration_plasma_t, 15
- integration_t, 15
- K0_coeffsA
 - bessel.c, 43
- K0_coeffsB
 - bessel.c, 43
- K1_coeffsA
 - bessel.c, 44
- K1_coeffsB
 - bessel.c, 44
- K_estimate
 - integration.c, 114
- kahan_sum
 - misc.c, 170
- kernel_args_t, 16
- kernel_logdet
 - matrix.c, 156
- L
 - casimir, 11
- LbyR
 - casimir, 11
- lda
 - matrix_t, 20
- ldim
 - casimir, 11
- lfac
 - logfac.c, 148
- lfac2
 - logfac.c, 149
- libcasimir.c, 115
 - casimir_M_elem, 135
 - casimir_M_free, 137
 - casimir_M_init, 137
 - casimir_build, 118
 - casimir_epsilonm1_drude, 118
 - casimir_epsilonm1_perf, 119
 - casimir_epsilonm1_plate, 119
 - casimir_epsilonm1_sphere, 120
 - casimir_estimate_lminmax, 120
 - casimir_free, 121
 - casimir_fresnel, 122
 - casimir_get_detalg, 122
 - casimir_get_epsrel, 123
 - casimir_get_ldim, 123
 - casimir_ht_drude, 123
 - casimir_ht_perf, 124
 - casimir_ht_plasma, 125
 - casimir_info, 126
 - casimir_init, 127
 - casimir_kernel_M0_EE, 129
 - casimir_kernel_M0_MM_plasma, 131
 - casimir_kernel_M0_MM, 130

- casimir_kernel_M, 128
- casimir_lnLambda, 132
- casimir_logdetD0, 134
- casimir_logdetD, 133
- casimir_mie, 138
- casimir_mie_perf, 140
- casimir_set_detalg, 141
- casimir_set_epsilonm1, 142
- casimir_set_epsilonm1_plate, 142
- casimir_set_epsilonm1_sphere, 143
- casimir_set_epsrel, 144
- casimir_set_ldim, 144
- libcasimir.h
 - CASIMIR_CACHE_ELEMS, 68
 - CASIMIR_EPSREL, 68
 - CASIMIR_FACTOR_LDIM, 68
 - CASIMIR_MINIMUM_LDIM, 68
 - casimir_M_elem, 87
 - casimir_M_free, 88
 - casimir_M_init, 89
 - casimir_build, 69
 - casimir_epsilonm1_drude, 69
 - casimir_epsilonm1_perf, 70
 - casimir_epsilonm1_plate, 71
 - casimir_epsilonm1_sphere, 71
 - casimir_estimate_lminmax, 72
 - casimir_free, 73
 - casimir_fresnel, 73
 - casimir_get_detalg, 74
 - casimir_get_epsrel, 74
 - casimir_get_ldim, 74
 - casimir_ht_drude, 75
 - casimir_ht_perf, 76
 - casimir_ht_plasma, 77
 - casimir_info, 78
 - casimir_init, 78
 - casimir_kernel_M0_EE, 80
 - casimir_kernel_M0_MM_plasma, 82
 - casimir_kernel_M0_MM, 81
 - casimir_kernel_M, 79
 - casimir_lnLambda, 83
 - casimir_logdetD0, 85
 - casimir_logdetD, 84
 - casimir_mie, 90
 - casimir_mie_perf, 91
 - casimir_set_detalg, 92
 - casimir_set_epsilonm1, 93
 - casimir_set_epsilonm1_plate, 93
 - casimir_set_epsilonm1_sphere, 94
 - casimir_set_epsrel, 95
 - casimir_set_ldim, 95
 - casimir_t, 68
 - polarization_t, 69
- libhodlr/include/hodlr.h, 145
- lnPl
 - plm.c, 178
- lnPlm
 - plm.c, 179
- lnPlm_downwards
 - plm.c, 180
- lnPlm_upwards
 - plm.c, 181
- log_t, 16
 - s, 17
 - v, 17
- logadd
 - misc.c, 171
- logadd_ms
 - misc.c, 171
- logfac.c, 147
 - lfac, 148
 - lfac2, 149
 - logi, 150
 - lookup_lfac, 150
 - lookup_logi, 150
- logi
 - logfac.c, 150
- lookup_lfac
 - logfac.c, 150
- lookup_logi
 - logfac.c, 150
- M
 - matrix_t, 20
- M_LOG2
 - constants.h, 63
- M_LOGPI
 - constants.h, 63
- M_PI
 - constants.h, 64
- MAX
 - constants.h, 64
- MIN
 - constants.h, 64
- MMIN
 - fcqs.c, 56
- material.c, 151
 - _parse, 152
 - material_epsilonm1, 152
 - material_free, 152
 - material_get_extrapolation, 153
 - material_info, 153
 - material_init, 154
- material_epsilonm1
 - material.c, 152
- material_free
 - material.c, 152
- material_get_extrapolation
 - material.c, 153
- material_info
 - material.c, 153
- material_init
 - material.c, 154
- material_t, 17
 - call, 17
 - epsm1, 18
 - filename, 18

- gamma_high, 18
- gamma_low, 18
- omegap_high, 18
- omegap_low, 18
- points, 18
- xi, 18
- xi_max, 19
- xi_min, 19
- matrix.c, 154
 - kernel_logdet, 156
 - matrix_alloc, 157
 - matrix_free, 158
 - matrix_load_from_file, 159
 - matrix_load_from_stream, 159
 - matrix_logdet_cholesky, 160
 - matrix_logdet_dense, 161
 - matrix_logdet_lu, 162
 - matrix_logdet_qr, 163
 - matrix_logdet_triangular, 164
 - matrix_norm_frobenius, 165
 - matrix_save_to_file, 165
 - matrix_save_to_stream, 166
 - matrix_setall, 167
 - matrix_trace, 167
 - matrix_trace2, 168
- matrix_alloc
 - matrix.c, 157
- matrix_free
 - matrix.c, 158
- matrix_load_from_file
 - matrix.c, 159
- matrix_load_from_stream
 - matrix.c, 159
- matrix_logdet_cholesky
 - matrix.c, 160
- matrix_logdet_dense
 - matrix.c, 161
- matrix_logdet_lu
 - matrix.c, 162
- matrix_logdet_qr
 - matrix.c, 163
- matrix_logdet_triangular
 - matrix.c, 164
- matrix_norm_frobenius
 - matrix.c, 165
- matrix_save_to_file
 - matrix.c, 165
- matrix_save_to_stream
 - matrix.c, 166
- matrix_setall
 - matrix.c, 167
- matrix_t, 19
 - dim, 19
 - dim2, 20
 - lda, 20
 - M, 20
- matrix_trace
 - matrix.c, 167
- matrix_trace2
 - matrix.c, 168
- misc.c, 169
 - kahan_sum, 170
 - logadd, 171
 - logadd_ms, 171
 - sqrtpm1, 172
- now
 - utils.c, 188
 - utils.h, 98
- omegap_high
 - material_t, 18
- omegap_low
 - material_t, 18
- plm.c, 173
 - _PI1, 174
 - _PI2, 175
 - _PI3, 176
 - _fn, 174
 - dlnPlm, 177
 - lnPI, 178
 - lnPlm, 179
 - lnPlm_downwards, 180
 - lnPlm_upwards, 181
 - Plm_continued_fraction, 182
- Plm_continued_fraction
 - plm.c, 182
- points
 - material_t, 18
- polarization_t
 - libcasimir.h, 69
- pow_2
 - constants.h, 64
- psd
 - psd.c, 185
- psd.c, 183
 - _eta, 184
 - dstemr_, 185
 - psd, 185
- quadpack.h
 - dqage, 51
 - dqagi, 52
 - dqags, 53
 - GK_10_21, 50
 - GK_15_31, 50
 - GK_20_41, 50
 - GK_25_51, 50
 - GK_30_61, 50
 - GK_7_15, 51
- R
 - casimir, 11
- s
 - log_t, 17
- SGN

- constants.h, [64](#)
- sign_t
 - constants.h, [64](#)
- size
 - buf, [9](#)
- sqrtpm1
 - misc.c, [172](#)
- strim
 - utils.c, [188](#)
 - utils.h, [99](#)
- strrep
 - utils.c, [188](#)
 - utils.h, [99](#)
- TERMINATE
 - utils.h, [97](#)
- time_as_string
 - utils.c, [189](#)
 - utils.h, [99](#)
- utils.c, [186](#)
 - disable_buffering, [188](#)
 - now, [188](#)
 - strim, [188](#)
 - strrep, [188](#)
 - time_as_string, [189](#)
 - xcalloc, [189](#)
 - xmalloc, [190](#)
 - xrealloc, [190](#)
- utils.h
 - COMPILER, [97](#)
 - disable_buffering, [98](#)
 - now, [98](#)
 - strim, [99](#)
 - strrep, [99](#)
 - TERMINATE, [97](#)
 - time_as_string, [99](#)
 - WARN, [98](#)
 - xcalloc, [100](#)
 - xfree, [98](#)
 - xmalloc, [100](#)
 - xrealloc, [101](#)
- v
 - log_t, [17](#)
- WARN
 - utils.h, [98](#)
- wi_finite
 - fcqs.c, [58](#)
- wi_semiinf
 - fcqs.c, [58](#)
- xcalloc
 - utils.c, [189](#)
 - utils.h, [100](#)
- xfree
 - utils.h, [98](#)
- xi
 - material_t, [18](#)
- xi_max
 - material_t, [19](#)
- xi_min
 - material_t, [19](#)
- xmalloc
 - utils.c, [190](#)
 - utils.h, [100](#)
- xrealloc
 - utils.c, [190](#)
 - utils.h, [101](#)
- y
 - casimir, [11](#)