

flavour

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Namespace Index</b>	<b>3</b>
2.1	Namespace List . . . . .	3
<b>3</b>	<b>Hierarchical Index</b>	<b>5</b>
3.1	Class Hierarchy . . . . .	5
<b>4</b>	<b>Class Index</b>	<b>7</b>
4.1	Class List . . . . .	7
<b>5</b>	<b>File Index</b>	<b>9</b>
5.1	File List . . . . .	9
<b>6</b>	<b>Namespace Documentation</b>	<b>11</b>
6.1	BGLmodels Namespace Reference . . . . .	11
6.1.1	Typedef Documentation . . . . .	12
6.1.1.1	CD . . . . .	12
6.1.1.2	Matrix3c . . . . .	12
6.1.1.3	Vector3c . . . . .	12
6.1.2	Enumeration Type Documentation . . . . .	12
6.1.2.1	BSpin . . . . .	12
6.1.2.2	FCharge . . . . .	13
6.1.2.3	FFlavour . . . . .	13
6.1.2.4	FHelicity . . . . .	13
6.1.2.5	FIsospin . . . . .	13

6.1.2.6	FType	14
6.1.3	Function Documentation	14
6.1.3.1	C7SM(double x)	14
6.1.3.2	C8SM(double x)	14
6.1.3.3	Vud(13.04 *M_PI/180, 0.201 *M_PI/180, 2.38 *M_PI/180, 1.2)	14
6.1.4	Variable Documentation	15
6.1.4.1	C7SM_Mb	15
6.1.4.2	C7SM_Mt	15
6.1.4.3	C7SM_MW	15
6.1.4.4	C8SM_Mb	15
6.1.4.5	C8SM_Mt	15
6.1.4.6	C8SM_MW	15
6.1.4.7	M_cos2	15
6.1.4.8	M_GF	15
6.1.4.9	M_Md	15
6.1.4.10	M_Ml	16
6.1.4.11	M_Mu	16
6.1.4.12	M_MW	16
6.1.4.13	M_MZ	16
6.1.4.14	mt_mb	16
6.1.4.15	mt_mt	16
6.1.4.16	mt_mW	16
6.1.4.17	Vnl	16
6.2	std Namespace Reference	17
6.2.1	Function Documentation	17
6.2.1.1	operator*(const Matrix &m1, const Matrix &m2)	17
6.2.1.2	operator+(const Matrix &m1, const Matrix &m2)	17

<b>7</b>	<b>Class Documentation</b>	<b>19</b>
7.1	BGLmodels::BGL Class Reference	19
7.1.1	Detailed Description	21
7.1.2	Constructor & Destructor Documentation	22
7.1.2.1	BGL(int genL=2, int genQ=2, int lup=0, int qup=0, int mssm=0)	22
7.1.2.2	~BGL()	30
7.1.3	Member Function Documentation	30
7.1.3.1	A0(ex x) const	30
7.1.3.2	A1(ex x) const	31
7.1.3.3	A2(ex x) const	31
7.1.3.4	A3(ex x) const	31
7.1.3.5	add(const char *s, ex pred, observable *ob, bool sb=0)	31
7.1.3.6	BranchingRatio(double *xx, double *p)	32
7.1.3.7	bsgammawidth(double tanb, double McH, double MR, double MI, int option=0)	32
7.1.3.8	CHdecaycoupling(Boson higgs, const Fermion &ff3, const Fermion &ff4) const	32
7.1.3.9	decaywidth(const Fermion &ff1, const Fermion &ff2, const Fermion &ff3, const Fermion &ff4, BSpin s=sAny) const	33
7.1.3.10	decaywidthtest2(const Fermion &ff1) const	34
7.1.3.11	fermiontomeson(const Fermion &ff4, const Fermion &ff3, const Meson &meson, BSpin s=sAny) const	34
7.1.3.12	fermiontomesontest(const Fermion &ff4, const Fermion &ff3, const Meson &meson, BSpin s=sAny) const	35
7.1.3.13	FH1(ex x) const	36
7.1.3.14	Fh1(ex x) const	36
7.1.3.15	FH2(ex x) const	37
7.1.3.16	Fh2(ex x) const	37
7.1.3.17	FW(ex x) const	37
7.1.3.18	generateparameters(int max=0) const	37
7.1.3.19	get_integral_symb(const multivector< ex, 3 > &a, ex m1) const	38
7.1.3.20	getlist(const parameters &p) const	39
7.1.3.21	GH1(ex x) const	39
7.1.3.22	GH2(ex x) const	40

7.1.3.23	<code>gRR2(const Fermion &amp;f1, const Fermion &amp;f3) const</code>	40
7.1.3.24	<code>GW(ex x) const</code>	40
7.1.3.25	<code>mesondw(const Meson &amp;meson, const Fermion &amp;ff3, const Fermion &amp;ff4, BSpin s=sAny) const</code>	41
7.1.3.26	<code>mesondwtest(const Meson &amp;meson, const Fermion &amp;ff3, const Fermion &amp;ff4, BSpin s=sAny) const</code>	42
7.1.3.27	<code>mesonmixing(ex mesonmass, const Fermion &amp;f1, const Fermion &amp;f2) const</code>	43
7.1.3.28	<code>tautomu_tautoe() const</code>	44
7.1.3.29	<code>topBranchingRatio(double *xx, double *p)</code>	44
7.1.3.30	<code>veto(const parameters &amp;p, int max=0) const</code>	45
7.1.3.31	<code>Y(ex x) const</code>	45
7.1.4	Member Data Documentation	45
7.1.4.1	<code>alpha</code>	45
7.1.4.2	<code>BGLtype</code>	45
7.1.4.3	<code>bosons</code>	46
7.1.4.4	<code>BR_Htotaunu</code>	46
7.1.4.5	<code>BR_toptoHq</code>	46
7.1.4.6	<code>Btaunu</code>	46
7.1.4.7	<code>BtoD2taunuR</code>	46
7.1.4.8	<code>BtoDtaunuR</code>	46
7.1.4.9	<code>BtotaunuR</code>	46
7.1.4.10	<code>cBmumu</code>	46
7.1.4.11	<code>cBsmumu</code>	46
7.1.4.12	<code>conjtoabs</code>	47
7.1.4.13	<code>cos2</code>	47
7.1.4.14	<code>cp</code>	47
7.1.4.15	<code>FB</code>	47
7.1.4.16	<code>FBs</code>	47
7.1.4.17	<code>FD</code>	47
7.1.4.18	<code>FDs</code>	47
7.1.4.19	<code>FK</code>	47

7.1.4.20	Fpi	47
7.1.4.21	g	47
7.1.4.22	GF	48
7.1.4.23	iBD2taunu	48
7.1.4.24	iBDtaunu	48
7.1.4.25	iBtaunu	48
7.1.4.26	MB0	48
7.1.4.27	MBp	48
7.1.4.28	MBs0	48
7.1.4.29	McH	48
7.1.4.30	Md	48
7.1.4.31	MD0	48
7.1.4.32	MDp	49
7.1.4.33	MDs0	49
7.1.4.34	MDsp	49
7.1.4.35	Mh	49
7.1.4.36	MI	49
7.1.4.37	mixes	49
7.1.4.38	MK0	49
7.1.4.39	MKp	49
7.1.4.40	mmmax	49
7.1.4.41	Mpi0	49
7.1.4.42	Mpip	50
7.1.4.43	MR	50
7.1.4.44	Mu	50
7.1.4.45	mu	50
7.1.4.46	MW	50
7.1.4.47	MZ	50
7.1.4.48	planck	50
7.1.4.49	replacements	50

7.1.4.50	<code>rho</code>	50
7.1.4.51	<code>stepsize</code>	50
7.1.4.52	<code>tanb</code>	51
7.1.4.53	<code>wc</code>	51
7.2	<b>BGL2 Class Reference</b>	51
7.2.1	<b>Detailed Description</b>	53
7.2.2	<b>Constructor &amp; Destructor Documentation</b>	54
7.2.2.1	<code>BGL2(int genL=2, int genQ=2, int lup=0, int qup=0, int mssm=0)</code>	54
7.2.2.2	<code>~BGL2()</code>	56
7.2.3	<b>Member Function Documentation</b>	57
7.2.3.1	<code>bsgammawidth(double tanb_, double McH_, double MR_, double MI_, int option=0)</code>	57
7.2.3.2	<code>epsK(double tanb_, double McH_, double MR_, double MI_, int option=0)</code>	57
7.2.3.3	<code>generateparameters(int max=0) const</code>	57
7.2.3.4	<code>getlist(const parameters &amp;p) const</code>	58
7.2.3.5	<code>mesonmixing(ex mesonmass, const Fermion &amp;f1, const Fermion &amp;f2) const</code>	58
7.2.4	<b>Member Data Documentation</b>	59
7.2.4.1	<code>alpha</code>	59
7.2.4.2	<code>BGLtype</code>	59
7.2.4.3	<code>bosons</code>	59
7.2.4.4	<code>BR_Htotaunu</code>	59
7.2.4.5	<code>BR_toptoHq</code>	59
7.2.4.6	<code>Btaunu</code>	59
7.2.4.7	<code>BtoD2taunuR</code>	59
7.2.4.8	<code>BtoDtaunuR</code>	59
7.2.4.9	<code>BtotaunuR</code>	59
7.2.4.10	<code>conjtoabs</code>	59
7.2.4.11	<code>cos2</code>	60
7.2.4.12	<code>cp</code>	60
7.2.4.13	<code>epsilonK</code>	60
7.2.4.14	<code>FB</code>	60



7.2.4.15	FBs	60
7.2.4.16	FD	60
7.2.4.17	FDs	60
7.2.4.18	FK	60
7.2.4.19	Fpi	60
7.2.4.20	g	60
7.2.4.21	GF	61
7.2.4.22	iBD2taunu	61
7.2.4.23	iBDtaunu	61
7.2.4.24	iBtaunu	61
7.2.4.25	inter1	61
7.2.4.26	inter2	61
7.2.4.27	MB0	61
7.2.4.28	MBp	61
7.2.4.29	MBs0	61
7.2.4.30	McH	61
7.2.4.31	Md	62
7.2.4.32	MD0	62
7.2.4.33	Md_	62
7.2.4.34	MDp	62
7.2.4.35	MDs0	62
7.2.4.36	MDsp	62
7.2.4.37	Mh	62
7.2.4.38	MI	62
7.2.4.39	mixes	62
7.2.4.40	MK0	62
7.2.4.41	MKp	63
7.2.4.42	mmmax	63
7.2.4.43	Mpi0	63
7.2.4.44	Mpip	63

7.2.4.45	MR	63
7.2.4.46	Mu	63
7.2.4.47	mu	63
7.2.4.48	Mu_	63
7.2.4.49	MW	63
7.2.4.50	MZ	63
7.2.4.51	planck	64
7.2.4.52	QCD1	64
7.2.4.53	QCD2	64
7.2.4.54	replacements	64
7.2.4.55	rho	64
7.2.4.56	Sparam	64
7.2.4.57	stepsize	64
7.2.4.58	tanb	64
7.2.4.59	Tparam	64
7.3	BGLmodels::Boson Class Reference	65
7.3.1	Detailed Description	65
7.3.2	Constructor & Destructor Documentation	66
7.3.2.1	Boson()	66
7.3.3	Member Function Documentation	66
7.3.3.1	coupling(const Fermion &f2, const Fermion &f1, ex mu)	66
7.3.3.2	couplingdaggerL(const Fermion &f2, const Fermion &f1) const	66
7.3.3.3	couplingdaggerR(const Fermion &f2, const Fermion &f1) const	67
7.3.3.4	couplingL(const Fermion &f2, const Fermion &f1) const	68
7.3.3.5	couplingR(const Fermion &f2, const Fermion &f1) const	69
7.3.3.6	reset()	69
7.3.4	Member Data Documentation	70
7.3.4.1	C	70
7.3.4.2	mass	70
7.3.4.3	s	70

7.4	calcu Class Reference	71
7.4.1	Detailed Description	71
7.4.2	Member Function Documentation	71
7.4.2.1	operator()(const parameters &p) const =0	71
7.5	calcuba Class Reference	72
7.5.1	Detailed Description	73
7.5.2	Constructor & Destructor Documentation	73
7.5.2.1	calcuba(observable *ob, const FUNCP_CUBA &e0)	73
7.5.3	Member Function Documentation	73
7.5.3.1	operator()(const parameters &p) const	73
7.5.4	Member Data Documentation	74
7.5.4.1	e	74
7.5.4.2	o	74
7.6	BGLmodels::calcuBmumu Class Reference	74
7.6.1	Detailed Description	76
7.6.2	Constructor & Destructor Documentation	76
7.6.2.1	calcuBmumu(const Mixes &mix, const Meson &m, const Fermion &f3, const Fermion &f4, observable *ob, const char *name)	76
7.6.3	Member Function Documentation	77
7.6.3.1	mesondwtest() const	77
7.6.3.2	obsvalue(const parameters &p) const	77
7.6.3.3	operator()(const parameters &p) const	78
7.6.3.4	Y(double x) const	78
7.6.4	Member Data Documentation	78
7.6.4.1	ff3	78
7.6.4.2	ff4	79
7.6.4.3	fp	79
7.6.4.4	gAi	79
7.6.4.5	gAr	79
7.6.4.6	gPi	79
7.6.4.7	gPr	79

7.6.4.8	gSi	79
7.6.4.9	gSr	79
7.6.4.10	meson	79
7.6.4.11	mixes	79
7.6.4.12	o	80
7.7	BGLmodels::calcubtosgamma2 Class Reference	80
7.7.1	Detailed Description	81
7.7.2	Constructor & Destructor Documentation	82
7.7.2.1	calcubtosgamma2(const Mixes &mixes)	82
7.7.3	Member Function Documentation	83
7.7.3.1	A0(double x) const	83
7.7.3.2	A1(double x) const	84
7.7.3.3	A2(double x) const	84
7.7.3.4	A3(double x) const	84
7.7.3.5	operator()(const parameters &p) const	84
7.7.3.6	width(const parameters &p, int option=0) const	86
7.7.4	Member Data Documentation	88
7.7.4.1	a	88
7.7.4.2	a77	88
7.7.4.3	a7ei	88
7.7.4.4	a7er	89
7.7.4.5	a7i	89
7.7.4.6	a7r	89
7.7.4.7	a87i	89
7.7.4.8	a87r	89
7.7.4.9	a88	89
7.7.4.10	a8ei	89
7.7.4.11	a8er	89
7.7.4.12	a8i	89
7.7.4.13	a8r	89

7.7.4.14	<a href="#">aee</a>	90
7.7.4.15	<a href="#">aei</a>	90
7.7.4.16	<a href="#">aer</a>	90
7.7.4.17	<a href="#">calN</a>	90
7.7.4.18	<a href="#">fp</a>	90
7.7.4.19	<a href="#">g1</a>	90
7.7.4.20	<a href="#">g2</a>	90
7.7.4.21	<a href="#">ii</a>	90
7.7.4.22	<a href="#">inter1</a>	90
7.7.4.23	<a href="#">inter2</a>	90
7.7.4.24	<a href="#">inter3</a>	91
7.7.4.25	<a href="#">inter4</a>	91
7.7.4.26	<a href="#">Md_</a>	91
7.7.4.27	<a href="#">Mu_</a>	91
7.7.4.28	<a href="#">ratio</a>	91
7.8	<a href="#">calcuex Class Reference</a>	91
7.8.1	<a href="#">Detailed Description</a>	92
7.8.2	<a href="#">Constructor &amp; Destructor Documentation</a>	92
7.8.2.1	<a href="#">calcuex(observable *ob, const ex &amp;e0)</a>	92
7.8.2.2	<a href="#">~calcuex()</a>	92
7.8.3	<a href="#">Member Function Documentation</a>	93
7.8.3.1	<a href="#">error(const parameters &amp;p) const</a>	93
7.8.3.2	<a href="#">operator()(const parameters &amp;p) const</a>	93
7.8.4	<a href="#">Member Data Documentation</a>	94
7.8.4.1	<a href="#">e</a>	94
7.8.4.2	<a href="#">o</a>	94
7.9	<a href="#">BGLmodels::calcuOblique Class Reference</a>	94
7.9.1	<a href="#">Detailed Description</a>	95
7.9.2	<a href="#">Constructor &amp; Destructor Documentation</a>	95
7.9.2.1	<a href="#">calcuOblique()</a>	95

7.9.3	Member Function Documentation . . . . .	96
7.9.3.1	F(double x, double y) const . . . . .	96
7.9.3.2	f(double t, double r) const . . . . .	96
7.9.3.3	G(double x, double y, double z) const . . . . .	96
7.9.3.4	Inxy_xy(double x, double y) const . . . . .	96
7.9.3.5	operator()(const parameters &p) const . . . . .	96
7.9.4	Member Data Documentation . . . . .	97
7.9.4.1	c1 . . . . .	97
7.9.4.2	c2 . . . . .	97
7.9.4.3	g1 . . . . .	97
7.9.4.4	g2 . . . . .	97
7.10	discreteparameter Class Reference . . . . .	97
7.10.1	Detailed Description . . . . .	98
7.10.2	Constructor & Destructor Documentation . . . . .	98
7.10.2.1	discreteparameter(int mi, int ma, TRandom3 *r) . . . . .	98
7.10.3	Member Data Documentation . . . . .	98
7.10.3.1	max . . . . .	98
7.10.3.2	min . . . . .	98
7.10.3.3	value . . . . .	99
7.11	BGLmodels::Fermion Class Reference . . . . .	99
7.11.1	Detailed Description . . . . .	99
7.11.2	Constructor & Destructor Documentation . . . . .	99
7.11.2.1	Fermion(FType t, FIsospin i, FFlavour f=fAny, FCharge p=cParticle, FHelicity h=hAny) . . . . .	99
7.11.3	Member Data Documentation . . . . .	100
7.11.3.1	flavour . . . . .	100
7.11.3.2	helicity . . . . .	100
7.11.3.3	isospin . . . . .	100
7.11.3.4	particle . . . . .	100
7.11.3.5	type . . . . .	100
7.12	freeparameter Class Reference . . . . .	100

7.12.1 Detailed Description . . . . .	101
7.12.2 Constructor & Destructor Documentation . . . . .	101
7.12.2.1 freeparameter(double mi, double ma, TRandom3 *r, double ss=1e-2) . . . . .	101
7.12.3 Member Function Documentation . . . . .	101
7.12.3.1 dist(double x) const . . . . .	101
7.12.3.2 invalid() const . . . . .	102
7.12.3.3 next(TRandom3 *r, double f=1) . . . . .	102
7.12.4 Member Data Documentation . . . . .	102
7.12.4.1 max . . . . .	102
7.12.4.2 min . . . . .	102
7.12.4.3 step . . . . .	103
7.12.4.4 value . . . . .	103
7.13 gauss2obs Class Reference . . . . .	103
7.13.1 Detailed Description . . . . .	104
7.13.2 Constructor & Destructor Documentation . . . . .	104
7.13.2.1 gauss2obs(measure v) . . . . .	104
7.13.2.2 gauss2obs(double mean, double sigma) . . . . .	104
7.13.2.3 ~gauss2obs() . . . . .	104
7.13.3 Member Function Documentation . . . . .	105
7.13.3.1 error(double hypothesis) const . . . . .	105
7.13.3.2 expected() const . . . . .	105
7.13.3.3 loglikelihood(double hypothesis) const . . . . .	105
7.13.4 Member Data Documentation . . . . .	105
7.13.4.1 m . . . . .	105
7.13.4.2 s . . . . .	105
7.14 gaussobs Class Reference . . . . .	106
7.14.1 Detailed Description . . . . .	107
7.14.2 Constructor & Destructor Documentation . . . . .	107
7.14.2.1 gaussobs(measure v) . . . . .	107
7.14.2.2 gaussobs(double mean, double sigma) . . . . .	107

7.14.2.3	<code>~gaussobs()</code>	107
7.14.3	Member Function Documentation	107
7.14.3.1	<code>error(double hipotesis) const</code>	107
7.14.3.2	<code>loglikelihood(double hipotesis) const</code>	107
7.14.4	Member Data Documentation	108
7.14.4.1	<code>m</code>	108
7.14.4.2	<code>s</code>	108
7.15	Juca Class Reference	108
7.15.1	Detailed Description	109
7.15.2	Constructor & Destructor Documentation	110
7.15.2.1	<code>Juca()</code>	110
7.15.2.2	<code>~Juca()</code>	110
7.15.3	Member Function Documentation	110
7.15.3.1	<code>add(const char *s, ex pred, observable *ob)</code>	110
7.15.3.2	<code>generateparameters() const</code>	110
7.15.3.3	<code>getlist(const parameters &amp;p) const</code>	111
7.15.4	Member Data Documentation	112
7.15.4.1	<code>A</code>	112
7.15.4.2	<code>eta</code>	112
7.15.4.3	<code>lambda</code>	112
7.15.4.4	<code>Mb</code>	112
7.15.4.5	<code>Mc</code>	112
7.15.4.6	<code>Md</code>	112
7.15.4.7	<code>Ms</code>	112
7.15.4.8	<code>Mt</code>	113
7.15.4.9	<code>Mu</code>	113
7.15.4.10	<code>replacements</code>	113
7.15.4.11	<code>rho</code>	113
7.16	limitedobs Class Reference	113
7.16.1	Detailed Description	114



7.16.2	Constructor & Destructor Documentation . . . . .	114
7.16.2.1	limitedobs(double limit, double cl=0.9, double m=0) . . . . .	114
7.16.2.2	~limitedobs() . . . . .	115
7.16.3	Member Function Documentation . . . . .	115
7.16.3.1	error(double hipotesis) const . . . . .	115
7.16.3.2	loglikelihood(double hipotesis) const . . . . .	115
7.16.4	Member Data Documentation . . . . .	115
7.16.4.1	lim . . . . .	115
7.16.4.2	min . . . . .	116
7.16.4.3	s . . . . .	116
7.17	std::Matrix Class Reference . . . . .	116
7.17.1	Detailed Description . . . . .	117
7.17.2	Constructor & Destructor Documentation . . . . .	118
7.17.2.1	Matrix() . . . . .	118
7.17.2.2	Matrix(const Matrix &m) . . . . .	118
7.17.2.3	Matrix(const char *m[3][3]) . . . . .	118
7.17.2.4	Matrix(const char *name, const char **index1, const char **index2) . . . . .	118
7.17.2.5	Matrix(ex m1, ex m2, ex m3) . . . . .	118
7.17.2.6	Matrix(ex m1) . . . . .	119
7.17.2.7	Matrix(ex t12, ex t13, ex t23, ex d13) . . . . .	119
7.17.3	Member Function Documentation . . . . .	119
7.17.3.1	conjugate() const . . . . .	119
7.17.3.2	cs(ex t12) . . . . .	120
7.17.3.3	sn(ex t12) . . . . .	120
7.18	BGLmodels::Matrixx Class Reference . . . . .	121
7.18.1	Detailed Description . . . . .	121
7.18.2	Constructor & Destructor Documentation . . . . .	122
7.18.2.1	Matrixx() . . . . .	122
7.18.2.2	Matrixx(const Matrix3c &a) . . . . .	122
7.18.2.3	Matrixx(double c12, double c13, double c23, double s12, double s13, double s23, CD e13, CD e13t) . . . . .	122

7.18.2.4	Matrixx(double t12, double t13, double t23, double d13)	122
7.18.3	Member Function Documentation	122
7.18.3.1	conjugate() const	122
7.19	measure Class Reference	123
7.19.1	Detailed Description	123
7.19.2	Constructor & Destructor Documentation	123
7.19.2.1	measure(double v=0, double e=0)	123
7.19.3	Member Function Documentation	123
7.19.3.1	operator*(measure m2) const	123
7.19.3.2	operator/(measure m2) const	124
7.19.4	Member Data Documentation	124
7.19.4.1	error	124
7.19.4.2	value	124
7.20	BGLmodels::Meson Class Reference	124
7.20.1	Detailed Description	125
7.20.2	Constructor & Destructor Documentation	125
7.20.2.1	Meson(const Fermion &qq1, const Fermion &qq2, ex m, ex d)	125
7.20.3	Member Data Documentation	125
7.20.3.1	decay_factor	125
7.20.3.2	mass	125
7.20.3.3	q1	125
7.20.3.4	q2	126
7.21	BGLmodels::Mixes Class Reference	126
7.21.1	Detailed Description	127
7.21.2	Constructor & Destructor Documentation	127
7.21.2.1	Mixes(ex tanb0, ex cp0, int genL=2, int genQ=2, int lup=0, int qup=0, int mssm=0)	127
7.21.3	Member Function Documentation	128
7.21.3.1	appendtolst(lst &reps) const	128
7.21.3.2	mass(const Fermion &f) const	129
7.21.3.3	massnum(const Fermion &f) const	129

7.21.4 Member Data Documentation . . . . .	129
7.21.4.1 cp . . . . .	129
7.21.4.2 M . . . . .	129
7.21.4.3 N . . . . .	129
7.21.4.4 N_ . . . . .	129
7.21.4.5 replacements . . . . .	130
7.21.4.6 reps . . . . .	130
7.21.4.7 tanb . . . . .	130
7.21.4.8 V . . . . .	130
7.21.4.9 VN . . . . .	130
7.21.4.10 VN_ . . . . .	130
7.22 Model Class Reference . . . . .	131
7.22.1 Detailed Description . . . . .	132
7.22.2 Constructor & Destructor Documentation . . . . .	132
7.22.2.1 Model() . . . . .	132
7.22.2.2 ~Model() . . . . .	132
7.22.3 Member Function Documentation . . . . .	132
7.22.3.1 generateparameters(int max=0) const =0 . . . . .	132
7.22.3.2 getlist(const parameters &p) const =0 . . . . .	132
7.22.3.3 likelihood(const parameters &p, bool check=1, int max=0) const . . . . .	132
7.22.3.4 loglike(const parameters &p, bool check=1, int max=0) const . . . . .	133
7.22.3.5 veto(const parameters &p, int max=0) const . . . . .	133
7.22.4 Member Data Documentation . . . . .	134
7.22.4.1 r . . . . .	134
7.23 std::multivector< T, N > Class Template Reference . . . . .	134
7.23.1 Detailed Description . . . . .	135
7.23.2 Member Typedef Documentation . . . . .	135
7.23.2.1 v . . . . .	135
7.23.3 Constructor & Destructor Documentation . . . . .	136
7.23.3.1 multivector() . . . . .	136

7.23.3.2	<a href="#">multivector(const multivector &amp;m)</a>	136
7.23.3.3	<a href="#">multivector(const T &amp;value,...)</a>	136
7.23.3.4	<a href="#">multivector(const T &amp;value, va_list &amp;listPointer)</a>	136
7.24	<a href="#">std::multivector&lt; T, 1 &gt; Class Template Reference</a>	137
7.24.1	<a href="#">Detailed Description</a>	138
7.24.2	<a href="#">Member Typedef Documentation</a>	138
7.24.2.1	<a href="#">v</a>	138
7.24.3	<a href="#">Constructor &amp; Destructor Documentation</a>	138
7.24.3.1	<a href="#">multivector()</a>	138
7.24.3.2	<a href="#">multivector(const multivector &amp;m)</a>	138
7.24.3.3	<a href="#">multivector(const T &amp;value, int x)</a>	138
7.24.3.4	<a href="#">multivector(const T &amp;value, va_list &amp;listPointer)</a>	139
7.25	<a href="#">observable Class Reference</a>	139
7.25.1	<a href="#">Detailed Description</a>	140
7.25.2	<a href="#">Constructor &amp; Destructor Documentation</a>	140
7.25.2.1	<a href="#">observable()</a>	140
7.25.2.2	<a href="#">~observable()</a>	140
7.25.3	<a href="#">Member Function Documentation</a>	140
7.25.3.1	<a href="#">error(double hipotesis) const =0</a>	140
7.25.3.2	<a href="#">loglikelihood(double hipotesis) const =0</a>	140
7.25.4	<a href="#">Member Data Documentation</a>	141
7.25.4.1	<a href="#">copies</a>	141
7.26	<a href="#">parameters Class Reference</a>	141
7.26.1	<a href="#">Detailed Description</a>	142
7.26.2	<a href="#">Member Function Documentation</a>	142
7.26.2.1	<a href="#">area() const</a>	142
7.26.2.2	<a href="#">dist(const parameters &amp;p) const</a>	143
7.26.2.3	<a href="#">gausslikelihood(const parameters &amp;p2) const</a>	143
7.26.2.4	<a href="#">invalid() const</a>	143
7.26.2.5	<a href="#">next(TRandom3 *r, double f=1)</a>	144

7.26.2.6	setvalues(const parameters &p)	144
7.26.3	Member Data Documentation	144
7.26.3.1	p	144
7.26.3.2	values	144
7.27	Peak Class Reference	145
7.27.1	Detailed Description	145
7.27.2	Constructor & Destructor Documentation	146
7.27.2.1	Peak(const Model *m, int maxx=0)	146
7.27.3	Member Function Documentation	146
7.27.3.1	adjuststeps()	146
7.27.3.2	findPeak()	147
7.27.4	Member Data Documentation	147
7.27.4.1	area	147
7.27.4.2	larea	148
7.27.4.3	lmax	148
7.27.4.4	lmax	148
7.27.4.5	max	148
7.27.4.6	model	148
7.27.4.7	pr	148
7.28	points Class Reference	148
7.28.1	Detailed Description	149
7.28.2	Constructor & Destructor Documentation	149
7.28.2.1	points(double p, double m, double e)	149
7.28.3	Member Data Documentation	149
7.28.3.1	error	149
7.28.3.2	measure	149
7.28.3.3	prediction	149
7.29	prediction Class Reference	149
7.29.1	Detailed Description	150
7.29.2	Constructor & Destructor Documentation	150

7.29.2.1	<a href="#">prediction(observable *ob, const FUNCP_CUBA &amp;e0)</a>	150
7.29.2.2	<a href="#">prediction(observable *ob, const ex &amp;e0)</a>	150
7.29.2.3	<a href="#">prediction(calcu *c)</a>	150
7.29.2.4	<a href="#">prediction(const prediction &amp;p)</a>	150
7.29.2.5	<a href="#">~prediction()</a>	150
7.29.3	<a href="#">Member Function Documentation</a>	151
7.29.3.1	<a href="#">loglikelihood(const parameters &amp;p) const</a>	151
7.29.4	<a href="#">Member Data Documentation</a>	151
7.29.4.1	<a href="#">calculate</a>	151
7.30	<a href="#">Proposal Class Reference</a>	151
7.30.1	<a href="#">Detailed Description</a>	152
7.30.2	<a href="#">Constructor &amp; Destructor Documentation</a>	152
7.30.2.1	<a href="#">Proposal(const Model *m)</a>	152
7.30.3	<a href="#">Member Function Documentation</a>	152
7.30.3.1	<a href="#">findPeaks(uint ns=1, int max=0)</a>	152
7.30.3.2	<a href="#">getNextPoint()</a>	153
7.30.3.3	<a href="#">getProposal()</a>	154
7.30.4	<a href="#">Member Data Documentation</a>	154
7.30.4.1	<a href="#">floatPeak</a>	154
7.30.4.2	<a href="#">model</a>	154
7.30.4.3	<a href="#">proposal</a>	154
7.30.4.4	<a href="#">total</a>	155
7.30.4.5	<a href="#">vPeak</a>	155
7.31	<a href="#">widthcalc Class Reference</a>	155
7.31.1	<a href="#">Detailed Description</a>	156
7.31.2	<a href="#">Constructor &amp; Destructor Documentation</a>	156
7.31.2.1	<a href="#">widthcalc()</a>	156
7.31.3	<a href="#">Member Function Documentation</a>	156
7.31.3.1	<a href="#">genM2()</a>	156
7.31.3.2	<a href="#">genM22()</a>	157

7.31.3.3	<code>get_integral(const multivector&lt; ex, 4 &gt; &amp;a, const vector&lt; ex &gt; &amp;mass, const vector&lt; int &gt; &amp;op, double m1, double m2, double m3, double m4) const</code>	159
7.31.3.4	<code>get_integral_meson(const multivector&lt; ex, 4 &gt; &amp;a, const vector&lt; ex &gt; &amp;mass, const vector&lt; int &gt; &amp;op, ex mm, ex m1, ex m2, ex m3, ex m4) const</code>	159
7.31.3.5	<code>get_integral_meson2(const multivector&lt; ex, 4 &gt; &amp;a, const vector&lt; ex &gt; &amp;mass, const vector&lt; int &gt; &amp;op, ex mm, ex m1, ex m2, ex m3, ex m4) const</code>	160
7.31.3.6	<code>get_integral_symb(const multivector&lt; ex, 4 &gt; &amp;a, const vector&lt; ex &gt; &amp;mass, const vector&lt; int &gt; &amp;op, ex m1) const</code>	160
7.31.4	Member Data Documentation	161
7.31.4.1	M2	161
7.31.4.2	M22	161
7.31.4.3	mq1	161
7.31.4.4	mq2	161
7.31.4.5	mq3	161
7.31.4.6	mq4	162
7.31.4.7	s2	162
7.31.4.8	s3	162
<b>8</b>	<b>File Documentation</b>	<b>163</b>
8.1	BGL.h File Reference	163
8.2	BGL.h	164
8.3	defs.h File Reference	183
8.3.1	Macro Definition Documentation	184
8.3.1.1	GL	184
8.3.1.2	GQ	184
8.3.1.3	UL	184
8.3.1.4	UQ	184
8.4	defs.h	184
8.5	draw.cpp File Reference	184
8.5.1	Function Documentation	185
8.5.1.1	<code>main(int argc, char *argv[])</code>	185
8.6	draw.cpp	189
8.7	eigen.cpp File Reference	198

8.7.1	Function Documentation	198
8.7.1.1	chi2(double *v)	198
8.7.1.2	main()	199
8.8	eigen.cpp	200
8.9	eigenteste.cpp File Reference	202
8.9.1	Function Documentation	202
8.9.1.1	main()	202
8.10	eigenteste.cpp	202
8.11	Formulas.h File Reference	203
8.12	Formulas.h	204
8.13	Juca.cpp File Reference	214
8.13.1	Function Documentation	215
8.13.1.1	main()	215
8.14	Juca.cpp	216
8.15	Juca.h File Reference	216
8.16	Juca.h	217
8.17	main.cpp File Reference	219
8.17.1	Function Documentation	220
8.17.1.1	main(int argc, char *argv[])	220
8.18	main.cpp	226
8.19	mainbk.cpp File Reference	233
8.19.1	Function Documentation	233
8.19.1.1	main()	233
8.20	mainbk.cpp	236
8.21	MCMC.h File Reference	239
8.22	MCMC.h	240
8.23	model.h File Reference	243
8.23.1	Macro Definition Documentation	244
8.23.1.1	_USE_MATH_DEFINES	244
8.24	model.h	244



8.25	<a href="#">multivector.h File Reference</a>	249
8.26	<a href="#">multivector.h</a>	251
8.27	<a href="#">script.cpp File Reference</a>	252
8.27.1	<a href="#">Typedef Documentation</a>	253
8.27.1.1	<a href="#">uint</a>	253
8.27.2	<a href="#">Function Documentation</a>	253
8.27.2.1	<a href="#">main()</a>	253
8.28	<a href="#">script.cpp</a>	254
8.29	<a href="#">script2.cpp File Reference</a>	254
8.29.1	<a href="#">Typedef Documentation</a>	255
8.29.1.1	<a href="#">uint</a>	255
8.29.2	<a href="#">Function Documentation</a>	255
8.29.2.1	<a href="#">main()</a>	255
8.30	<a href="#">script2.cpp</a>	256
8.31	<a href="#">scriptplots.cpp File Reference</a>	257
8.31.1	<a href="#">Function Documentation</a>	257
8.31.1.1	<a href="#">main()</a>	257
8.32	<a href="#">scriptplots.cpp</a>	258
8.33	<a href="#">scripttable.cpp File Reference</a>	260
8.33.1	<a href="#">Typedef Documentation</a>	260
8.33.1.1	<a href="#">uint</a>	260
8.33.2	<a href="#">Function Documentation</a>	261
8.33.2.1	<a href="#">main()</a>	261
8.34	<a href="#">scripttable.cpp</a>	261
8.35	<a href="#">update.cpp File Reference</a>	262
8.35.1	<a href="#">Function Documentation</a>	262
8.35.1.1	<a href="#">main(int argc, char *argv[])</a>	262
8.36	<a href="#">update.cpp</a>	263
8.37	<a href="#">widthcalc.h File Reference</a>	264
8.37.1	<a href="#">Macro Definition Documentation</a>	265
8.37.1.1	<a href="#">_USE_MATH_DEFINES</a>	265
8.38	<a href="#">widthcalc.h</a>	266



# Chapter 1

## Introduction

The program produces figures presenting 68%, 95% and 99% CL allowed regions in parameter space. To wit, we represent regions where the specific BGL model is able to fit the imposed experimental information at least as well as the corresponding goodness levels. Some comments are in order. This procedure corresponds to the profile likelihood method. In brief, for a model with parameters  $\vec{p}$ , we compute the predictions for the considered set of observables  $\vec{O}_{\text{Th}}(\vec{p})$ . Then, using the experimental information  $\vec{O}_{\text{Exp}}$  available for those observables, we build a likelihood function  $\mathcal{L}(\vec{O}_{\text{Exp}}|\vec{O}_{\text{Th}}(\vec{p}))$  which gives the probability of obtaining the experimental results  $\vec{O}_{\text{Exp}}$  assuming that the model is correct. The likelihood function  $\mathcal{L}(\vec{O}_{\text{Exp}}|\vec{O}_{\text{Th}}(\vec{p}))$  encodes all the information on how the model is able to reproduce the observed data all over parameter space. Nevertheless, the knowledge of  $\mathcal{L}(\vec{O}_{\text{Exp}}|\vec{O}_{\text{Th}}(\vec{p}))$  in a multidimensional parameter space can be hardly represented and one is led to the problem of reducing that information to one or two-dimensional subspaces. In the profile likelihood method, for each point in the chosen subspace, the highest likelihood over the complementary, marginalized space, is retained. Let us clarify that likelihood – or chi-squared  $\chi^2 \equiv -2 \log \mathcal{L}$  – profiles and derived regions such as the ones we represent, are thus insensitive to the size of the space over which one marginalizes; this would not be the case in a Bayesian analysis, where an integration over the marginalized space is involved. The profile likelihood method seems adequate to our purpose, which is none other than exploring where in parameter space are the different BGL models able to satisfy experimental constraints, without weighting in eventual fine tunings of the models or parameter space volumes. For the numerical computations the libraries GiNaC and ROOT are used. \*



## Chapter 2

# Namespace Index

### 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">BGLmodels</a>	.....	<a href="#">11</a>
<a href="#">std</a>	.....	<a href="#">17</a>



## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

BGLmodels::Boson . . . . .	65
calcu . . . . .	71
BGLmodels::calcuBmumu . . . . .	74
BGLmodels::calcubtosgamma2 . . . . .	80
BGLmodels::calcuOblique . . . . .	94
calcuba . . . . .	72
calcuex . . . . .	91
discretparameter . . . . .	97
BGLmodels::Fermion . . . . .	99
freeparameter . . . . .	100
Matrix3c	
BGLmodels::Matrixx . . . . .	121
measure . . . . .	123
BGLmodels::Meson . . . . .	124
BGLmodels::Mixes . . . . .	126
observable . . . . .	139
gauss2obs . . . . .	103
gaussobs . . . . .	106
limitedobs . . . . .	113
Peak . . . . .	145
points . . . . .	148
prediction . . . . .	149
Proposal . . . . .	151
vector	
Model . . . . .	131
BGL2 . . . . .	51
BGLmodels::BGL . . . . .	19
Juca . . . . .	108
std::multivector< ex, 2 > . . . . .	134
std::Matrix . . . . .	116
std::multivector< ex, 6 > . . . . .	134
std::multivector< ex, 7 > . . . . .	134
std::multivector< Matrix, 2 > . . . . .	134
std::multivector< Matrix, 4 > . . . . .	134
parameters . . . . .	141
std::multivector< T, N > . . . . .	134
std::multivector< T, 1 > . . . . .	137
widthcalc . . . . .	155





## Chapter 4

# Class Index

### 4.1 Class List

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

<a href="#">BGLmodels::BGL</a>	Implementation of the <a href="#">BGL</a> model . . . . .	19
<a href="#">BGL2</a>	A second implementation of the BGL model, for testing purposes . . . . .	51
<a href="#">BGLmodels::Boson</a>	Gauge boson . . . . .	65
<a href="#">calcu</a>	Base class to do the calculus of a constraint to the model . . . . .	71
<a href="#">calcuba</a>	Class to do the calculus of a constraint based on a GiNaC compiled expression . . . . .	72
<a href="#">BGLmodels::calcuBmumu</a>	Calculus of the constraints coming from the $B \rightarrow \mu \mu$ decay . . . . .	74
<a href="#">BGLmodels::calcubtosgamma2</a>	Calculus of the constraints coming from the $b \rightarrow s \gamma$ decay . . . . .	80
<a href="#">calcuex</a>	Class to do the calculus of a constraint based on a GiNaC symbolic expression . . . . .	91
<a href="#">BGLmodels::calcuOblique</a>	Calculus of the constraints coming from the oblique parameters . . . . .	94
<a href="#">discreteparameter</a>	A parameter which will be fitted in the simulation . . . . .	97
<a href="#">BGLmodels::Fermion</a>	<a href="#">Fermion</a> properties . . . . .	99
<a href="#">freeparameter</a>	A parameter which will be fitted in the simulation . . . . .	100
<a href="#">gauss2obs</a>	Same as gaussobs but with a different initializer, such that the uncertainty sigma is absolute . . . . .	103
<a href="#">gaussobs</a>	An experimental measure of a parameter which is a mean value and a standard deviation . . . . .	106
<a href="#">Juca</a>	. . . . .	108
<a href="#">limitedobs</a>	An experimental measure which is an upper limit on a parameter with a given Confidence Level . . . . .	113
<a href="#">std::Matrix</a>	. . . . .	116
<a href="#">BGLmodels::Matrixx</a>	Class to represent the mixing matrices VCKM and VPMNS . . . . .	121
<a href="#">measure</a>	A class containing the value and uncertainty of an experimental measure . . . . .	123

<a href="#">BGLmodels::Meson</a>	
<a href="#">Meson</a> properties . . . . .	124
<a href="#">BGLmodels::Mixes</a>	
Definition of the couplings for the different <a href="#">BGL</a> models . . . . .	126
<a href="#">Model</a>	
Abstract class for a model . . . . .	131
<a href="#">std::multivector&lt; T, N &gt;</a>	
A vector of vectors of vectors of... (N times) of class T objects . . . . .	134
<a href="#">std::multivector&lt; T, 1 &gt;</a>	
Specialization template class of <a href="#">multivector&lt;T,N&gt;</a> for N=1 . . . . .	137
<a href="#">observable</a>	
A base class representing an experimental measure . . . . .	139
<a href="#">parameters</a>	
Vector of parameters . . . . .	141
<a href="#">Peak</a>	
A class containing the parameters of a maximum of the likelihood function . . . . .	145
<a href="#">points</a> . . . . .	148
<a href="#">prediction</a>	
Theoretical expression for an experimental measure . . . . .	149
<a href="#">Proposal</a>	
A class containing the parameters of a proposal for the next step in the Markov Chain . . . . .	151
<a href="#">widthcalc</a>	
This class calculates decay widths of one lepton to 3 leptons . . . . .	155

## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

BGL.h	163
defs.h	183
draw.cpp	184
eigen.cpp	198
eigenteste.cpp	202
Formulas.h	203
Juca.cpp	214
Juca.h	216
main.cpp	219
mainbk.cpp	233
MCMC.h	239
model.h	243
multivector.h	249
script.cpp	252
script2.cpp	254
scriptplots.cpp	257
scripttable.cpp	260
update.cpp	262
widthcalc.h	264



## Chapter 6

# Namespace Documentation

### 6.1 BGLmodels Namespace Reference

#### Classes

- class [BGL](#)  
*Implementation of the [BGL](#) model.*
- class [Boson](#)  
*Gauge boson.*
- class [calcuBmumu](#)  
*calculus of the constraints coming from the  $B \rightarrow \mu \mu$  decay*
- class [calcubtosgamma2](#)  
*calculus of the constraints coming from the  $b \rightarrow s$  gamma decay*
- class [calcuOblique](#)  
*calculus of the constraints coming from the oblique parameters*
- class [Fermion](#)  
*a fermion properties*
- class [Matrixx](#)  
*a class to represent the mixing matrices VCKM and VPMNS*
- class [Meson](#)  
*a meson properties*
- class [Mixes](#)  
*definition of the couplings for the different [BGL](#) models*

#### Typedefs

- typedef std::complex< double > [CD](#)
- typedef std::array< [CD](#), 3 > [Vector3c](#)
- typedef std::array< std::array< [CD](#), 3 >, 3 > [Matrix3c](#)

#### Enumerations

- enum [FType](#) { [tLepton](#), [tQuark](#) }
- enum [Flsospin](#) { [iUp](#), [iDown](#) }
- enum [FFlavour](#) { [fElectron](#), [fMuon](#), [fTau](#), [fAny](#) }
- enum [FCharge](#) { [cParticle](#), [cAntiParticle](#) }
- enum [FHelicity](#) { [hLeft](#), [hRight](#), [hAny](#) }
- enum [BSpin](#) { [sScalar](#), [sVector](#), [sAny](#) }

## Functions

- const [Matrixx Vud](#) (13.04 \*M\_PI/180, 0.201 \*M\_PI/180, 2.38 \*M\_PI/180, 1.2)
- constexpr double [C7SM](#) (double x)
- constexpr double [C8SM](#) (double x)

## Variables

- constexpr double [M\\_GF](#) =1.166371e-5
- constexpr double [M\\_MZ](#) =91.1876
- constexpr double [M\\_MW](#) =80.398
- constexpr double [M\\_cos2](#) =std::pow([M\\_MW/M\\_MZ](#),2)
- constexpr double [M\\_Mu](#) [3] ={2.4e-3,1.29,172.9}
- constexpr double [M\\_Md](#) [3] ={5.3e-3,95e-3,4.2}
- constexpr double [M\\_MI](#) [3] ={0.510998910e-3,105.6583715e-3,1776.82e-3}
- const [Matrixx Vnl](#) =[Matrixx](#)(33.6\*M\_PI/180,9.11\*M\_PI/180,40.4\*M\_PI/180,M\_PI/4).conjugate()
- constexpr double [mt\\_mt](#) =163.3
- constexpr double [mt\\_mW](#) =174.2
- constexpr double [mt\\_mb](#) =261.8
- constexpr double [C7SM\\_MW](#) =[C7SM](#)(std::pow([mt\\_mW/M\\_MW](#),2))
- constexpr double [C7SM\\_Mt](#) =[C7SM](#)(std::pow([mt\\_mt/M\\_MW](#),2))
- constexpr double [C7SM\\_Mb](#) =-0.353
- constexpr double [C8SM\\_MW](#) =[C8SM](#)(std::pow([mt\\_mW/M\\_MW](#),2))
- constexpr double [C8SM\\_Mt](#) =[C8SM](#)(std::pow([mt\\_mt/M\\_MW](#),2))
- constexpr double [C8SM\\_Mb](#) =[C8SM](#)(std::pow([mt\\_mb/M\\_MW](#),2))

## 6.1.1 Typedef Documentation

### 6.1.1.1 typedef std::complex<double> BGLmodels::CD

Definition at line 65 of file [Formulas.h](#).

### 6.1.1.2 typedef std::array<std::array<CD,3>,3> BGLmodels::Matrix3c

Definition at line 67 of file [Formulas.h](#).

### 6.1.1.3 typedef std::array<CD,3> BGLmodels::Vector3c

Definition at line 66 of file [Formulas.h](#).

## 6.1.2 Enumeration Type Documentation

### 6.1.2.1 enum BGLmodels::BSpin

Enumerator

***sScalar***  
***sVector***  
***sAny***

Definition at line 27 of file [Formulas.h](#).

```
00027 {sScalar, sVector, sAny};
```

## 6.1.2.2 enum BGLmodels::FCharge

Enumerator

***cParticle***  
***cAntiParticle***

Definition at line 25 of file [Formulas.h](#).

```
00025 {cParticle, cAntiParticle};
```

## 6.1.2.3 enum BGLmodels::FFlavour

Enumerator

***fElectron***  
***fMuon***  
***fTau***  
***fAny***

Definition at line 24 of file [Formulas.h](#).

```
00024 {fElectron, fMuon, fTau, fAny};
```

## 6.1.2.4 enum BGLmodels::FHelicity

Enumerator

***hLeft***  
***hRight***  
***hAny***

Definition at line 26 of file [Formulas.h](#).

```
00026 {hLeft, hRight, hAny};
```

## 6.1.2.5 enum BGLmodels::FIsospin

Enumerator

***iUp***  
***iDown***

Definition at line 23 of file [Formulas.h](#).

```
00023 {iUp, iDown};
```

### 6.1.2.6 enum BGLmodels::FType

Enumerator

***tLepton***

***tQuark***

Definition at line 22 of file [Formulas.h](#).

```
00022 {tLepton,tQuark};
```

## 6.1.3 Function Documentation

### 6.1.3.1 constexpr double BGLmodels::C7SM ( double x )

Definition at line 264 of file [Formulas.h](#).

```
00264                                     {
00265     return ((1/(x-1)+3)*x*std::log(x)+(-8*x*x-5*x+7)/6)*x/4/std::pow(x-1,3);
00266 }
```

### 6.1.3.2 constexpr double BGLmodels::C8SM ( double x )

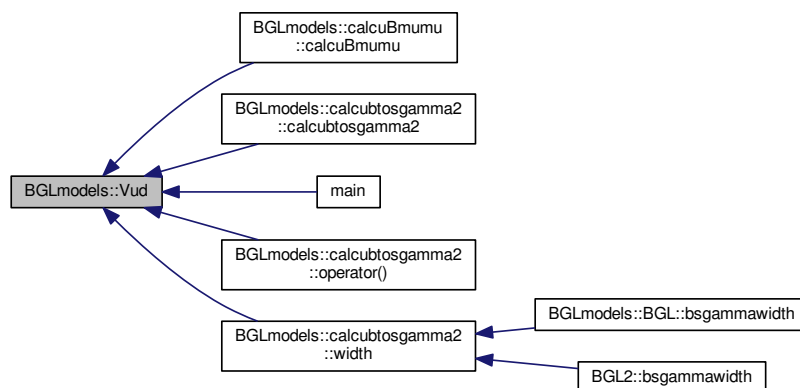
Definition at line 268 of file [Formulas.h](#).

```
00268                                     {
00269     return (-3/(x-1)*x*std::log(x)+(-x*x+5*x+2)/2)*x/4/std::pow(x-1,3);
00270 }
```

### 6.1.3.3 const Matrixx BGLmodels::Vud ( 13.04 \*M\_PI/ 180, 0.201 \*M\_PI/ 180, 2.38 \*M\_PI/ 180, 1. 2 )

Referenced by [BGLmodels::calcuBmumu::calcuBmumu\(\)](#), [BGLmodels::calcubtosgamma2::calcubtosgamma2\(\)](#), [main\(\)](#), [BGLmodels::calcubtosgamma2::operator\(\)](#), and [BGLmodels::calcubtosgamma2::width\(\)](#).

Here is the caller graph for this function:





### 6.1.4 Variable Documentation

6.1.4.1 `constexpr double BGLmodels::C7SM_Mb = -0.353`

Definition at line 272 of file [Formulas.h](#).

6.1.4.2 `constexpr double BGLmodels::C7SM_Mt = C7SM(std::pow(mt_mt/M_MW,2))`

Definition at line 272 of file [Formulas.h](#).

Referenced by [BGLmodels::calcubtosgamma2::calcubtosgamma2\(\)](#), [BGLmodels::calcubtosgamma2::operator\(\)\(\)](#), and [BGLmodels::calcubtosgamma2::width\(\)](#).

6.1.4.3 `constexpr double BGLmodels::C7SM_MW = C7SM(std::pow(mt_mW/M_MW,2))`

Definition at line 272 of file [Formulas.h](#).

6.1.4.4 `constexpr double BGLmodels::C8SM_Mb = C8SM(std::pow(mt_mb/M_MW,2))`

Definition at line 273 of file [Formulas.h](#).

6.1.4.5 `constexpr double BGLmodels::C8SM_Mt = C8SM(std::pow(mt_mt/M_MW,2))`

Definition at line 273 of file [Formulas.h](#).

Referenced by [BGLmodels::calcubtosgamma2::calcubtosgamma2\(\)](#), [BGLmodels::calcubtosgamma2::operator\(\)\(\)](#), and [BGLmodels::calcubtosgamma2::width\(\)](#).

6.1.4.6 `constexpr double BGLmodels::C8SM_MW = C8SM(std::pow(mt_mW/M_MW,2))`

Definition at line 273 of file [Formulas.h](#).

6.1.4.7 `constexpr double BGLmodels::M_cos2 = std::pow(M_MW/M_MZ,2)`

Definition at line 60 of file [Formulas.h](#).

6.1.4.8 `constexpr double BGLmodels::M_GF = 1.166371e-5`

Definition at line 57 of file [Formulas.h](#).

6.1.4.9 `constexpr double BGLmodels::M_Md[3] = {5.3e-3, 95e-3, 4.2}`

Definition at line 62 of file [Formulas.h](#).

6.1.4.10 `constexpr double BGLmodels::M_MI[3] = {0.510998910e-3, 105.6583715e-3, 1776.82e-3}`

Definition at line 63 of file [Formulas.h](#).

6.1.4.11 `constexpr double BGLmodels::M_Mu[3] = {2.4e-3, 1.29, 172.9}`

Definition at line 61 of file [Formulas.h](#).

6.1.4.12 `constexpr double BGLmodels::M_MW = 80.398`

Definition at line 59 of file [Formulas.h](#).

Referenced by [BGLmodels::BGL::BGL\(\)](#), and [BGL2::BGL2\(\)](#).

6.1.4.13 `constexpr double BGLmodels::M_MZ = 91.1876`

Definition at line 58 of file [Formulas.h](#).

Referenced by [BGLmodels::BGL::BGL\(\)](#), and [BGL2::BGL2\(\)](#).

6.1.4.14 `constexpr double BGLmodels::mt_mb = 261.8`

Definition at line 262 of file [Formulas.h](#).

6.1.4.15 `constexpr double BGLmodels::mt_mt = 163.3`

Definition at line 262 of file [Formulas.h](#).

Referenced by [BGLmodels::calcubtosgamma2::operator\(\)\(\)](#), and [BGLmodels::calcubtosgamma2::width\(\)](#).

6.1.4.16 `constexpr double BGLmodels::mt_mW = 174.2`

Definition at line 262 of file [Formulas.h](#).

6.1.4.17 `const Matrixx BGLmodels::Vnl = Matrixx(33.6*M_PI/180, 9.11*M_PI/180, 40.4*M_PI/180, M_PI/4).conjugate()`

Definition at line 98 of file [Formulas.h](#).

## 6.2 std Namespace Reference

### Classes

- class [Matrix](#)
- class [multivector](#)  
*A vector of vectors of vectors of... (N times) of class T objects.*
- class [multivector< T, 1 >](#)  
*Specialization template class of [multivector< T,N>](#) for N=1.*

### Functions

- [Matrix operator\\*](#) (const [Matrix](#) &m1, const [Matrix](#) &m2)  
*computes the matrix product*
- [Matrix operator+](#) (const [Matrix](#) &m1, const [Matrix](#) &m2)  
*computes the matrix sum*

#### 6.2.1 Function Documentation

##### 6.2.1.1 [Matrix std::operator\\* \( const Matrix & m1, const Matrix & m2 \)](#)

computes the matrix product

Definition at line 136 of file [multivector.h](#).

```
00136                                     {
00137     Matrix res;
00138     for(uint i=0; i<3; i++)
00139         for(uint j=0; j<3; j++)
00140             for(uint k=0; k<3; k++)
00141                 res[i][j]=res[i][j]+m1[i][k]*m2[k][j];
00142     return res;
00143 }
```

##### 6.2.1.2 [Matrix std::operator+ \( const Matrix & m1, const Matrix & m2 \)](#)

computes the matrix sum

Definition at line 146 of file [multivector.h](#).

```
00146                                     {
00147     Matrix res;
00148     for(uint i=0; i<3; i++)
00149         for(uint j=0; j<3; j++)
00150             res[i][j]=m1[i][j]+m2[i][j];
00151     return res;
00152 }
```



## Chapter 7

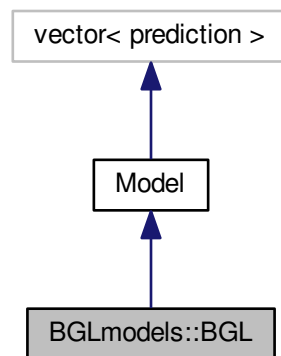
# Class Documentation

### 7.1 BGLmodels::BGL Class Reference

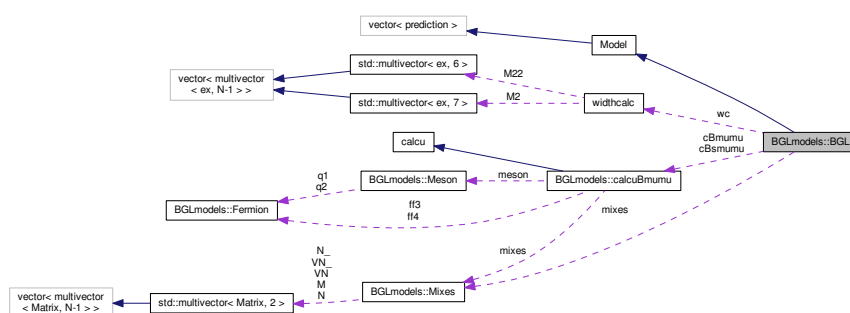
Implementation of the [BGL](#) model.

```
#include <BGL.h>
```

Inheritance diagram for BGLmodels::BGL:



Collaboration diagram for BGLmodels::BGL:



## Public Member Functions

- [BGL](#) (int genL=2, int genQ=2, int lup=0, int qup=0, int mssm=0)
- [~BGL](#) ()
- ex [Y](#) (ex x) const
- ex [GW](#) (ex x) const
- ex [GH1](#) (ex x) const
- ex [GH2](#) (ex x) const
- ex [FW](#) (ex x) const
- ex [FH1](#) (ex x) const
- ex [FH2](#) (ex x) const
- ex [Fh1](#) (ex x) const
- ex [Fh2](#) (ex x) const
- ex [A0](#) (ex x) const
- ex [A1](#) (ex x) const
- ex [A2](#) (ex x) const
- ex [A3](#) (ex x) const
- void [add](#) (const char \*s, ex pred, [observable](#) \*ob, bool sb=0)
- int [veto](#) (const [parameters](#) &p, int max=0) const
- [parameters](#) [generateparameters](#) (int max=0) const
- [parameters](#) [getlist](#) (const [parameters](#) &p) const
- double [bsgammawidth](#) (double [tanb](#), double [McH](#), double [MR](#), double [MI](#), int option=0)
- ex [decaywidth](#) (const [Fermion](#) &ff1, const [Fermion](#) &ff2, const [Fermion](#) &ff3, const [Fermion](#) &ff4, [BSpin](#) s=[sAny](#)) const
- ex [get\\_integral\\_symb](#) (const [multivector](#)< ex, 3 > &a, ex m1) const
- ex [decaywidthtest2](#) (const [Fermion](#) &ff1) const
- ex [gRR2](#) (const [Fermion](#) &f1, const [Fermion](#) &f3) const
- ex [tautomu\\_tautoe](#) () const
- ex [mesondw](#) (const [Meson](#) &meson, const [Fermion](#) &ff3, const [Fermion](#) &ff4, [BSpin](#) s=[sAny](#)) const
- ex [mesondwtest](#) (const [Meson](#) &meson, const [Fermion](#) &ff3, const [Fermion](#) &ff4, [BSpin](#) s=[sAny](#)) const
- ex [fermiontomeson](#) (const [Fermion](#) &ff4, const [Fermion](#) &ff3, const [Meson](#) &meson, [BSpin](#) s=[sAny](#)) const
- ex [fermiontomesontest](#) (const [Fermion](#) &ff4, const [Fermion](#) &ff3, const [Meson](#) &meson, [BSpin](#) s=[sAny](#)) const
- ex [mesonmixing](#) (ex mesonmass, const [Fermion](#) &f1, const [Fermion](#) &f2) const
- ex [CHdecaycoupling](#) ([Boson](#) higgs, const [Fermion](#) &ff3, const [Fermion](#) &ff4) const
- double [BranchingRatio](#) (double \*xx, double \*p)
- double [topBranchingRatio](#) (double \*xx, double \*p)

## Public Attributes

- [widthcalc](#) [wc](#)
- const double [planck](#)
- const possymbol [GF](#)
- const possymbol [MZ](#)
- const possymbol [MW](#)
- const possymbol [Mh](#)
- const constant [Mpip](#)
- const constant [Mpi0](#)
- const constant [MBp](#)
- const constant [MB0](#)
- const constant [MBs0](#)
- const constant [MKp](#)
- const constant [MK0](#)
- const constant [MDp](#)
- const constant [MD0](#)

- const constant [MDsp](#)
- const constant [MDs0](#)
- const constant [Fpi](#)
- const constant [FB](#)
- const constant [FBs](#)
- const constant [FK](#)
- const constant [FD](#)
- const constant [FDs](#)
- ex [cos2](#)
- ex [g](#)
- ex [alpha](#)
- const possymbol [tanb](#)
- const possymbol [cp](#)
- const possymbol [McH](#)
- const possymbol [MR](#)
- const possymbol [MI](#)
- const possymbol [rho](#)
- possymbol [Mu](#) [3]
- possymbol [Md](#) [3]
- vector< [Boson](#) > [bosons](#)
- lst [replacements](#)
- ex [Btaunu](#)
- ex [BR\\_Htotaunu](#)
- ex [BR\\_toptoHq](#)
- ex [BtotaunuR](#)
- ex [BtoDtaunuR](#)
- ex [BtoD2taunuR](#)
- const [Mixes mixes](#)
- lst [conjtoabs](#)
- realsymbol [mu](#)
- int [iBtaunu](#)
- int [iBDtaunu](#)
- int [iBD2taunu](#)
- vector< int > [BGLtype](#)
- double [mmax](#)
- double [stepsize](#)
- [calcuBmumu](#) \* [cBmumu](#)
- [calcuBmumu](#) \* [cBsmumu](#)

### 7.1.1 Detailed Description

Implementation of the [BGL](#) model.

Definition at line 78 of file [BGL.h](#).

## 7.1.2 Constructor & Destructor Documentation

### 7.1.2.1 BGLmodels::BGL::BGL ( int *genL* = 2, int *genQ* = 2, int *lup* = 0, int *qup* = 0, int *mssm* = 0 ) [inline]

Definition at line 81 of file [BGL.h](#).

References [BGLmodels::Boson::C](#), [std::Matrix::conjugate\(\)](#), [BGLmodels::Boson::couplingdaggerL\(\)](#), [BGLmodels::Boson::couplingL\(\)](#), [BGLmodels::Boson::couplingR\(\)](#), [BGLmodels::cParticle](#), [BGLmodels::fElectron](#), [BGLmodels::fMuon](#), [BGLmodels::fTau](#), [BGLmodels::hLeft](#), [BGLmodels::hRight](#), [BGLmodels::iDown](#), [BGLmodels::iUp](#), [BGLmodels::M\\_MW](#), [BGLmodels::M\\_MZ](#), [BGLmodels::Boson::mass](#), [BGLmodels::Boson::reset\(\)](#), [BGLmodels::Boson::s](#), [BGLmodels::sScalar](#), [BGLmodels::sVector](#), [BGLmodels::tLepton](#), and [BGLmodels::tQuark](#).

```
00081                                     :
00082         planck(6.58211928e-25),
00083         GF("G_F"),
00084         MZ("M_Z"),
00085         MW("M_W"),
00086         Mpip("Mpip",0.1396,"M_{\\pi^+}",domain::real),
00087         Mpi0("Mpi0",0.1349766,"M_{\\pi^0}",domain::real),
00088         MBp("MBp",5.279,"M_{B^+}",domain::real),
00089         MB0("MB0",5.2795,"M_{B^0}",domain::real),
00090         MBs0("MBs0",5.3663,"M_{B_s^0}",domain::real),
00091         MKp("MKp",0.493677,"MKp",domain::real),
00092         MK0("MK0",0.497614,"MK0",domain::real),
00093         MDp("MDp",1.86957,"MDp",domain::real),
00094         MD0("MD0",1.86480,"MD0",domain::real),
00095         MDsp("MDsp",1.96845,"MDsp",domain::real),
00096         MDs0("MDs0",0),
00097         Fpi("Fpi",0.132,"Fpi",domain::real),
00098         FB("FB",0.189,"FB",domain::real),
00099         FBs("FBs",0.225,"FBs",domain::real),
00100         FK("FK",0.159,"FK",domain::real),
00101         FD("FD",0.208,"FD",domain::real),
00102         FDs("FDs",0.248,"FDs",domain::real),
00103         //alpha(7.297352e-3*4*M_PI),
00104         cos2(pow(MW/MZ,2)),
00105         g(sqrt(GF*8/sqrt(ex(2)))*MW),
00106         //g(sqrt(4*Pi*alpha/(1-cos2))),
00107         tanb("tg\\beta"),
00108         cp("cp"),
00109         McH("M_{H^+}"),
00110         MR("M_{R}"),
00111         MI("M_{I}"),
00112         mixes(tanb,cp, genL,genQ, lup, qup, mssm),
00113         mu("\\mu"),
00114         BGLtype(4,0),
00115         mmmmax(1000),
00116         stepsize(1e-2)
00117         //muwidth(planck/2.197034e-6)
00118         {
00119             alpha=pow(g,2)*(1-cos2)/(4*Pi);
00120             replacements.append(GF==1.166371e-5);
00121             replacements.append(MZ==M_MZ);
00122             replacements.append(MW==M_MW);
00123
00124             mixes.appendtolst(replacements);
00125
00126             replacements.append(Pi==M_PI);
00127             replacements.append(sqrt(ex(2))==sqrt(2));
00128
00129             //cout<<pow(sqrt(2)/8*pow(g/MW,2),2)<<endl;
00130             //cout<<pow(1.166,2)<<endl;
00131
00132             Boson boson;
00133
00134             realsymbol q3("q3");
00135             ex vq3=dirac_slash(q3,4);
00136             varidx jmu(mu,4,1);
00137
00138             for(uint i=0;i<2;i++)
00139                 for(uint j=0;j<3;j++)
00140                     for(uint k=0;k<3;k++){
00141                         conjtoabs.append(conjugate(mixes.V[i][j][k])==pow(abs(
00142                             mixes.V[i][j][k]),2)/mixes.V[i][j][k]);
00143                     }
00144             /*
00145             //Gamma boson
00146             boson.mass=0;
00147             boson.s=Boson::vector;
```



```

00147
00148     boson.coupsL[0][0]=Matrix(g*sqrt(1-cos2)*0);
00149     boson.coupsL[1][1]=Matrix(g*sqrt(1-cos2)*(-1));
00150     boson.coupsL[2][2]=Matrix(g*sqrt(1-cos2)*ex(2)/3);
00151     boson.coupsL[3][3]=Matrix(g*sqrt(1-cos2)*ex(-1)/3);
00152
00153     boson.coupsR[0][0]=Matrix(g*sqrt(1-cos2)*0);
00154     boson.coupsR[1][1]=Matrix(g*sqrt(1-cos2)*(-1));
00155     boson.coupsR[2][2]=Matrix(g*sqrt(1-cos2)*ex(2)/3);
00156     boson.coupsR[3][3]=Matrix(g*sqrt(1-cos2)*ex(-1)/3);
00157
00158     bosons.push_back(boson);
00159     boson.reset();
00160     */
00161     //W+ boson
00162     boson.mass=MW;
00163     boson.s=sVector;
00164
00165     for(uint t=tLepton;t<=tQuark;t++) boson.C[t][iUp][
iDown][hLeft]=mixes.V[t]*Matrix(g/sqrt(ex(2)));
00166     Bson wboson=boson;
00167     bosons.push_back(boson);
00168     boson.reset();
00169
00170     //H+ boson
00171     boson.mass=MCH;
00172     boson.s=sScalar;
00173
00174     for(uint t=tLepton;t<=tQuark;t++)
00175     for(uint i=iUp;i<=iDown;i++) boson.C[t][iUp][iDown][i]=
mixes.VN[t][i]*Matrix(g/MW/sqrt(ex(2)));
00176     Bson chiggs=boson;
00177     bosons.push_back(boson);
00178     boson.reset();
00179
00180     for(int b=bosons.size()-1;b>=0;b--){
00181         boson.mass=bosons[b].mass;
00182         boson.s=bosons[b].s;
00183         if(boson.s==sVector)
00184             for(uint t=tLepton;t<=tQuark;t++)
00185                 for(uint i=iUp;i<=iDown;i++)
00186                     for(uint j=iUp;j<=iDown;j++)
00187                         for(uint h=hLeft;h<=hRight;h++){
00188                             boson.C[t][i][j][h]=bosons[b].C[t][j][i][h].conjugate();
00189                         }
00190             else for(uint t=tLepton;t<=tQuark;t++)
00191                 for(uint i=iUp;i<=iDown;i++)
00192                     for(uint j=iUp;j<=iDown;j++)
00193                         for(uint h=hLeft;h<=hRight;h++){
00194                             boson.C[t][i][j][hLeft]=bosons[b].C[t][j][i][
hRight].conjugate();
00195                             boson.C[t][i][j][hRight]=bosons[b].C[t][j][i][
hLeft].conjugate();
00196                         }
00197         bosons.push_back(boson);
00198         boson.reset();
00199     }
00200
00201     //(R+iI)/sqrt(2) boson
00202     boson.mass=MR;
00203     boson.s=sScalar;
00204
00205     for(uint t=tLepton;t<=tQuark;t++){
00206         boson.C[t][iDown][iDown][hRight]=mixes.
N[t][iDown]*Matrix(g/MW/ex(2));
00207         boson.C[t][iUp][iUp][hLeft]=mixes.N[t][
iUp].conjugate()*Matrix(g/MW/ex(2));
00208         boson.C[t][iDown][iDown][hLeft]=mixes.
N[t][iDown].conjugate()*Matrix(g/MW/ex(2));
00209         boson.C[t][iUp][iUp][hRight]=mixes.N[t][
iUp]*Matrix(g/MW/ex(2));
00210     }
00211     bosons.push_back(boson);
00212     boson.reset();
00213
00214     //(R+iI)/sqrt(2) boson
00215     boson.mass=MI;
00216     boson.s=sScalar;
00217
00218     for(uint t=tLepton;t<=tQuark;t++){
00219         boson.C[t][iDown][iDown][hRight]=mixes.
N[t][iDown]*Matrix(I*g/MW/ex(2));
00220         boson.C[t][iUp][iUp][hLeft]=mixes.N[t][
iUp].conjugate()*Matrix(I*g/MW/ex(2));
00221         boson.C[t][iDown][iDown][hLeft]=mixes.
N[t][iDown].conjugate()*Matrix(-I*g/MW/ex(2));
00222         boson.C[t][iUp][iUp][hRight]=mixes.N[t][

```

```

        iUp]*Matrix(-I*g/MW/ex(2));
00223     }
00224     bosons.push_back(boson);
00225     boson.reset();
00226
00227     Fermion electron(tLepton,iDown,fElectron);
00228     Fermion electronR(tLepton,iDown,fElectron,cParticle,
hRight);
00229
00230     Fermion muon(tLepton,iDown,fMuon);
00231     Fermion muonR(tLepton,iDown,fMuon,cParticle,
hRight);
00232
00233     Fermion tau(tLepton,iDown,fTau);
00234     Fermion tauR(tLepton,iDown,fTau,cParticle,
hRight);
00235     Fermion neutrino(tLepton,iUp);
00236     Fermion neutrinoR(tLepton,iUp,fTau);
00237     Fermion neutrinoMuon(tLepton,iUp,fMuon);
00238     Fermion neutrinoElectron(tLepton,iUp,fElectron);
00239
00240     Fermion up(tQuark,iUp,fElectron);
00241     Fermion down(tQuark,iDown,fElectron);
00242     Fermion bottom(tQuark,iDown,fTau);
00243     Fermion strange(tQuark,iDown,fMuon);
00244     Fermion charm(tQuark,iUp,fMuon);
00245     Fermion top(tQuark,iUp,fTau);
00246
00247     Meson Pi0d(down,down,Mpi0,Fpi);
00248     Meson Pi0u(down,down,Mpi0,Fpi);
00249     Meson Pip(up,down,Mpip,Fpi);
00250     Meson Pim(down,up,Mpip,Fpi);
00251
00252     Meson K0(down,strange,MK0,FK);
00253     Meson Kp(up,strange,MKp,FK);
00254
00255     Meson D0(charm,up,MD0,FD);
00256     Meson Dp(charm,down,MDp,FD);
00257     Meson Dsp(charm,strange,MDsp,FDs);
00258
00259     Meson B0(down,bottom,MB0,FB);
00260     Meson Bp(up,bottom,MBp,FB);
00261     Meson Bs0(strange,bottom,MBs0,FBs);
00262
00263     lst_sb;
00264     //sb.append(mixes.M[tQuark][iUp][0][0]==0);
00265     sb.append(pow(abs(mixes.V[0][2][2]),2)==1-pow(abs(mixes.V[0][1][2]),2)-pow(abs(
mixes.V[0][0][2]),2));
00266     sb.append(pow(abs(mixes.V[0][2][1]),2)==1-pow(abs(mixes.V[0][1][1]),2)-pow(abs(
mixes.V[0][0][1]),2));
00267
00268     //cout<<"Btaunu ";<<collect_common_factors(expand(Btaunu.subs(sb).subs(conjtoabs)))<<endl;
00269
00270     cout<<latex;
00271
00272     ex mutoenunu=decaywidth(muon,neutrino,electron,neutrino);
00273
00274     //cout<<"mutoenunu ";<<mutoenunu<<endl;
00275     //add("mutoenunu",decaywidth(muon,neutrino,electron,neutrino),new
gaussobs(planck/2.197034e-6,0.03));
00276
00277     add("muRtoeRnunu",gRR2(muon,electron),new limitedobs(std::pow(0.035,2),0.95));
00278
00279     //add("tautoenunu",decaywidth(tau,neutrino,electron,neutrino),new
gaussobs(planck/290.6e-15*0.1782,0.03));
00280     add("tauRtoeRnunu",gRR2(tau,electron),new limitedobs(std::pow(0.7,2),0.95));
00281
00282     //add("tautomununu",decaywidth(tau,neutrino,muon,neutrino),new
gaussobs(planck/290.6e-15*0.1739,0.03));
00283     add("tauRtomuRnunu",gRR2(tau,muon),new limitedobs(std::pow(0.72,2),0.95));
00284
00285     add("tautomu_tautoe",tautomu_tautoe(),new gaussobs(1.0018,0.0014/1.0018));
00286     //PROBLEM!!!
00287     cout<<"tautomu_tautoe: "<<1/1.0018<< "ERROR: "<<0.0014/1.0018<<endl;
00288     cout<<"ratio1 "<<tautomu_tautoe().subs(replacements)<<endl;
00289     cout<<"ratio2 "<<(decaywidth(tau,neutrino,muon,neutrino,
sVector)/decaywidth(tau,neutrino,electron,neutrino,sVector)).subs(
replacements)<<endl;
00289     //muto3e
00290     ex mu3e=decaywidth(muon,electron,electron,electron);
00291     add("muto3e", mu3e,new limitedobs(planck/2.197034e-6*1e-12));
00292     cout<<"mu3e "<<decaywidthtest2(muon)<<endl;
00293
00294     //tauto3e
00295     add("tauto3e", decaywidth(tau,electron,electron,electron),new
limitedobs(planck/290.6e-15*2.7e-8));
00296     //tauto2elmu+

```

```

00297         add("tauto2elmup", decaywidth(tau,electron,electron,muon), new
limitedobs(planck/290.6e-15*1.5e-8));
00298         //tauto2elmu-
00299         add("tauto2elmu", decaywidth(tau,electron,muon,electron), new
limitedobs(planck/290.6e-15*1.8e-8));
00300         //tauto2mule+
00301
00302         add("tauto2mulep", decaywidth(tau,muon,muon,electron), new
limitedobs(planck/290.6e-15*1.7e-8));
00303         cout<<"tauto2mulep "<<decaywidthtest2(tau)<<endl;
00304         //tauto2mule-
00305         add("tauto2mulep", decaywidth(tau,muon,electron,muon), new
limitedobs(planck/290.6e-15*2.7e-8));
00306         cout<<"tauto2mule "<<decaywidthtest2(tau)<<endl;
00307
00308         //tauto3mu
00309         add("tauto3mu", decaywidth(tau,muon,muon,muon),new
limitedobs(planck/290.6e-15*2.1e-8));
00310         //cout<<"tauto3mu "<<collect_common_factors(expand(decaywidth(tau,muon,muon,muon)))<<endl;
00311
00312
00313         ex piratio=1.2352e-4/(mesondw(Pip,neutrino,electron,sVector)/
mesondw(Pip,neutrino,muon,sVector));
00314         ex picorrection=piratio.subs(replacements);
00315         ex pierror=picorrection*0.0001/1.2352;
00316         cout<<"PiRatio "<<picorrection-1<<" +/- "<<pierror<<endl;
00317         piratio*=mesondw(Pip,neutrino,electron)/mesondw(Pip,neutrino,muon);
00318         add("piontoenu_munu",piratio,new gaussobs(1.230e-4,0.003)); //PROBLEM!!!
00319         cout<<"piontoenu_munu: "<<1.2352e-4/1.230e-4<<" ERROR: "<<0.003<<endl;
00320
00321
00322         add("tautopinu_pitomunu", (1+0.16e-2)*fermiontomeson(tau,neutrino,Pip)/
mesondw(Pip,neutrino,muon),new gaussobs((10.83e-2/290.6e-15/(0.9998770/2.6033e-8)),0.06/10.8
3));
00323         cout<<"tautopinu/pitomunu: "<<(1+0.16e-2)*(fermiontomeson(tau,neutrino,Pip,
sVector)/mesondw(Pip,neutrino,muon,sVector)).subs(
replacements)/((10.83e-2/290.6e-15/(0.9998770/2.6033e-8)))<<" ERROR: "<<0.06/10.83<<endl;
00324         cout<<"tautopinu: "<<fermiontomesontest(tau,neutrino,Pip)<<endl;
00325         cout<<"tautopinu_pitomunu: "<<(10.83e-2/290.6e-15/(0.9998770/2.6033e-8)<<" +/- "<<0.06e-2/290.6e-15/
(0.9998770/2.6033e-8)<<endl;
00326
00327         add("tautoKnu_Ktomunu", (1+0.9e-2)*fermiontomeson(tau,neutrino,Kp)/
mesondw(Kp,neutrino,muon),new gaussobs((7e-3/290.6e-15)/(0.6355/1.238e-8),0.1/7));
00328         cout<<"tautoKnu/Ktomunu: "<<(1+0.9e-2)*(fermiontomeson(tau,neutrino,Kp,
sVector)/mesondw(Kp,neutrino,muon,sVector)).subs(
replacements)/((7e-3/290.6e-15)/(0.6355/1.238e-8))<<" ERROR: "<<0.1/7<<endl;
00329         cout<<"tautoKnu/Ktomunu: "<<(7e-3/290.6e-15)/(0.6355/1.238e-8)<<" +/- "<<(0.1e-3/290.6e-15)/(0.6355
/1.238e-8)<<endl;
00330
00331         //ex
pi0toemu=(mesondecaywidth(Mpi0,down,down,electron,muon)+mesondecaywidth(Mpi0,up,up,electron,muon)+mesondecaywidth(Mpi0,up,up,electron,muon)+mesondecaywidth(Mpi0,down,down,electron,muon))/2;
00332         ex pi0toemu=(mesondw(Pi0d,electron,muon)+mesondw(Pi0d,muon,electron)+
mesondw(Pi0u,electron,muon)+mesondw(Pi0u,muon,electron))/2;
00333         add("pi0toemu",pi0toemu,new limitedobs(3.6e-10*planck/8.52e-17));
00334
00335         ex Kratio=2.477e-5/(mesondw(Kp,neutrino,electron,sVector)/
mesondw(Kp,neutrino,muon,sVector));
00336         ex Kcorrection=Kratio.subs(replacements);
00337         ex Kerror=Kcorrection*0.001/2.477;
00338         cout<<"KRatio "<<Kcorrection-1<<" +/- "<<Kerror<<endl;
00339         Kratio*=mesondw(Kp,neutrino,electron)/mesondw(Kp,neutrino,muon);
00340         add("Ktoenu_munu",Kratio,new gaussobs(2.488e-5,0.005));
00341         cout<<"Ktoenu_munu: "<<2.477e-5/2.488e-5<<" ERROR: "<<0.005<<endl;
00342
00343         ex k0Ltoemu=mesondw(K0,electron,muon)+mesondw(K0,muon,electron);
00344         add("K0Ltoemu",k0Ltoemu,new limitedobs((4.7e-12*planck/5.116e-8));
00345         add("K0Ltoee",mesondw(K0,electron,electron),new limitedobs((9e-12*
planck/5.116e-8));
00346
00347         //add("K0Ltomumu",mesondw(K0,muon,muon),new limitedobs((6.84e-9*planck/5.116e-8));
00348
00349         add("Dtoenu",mesondw(Dp,neutrino,electron),new limitedobs(8.8e-6*
planck/1040e-15));
00350         add("Dtomunu",mesondw(Dp,neutrino,muon),new gaussobs(3.82e-4*
planck/1040e-15,0.1)); //PROBLEM!!!
00351         cout<<"Dtomunu: "<<mesondw(Dp,neutrino,muon,sVector).subs(
replacements)/(3.82e-4*planck/1040e-15)<<" ERROR: "<<0.1<<endl;
00352
00353         add("Dtotaunu",mesondw(Dp,neutrino,tau),new limitedobs(1.2e-3*
planck/1040e-15)); //PROBLEM!!!
00354         cout<<"Dtotaunu: "<<mesondw(Dp,neutrino,tau,sVector).subs(
replacements)/(1.2e-3*planck/1040e-15)<<" LIMIT"<<endl;
00355
00356         //D0 2.6e-7/410.1e-15
00357
00358         ex D0toemu=mesondw(D0,electron,muon)+mesondw(D0,muon,electron);
00359         add("D0toemu",D0toemu,new limitedobs((2.6e-7*planck/410.1e-15));

```

```

00360     ex D0toee=mesondw(D0,electron,electron);
00361     add("D0toee",D0toee,new limitedobs((7.9e-8*planck/410.1e-15)));
00362     ex D0tomumu=mesondw(D0,muon,muon);
00363     add("D0tomumu",D0tomumu,new limitedobs((1.4e-7*planck/410.1e-15)));
00364
00365     //ex Dstomunu=mesondecaywidth(MDsp,strange,charm,muon,neutrino); //500e-15
00366     add("Dstomunu",mesondw(Dsp,neutrino,muon),new gaussobs(5.9e-3*
planck/500e-15,0.33/5.9)); //PROBLEM!!!
00367     cout<<"Dstomunu: "<<mesondw(Dsp,neutrino,muon,sVector).subs(
replacements)/(5.9e-3*planck/500e-15)<<" ERROR: "<<0.33/5.9<<endl;
00368
00369     add("Dstoenue",mesondw(Dsp,neutrino,electron),new limitedobs(1.2e-4*
planck/500e-15));
00370     add("Dstotaunu",mesondw(Dsp,neutrino,tau),new gaussobs(5.43e-2*
planck/500e-15,0.31/5.43)); //PROBLEM!!!
00371     cout<<"Dstotaunu: "<<mesondw(Dsp,neutrino,tau,sVector).subs(
replacements)/(5.43e-2*planck/500e-15)<<" ERROR: "<<0.31/5.43<<endl;
00372
00373     add("Btomunu",mesondw(Bp,neutrino,muon),new limitedobs(9.8e-7*
planck/1.641e-12));
00374     add("Btoenu",mesondw(Bp,neutrino,electron),new limitedobs(1e-6*
planck/1.641e-12));
00375
00376     add("Btotaunu",mesondw(Bp,neutrino,tau),new gaussobs(1.15e-4*
planck/1.641e-12,0.23/1.15));
00377     //add("Btotaunu",mesondw(Bp,neutrino,tau),new gaussobs(0.79e-4*planck/1.641e-12,0.23/1.15));
00378
00379     //calcuBmumu
00380     calculatest(mixes,Bs0,muon,muon,2.9e-9*planck/1.516e-12,0.7e-9*planck/1.516e-12,"Bs_to_mumu");
    //calcuBmumu
    calculatest2(mixes,B0,muon,muon,3.6e-10*planck/1.519e-12,1.6e-10*planck/1.516e-19,"B_to_mumu");
00381     //cout<<"TESTE "<<endl;
00382     //double ps[4]={1,1e16,1e16,1e16};
00383     //double resteste=0,resteste2=0;
00384     //int nt=4,mt=1;
00385     //calculatest.fp(&nt,ps,&mt,&resteste);
00386     //calculatest2.fp(&nt,ps,&mt,&resteste2);
00387     //cout<<"TESTE "<<resteste/(2.9e-9*planck/1.516e-12)<<"
    "<<resteste2/(3.6e-10*planck/1.519e-12)<<endl;
00388     //ex B0tomumu=mesondw(B0,muon,muon);
00389     //cout<<"B0tomumu "<<collect_common_factors(B0tomumu)<<endl;
00390     //1.65e-4
00391     //add("B0tomumu",B0tomumu,new limitedobs((8e-10*planck/1.519e-12)));
00392
00393     push_back(prediction(new calcuBmumu(mixes,B0,muon,muon,new
limitedobs(6.3e-10*planck/1.519e-12),"B_to_mumu")));
00394     //push_back(prediction(new calcuBmumu(mixes,B0,muon,muon,new
gauss2obs(3.6e-10*planck/1.519e-12,1.6e-10*planck/1.519e-12),"B_to_mumu")));
00395     push_back(prediction(new calcuBmumu(mixes,Bs0,muon,muon,new
gauss2obs(2.9e-9*planck/1.516e-12,0.7e-9*planck/1.516e-12),"Bs_to_mumu")));
00396     push_back(prediction(new calcuBmumu(mixes,K0,muon,muon,new
limitedobs(2.3e-9*planck/5.116e-8),"K0L_to_mumu")));
00397
00398     cBmumu=new calcuBmumu(mixes,B0,muon,muon,new limitedobs(6.3e-10*
planck/1.519e-12),"B_to_mumu");
00399     cBsmumu=new calcuBmumu(mixes,Bs0,muon,muon,new gauss2obs(2.9e-9*
planck/1.516e-12,0.7e-9*planck/1.516e-12),"Bs_to_mumu");
00400
00401     ex B0toetau=mesondw(B0,electron,tau)+mesondw(B0,tau,electron);
00402     add("B0toetau",B0toetau,new limitedobs((2.8e-5*planck/1.519e-12)));
00403     ex B0tomutau=mesondw(B0,muon,tau)+mesondw(B0,tau,muon);
00404     add("B0tomutau",B0tomutau,new limitedobs((2.2e-5*planck/1.519e-12)));
00405     ex B0toee=mesondw(B0,electron,electron);
00406     add("B0toee",B0toee,new limitedobs((8.3e-8*planck/1.519e-12)));
00407     ex B0totatautau=mesondw(B0,tau,tau);
00408     add("B0totatautau",B0totatautau,new limitedobs((4.1e-3*planck/1.519e-12)));
00409
00410     //Bs m=5.3663, life=1.472e-12 emu=2e-7, ee=2.8e-7 mumu=4.2e-8
00411     ex Bs0toemu=mesondw(Bs0,electron,muon)+mesondw(Bs0,muon,electron);
00412     add("Bs0toemu",Bs0toemu,new limitedobs((2e-7*planck/1.516e-12)));
00413     ex Bs0toee=mesondw(Bs0,electron,electron);
00414     add("Bs0toee",Bs0toee,new limitedobs((2.8e-7*planck/1.516e-12)));
00415     // ex Bs0tomumu=mesondw(Bs0,muon,muon);
00416     // add("Bs0tomumu",Bs0tomumu,new limitedobs((3.2e-9*planck/1.516e-12)));
00417
00418     // add("chargedHiggs",pow(McH,-2),new limitedobs(std::pow(80.0,-2),0.9));
00419
00420     cout<<"Bs0tomumu: "<<mesondwtest(Bs0,muon,muon)<<endl;
00421     //add("chargedHiggs",1/McH,new limitedobs(1/80.0,0));
00422
00423     /*
00424     Matrix llgamma2loop=Matrix(sqrt(ex(2))*mixes.N[tQuark][iUp][2][2]*mixes.M[tQuark][iUp][fTau][fTau]*
pow(1/McH*log(mixes.M[tQuark][iUp][fTau][fTau]/McH),2))*mixes.N[tLepton][iDown];
00425     for(uint i=0;i<3;i++)
00426         for(uint j=0;j<3;j++)
00427             if(j<i) llgamma2loop[i][j]=ex(3)*pow(g*g*(1-cos2)/4/Pi/Pi,3)*llgamma2loop[j][i]*
llgamma2loop[j][i].conjugate()/pow(mixes.M[tLepton][iDown][i][i],2);

```

```

00428         else llgamma2loop[i][j]=0;
00429         add("mutoegamma", llgamma2loop[1][0], new limitedobs(1.2e-11));
00430         add("tautoegamma", llgamma2loop[2][0], new limitedobs(3.3e-8));
00431         add("tautomugamma", llgamma2loop[2][1], new limitedobs(4.4e-8));
00432         //add("mutoegamma", llgamma2loop[1][0], new limitedobs(planck/2.197034e-6*1.2e-11));
00433         //add("tautoegamma", llgamma2loop[2][0], new limitedobs(planck/290.6e-15*3.3e-8));
00434         //add("tautomugamma", llgamma2loop[2][1], new limitedobs(planck/290.6e-15*4.4e-8));
00435     */
00436
00437     Matrix llgammaCH;
00438     //Matrix
00439     llgammaCH=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH*12))*mixes.M[tLepton][iDown]*mixes.VN[tLepton][1].conjugate()*mixes.VN[tLepton][1];
00440     Matrix llgammaHOM, llgammaHOE;
00441     /*for(uint i=0; i<3; i++)
00442         for(uint j=0; j<3; j++)
00443             for(uint k=0; k<3; k++){
00444                 ex z=pow(fmasses[1][i][j]/McH, 2); mixes.M[tQuark][iUp][i][j]/MR, 2);
00445                 llgammaHOM[j][k]=llgammaHOM[j][k]+(mixes.VN[0][1][i][j]*
00446                 mixes.VN[0][1][k][i]+mixes.VN[0][1][i][j]*mixes.VN[0][1][i][k].conjugate())/pow(mixes.M[tLepton][iDown][i][j], 2)*(
00447                 *z+6*z*log(z))/6;
00448                 llgammaHOM[j][k]=llgammaHOM[j][k]+(mixes.VN[0][1][i][j]*
00449                 mixes.VN[0][1][k][i])/mixes.M[tLepton][iDown][i][j]/mixes.M[tLepton][iDown][j][j]*(3*z+2*z*log(z));
00450                 llgammaHOE[j][k]=llgammaHOE[j][k]+(mixes.VN[0][1][i][j].conjugate()*
00451                 mixes.VN[0][1][k][i]-mixes.VN[0][1][i][j]*mixes.VN[0][1][i][k].conjugate())/pow(mixes.M[tLepton][iDown][i][j], 2)*(
00452                 *z+6*z*log(z))/6;
00453                 llgammaHOE[j][k]=llgammaHOE[j][k]+(mixes.VN[0][1][i][j]*
00454                 mixes.VN[0][1][k][i])/mixes.M[tLepton][iDown][i][j]/mixes.M[tLepton][iDown][j][j]*(3*z+2*z*log(z));
00455             }
00456     */
00457     llgammaHOM=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH))*
00458     mixes.M[tLepton][iDown]*llgammaHOM;
00459     llgammaHOE=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH))*
00460     mixes.M[tLepton][iDown]*llgammaHOE;
00461
00462     Matrix llgamma, llgamma2;
00463
00464     for(uint i=0; i<3; i++)
00465         for(uint j=0; j<3; j++){
00466             //if(j<i)
00467             llgamma[i][j]=(llgammaCH[i][j]*llgammaCH[i][j].conjugate()+llgammaHOE[i][j]*llgammaHOM[i][j].conjugate()+llgammaHOM[i][j]*llgammaHOE[i][j].conjugate()+g*g*(1-cos2)*pow((pow(mixes.M[tLepton][iDown][i][j])
00468             ex mmuon=mixes.M[tLepton][iDown][i][j];
00469             ex A,B;
00470
00471             if(j<i){ for(uint k=0; k<3; k++){
00472                 ex mtau=mixes.M[tLepton][iDown][k][k];
00473                 B+=-mixes.VN[tLepton][1][k][j].conjugate()*
00474                 mixes.VN[tLepton][1][k][i]/(12*pow(McH, 2));
00475                 B+=mixes.N[tLepton][1][k][j].conjugate()*
00476                 mixes.N[tLepton][1][k][i]/12*(pow(MR, -2)+pow(MI, -2));
00477                 A+=mixes.N[tLepton][1][j][k]*mixes.N[tLepton][1][i][k].conjugate()*
00478                 (pow(MR, -2)+pow(MI, -2))/12;
00479                 A+=mixes.N[tLepton][1][j][k]*mixes.N[tLepton][1][i][k]/mtau/mmuon*(Fh2(pow(mtau/MR, 2))-Fh2(pow(mtau/MI, 2)))/4;
00480             }
00481             llgamma[i][j]=(A*A.conjugate()+B*B.conjugate())*alpha*pow(mmuon, 5)*
00482             GF*GF/(128*pow(Pi, 4));
00483         }
00484     else if(j==i){
00485         for(uint k=0; k<3; k++){
00486             ex mtau=mixes.M[tLepton][iDown][k][k];
00487             B+=-mixes.VN[tLepton][1][k][j].conjugate()*
00488             mixes.VN[tLepton][1][k][i]/(12*pow(McH, 2));
00489             B+=mixes.N[tLepton][1][k][j].conjugate()*
00490             mixes.N[tLepton][1][k][i]/12*(pow(MR, -2)+pow(MI, -2));
00491             B+=mixes.N[tLepton][1][j][k].conjugate()*
00492             mixes.N[tLepton][1][i][k]/12*(pow(MR, -2)+pow(MI, -2));
00493         }
00494         llgamma[i][j]=-B*GF*sqrt(1/2)/(8*pow(Pi, 2))*2*mmuon; //e (GeV)^-1=1/(516) (e cm)
00495     where e=sqrt(alpha*4*Pi)
00496     }
00497     }
00498     add("mutoegamma", llgamma[1][0], new limitedobs(planck/2.197034e-6*2.4e-12));
00499     add("tautoegamma", llgamma[2][0], new limitedobs(planck/290.6e-15*3.3e-8));
00500     add("tautomugamma", llgamma[2][1], new limitedobs(planck/290.6e-15*4.4e-8));
00501
00502     //add("d_e", abs(llgamma[0][0].imag_part()), new limitedobs(10.5e-28*516));
00503     add("d_mu", abs(llgamma[1][1].imag_part()), new limitedobs(1.9e-19*516));
00504     add("d_tau", llgamma[2][2].imag_part(), new gaussobs(-0.85e-17*516, 0.825/0.85));
00505     cout<<"EDM: "<<llgamma[0][0].subs(conjtoabs).subs(replacements).imag_part()<<endl;
00506     add("a_mu", -llgamma[1][1].real_part()*2*mixes.M[tLepton][iDown][1][1], new gaussobs(3e-9, 1.0/3.0));
00507     */
00508     //llgammaCH=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH*12))*mixes.M[tQuark][iDown]*
00509     mixes.VN[1][1].conjugate()*mixes.VN[1][1]; //4+1
00510     //Matrix llgammaHOM, llgammaHOE;

```

```

00494         for(uint i=0;i<3;i++)
00495             for(uint j=0;j<3;j++)
00496                 for(uint k=0;k<3;k++){
00497                     ex z=pow(mixes.M[tQuark][iUp][i][i]/MR,2);
00498                     llgammaHOM[j][k]=llgammaHOM[j][k]+(mixes.VN[1][1][j][i].conjugate()*
mixes.VN[1][1][k][i]+mixes.VN[1][1][i][j]*mixes.VN[1][1][i][k].conjugate())/pow(mixes.M[tQuark][iDown][i][i],2)*(2*
z+6*z*z*log(z))/6;
00499                     llgammaHOM[j][k]=llgammaHOM[j][k]+(mixes.VN[1][1][i][j]*
mixes.VN[1][1][k][i])/mixes.M[tQuark][iDown][i][i]/mixes.M[tQuark][iDown][j][j]*(3*z+2*z*log(z));
00500
00501                     llgammaHOE[j][k]=llgammaHOE[j][k]+(mixes.VN[1][1][j][i].conjugate()*
mixes.VN[1][1][k][i]-mixes.VN[1][1][i][j]*mixes.VN[1][1][i][k].conjugate())/pow(mixes.M[tQuark][iDown][i][i],2)*(2*
z+6*z*z*log(z))/6;
00502                     llgammaHOE[j][k]=llgammaHOE[j][k]+(mixes.VN[1][1][i][j]*
mixes.VN[1][1][k][i])/mixes.M[tQuark][iDown][i][i]/mixes.M[tQuark][iDown][j][j]*(3*z+2*z*log(z));
00503                 }
00504                 llgammaHOM=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH))*mixes.M[tQuark][iDown]*llgammaHOM;
00505                 llgammaHOE=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH))*mixes.M[tQuark][iDown]*llgammaHOE;
00506
00507                 //Matrix llgamma;
00508                 for(uint i=0;i<3;i++)
00509                     for(uint j=0;j<3;j++)
00510                         if(j<i) {llgamma[i][j]=(llgammaCH[i][j]*llgammaCH[i][j].conjugate()+llgammaHOE[i][j]*
llgammaHOE[i][j].conjugate()+llgammaHOM[i][j]*llgammaHOM[i][j].conjugate())*g*g*(1-cos2)*
pow((pow(mixes.M[tQuark][iDown][i][i],2)-pow(mixes.M[tQuark][iDown][j][j],2))/mixes.M[tQuark][iDown][i][i],3)/(4*Pi)*
00511                         //llgamma[i][j]=llgamma[i][j].subs(1st(abs(wild()*pow(MH0,-2)))==abs(wild())
*pow(MH0,-2)));
00512                     }
00513                     else llgamma[i][j]=0;
00514                 */
00515
00516
00517                 push_back(prediction(new calculbtosgamma2(mixes)));
00518
00519                 //add("btosgamma",llgamma[2][1],new gaussobs(3.55e-4,sqrt(2)*0.25/3.55),1);
00520                 //cout<<csrc<<llgamma[2][1]<<endl;
00521                 //cout<<latex;
00522
00523
00524                 BR_Htotaunu=(CHdecaycoupling(chiggs,tau,neutrino)+3*
CHdecaycoupling(chiggs, strange, charm))/factor(CHdecaycoupling(chiggs,Fermion(
tLepton,iDown),neutrino)+3*CHdecaycoupling(chiggs,Fermion(
tQuark,iDown),charm)+3*CHdecaycoupling(chiggs,Fermion(
tQuark,iDown),up));
00525                 BR_Htotaunu=BR_Htotaunu.subs(replacements);
00526
00527                 //BR_toptoHq=decaywidth(top,bottom,chiggs);
00528                 //ex toptoWb=decaywidth(top,bottom,wboson);
00529                 //BR_toptoHq=BR_toptoHq/(BR_toptoHq+toptoWb);
00530                 //BR_toptoHq=BR_toptoHq.subs(replacements);
00531
00532                 //cout<<"toptoWb "<<toptoWb.subs(replacements).evalf()<<endl;
00533
00534                 //b to c tau- nu/b to c e- nu
00535                 //ex
btocR=decaywidth(bottom,charm,tau,neutrino,sVector)/(decaywidth(bottom,charm,electron,neutrino,sVector)+decaywidth(bot
00536                 //cout<<btocR.subs(replacements)<<endl;
00537
00538                 ex BtoDtaunu,BtoD2taunu, BtoDtaunuSM, KtoPi;
00539                 for(uint i=0; i<3; i++){
00540                     ex Wcoup=wboson.couplingL(charm,bottom)*wboson.couplingdaggerL(tau,Fermion(
tLepton,iUp,FFlavour(i)));
00541                     if(Wcoup.subs(replacements)==ex(0)) continue;
00542                     ex chcoup_Wcoup=-pow(MW/McH,2)*(chiggs.couplingR(charm,bottom)+chiggs.couplingL(charm,
bottom))*chiggs.couplingdaggerL(tau,Fermion(tLepton,iUp,FFlavour(i)))/Wcoup;
00543                     ex chcoup2_Wcoup=-pow(MW/McH,2)*(chiggs.couplingR(charm,bottom)-chiggs.couplingL(charm
,bottom))*chiggs.couplingdaggerL(tau,Fermion(tLepton,iUp,FFlavour(i)))/Wcoup;
00544
00545                     BtoDtaunuSM+=Wcoup*Wcoup.conjugate();
00546                     BtoDtaunu+=Wcoup*Wcoup.conjugate()*(1+1.5*chcoup_Wcoup.real_part()+chcoup_Wcoup.conjugate()
*chcoup_Wcoup);
00547                     BtoD2taunu+=Wcoup*Wcoup.conjugate()*(1+0.12*chcoup2_Wcoup.real_part()+0.05*chcoup2_Wcoup.
conjugate()*chcoup2_Wcoup);
00548                 }
00549                 1st r2(pow(mixes.V[1][1][2].imag_part(),2)==pow(abs(mixes.V[1][1][2]),2)-pow(
mixes.V[1][1][2].real_part(),2));
00550                 r2.append(pow(mixes.V[0][2][2].imag_part(),2)==pow(abs(mixes.
V[0][2][2]),2)-pow(mixes.V[0][2][2].real_part(),2));
00551
00552                 r2.append(mixes.M[1][0][1][1]==0);
00553                 r2.append(pow(abs(mixes.V[0][2][2]),2)==1-pow(abs(mixes.V[0][1][2]),2)-pow(abs(
mixes.V[0][0][2]),2));
00554                 r2.append(pow(abs(mixes.V[0][2][1]),2)==1-pow(abs(mixes.V[0][1][1]),2)-pow(abs(
mixes.V[0][0][1]),2));
00555                 r2.append(abs(sqrt(ex(2))*GF)==sqrt(ex(2))*GF);
00556
00557                 BtoDtaunuSM=collect_common_factors(BtoDtaunuSM.subs(conjtoabs).subs(r2));

```

```

00558         BtoDtaunu=collect_common_factors(BtoDtaunu.subs(conjtoabs).subs(r2));
00559
00560         BtoDtaunuR=(BtoDtaunu/BtoDtaunuSM).subs(replacements).real_part();
00561
00562         BtoD2taunu=BtoD2taunu.subs(conjtoabs).subs(r2);
00563         BtoD2taunuR=(BtoD2taunu/BtoDtaunuSM).subs(replacements).real_part();
00564
00565
00566         //cout<<"BtoDtaunu/BtoDtaunuSM "<<expand(BtoDtaunu/BtoDtaunuSM)<<endl;
00567         iBDtaunu=size();
00568         add("BtoDtaunu_BtoDtaunuSM",BtoDtaunu/BtoDtaunuSM,new gaussobs(440.0/296, 1.4*58.0/440))
;
00569
00570         iBD2taunu=size();
00571         //cout<<"BtoD2taunu/BtoD2taunuSM
"<<1+collect_common_factors(expand(BtoD2taunu/BtoDtaunuSM-1))<<endl;
00572         add("BtoD2taunu_BtoD2taunuSM",BtoD2taunu/BtoDtaunuSM,new gaussobs(332.0/252, 1.4*24.0/33
2.0));
00573
00574
00575
00576         for(uint j=0; j<2; j++){
00577             ex KtoPimunu, KtoPimunuSM;
00578             for(uint i=0; i<3; i++){
00579                 ex Wcoup=wboson.couplingL(up, strange)*wboson.couplingdaggerL(Fermion(
tLepton,iDown,Fflavour(j)),Fermion(tLepton,iUp,
FFlavour(i)));
00580                 if(Wcoup.subs(replacements)==ex(0)) continue;
00581                 ex chcoup_Wcoup=-pow(MW/McH,2)*(chiggs.couplingR(up, strange)+chiggs.couplingL(up,
strange))\
00582                     *chiggs.couplingdaggerL(muon,Fermion(tLepton,
iUp,FFlavour(i)))/Wcoup*(pow(MKp,2)-pow(Mpip,2))\
00583                     /(mixes.mass(Fermion(tLepton,
iDown,FFlavour(j)))*(mixes.mass(strange)-mixes.mass(up)));
00584                 chcoup_Wcoup=collect_common_factors(expand(chcoup_Wcoup));
00585                 KtoPimunuSM+=collect_common_factors(expand(Wcoup*Wcoup.conjugate()));
00586                 KtoPimunu+=collect_common_factors(expand(Wcoup*Wcoup.conjugate()*pow(1+chcoup_Wcoup,2)));
00587             }
00588             KtoPimunuSM=collect_common_factors(expand(KtoPimunuSM.subs(conjtoabs).subs(r2)));
00589             KtoPimunu=collect_common_factors(expand(KtoPimunu.subs(conjtoabs).subs(r2)));
00590             KtoPimunu=expand(KtoPimunu.subs(replacements).real_part().subs(1st(abs(wild()*pow(
MR,-2))==abs(wild()*pow(MR,-2)).subs(1st(log(wild()*pow(MR,-2))==log(wild())-2*log(
MR))));
00591             KtoPimunu=expand(KtoPimunu.evalf());
00592             KtoPimunuSM=expand(KtoPimunuSM.subs(replacements).real_part().subs(1st(abs(wild()*
pow(MR,-2))==abs(wild()*pow(MR,-2)).subs(1st(log(wild()*pow(MR,-2))==log(wild())-2*log(
MR))));
00593             KtoPimunuSM=expand(KtoPimunuSM.evalf());
00594             KtoPi+=0.5*log(KtoPimunu/KtoPimunuSM);
00595         }
00596
00597         add("KtoPi",KtoPi/(pow(MKp,2)-pow(Mpip,2)),new gaussobs(0.08, 0.11/0.08));
00598
00599
00600         //add("b to c tau- nu/b to c e- nu", decaywidth(bottom,charm,electron,neutrino), new
limitedobs(planck/290.6e-15*2.7e-8));
00601
00602         double fD=0.207;
00603         ex DDbbar=ex(std::pow(fD,2))*mesonmixing(MD0,charm,up);
00604         DDbbar=expand(DDbbar.subs(replacements).subs(1st(abs(wild()*pow(MR,-2))==abs(wild())*pow(
MR,-2)).subs(1st(log(wild()*pow(MR,-2))==log(wild())-2*log(MR))));
00605         DDbbar=expand(DDbbar.evalf());
00606         ex aDDbbar=sqrt(DDbbar.real_part()*DDbbar.real_part()+DDbbar.imag_part()*DDbbar.imag_part());
00607         add("DDbbar",aDDbbar,new limitedobs(9.47e-15));
00608         cout<<DDbbar<<endl;
00609         //2|M12|<6.6e-15GeV
00610
00611         double fK=0.156;
00612         ex KKbar=ex(std::pow(fK,2))*mesonmixing(MK0,strange,down);
00613         KKbar=expand(KKbar.subs(replacements).subs(1st(abs(wild()*pow(MR,-2))==abs(wild())*pow(
MR,-2)).subs(1st(log(wild()*pow(MR,-2))==log(wild())-2*log(MR))));
00614         KKbar=expand(KKbar.evalf());
00615         ex aKKbar=sqrt(KKbar.real_part()*KKbar.real_part()+KKbar.imag_part()*KKbar.imag_part());
00616         add("KKbar",aKKbar,new limitedobs(3.5e-15));
00617         ex eK=0.94*imag_part(KKbar)/3.5e-15/sqrt(2);
00618         //add("a_eK",abs(eK),new limitedobs(2.2e-3));
00619         add("a_eK",abs(eK),new limitedobs(20*0.0114e-3));
00620         cout<<abs(KKbar)<<endl;
00621
00622         double fB=0.189;
00623         ex Vtb=mixes.V[tQuark][2][2]/mixes.V[tQuark][2][2].conjugate();
00624         ex Vtd=mixes.V[tQuark][2][0]/mixes.V[tQuark][2][0].conjugate();
00625         ex Vts=mixes.V[tQuark][2][1]/mixes.V[tQuark][2][1].conjugate();
00626
00627         ex BBbar=1+ex(std::pow(fB,2))*mesonmixing(MB0,bottom,down)/(3.337e-13*Vtb*Vtd.conjugate()
);
00628         add("BBbarimag",imag_part(BBbar),new gauss2obs(-0.199,0.062));

```

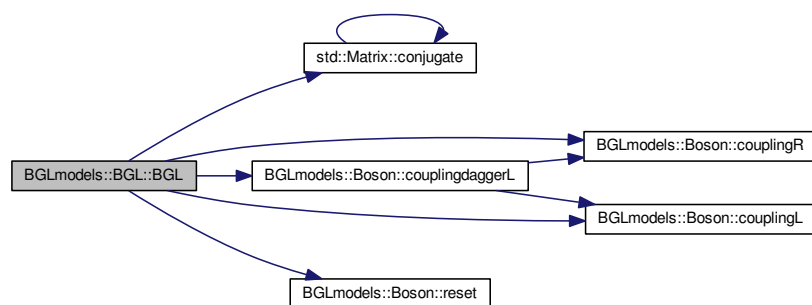


```

00629         add("BBbarreal", real_part(BBbar), new gauss2obs(0.823, 0.143));
00630         cout<<BBbar<<endl;
00631         BBbar=3.337e-13*Vtb*Vtd.conjugate();
00632         cout<<"Bbar " << (abs(imag_part(BBbar))/abs(BBbar)).subs(replacements)<<endl;
00633         double fBs=0.225;
00634         ex BsBsbar=1+ex(std::pow(fBs, 2))*mesonmixing(MBs0, bottom, strange)/(1.186e-11*Vtb*Vts.
conjugate());
00635         add("BsBsbarimag", imag_part(BsBsbar), new gauss2obs(0, 0.1));
00636         add("BsBsbarreal", real_part(BsBsbar), new gauss2obs(0.965, 0.133));
00637         cout<<BsBsbar<<endl;
00638         BsBsbar=1.186e-11*Vtb*Vts.conjugate();
00639         cout<<"Bbar " << (abs(imag_part(BsBsbar))/abs(BsBsbar)).subs(replacements)<<endl;
00640
00641         ex McH2=MCH*MCH;
00642         ex MR2=MR*MR;
00643         ex MI2=MI*MI;
00644
00645         ex cu=collect_common_factors(expand(chiggs.couplingL(top, bottom)))/mixes.
mass(top)/(g/MW/sqrt(ex(2)))/mixes.V[1][2][2];
00646         cout<<"cu " <<cu<<endl;
00647         ex Zbb=(cu-0.72)/MCH;
00648         add("Zbb", Zbb, new limitedobs(0.0024));
00649         cout<<"Zbb " <<Zbb<<endl;
00650         cout<<"SIZE " <<size()<<endl;
00651
00652         push_back(prediction(new calcuOblique()));
00653     }

```

Here is the call graph for this function:



### 7.1.2.2 BGLmodels::BGL::~~BGL( ) [inline]

Definition at line 655 of file [BGL.h](#).

```

00655     {
00656         delete cBmuu;
00657         delete cBsmumu;
00658     }

```

## 7.1.3 Member Function Documentation

### 7.1.3.1 ex BGLmodels::BGL::A0( ex x ) const [inline]

Definition at line 700 of file [BGL.h](#).

```

00700     {
00701         return x*(2+3*x-6*x*x+ x*x*x+6*x*log(x))/(24*pow(1-x, 4));
00702     }

```



7.1.3.2 `ex BGLmodels::BGL::A1 ( ex x ) const` `[inline]`

Definition at line 704 of file [BGL.h](#).

```
00704         {
00705         return x*(-3+4*x-x*x-2*log(x))/(4*pow(1-x,3));
00706     }
```

7.1.3.3 `ex BGLmodels::BGL::A2 ( ex x ) const` `[inline]`

Definition at line 708 of file [BGL.h](#).

```
00708         {
00709         return x/(6*pow(1-x,3))*((-7+5*x-8*x*x)/6.0+x*log(x)/(1-x)*(-2+3*x));
00710     }
```

7.1.3.4 `ex BGLmodels::BGL::A3 ( ex x ) const` `[inline]`

Definition at line 712 of file [BGL.h](#).

```
00712         {
00713         return (-3+8*x-5*x*x+(6*x-4)*log(x))*x/(6*pow(1-x,3));
00714     }
```

7.1.3.5 `void BGLmodels::BGL::add ( const char * s, ex pred, observable * ob, bool sb = 0 )` `[inline]`

Definition at line 716 of file [BGL.h](#).

```
00716         {
00717         //cout<<s<<endl;
00718         //cout<<"prediction symb"<<pred<<endl;
00719         //,pow(sin(wild()), 2) == 1-pow(cos(wild()), 2)
00720         //ex
00721         p=expand(pred.subs(replacements).real_part()).subs(1st(abs(wild()*pow(MR,-2))==abs(wild())*pow(MR,-2))).subs(1st(log(wi
00722         ex p=pred.subs(replacements).real_part());
00723         p=collect_common_factors(expand(p.evalf()));
00724         FUNCP_CUBA fp;
00725
00726         lst l(tanb,McH,MR,MI);
00727
00728         for(uint i=0;i<3;i++){
00729             l.append(Mu[i]);
00730             l.append(Md[i]);
00731         }
00732         if(sb) push_back(prediction(ob,p));
00733         else {
00734             compile_ex(lst(p), l, fp);
00735             //cout<<"prediction numeric"<<p<<endl;
00736             //cout<<"exp "<<ob->expected()<<endl<<endl;
00737             push_back(prediction(ob,fp));
00738         }
00739     }
```

### 7.1.3.6 double BGLmodels::BGL::BranchingRatio ( double \* xx, double \* p ) [inline]

Definition at line 1477 of file BGL.h.

```
01477 {
01478     return ex_to<numeric>(BR_Htotaunu.subs(tanb==pow(10.0,xx[0])).evalf()).to_double();
01479 }
```

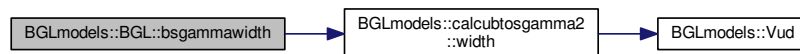
### 7.1.3.7 double BGLmodels::BGL::bsgammawidth ( double tanb, double McH, double MR, double MI, int option = 0 ) [inline]

Definition at line 799 of file BGL.h.

References [BGLmodels::calcubtosgamma2::width\(\)](#).

```
00799 {
00800     parameters p=generateparameters();
00801     p[0].value=pow(10.0,tanb);
00802     p[1].value=McH;
00803     p[2].value=MR;
00804     p[3].value=MI;
00805
00806     calcubtosgamma2 cal(mixes);
00807
00808     return cal.width(p,option);
00809 }
```

Here is the call graph for this function:



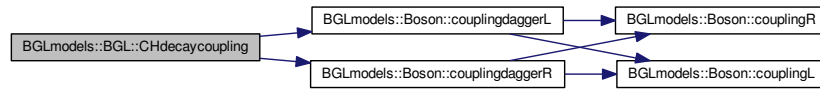
### 7.1.3.8 ex BGLmodels::BGL::CHdecaycoupling ( Boson higgs, const Fermion & ff3, const Fermion & ff4 ) const [inline]

Definition at line 1461 of file BGL.h.

References [BGLmodels::Boson::couplingdaggerL\(\)](#), [BGLmodels::Boson::couplingdaggerR\(\)](#), [BGLmodels::fAny](#), [BGLmodels::fElectron](#), [BGLmodels::Fermion::flavour](#), and [BGLmodels::fTau](#).

```
01461 {
01462
01463     Fermion f3=ff3, f4=ff4;
01464     ex ret=0;
01465     for(uint k=fElectron;k<=fTau;k++){
01466         if(ff3.flavour==fAny || ff3.flavour==k){
01467             f3.flavour=(FFlavour)k;
01468             for(uint l=fElectron;l<=fTau;l++){
01469                 if(ff4.flavour==fAny || ff4.flavour==l){
01470                     f4.flavour=(FFlavour)l;
01471                     ret+=higgs.couplingdaggerL(f3,f4)*higgs.couplingdaggerR(f3,f4).conjugate()+higgs.
01472 couplingdaggerR(f3,f4)*higgs.couplingdaggerL(f3,f4).conjugate();
01473                 }
01474             }
01475         }
01476     }
01477     return collect_common_factors(ret.subs(conjtoabs));
01478 }
```

Here is the call graph for this function:



#### 7.1.3.9 ex BGLmodels::BGL::decaywidth ( const Fermion & ff1, const Fermion & ff2, const Fermion & ff3, const Fermion & ff4, BSpin s = sAny ) const [inline]

Definition at line 879 of file BGL.h.

References [BGLmodels::fAny](#), [BGLmodels::fElectron](#), [BGLmodels::Fermion::flavour](#), [BGLmodels::fTau](#), [BGLmodels::Boson::mass](#), [BGLmodels::Boson::s](#), and [BGLmodels::sAny](#).

```

00879
00880 {
00881     multivector<ex,4> a(0,bosons.size(),2,2,2);
00882     vector<ex> mass(bosons.size(),0);
00883     vector<int> op(bosons.size(),0);
00884     ex ret=0;
00885     Fermion f1=ff1, f2=ff2,f3=ff3, f4=ff4;
00886
00887     for(uint i=fElectron;i<=fTau;i=i+1)
00888     if(ff1.flavour==fAny || ff1.flavour==i){
00889         f1.flavour=(Fflavour)i;
00890     for(uint j=fElectron;j<=fTau;j++)
00891     if(ff2.flavour==fAny || ff2.flavour==j){
00892         f2.flavour=(Fflavour)j;
00893     for(uint k=fElectron;k<=fTau;k++)
00894     if(ff3.flavour==fAny || ff3.flavour==k){
00895         f3.flavour=(Fflavour)k;
00896     for(uint l=fElectron;l<=fTau;l++)
00897     if(ff4.flavour==fAny || ff4.flavour==l){
00898         f4.flavour=(Fflavour)l;
00899     for(uint i=0;i<bosons.size();i++) if(bosons[i].s==s || s==
sAny) {
00900         op[i]=bosons[i].s;
00901         mass[i]=bosons[i].mass;
00902         a[i][0][0][0]=bosons[i].couplingdaggerL(f2,f1)*
bosons[i].couplingL(f3,f4);
00903         a[i][0][0][1]=bosons[i].couplingdaggerL(f2,f1)*
bosons[i].couplingR(f3,f4);
00904         a[i][0][1][0]=bosons[i].couplingdaggerR(f2,f1)*
bosons[i].couplingL(f3,f4);
00905         a[i][0][1][1]=bosons[i].couplingdaggerR(f2,f1)*
bosons[i].couplingR(f3,f4);
00906         a[i][1][0][0]=bosons[i].couplingdaggerL(f3,f1)*
bosons[i].couplingL(f2,f4);
00907         a[i][1][0][1]=bosons[i].couplingdaggerL(f3,f1)*
bosons[i].couplingR(f2,f4);
00908         a[i][1][1][0]=bosons[i].couplingdaggerR(f3,f1)*
bosons[i].couplingL(f2,f4);
00909         a[i][1][1][1]=bosons[i].couplingdaggerR(f3,f1)*
bosons[i].couplingR(f2,f4);
00910     }
00911     }
00912     ret+=wc.get_integral_symb(a,mass,op,mixes.mass(f1));
00913     //
00914     ret+=wc.get_integral(a,mass,op,mixes.massnum(f1),mixes.massnum(f2),mixes.massnum(f3),mixes.massnum(f4))/pow(mixes.massnum(f1),2);
00915     }
00916     if(ff2.flavour==ff4.flavour) ret=ret/2;
00917     return collect_common_factors(ret.subs(conjtoabs));
00918     //return
    expand(ret.subs(1st(exp(-I*wild()))==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
00919 }

```

### 7.1.3.10 `ex BGLmodels::BGL::decaywidthtest2 ( const Fermion & ff1 ) const [inline]`

Definition at line 989 of file [BGL.h](#).

```

00989                                     {
00990         multivector<ex,3> a(0,2,2,2);
00991         symbol gLL("g_{LL}"), gLR("g_{LR}"), gRL("g_{RL}"), gRR("g_{RR}"), cLL("c_{LL}"), cLR("c_{LR}"), cRL("
c_{RL}"), cRR("c_{RR}");
00992
00993         a[0][0][0]=gLL;
00994         a[0][0][1]=gLR;
00995         a[0][1][0]=gRL;
00996         a[0][1][1]=gRR;
00997
00998         a[1][0][0]=cLL;
00999         a[1][0][1]=cLR;
01000         a[1][1][0]=cRL;
01001         a[1][1][1]=cRR;
01002
01003         ex ret=get_integral_symb(a,mixes.mass(ff1));
01004         //
ret+=wc.get_integral(a,mass,op,mixes.massnum(f1),mixes.massnum(f2),mixes.massnum(f3),mixes.massnum(f4))/pow(mixes.massnum(f1),mixes.massnum(f2),mixes.massnum(f3),mixes.massnum(f4));
01005
01006         return collect_common_factors(ret.subs(conjtoabs));
01007         //return
expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
01008 }

```

### 7.1.3.11 `ex BGLmodels::BGL::fermiontomeson ( const Fermion & ff4, const Fermion & ff3, const Meson & meson, BSpin s=sAny ) const [inline]`

Definition at line 1314 of file [BGL.h](#).

References [BGLmodels::Boson::couplingL\(\)](#), [BGLmodels::Boson::couplingR\(\)](#), [BGLmodels::Meson::decay\\_factor](#), [BGLmodels::fAny](#), [BGLmodels::fElectron](#), [BGLmodels::Fermion::flavour](#), [BGLmodels::fTau](#), [BGLmodels::Meson::mass](#), [BGLmodels::Boson::mass](#), [BGLmodels::Meson::q1](#), [BGLmodels::Meson::q2](#), [BGLmodels::Boson::s](#), and [BGLmodels::sAny](#).

```

01314                                     {
01315
01316         const Fermion& f1(meson.q1), f2(meson.q2);
01317         ex mesonmass=meson.mass;
01318
01319         Fermion f3=ff3, f4=ff4;
01320
01321         ex ret=0;
01322
01323         realsymbol q3("q3"), q4("q4");
01324         ex s2=pow(mesonmass,2);
01325
01326         for(uint k=fElectron;k<=fTau;k++){
01327             if(ff3.flavour==fAny || ff3.flavour==k){
01328                 f3.flavour=(Fflavour)k;
01329             for(uint l=fElectron;l<=fTau;l++){
01330                 if(ff4.flavour==fAny || ff4.flavour==l){
01331                     f4.flavour=(Fflavour)l;
01332                     ex v1=0, v2=0;
01333                     ex mq1=mixes.mass(f1),mq2=mixes.mass(f2),mq3=
mixes.mass(f3),mq4=mixes.mass(f4);
01334                     ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;
01335                     scalar_products sp;
01336                     sp.add(q4, q3, -(s2-m2q4-m2q3)/2);
01337                     sp.add(q3, q3, m2q3);
01338                     sp.add(q4, q4, m2q4);
01339                     //ex qm0=(s2-mq3*mq3+mq4*mq4)/(2*mq4), lqm1=sqrt(qm0*qm0-s2);
01340                     ex q30=(-s2+mq3*mq3+mq4*mq4)/(2*mq4), lq3l=sqrt(q30*q30-mq3*mq3);
01341                     //ex q30=(-(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);
01342
01343                     for(uint i=0;i<bosons.size();i++) if(bosons[i].s==s || s==
sAny){
01344                         if(bosons[i].s==0){
01345                             ex a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1)
)*bosons[i].couplingL(f3,f4)*s2/(mq1+mq2)/pow(bosons[i].mass,2);

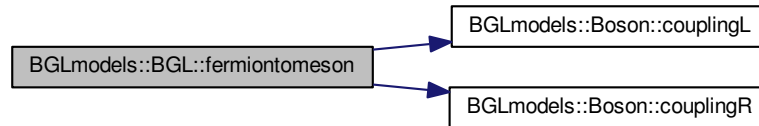
```

```

01346         v1=v1+a*dirac_gammaL();
01347         v2=v2+a.conjugate()*dirac_gammaR();
01348         a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*
bosons[i].couplingR(f3,f4)*s2/(mq1+mq2)/pow(bosons[i].mass,2);
01349         v1=v1+a*dirac_gammaL();
01350         v2=v2+a.conjugate()*dirac_gammaL();
01351     }
01352     else{
01353         ex s1=(dirac_slash(q3,4)+dirac_slash(q4,4));
01354         ex a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))
)*bosons[i].couplingL(f3,f4)/pow(bosons[i].mass,2);
01355         v1=v1+a*s1*dirac_gammaL();
01356         v2=v2+a.conjugate()*s1*dirac_gammaL();
01357         a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*
bosons[i].couplingR(f3,f4)/pow(bosons[i].mass,2);
01358         v1=v1+a*s1*dirac_gammaR();
01359         v2=v2+a.conjugate()*s1*dirac_gammaR();
01360     }
01361 }
01362 ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)+mq4*dirac_ONE();
01363 ex dt=dirac_trace(vq3*v1+vq4*v2).simplify_indexed(sp);
01364 ex result=expand(dt*2*lq3l/mq4/mq4/Pi/128);
01365
01366 ret+=result;
01367 }
01368 }
01369
01370
01371
01372 return pow(meson.decay_factor,2)*collect_common_factors(ret.subs(
conjtoabs));
01373 //return
expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
01374 }

```

Here is the call graph for this function:



#### 7.1.3.12 ex BGLmodels::BGL::fermionmesontest ( const Fermion & ff4, const Fermion & ff3, const Meson & meson, BSpin s=sAny ) const [inline]

Definition at line 1376 of file BGL.h.

References [BGLmodels::Meson::decay\\_factor](#), [BGLmodels::Any](#), [BGLmodels::fElectron](#), [BGLmodels::Fermion](#), [BGLmodels::flavour](#), [BGLmodels::fTau](#), [BGLmodels::Meson::mass](#), [BGLmodels::Meson::q1](#), and [BGLmodels::Meson::q2](#).

```

01376
01377
01378     const Fermion& f1(meson.q1), f2(meson.q2);
01379     ex mesonmass=meson.mass;
01380
01381     Fermion f3=ff3, f4=ff4;
01382
01383     ex ret=0;
01384
01385     realsymbol q3("q3"), q4("q4");
01386
01387     symbol sL("sL"), sR("sR"), vL("vL"), vR("vR");
01388     ex s2=pow(mesonmass,2);
01389

```

```

01390         for(uint k=fElectron;k<=fTau;k++)
01391         if(ff3.flavour==fAny || ff3.flavour==k){
01392             f3.flavour=(FFlavour)k;
01393         for(uint l=fElectron;l<=fTau;l++)
01394         if(ff4.flavour==fAny || ff4.flavour==l){
01395             f4.flavour=(FFlavour)l;
01396             ex v1=0, v2=0;
01397             ex mq1=mixes.mass(f1),mq2=mixes.mass(f2),mq3=
mixes.mass(f3),mq4=mixes.mass(f4);
01398             ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;
01399             scalar_products sp;
01400             sp.add(q4, q3, -(s2-m2q4-m2q3)/2);
01401             sp.add(q3, q3, m2q3);
01402             sp.add(q4, q4, m2q4);
01403             //ex qm0=(s2-mq3*mq3+mq4*mq4)/(2*mq4), lqm1=sqrt(qm0*qm0-s2);
01404             ex q30=(-s2+mq3*mq3+mq4*mq4)/(2*mq4), lq3l=sqrt(q30*q30-mq3*mq3);
01405             //ex q30=-(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);
01406
01407
01408             ex a=sL;
01409             v1=v1+a*dirac_gammaL();
01410             v2=v2+a.conjugate()*dirac_gammaR();
01411             a=sR;
01412             v1=v1+a*dirac_gammaL();
01413             v2=v2+a.conjugate()*dirac_gammaL();
01414
01415             ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
01416             a=vL;
01417             v1=v1+a*sl*dirac_gammaL();
01418             v2=v2+a.conjugate()*sl*dirac_gammaL();
01419             a=vR;
01420             v1=v1+a*sl*dirac_gammaR();
01421             v2=v2+a.conjugate()*sl*dirac_gammaR();
01422
01423             ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)+mq4*dirac_ONE();
01424             ex dt=dirac_trace(vq3*v1*vq4*v2).simplify_indexed(sp);
01425             ex result=expand(dt*2*lq3l/mq4/mq4/Pi/128);
01426
01427             ret+=result;
01428         }
01429     }
01430
01431     return pow(meson.decay_factor,2)*collect_common_factors(ret.subs(
conjtoabs));
01432     //return
expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
01433 }

```

### 7.1.3.13 ex BGLmodels::BGL::FH1 ( ex x ) const [inline]

Definition at line 682 of file [BGL.h](#).

```

00682         {
00683             return -x/12;
00684         }

```

### 7.1.3.14 ex BGLmodels::BGL::Fh1 ( ex x ) const [inline]

Definition at line 690 of file [BGL.h](#).

```

00690         {
00691             //return (2*x+3*pow(x,2)-6*pow(x,3)+pow(x,4)+6*pow(x,2)*log(x))/(6*pow(1-x,4));
00692             return x/3;
00693         }

```

7.1.3.15 `ex BGLmodels::BGL::FH2 ( ex x ) const [inline]`

Definition at line 686 of file [BGL.h](#).

```
00686     {
00687         return x/2;
00688     }
```

7.1.3.16 `ex BGLmodels::BGL::Fh2 ( ex x ) const [inline]`

Definition at line 695 of file [BGL.h](#).

```
00695     {
00696         //return (-3*x+4*pow(x,2)-pow(x,3)-2*x*log(x))/pow(1-x,3);
00697         return -2*(3/2+log(x))*x;
00698     }
```

7.1.3.17 `ex BGLmodels::BGL::FW ( ex x ) const [inline]`

Definition at line 678 of file [BGL.h](#).

```
00678     {
00679         return (94-x*(179+x*(-55+12*x)))/(36*pow(1-x,3))+x*(16+x*(-32+9*x))*log(x)/(6*pow(1-x,4));
00680     }
```

7.1.3.18 `parameters BGLmodels::BGL::generateparameters ( int max = 0 ) const [inline],[virtual]`

Implements [Model](#).

Definition at line 759 of file [BGL.h](#).

Referenced by [main\(\)](#).

```
00759     {
00760         parameters p;
00761         //x=log_10(tanb)
00762         p.push_back(freeparameter(-3,3,r,stepsize));
00763         //y=log_10(McH)
00764         if(max==1) p.push_back(freeparameter(10,10000,r,stepsize));
00765         else p.push_back(freeparameter(10,mmmax,r,stepsize));
00766         //log_10(massR)
00767         p.push_back(freeparameter(-200,200,r,stepsize));
00768         //log_10(massI)
00769         p.push_back(freeparameter(-50,50,r,stepsize));
00770
00771         return p;
00772     }
```

Here is the caller graph for this function:



### 7.1.3.19 ex BGLmodels::BGL::get\_integral\_symb ( const multivector< ex, 3 > &a, ex m1 ) const [inline]

Definition at line 921 of file BGL.h.

References [BGLmodels::Boson::s](#).

```

00921                                     {
00922     realsymbol s2("s2"), s3("s3");
00923     realsymbol q1("q1"), q2("q2"), q3("q3"), q4("q4");
00924
00925     ex m2q1=m1*m1;
00926
00927     ex vq1=dirac_slash(q2,4)+dirac_slash(q3,4)+dirac_slash(q4,4)+m1*dirac_ONE(), vq2=dirac_slash(q2,4);
00928     ex vq3=dirac_slash(q3,4), vq4=dirac_slash(q4,4);
00929
00930     ex s4=m2q1-s2-s3;
00931     scalar_products sp;
00932     sp.add(q2, q3, (s4)/2);
00933     sp.add(q4, q3, (s2)/2);
00934     sp.add(q2, q4, (s3)/2);
00935
00936     sp.add(q2, q2, 0);
00937     sp.add(q3, q3, 0);
00938     sp.add(q4, q4, 0);
00939
00940     multivector<ex,2> v(0,2,2);
00941     v[0][0]=dirac_gammaL(); v[0][1]=dirac_gammaR();
00942     v[1][0]=dirac_gammaR(); v[1][1]=dirac_gammaL();
00943
00944     multivector<ex,5> traces(0,2,2,2,2,2);
00945     for(uint k=0;k<2;k++)
00946         for(uint l=0;l<2;l++)
00947             for(uint m=0;m<2;m++)
00948                 for(uint n=0;n<2;n++){
00949                     ex vk=v[k][0];
00950                     ex vm=v[m][0];
00951                     ex vl=v[l][1];
00952                     ex vn=v[n][1];
00953
00954                     traces[k][l][m][n][0]=dirac_trace(vq2*vk*vq1*vl)*dirac_trace(vq3*vm*vq4*vn)
;
00955                     traces[k][l][m][n][1]=-dirac_trace(vq2*vk*vq1*vl*vq3*vm*vq4*vn);
00956                 }
00957
00958     for(uint k=0;k<2;k++)
00959         for(uint l=0;l<2;l++)
00960             for(uint m=0;m<2;m++)
00961                 for(uint n=0;n<2;n++)
00962                     for(uint o=0;o<2;o++){
00963                         {
00964                             traces[k][l][m][n][o]=(traces[k][l][m][n][o]).
simplify_indexed(sp);
00965                         }
00966
00967     ex q10=(s2+m1*m1)/(2*sqrt(s2)), lq11=(m1*m1-s2)/(2*sqrt(s2));
00968     ex q30=sqrt(s2)/2, lq31=q30;
00969     ex q20=(m1*m1-s2)/(2*m1), lq21=q20;
00970
00971     ex total=0;
00972     for(uint k=0;k<2;k++)
00973         for(uint l=0;l<2;l++)
00974             for(uint m=0;m<2;m++)
00975                 for(uint n=0;n<2;n++)
00976                     for(uint r=0;r<2;r++)
00977                         for(uint s=0;s<2;s++){
00978                             ex coup=a[r][k][m]*a[s][l][n].conjugate();
00979                             ex integrand=traces[k][l][m][n][(r+s)%2];
00980                             integrand=expand(integral(s3, 0, m1*m1-s2, integrand).eval_integ())/lq11/sqrt(s2)*lq21/m1/m1
);
00981                             //double mm2=0, mm3=0, m4=0;
00982                             ex result=integral(s2,0,m1*m1,integrand).eval_integ()/pow(Pi,3)/512;
00983                             ex partial=result*coup;
00984                             total=total+partial;
00985                         }
00986     return total;
00987 }
```



### 7.1.3.20 parameters BGLmodels::BGL::getlist ( const parameters & p ) const [inline], [virtual]

Implements [Model](#).

Definition at line 774 of file [BGL.h](#).

References [parameters::p](#), and [parameters::values](#).

Referenced by [main\(\)](#).

```

00774                                     {
00775         //cout<<aux<<endl;
00776         //double
00777         c2=(1+sqrt(1-4*sqrt(ex_to<numeric>(mudecay.subs(1st(tanb==exp(p[0].value),McH==p[1].value)).to_double())))/2;
00778         double x=pow(10.0,p[0].value);
00779         //double y=pow(10.0,p[1].value);
00780         //double z=pow(10.0,p[2].value);
00781         //double w=pow(10.0,p[3].value);
00782
00783         double y=p[1].value;
00784         double z=y+p[2].value;
00785         double w=z+p[3].value;
00786
00787         parameters pp(p);
00788         pp[0].value=x;
00789         pp[2].value+=pp[1].value;
00790         pp[3].value+=pp[2].value;
00791         pp.values=vector<double>();
00792         for(uint i=0; i<4; i++) pp.values.push_back(pp[i].value);
00793         lst &l=pp.p;
00794         l=lst(tanb==x,McH==y,MR==z,MI==w);
00795
00796         return pp;
00797     }

```

Here is the caller graph for this function:



### 7.1.3.21 ex BGLmodels::BGL::GH1 ( ex x ) const [inline]

Definition at line 668 of file [BGL.h](#).

```

00668     {
00669         return x*(x*((39-14*x)*x-6)+6*x*(3*x-8)*log(x)-19)/(36*pow(x-1,4));
00670         //return -x/12;
00671     }

```

### 7.1.3.22 ex BGLmodels::BGL::GH2 ( ex x ) const [inline]

Definition at line 673 of file [BGL.h](#).

```
00673         {
00674             return x * ((x-1) * (11*x-21) + (16-6*x) * log(x)) / (6*pow(x-1,3));
00675             //return x/2;
00676         }
```

### 7.1.3.23 ex BGLmodels::BGL::gRR2 ( const Fermion & f1, const Fermion & f3 ) const [inline]

Definition at line 1085 of file [BGL.h](#).

References [BGLmodels::fElectron](#), [BGLmodels::Fermion::flavour](#), [BGLmodels::fTau](#), [BGLmodels::iUp](#), [BGLmodels::Boson::mass](#), [BGLmodels::Boson::s](#), [BGLmodels::sScalar](#), [BGLmodels::sVector](#), and [BGLmodels::Lepton](#).

```
01085         {
01086             ex ret1=0,ret2=0;
01087             Fermion f2(tLepton,iUp);
01088             Fermion f4(tLepton,iUp);
01089
01090             for(uint k=fElectron;k<=fTau;k++){
01091                 f2.flavour=(Fflavour)k;
01092                 for(uint l=fElectron;l<=fTau;l++){
01093                     f4.flavour=(Fflavour)l;
01094                     for(uint i=0;i<bosons.size();i++){
01095                         if(bosons[i].s==sScalar) {
01096                             ex x=bosons[i].couplingdaggerR(f2,f1)*bosons[i].couplingL(f3,f4)/pow(
01097                                 bosons[i].mass,2);
01098                             ret1+=x*x.conjugate();
01099                         }
01100                         else if(bosons[i].s==sVector) {
01101                             ex x=bosons[i].couplingdaggerL(f2,f1)*bosons[i].couplingL(f3,f4)/pow(
01102                                 bosons[i].mass,2);
01103                             ret2+=x*x.conjugate();
01104                         }
01105                     }
01106                 }
01107                 //r2.append();
01108                 ret2=ret2.subs(conjtoabs);
01109                 ret1=ret1.subs(conjtoabs);
01110                 for(uint i=0;i<3;i++){
01111                     ret1=collect_common_factors(expand(ret1.subs(pow(abs(mixes.
01112                         V[0][2][i]),2)==1-pow(abs(mixes.V[0][1][i]),2)-pow(abs(mixes.V[0][0][i]),2))));
01113                     ret2=collect_common_factors(expand(ret2.subs(pow(abs(mixes.
01114                         V[0][2][i]),2)==1-pow(abs(mixes.V[0][1][i]),2)-pow(abs(mixes.V[0][0][i]),2))));
01115                 }
01116             }
01117             cout<<ret2<<endl;
01118             return collect_common_factors(ret1/ret2);
01119         }
```

### 7.1.3.24 ex BGLmodels::BGL::GW ( ex x ) const [inline]

Definition at line 664 of file [BGL.h](#).

```
00664         {
00665             return (94-x*(179+x*(-55+12*x)))/(36*pow(1-x,3))+x*(16+x*(-32+9*x))*log(x)/(6*pow(1-x,4));
00666         }
```

7.1.3.25 `ex BGLmodels::BGL::mesondw ( const Meson & meson, const Fermion & ff3, const Fermion & ff4, BSpin s = sAny ) const [inline]`

Definition at line 1170 of file [BGL.h](#).

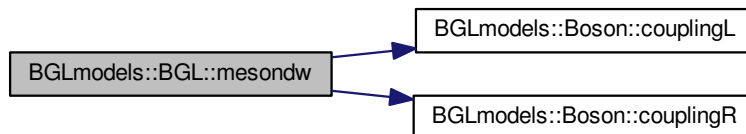
References [BGLmodels::Boson::couplingL\(\)](#), [BGLmodels::Boson::couplingR\(\)](#), [BGLmodels::Meson::decay\\_factor](#), [BGLmodels::fAny](#), [BGLmodels::fElectron](#), [BGLmodels::Fermion::flavour](#), [BGLmodels::fTau](#), [BGLmodels::Meson::mass](#), [BGLmodels::Boson::mass](#), [BGLmodels::Meson::q1](#), [BGLmodels::Meson::q2](#), [BGLmodels::Boson::s](#), and [BGLmodels::sAny](#).

```

01170
01171
01172     const Fermion& f1(meson.q1), f2(meson.q2);
01173     ex mesonmass=meson.mass;
01174
01175     Fermion f3=ff3, f4=ff4;
01176
01177     ex ret=0;
01178
01179     realsymbol q3("q3"), q4("q4");
01180     ex s2=pow(mesonmass,2);
01181
01182     for(uint k=fElectron;k<=fTau;k++){
01183         if(ff3.flavour==fAny || ff3.flavour==k){
01184             f3.flavour=(FFlavour)k;
01185             for(uint l=fElectron;l<=fTau;l++){
01186                 if(ff4.flavour==fAny || ff4.flavour==l){
01187                     f4.flavour=(FFlavour)l;
01188                     ex v1=0, v2=0;
01189                     ex mql=mixes.mass(f1),mq2=mixes.mass(f2),mq3=
01190 mixes.mass(f3),mq4=mixes.mass(f4);
01191                     ex m2q1=mql*mql, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;
01192                     scalar_products sp;
01193                     sp.add(q4, q3, (s2-m2q4-m2q3)/2);
01194                     sp.add(q3, q3, m2q3);
01195                     sp.add(q4, q4, m2q4);
01196                     ex q10=(s2+mql*mql-mq2*mq2)/(2*sqrt(s2)), lq1l=sqrt(q10*q10-mql*mql);
01197                     ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);
01198
01199                     for(uint i=0;i<bosons.size();i++) if(bosons[i].s==s || s==
01200 sAny){
01201                         if(bosons[i].s==0){
01202                             ex a=-(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1
01203 ))*bosons[i].couplingL(f3,f4)*s2/(mql+mq2)/pow(bosons[i].mass,2);
01204                             v1=v1+a*dirac_gammaL();
01205                             v2=v2+a.conjugate()*dirac_gammaR();
01206                             a=-(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*
01207 bosons[i].couplingR(f3,f4)*s2/(mql+mq2)/pow(bosons[i].mass,2);
01208                             v1=v1+a*dirac_gammaL();
01209                             v2=v2+a.conjugate()*dirac_gammaR();
01210                         }
01211                         else{
01212                             ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
01213                             ex a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1)
01214 )*bosons[i].couplingL(f3,f4)/pow(bosons[i].mass,2);
01215                             v1=v1+a*sl*dirac_gammaL();
01216                             v2=v2+a.conjugate()*sl*dirac_gammaR();
01217                             a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*
01218 bosons[i].couplingR(f3,f4)/pow(bosons[i].mass,2);
01219                             v1=v1+a*sl*dirac_gammaL();
01220                             v2=v2+a.conjugate()*sl*dirac_gammaR();
01221                         }
01222                     }
01223                     ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)-mq4*dirac_ONE();
01224                     ex dt=dirac_trace(vq3*v1*vq4*v2).simplify_indexed(sp);
01225                     ex result=expand(dt*4*lq3l/s2/Pi/128);
01226
01227                     ret+=result;
01228                 }
01229             }
01230         }
01231     }
01232     return pow(meson.decay_factor,2)*collect_common_factors(ret.subs(
01233 conjtoabs));
01234     //return
01235     expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()), sin(wild())==sqrt(1-pow(cos(wild()),2)))));
01236 }

```

Here is the call graph for this function:



**7.1.3.26** `ex BGLmodels::BGL::mesondwtest ( const Meson & meson, const Fermion & ff3, const Fermion & ff4, BSpin s = sAny ) const [inline]`

Definition at line 1230 of file [BGL.h](#).

References [BGLmodels::Meson::decay\\_factor](#), [BGLmodels::fAny](#), [BGLmodels::fElectron](#), [BGLmodels::Fermion::flavour](#), [BGLmodels::fTau](#), [BGLmodels::Meson::mass](#), [BGLmodels::Meson::q1](#), and [BGLmodels::Meson::q2](#).

```

01230
01231
01232     const Fermion& f1(meson.q1), f2(meson.q2);
01233     ex mesonmass=meson.mass;
01234
01235     Fermion f3=ff3, f4=ff4;
01236
01237     ex ret=0;
01238
01239     realsymbol q3("q3"), q4("q4");
01240     symbol gL("gL"), gR("gR"), gVL("gVL"), gVR("gVR");
01241     symbol gS("gS"), gP("gP"), gA("gA");
01242
01243     ex s2=pow(mesonmass,2);
01244
01245     for(uint k=fElectron;k<=fTau;k++)
01246     if(ff3.flavour==fAny || ff3.flavour==k){
01247         f3.flavour=(Fflavour)k;
01248     for(uint l=fElectron;l<=fTau;l++)
01249     if(ff4.flavour==fAny || ff4.flavour==l){
01250         f4.flavour=(Fflavour)l;
01251         ex v1=0, v2=0;
01252         ex mql=mixes.mass(f1),mq2=mixes.mass(f2),mq3=
mixes.mass(f3),mq4=mixes.mass(f4);
01253         ex m2q1=mql*mql, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;
01254         scalar_products sp;
01255         sp.add(q4, q3, (s2-m2q4-m2q3)/2);
01256         sp.add(q3, q3, m2q3);
01257         sp.add(q4, q4, m2q4);
01258         ex q10=(s2+mql*mql-mq2*mq2)/(2*sqrt(s2)), lq1l=sqrt(q10*q10-mql*mql);
01259         ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);
01260
01261         ex a;
01262         /*
01263         a=-gL*s2/(mql+mq2);
01264         v1=v1+a*dirac_gammaL();
01265         v2=v2+a.conjugate()*dirac_gammaR();
01266         a=-gR*s2/(mql+mq2);
01267         v1=v1+a*dirac_gammaR();
01268         v2=v2+a.conjugate()*dirac_gammaL();
01269
01270         ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
01271         a=gA;
01272         v1=v1+a*sl*dirac_gamma5();
01273         v2=v2+a.conjugate()*sl*dirac_gamma5();
01274         */
01275         a=-gS*s2/(mql+mq2);
01276         v1=v1+a*dirac_ONE();
01277         v2=v2+a.conjugate()*dirac_ONE();
01278         a=-gP*s2/(mql+mq2);
01279         v1=v1+a*dirac_gamma5();

```

```

01279         v2=v2-a.conjugate()*dirac_gamma5();
01280         ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
01281         // a=gA;
01282         // v1=v1+a*sl*dirac_gamma5();
01283         // v2=v2+a.conjugate()*sl*dirac_gamma5();
01284
01285         /*}
01286         else{
01287             ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
01288             ex a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*
bosons[i].couplingL(f3,f4)/pow(bosons[i].mass,2);
01289             v1=v1+a*sl*dirac_gammaL();
01290             v2=v2+a.conjugate()*sl*dirac_gammaL();
01291             a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*
bosons[i].couplingR(f3,f4)/pow(bosons[i].mass,2);
01292             v1=v1+a*sl*dirac_gammaR();
01293             v2=v2+a.conjugate()*sl*dirac_gammaR();
01294         }*/
01295
01296         ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)-mq4*dirac_ONE();
01297         ex dt=dirac_trace(vq3*v1+vq4*v2).simplify_indexed(sp);
01298         ex result=expand(dt*4*1q3l/s2/Pi/128);
01299
01300         ret+=result;
01301     }
01302 }
01303
01304 ltest.append(conjugate(gL)==pow(abs(gL),2)/gL);
01305 ltest.append(conjugate(gR)==pow(abs(gR),2)/gR);
01306 ltest.append(conjugate(gS)==pow(abs(gS),2)/gS);
01307 ltest.append(conjugate(gP)==pow(abs(gP),2)/gP);
01308 ltest.append(conjugate(gA)==pow(abs(gA),2)/gA);
01309
01310     return pow(meson.decay_factor,2)*collect_common_factors(expand(ret.subs(
conjtoabs).subs(ltest)));
01311     //return
expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
01312 }

```

### 7.1.327 ex BGLmodels::BGL::mesonmixing ( ex mesonmass, const Fermion & f1, const Fermion & f2 ) const [inline]

Definition at line 1435 of file BGL.h.

References [BGLmodels::Boson::mass](#), and [BGLmodels::Boson::s](#).

```

01435
01436
01437     ex ret=0;
01438
01439         ex v1=0, v2=0;
01440         ex mql=mixes.mass(f1),mq2=mixes.mass(f2);
01441         ex m2q1=mql*mql, m2q2=mq2*mq2;
01442
01443         for(uint i=0;i<bosons.size();i++)
01444             if(bosons[i].s==0){
01445                 ex a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1)
);
01446                 v1=v1+pow(a/bosons[i].mass,2);
01447
01448                 ex b=(bosons[i].couplingdaggerR(f2,f1)+bosons[i].couplingdaggerL(f2,f1)
);
01449                 v2=v2+pow(b/bosons[i].mass,2);
01450             }
01451
01452         ex fc=mesonmass/(mixes.massnum(f1)+mixes.massnum(f2));
01453         fc=pow(fc,2);
01454
01455         ret=2*(-v1*(1+11*fc)+v2*(1+fc))*mesonmass/96;
01456
01457         return collect_common_factors(ret.subs(conjtoabs));
01458         //return
expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
01459     }

```

### 7.1.3.28 ex BGLmodels::BGL::tautomu\_tautoe ( ) const [inline]

Definition at line 1117 of file BGL.h.

References [BGLmodels::fElectron](#), [BGLmodels::Fermion::flavour](#), [BGLmodels::fMuon](#), [BGLmodels::fTau](#), [BGLmodels::iDown](#), [BGLmodels::iUp](#), [BGLmodels::Boson::mass](#), [BGLmodels::Boson::s](#), [BGLmodels::sScalar](#), [BGLmodels::sVector](#), and [BGLmodels::tLepton](#).

```

01117         {
01118
01119             ex ret1=0,ret2=0, rety1=0, rety2=0;
01120
01121             Fermion f1(tLepton,iDown,fTau);
01122             Fermion f31(tLepton,iDown,fMuon);
01123             Fermion f32(tLepton,iDown,fElectron);
01124
01125             Fermion f2(tLepton,iUp);
01126             Fermion f4(tLepton,iUp);
01127
01128
01129             for(uint k=fElectron;k<=fTau;k++){
01130                 f2.flavour=(Fflavour)k;
01131                 for(uint l=fElectron;l<=fTau;l++){
01132                     f4.flavour=(Fflavour)l;
01133                     ex x1=0, x2=0, y1=0, y2=0;
01134                     for(uint i=0;i<bosons.size();i++){
01135                         if(bosons[i].s==sScalar) {
01136                             x1+=bosons[i].couplingdaggerR(f2,f1)*bosons[i].couplingL(f31,f4)/pow(
01137                                 bosons[i].mass,2);
01138                             x2+=bosons[i].couplingdaggerR(f2,f1)*bosons[i].couplingL(f32,f4)/pow(
01139                                 bosons[i].mass,2);
01140                         }
01141                         else if(bosons[i].s==sVector) {
01142                             y1+=bosons[i].couplingdaggerL(f2,f1)*bosons[i].couplingL(f31,f4)/pow(
01143                                 bosons[i].mass,2);
01144                             y2+=bosons[i].couplingdaggerL(f2,f1)*bosons[i].couplingL(f32,f4)/pow(
01145                                 bosons[i].mass,2);
01146                         }
01147                     }
01148                     ret1+=(x1+y1.conjugate()).real_part();
01149                     ret2+=(x2+y2.conjugate()).real_part();
01150                     rety1+=y1+y1.conjugate();
01151                     rety2+=y2+y2.conjugate();
01152                 }
01153                 ret2=(ret2/rety2*mixes.mass(f32)/mixes.mass(f1)).subs(
01154                     conjtoabs);
01155                 ret1=(ret1/rety1*mixes.mass(f31)/mixes.mass(f1)).subs(
01156                     conjtoabs);
01157                 for(uint i=0;i<3;i++){
01158                     ret1=collect_common_factors(expand(ret1.subs(pow(abs(mixes.
01159 V[0][2][i]),2)==1-pow(abs(mixes.V[0][1][i]),2)-pow(abs(mixes.V[0][0][i]),2))));
01160                     ret2=collect_common_factors(expand(ret2.subs(pow(abs(mixes.
01161 V[0][2][i]),2)==1-pow(abs(mixes.V[0][1][i]),2)-pow(abs(mixes.V[0][0][i]),2))));
01162                 }
01163                 ex x=pow(mixes.mass(f31)/mixes.mass(f1),2);
01164                 ex F1=1-8*x+8*pow(x,3)-pow(x,4)-12*pow(x,2)*log(x);
01165                 ex g1=1+9*x-9*pow(x,2)-pow(x,3)+6*x*(1+x)*log(x);
01166                 ex N1=1+gRR2(f1,f31)/4;
01167
01168                 x=pow(mixes.mass(f32)/mixes.mass(f1),2);
01169                 ex F2=1-8*x+8*pow(x,3)-pow(x,4)-12*pow(x,2)*log(x);
01170                 ex g2=1+9*x-9*pow(x,2)-pow(x,3)+6*x*(1+x)*log(x);
01171                 ex N2=1+gRR2(f1,f32)/4;
01172
01173                 return collect_common_factors(N1*(F1+2/N1*ret1*g1)/N2/(F2+2/N2*ret2*g2)*F2/F1);
01174             }
01175         }

```

### 7.1.3.29 double BGLmodels::BGL::topBranchingRatio ( double \* xx, double \* p ) [inline]

Definition at line 1482 of file BGL.h.

```

01482         {
01483             return ex_to<numeric>(BR_toptoHq.subs(1st(tanb==pow(10.0,xx[0]),
01484 McH==xx[1])).evalf()).to_double();
01485         }

```

**7.1.3.30** `int BGLmodels::BGL::veto ( const parameters & p, int max = 0 ) const` `[inline]`, `[virtual]`

Reimplemented from [Model](#).

Definition at line 741 of file [BGL.h](#).

References [parameters::isvalid\(\)](#).

```

00741                                     {
00742         if (!p.isvalid()) return 1;
00743         if (max==1) {
00744             double mr=p[1].value+p[2].value;
00745             if (mr<10 || mr>10000) return 1;
00746             mr+=p[3].value;
00747             if (mr<10 || mr>10000) return 1;
00748             return 0;
00749         }
00750         else{
00751             double mr=p[1].value+p[2].value;
00752             if (mr<10 || mr>mmmax) return 1;
00753             mr+=p[3].value;
00754             if (mr<10 || mr>mmmax) return 1;
00755             return 0;
00756         }
00757     }

```

Here is the call graph for this function:



**7.1.3.31** `ex BGLmodels::BGL::Y ( ex x ) const` `[inline]`

Definition at line 660 of file [BGL.h](#).

```

00660         {
00661             return 1.0113*x/8/(1-x) * (4-x+3*x*log(x)/(1-x));
00662         }

```

## 7.1.4 Member Data Documentation

**7.1.4.1** `ex BGLmodels::BGL::alpha`

Definition at line 1492 of file [BGL.h](#).

**7.1.4.2** `vector<int> BGLmodels::BGL::BGLtype`

Definition at line 1510 of file [BGL.h](#).

#### 7.1.4.3 `vector< Boson > BGLmodels::BGL::bosons`

Definition at line 1495 of file [BGL.h](#).

#### 7.1.4.4 `ex BGLmodels::BGL::BR_Htotaunu`

Definition at line 1499 of file [BGL.h](#).

Referenced by [main\(\)](#).

#### 7.1.4.5 `ex BGLmodels::BGL::BR_toptoHq`

Definition at line 1500 of file [BGL.h](#).

#### 7.1.4.6 `ex BGLmodels::BGL::Btaunu`

Definition at line 1498 of file [BGL.h](#).

#### 7.1.4.7 `ex BGLmodels::BGL::BtoD2taunuR`

Definition at line 1503 of file [BGL.h](#).

#### 7.1.4.8 `ex BGLmodels::BGL::BtoDtaunuR`

Definition at line 1502 of file [BGL.h](#).

#### 7.1.4.9 `ex BGLmodels::BGL::BtotaunuR`

Definition at line 1501 of file [BGL.h](#).

#### 7.1.4.10 `calcuBmumu* BGLmodels::BGL::cBmumu`

Definition at line 1514 of file [BGL.h](#).

Referenced by [main\(\)](#).

#### 7.1.4.11 `calcuBmumu* BGLmodels::BGL::cBsmumu`

Definition at line 1515 of file [BGL.h](#).

Referenced by [main\(\)](#).



#### 7.1.4.12 `lst` BGLmodels::BGL::conjtoabs

Definition at line [1506](#) of file [BGL.h](#).

#### 7.1.4.13 `ex` BGLmodels::BGL::cos2

Definition at line [1492](#) of file [BGL.h](#).

#### 7.1.4.14 `const` `possymbol` BGLmodels::BGL::cp

Definition at line [1493](#) of file [BGL.h](#).

#### 7.1.4.15 `const` `constant` BGLmodels::BGL::FB

Definition at line [1491](#) of file [BGL.h](#).

#### 7.1.4.16 `const` `constant` BGLmodels::BGL::FBs

Definition at line [1491](#) of file [BGL.h](#).

#### 7.1.4.17 `const` `constant` BGLmodels::BGL::FD

Definition at line [1491](#) of file [BGL.h](#).

#### 7.1.4.18 `const` `constant` BGLmodels::BGL::FDs

Definition at line [1491](#) of file [BGL.h](#).

#### 7.1.4.19 `const` `constant` BGLmodels::BGL::FK

Definition at line [1491](#) of file [BGL.h](#).

#### 7.1.4.20 `const` `constant` BGLmodels::BGL::Fpi

Definition at line [1491](#) of file [BGL.h](#).

#### 7.1.4.21 `ex` BGLmodels::BGL::g

Definition at line [1492](#) of file [BGL.h](#).

7.1.4.22 `const possymbol BGLmodels::BGL::GF`

Definition at line 1489 of file [BGL.h](#).

7.1.4.23 `int BGLmodels::BGL::iBD2taunu`

Definition at line 1509 of file [BGL.h](#).

7.1.4.24 `int BGLmodels::BGL::iBDtaunu`

Definition at line 1509 of file [BGL.h](#).

7.1.4.25 `int BGLmodels::BGL::iBtaunu`

Definition at line 1509 of file [BGL.h](#).

7.1.4.26 `const constant BGLmodels::BGL::MB0`

Definition at line 1490 of file [BGL.h](#).

7.1.4.27 `const constant BGLmodels::BGL::MBp`

Definition at line 1490 of file [BGL.h](#).

7.1.4.28 `const constant BGLmodels::BGL::MBs0`

Definition at line 1490 of file [BGL.h](#).

7.1.4.29 `const possymbol BGLmodels::BGL::McH`

Definition at line 1493 of file [BGL.h](#).

7.1.4.30 `possymbol BGLmodels::BGL::Md[3]`

Definition at line 1494 of file [BGL.h](#).

7.1.4.31 `const constant BGLmodels::BGL::MD0`

Definition at line 1490 of file [BGL.h](#).

**7.1.4.32** const constant BGLmodels::BGL::MDp

Definition at line 1490 of file [BGL.h](#).

**7.1.4.33** const constant BGLmodels::BGL::MDs0

Definition at line 1490 of file [BGL.h](#).

**7.1.4.34** const constant BGLmodels::BGL::MDsp

Definition at line 1490 of file [BGL.h](#).

**7.1.4.35** const possymbol BGLmodels::BGL::Mh

Definition at line 1489 of file [BGL.h](#).

**7.1.4.36** const possymbol BGLmodels::BGL::MI

Definition at line 1493 of file [BGL.h](#).

**7.1.4.37** const Mixes BGLmodels::BGL::mixes

Definition at line 1505 of file [BGL.h](#).

**7.1.4.38** const constant BGLmodels::BGL::MK0

Definition at line 1490 of file [BGL.h](#).

**7.1.4.39** const constant BGLmodels::BGL::MKp

Definition at line 1490 of file [BGL.h](#).

**7.1.4.40** double BGLmodels::BGL::mmax

Definition at line 1512 of file [BGL.h](#).

Referenced by [main\(\)](#).

**7.1.4.41** const constant BGLmodels::BGL::Mpi0

Definition at line 1490 of file [BGL.h](#).

#### 7.1.4.42 `const constant BGLmodels::BGL::Mpip`

Definition at line [1490](#) of file [BGL.h](#).

#### 7.1.4.43 `const possymbol BGLmodels::BGL::MR`

Definition at line [1493](#) of file [BGL.h](#).

#### 7.1.4.44 `possymbol BGLmodels::BGL::Mu[3]`

Definition at line [1494](#) of file [BGL.h](#).

#### 7.1.4.45 `realsymbol BGLmodels::BGL::mu`

Definition at line [1507](#) of file [BGL.h](#).

#### 7.1.4.46 `const possymbol BGLmodels::BGL::MW`

Definition at line [1489](#) of file [BGL.h](#).

#### 7.1.4.47 `const possymbol BGLmodels::BGL::MZ`

Definition at line [1489](#) of file [BGL.h](#).

#### 7.1.4.48 `const double BGLmodels::BGL::planck`

Definition at line [1488](#) of file [BGL.h](#).

Referenced by [main\(\)](#).

#### 7.1.4.49 `lst BGLmodels::BGL::replacements`

Definition at line [1497](#) of file [BGL.h](#).

#### 7.1.4.50 `const possymbol BGLmodels::BGL::rho`

Definition at line [1493](#) of file [BGL.h](#).

#### 7.1.4.51 `double BGLmodels::BGL::stepsize`

Definition at line [1512](#) of file [BGL.h](#).

Referenced by [main\(\)](#).

#### 7.1.4.52 `const possymbol BGLmodels::BGL::tanb`

Definition at line [1493](#) of file [BGL.h](#).

#### 7.1.4.53 `widthcalc BGLmodels::BGL::wc`

Definition at line [1486](#) of file [BGL.h](#).

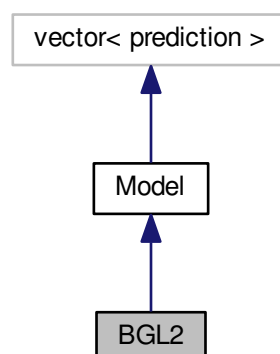
The documentation for this class was generated from the following file:

- [BGL.h](#)

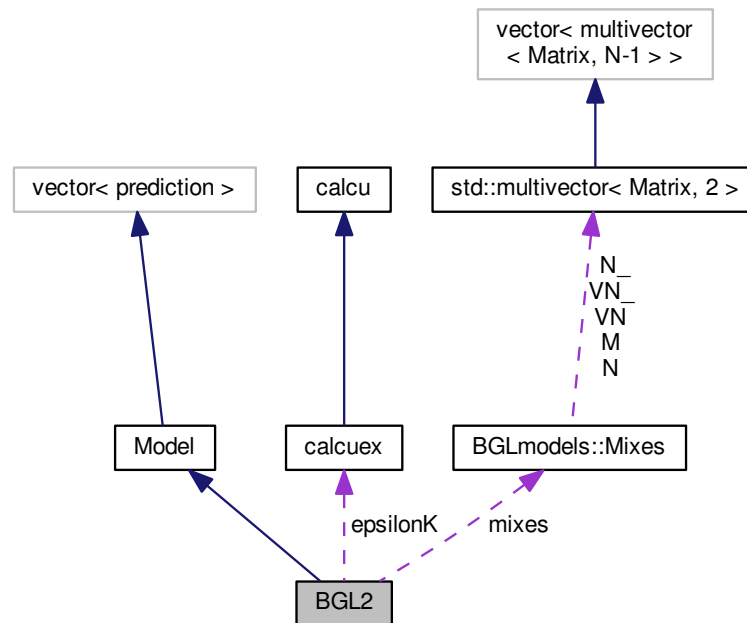
## 7.2 BGL2 Class Reference

A second implementation of the BGL model, for testing purposes.

Inheritance diagram for BGL2:



Collaboration diagram for BGL2:



## Public Member Functions

- [BGL2](#) (int genL=2, int genQ=2, int lup=0, int qup=0, int mssm=0)
- [~BGL2](#) ()
- [parameters generateparameters](#) (int max=0) const
- [parameters getlist](#) (const [parameters](#) &p) const
- ex [mesonmixing](#) (ex mesonmass, const [Fermion](#) &f1, const [Fermion](#) &f2) const
- double [bsgammawidth](#) (double tanb\_, double McH\_, double MR\_, double MI\_, int option=0)
- double [epsK](#) (double tanb\_, double McH\_, double MR\_, double MI\_, int option=0)

## Public Attributes

- const double [planck](#)
- const possymbol [GF](#)
- const possymbol [MZ](#)
- const possymbol [MW](#)
- const possymbol [Mh](#)
- const constant [Mpip](#)
- const constant [Mpi0](#)
- const constant [MBp](#)
- const constant [MB0](#)
- const constant [MBs0](#)
- const constant [MKp](#)
- const constant [MK0](#)
- const constant [MDp](#)

- const constant [MD0](#)
- const constant [MDsp](#)
- const constant [MDs0](#)
- const constant [Fpi](#)
- const constant [FB](#)
- const constant [FBs](#)
- const constant [FK](#)
- const constant [FD](#)
- const constant [FDs](#)
- ex [cos2](#)
- ex [g](#)
- ex [alpha](#)
- const possymbol [tanb](#)
- const possymbol [cp](#)
- const possymbol [McH](#)
- const possymbol [MR](#)
- const possymbol [MI](#)
- const possymbol [rho](#)
- const realsymbol [Tparam](#)
- const realsymbol [Sparam](#)
- const realsymbol [QCD1](#)
- const realsymbol [QCD2](#)
- possymbol [Mu](#) [3]
- possymbol [Md](#) [3]
- vector< [Boson](#) > [bosons](#)
- lst [replacements](#)
- ex [Btaunu](#)
- ex [BR\\_Htotaunu](#)
- ex [BR\\_toptoHq](#)
- ex [BtotaunuR](#)
- ex [BtoDtaunuR](#)
- ex [BtoD2taunuR](#)
- const [Mixes mixes](#)
- lst [conjtoabs](#)
- realsymbol [mu](#)
- int [iBtaunu](#)
- int [iBDtaunu](#)
- int [iBD2taunu](#)
- vector< int > [BGLtype](#)
- ROOT::Math::Interpolator [inter1](#)
- ROOT::Math::Interpolator [inter2](#)
- ROOT::Math::Interpolator [Mu\\_](#) [3]
- ROOT::Math::Interpolator [Md\\_](#) [3]
- double [mmax](#)
- double [stepsize](#)
- [calcuex](#) \* [epsilonK](#)

### 7.2.1 Detailed Description

A second implementation of the BGL model, for testing purposes.

Definition at line 28 of file [draw.cpp](#).

## 7.2.2 Constructor & Destructor Documentation

### 7.2.2.1 BGL2::BGL2 ( int *genL* = 2, int *genQ* = 2, int *lup* = 0, int *qup* = 0, int *mssm* = 0 ) [inline]

Definition at line 31 of file [draw.cpp](#).

References [BGLmodels::Boson::C](#), [BGLmodels::cParticle](#), [BGLmodels::fElectron](#), [BGLmodels::fMuon](#), [BGLmodels::fTau](#), [BGLmodels::hLeft](#), [BGLmodels::hRight](#), [BGLmodels::iDown](#), [BGLmodels::iUp](#), [BGLmodels::M\\_MW](#), [BGLmodels::M\\_MZ](#), [BGLmodels::Boson::mass](#), [BGLmodels::Boson::reset\(\)](#), [BGLmodels::Boson::s](#), [BGLmodels::Scalar](#), [BGLmodels::sVector](#), [BGLmodels::tLepton](#), and [BGLmodels::tQuark](#).

```
00031                                     :
00032         planck(6.58211928e-25),
00033         GF("G_F"),
00034         MZ("M_Z"),
00035         MW("M_W"),
00036         Mpip("Mpip",0.1396,"M_{\\pi^+}",domain::real),
00037         Mpi0("Mpi0",0.1349766,"M_{\\pi^0}",domain::real),
00038         MBp("MBp",5.279,"M_{B^+}",domain::real),
00039         MB0("MB0",5.2795,"M_{B^0}",domain::real),
00040         MBS0("MBS0",5.3663,"M_{B_s^0}",domain::real),
00041         MKp("MKp",0.493677,"MKp",domain::real),
00042         MK0("MK0",0.497614,"MK0",domain::real),
00043         MDp("MDp",1.86957,"MDp",domain::real),
00044         MD0("MD0",1.86480,"MD0",domain::real),
00045         MDsp("MDsp",1.96845,"MDsp",domain::real),
00046         MDs0("MDs0",0),
00047         Fpi("Fpi",0.132,"Fpi",domain::real),
00048         FB("FB",0.189,"FB",domain::real),
00049         FBS("FBS",0.225,"FBS",domain::real),
00050         FK("FK",0.159,"FK",domain::real),
00051         FD("FD",0.208,"FD",domain::real),
00052         FDS("FDS",0.248,"FDS",domain::real),
00053         //alpha(7.297352e-3*4*M_PI),
00054         cos2(pow(MW/MZ,2)),
00055         g(sqrt(GF*8/sqrt(ex(2))*MW),
00056         //g(sqrt(4*Pi*alpha/(1-cos2))),
00057         tanb("tg\\beta"),
00058         cp("cp"),
00059         McH("M_{H^+}"),
00060         MR("M_{R}"),
00061         MI("M_{I}"),
00062         Tparam("T_param"),
00063         Sparam("S_param"),
00064         QCD1("QCD_1"),
00065         QCD2("QCD_2"),
00066         mixes(tanb,cp, genL,genQ, lup, qup, mssm),
00067         mu("\\mu"),
00068         BGLtype(4,0),
00069         mmmmax(1000),
00070         stepsize(1e-2)
00071     {
00072         alpha=pow(g,2)*(1-cos2)/(4*Pi);
00073         replacements.append(GF==1.166371e-5);
00074         replacements.append(MZ==M_MZ);
00075         replacements.append(MW==M_MW);
00076
00077         mixes.appenddtolst(replacements);
00078
00079         replacements.append(Pi==M_PI);
00080         replacements.append(sqrt(ex(2))==sqrt(2));
00081         replacements.append(Pi==M_PI);
00082         replacements.append(sqrt(ex(2))==sqrt(2));
00083
00084         Boson boson;
00085
00086         realsymbol q3("q3");
00087         ex vq3=dirac_slash(q3,4);
00088         varidx jmu(mu,4,1);
00089
00090         for(uint i=0;i<2;i++)
00091             for(uint j=0;j<3;j++)
00092                 for(uint k=0;k<3;k++){
00093                     conjtoabs.append(conjugate(mixes.V[i][j][k])==pow(abs(
00094                         mixes.V[i][j][k]),2)/mixes.V[i][j][k]);
00095                 }
00096
00097         //W+ boson
00098         boson.mass=MW;
```



```

00098         boson.s=sVector;
00099
00100         for(uint t=tLepton;t<=tQuark;t++) boson.C[t][iUp][
iDown][hLeft]=mixes.V[t]*Matrix(g/sqrt(ex(2)));
00101         Boson wboson=boson;
00102         bosons.push_back(boson);
00103         boson.reset();
00104
00105         //H+ boson
00106         boson.mass=MCH;
00107         boson.s=sScalar;
00108
00109         for(uint t=tLepton;t<=tQuark;t++)
00110         for(uint i=iUp;i<=iDown;i++) boson.C[t][iUp][iDown][i]=
mixes.VN[t][i]*Matrix(g/MW/sqrt(ex(2)));
00111         Boson chiggs=boson;
00112         bosons.push_back(boson);
00113         boson.reset();
00114
00115         for(int b=bosons.size()-1;b>=0;b--){
00116             boson.mass=bosons[b].mass;
00117             boson.s=bosons[b].s;
00118             if(boson.s==sVector)
00119                 for(uint t=tLepton;t<=tQuark;t++)
00120                 for(uint i=iUp;i<=iDown;i++)
00121                 for(uint j=iUp;j<=iDown;j++)
00122                 for(uint h=hLeft;h<=hRight;h++){
00123                     boson.C[t][i][j][h]=bosons[b].C[t][j][i][h].conjugate();
00124                 }
00125             else for(uint t=tLepton;t<=tQuark;t++)
00126             for(uint i=iUp;i<=iDown;i++)
00127             for(uint j=iUp;j<=iDown;j++)
00128             for(uint h=hLeft;h<=hRight;h++){
00129                 boson.C[t][i][j][hLeft]=bosons[b].C[t][j][i][
hRight].conjugate();
00130                 boson.C[t][i][j][hRight]=bosons[b].C[t][j][i][
hLeft].conjugate();
00131             }
00132             bosons.push_back(boson);
00133             boson.reset();
00134         }
00135
00136         //(R+iI)/sqrt(2) boson
00137         boson.mass=MR;
00138         boson.s=sScalar;
00139
00140         for(uint t=tLepton;t<=tQuark;t++){
00141             boson.C[t][iDown][iDown][hRight]=mixes.
N[t][iDown]*Matrix(g/MW/ex(2));
00142             boson.C[t][iUp][iUp][hLeft]=mixes.N[t][
iUp].conjugate()*Matrix(g/MW/ex(2));
00143             boson.C[t][iDown][iDown][hLeft]=mixes.
N[t][iDown].conjugate()*Matrix(g/MW/ex(2));
00144             boson.C[t][iUp][iUp][hRight]=mixes.N[t][
iUp]*Matrix(g/MW/ex(2));
00145         }
00146         bosons.push_back(boson);
00147         boson.reset();
00148
00149         //(R+iI)/sqrt(2) boson
00150         boson.mass=MI;
00151         boson.s=sScalar;
00152
00153         for(uint t=tLepton;t<=tQuark;t++){
00154             boson.C[t][iDown][iDown][hRight]=mixes.
N[t][iDown]*Matrix(I*g/MW/ex(2));
00155             boson.C[t][iUp][iUp][hLeft]=mixes.N[t][
iUp].conjugate()*Matrix(I*g/MW/ex(2));
00156             boson.C[t][iDown][iDown][hLeft]=mixes.
N[t][iDown].conjugate()*Matrix(-I*g/MW/ex(2));
00157             boson.C[t][iUp][iUp][hRight]=mixes.N[t][
iUp]*Matrix(-I*g/MW/ex(2));
00158         }
00159         bosons.push_back(boson);
00160         boson.reset();
00161
00162         Fermion electron(tLepton,iDown,fElectron);
00163         Fermion electronR(tLepton,iDown,fElectron,
cParticle,hRight);
00164
00165         Fermion muon(tLepton,iDown,fMuon);
00166         Fermion muonR(tLepton,iDown,fMuon,cParticle,
hRight);
00167
00168         Fermion tau(tLepton,iDown,fTau);
00169         Fermion tauR(tLepton,iDown,fTau,cParticle,
hRight);

```

```

00170         Fermion neutrino(tLepton,iUp);
00171         Fermion neutrinotau(tLepton,iUp,fTau);
00172         Fermion neutrinomuon(tLepton,iUp,fMuon);
00173         Fermion neutrinoe(tLepton,iUp,fElectron);
00174
00175         Fermion up(tQuark,iUp,fElectron);
00176         Fermion down(tQuark,iDown,fElectron);
00177         Fermion bottom(tQuark,iDown,fTau);
00178         Fermion strange(tQuark,iDown,fMuon);
00179         Fermion charm(tQuark,iUp,fMuon);
00180         Fermion top(tQuark,iUp,fTau);
00181
00182         Meson Pi0d(down,down,Mpi0,Fpi);
00183         Meson Pi0u(down,down,Mpi0,Fpi);
00184         Meson Pip(up,down,Mpip,Fpi);
00185         Meson Pim(down,up,Mpip,Fpi);
00186
00187         Meson K0(down,strange,MK0,FK);
00188         Meson Kp(up,strange,MKp,FK);
00189
00190         Meson D0(charm,up,MD0,FD);
00191         Meson Dp(charm,down,MDp,FD);
00192         Meson Dsp(charm,strange,MDsp,FDs);
00193
00194         Meson B0(down,bottom,MB0,FB);
00195         Meson Bp(up,bottom,MBp,FB);
00196         Meson Bs0(strange,bottom,MBs0,FBs);
00197
00198         lst sb;
00199         //sb.append(mixes.M[tQuark][iUp][0][0]==0);
00200         sb.append(pow(abs(mixes.V[0][2][2]),2)==1-pow(abs(mixes.V[0][1][2]),2)-pow(abs(
mixes.V[0][0][2]),2));
00201         sb.append(pow(abs(mixes.V[0][2][1]),2)==1-pow(abs(mixes.V[0][1][1]),2)-pow(abs(
mixes.V[0][0][1]),2));
00202
00203         //cout<<pow(sqrt(2)/8*pow(g/MW,2),2)<<endl;
00204         //cout<<pow(1.166,2)<<endl;
00205         double fK=0.156;
00206         ex KKbar=ex(std::pow(fK,2))*mesonmixing(MK0,strange,down);
00207         ex eK=0.94*imag_part(KKbar)/3.5e-15/sqrt(2);
00208         cout<<"KKbar "<<KKbar<<endl;
00209         KKbar=expand(KKbar.subs(replacements).subs(lst(abs(wild()*pow(MR,-2))==abs(wild()*pow(
MR,-2))).subs(lst(log(wild()*pow(MR,-2))==log(wild())-2*log(MR))));
00210         KKbar=expand(KKbar.evalf());
00211         ex aKKbar=sqrt(KKbar.real_part()*KKbar.real_part()+KKbar.imag_part()*KKbar.imag_part());
00212
00213         eK=0.94*imag_part(KKbar)/3.5e-15/sqrt(2);
00214         eK=eK.subs(replacements).real_part();
00215         eK=collect_common_factors(expand(eK.evalf()));
00216         cout<<"eK"<<eK<<endl;
00217         //add("a_eK",abs(eK),new limitedobs(2.2e-3));
00218         epsilonK=new calcuex(new limitedobs(2*0.011e-3),abs(eK));
00219
00220
00221 }

```

Here is the call graph for this function:



### 7.2.2.2 BGL2::~~BGL2( ) [inline]

Definition at line 223 of file [draw.cpp](#).

```

00223 {epsilonK->~calcuex();}

```

### 7.2.3 Member Function Documentation

#### 7.2.3.1 double BGL2::bsgammawidth ( double *tanb\_*, double *McH\_*, double *MR\_*, double *MI\_*, int *option* = 0 ) [inline]

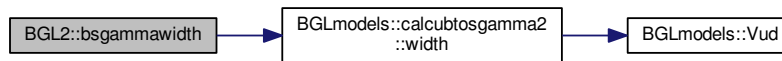
Definition at line 298 of file [draw.cpp](#).

References [BGLmodels::calcubtsgamma2::width\(\)](#).

```

00298                                     {
00299     parameters p=generateparameters();
00300     p[0].value=pow(10.0,tanb_);
00301     p[1].value=McH_;
00302     p[2].value=MR_;
00303     p[3].value=MI_;
00304     calcubtsgamma2 cal(mixes);
00305
00306     return cal.width(p,option);
00307 }
```

Here is the call graph for this function:



#### 7.2.3.2 double BGL2::epsK ( double *tanb\_*, double *McH\_*, double *MR\_*, double *MI\_*, int *option* = 0 ) [inline]

Definition at line 309 of file [draw.cpp](#).

References [parameters::p](#).

```

00309                                     {
00310     parameters p=generateparameters();
00311     p[0].value=pow(10.0,tanb_);
00312     p[1].value=McH_;
00313     p[2].value=MR_;
00314     p[3].value=MI_;
00315     p.p=1st(tanb==p[0].value,McH==p[1].value,MR==p[2].value,MI==p[3].value);
00316
00317
00318     return epsilonK->error(p);
00319 }
```

#### 7.2.3.3 parameters BGL2::generateparameters ( int *max* = 0 ) const [inline],[virtual]

Implements [Model](#).

Definition at line 225 of file [draw.cpp](#).

```

00225                                     {
00226     parameters p;
00227     //x=log_10(tanb)
00228     p.push_back(freeparameter(-3,3,r,stepsize));
00229     //y=log_10(McH)
00230     if(max==1) p.push_back(freeparameter(10,10000,r,stepsize));
00231     else p.push_back(freeparameter(10,mmmax,r,stepsize));
00232     //log_10(massR)
00233     p.push_back(freeparameter(-200,200,r,stepsize));
00234     //log_10(massI)
00235     p.push_back(freeparameter(-50,50,r,stepsize));
00236
00237     return p;
00238 }
```

### 7.2.3.4 parameters BGL2::getlist ( const parameters & p ) const [inline], [virtual]

Implements [Model](#).

Definition at line 241 of file [draw.cpp](#).

References [parameters::p](#), and [parameters::values](#).

```

00241                                     {
00242                                     //cout<<aux<<endl;
00243                                     //double
00244                                     c2=(1+sqrt(1-4*sqrt(ex_to<numeric>(mudecay.subs(1st(tanb==exp(p[0].value),McH==p[1].value)))>.to_double())))/2;
00245                                     double x=pow(10.0,p[0].value);
00246                                     //double y=pow(10.0,p[1].value);
00247                                     //double z=pow(10.0,p[2].value);
00248                                     //double w=pow(10.0,p[3].value);
00249
00250                                     double y=p[1].value;
00251                                     double z=y+p[2].value;
00252                                     double w=z+p[3].value;
00253
00254                                     parameters pp(p);
00255                                     pp[0].value=x;
00256                                     pp[2].value+=pp[1].value;
00257                                     pp[3].value+=pp[2].value;
00258                                     pp.values=vector<double>();
00259                                     for(uint i=0; i<4; i++) pp.values.push_back(pp[i].value);
00260                                     lst &l=pp.p;
00261                                     l=lst(tanb==x,McH==y,MR==z,MI==w);
00262                                     l.append(QCD1==inter1.Eval(y));
00263                                     l.append(QCD2==inter2.Eval(y));
00264
00265                                     for(uint i=0;i<3;i++){
00266                                         l.append(Mu[i]==Mu__[i].Eval(log(y)));
00267                                         l.append(Md[i]==Md__[i].Eval(log(y)));
00268                                     }
00269                                     return pp;
00270 }

```

### 7.2.3.5 ex BGL2::mesonmixing ( ex mesonmass, const Fermion & f1, const Fermion & f2 ) const [inline]

Definition at line 272 of file [draw.cpp](#).

```

00272                                     {
00273
00274                                     ex ret=0;
00275
00276                                     ex v1=0, v2=0;
00277                                     ex mq1=mixes.mass(f1),mq2=mixes.mass(f2);
00278                                     ex m2q1=mq1*mq1, m2q2=mq2*mq2;
00279
00280                                     for(uint i=0;i<bosons.size();i++)
00281                                         if(bosons[i].s==0){
00282                                             ex a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1)
00283 );
00284                                             v1=v1+pow(a/bosons[i].mass,2);
00285                                             ex b=(bosons[i].couplingdaggerR(f2,f1)+bosons[i].couplingdaggerL(f2,f1)
00286 );
00287                                             v2=v2+pow(b/bosons[i].mass,2);
00288                                         }
00289                                     ex fc=mesonmass/(mixes.massnum(f1)+mixes.massnum(f2));
00290                                     fc=pow(fc,2);
00291
00292                                     ret=2*(-v1*(1+l1*fc)+v2*(1+fc))*mesonmass/96;
00293
00294                                     return collect_common_factors(ret.subs(conjtoabs));
00295                                     //return
00296                                     expand(ret.subs(1st(exp(-I*wild()))==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
00297 }

```

## 7.2.4 Member Data Documentation

### 7.2.4.1 ex BGL2::alpha

Definition at line 325 of file [draw.cpp](#).

### 7.2.4.2 vector<int> BGL2::BGLtype

Definition at line 344 of file [draw.cpp](#).

### 7.2.4.3 vector< Boson > BGL2::bosons

Definition at line 329 of file [draw.cpp](#).

### 7.2.4.4 ex BGL2::BR\_Htotaunu

Definition at line 333 of file [draw.cpp](#).

### 7.2.4.5 ex BGL2::BR\_toptoHq

Definition at line 334 of file [draw.cpp](#).

### 7.2.4.6 ex BGL2::Btaunu

Definition at line 332 of file [draw.cpp](#).

### 7.2.4.7 ex BGL2::BtoD2taunuR

Definition at line 337 of file [draw.cpp](#).

### 7.2.4.8 ex BGL2::BtoDtaunuR

Definition at line 336 of file [draw.cpp](#).

### 7.2.4.9 ex BGL2::BtotaunuR

Definition at line 335 of file [draw.cpp](#).

### 7.2.4.10 Ist BGL2::conjtoabs

Definition at line 340 of file [draw.cpp](#).

#### 7.2.4.11 `ex BGL2::cos2`

Definition at line 325 of file [draw.cpp](#).

#### 7.2.4.12 `const possymbol BGL2::cp`

Definition at line 326 of file [draw.cpp](#).

#### 7.2.4.13 `calcuex* BGL2::epsilonK`

Definition at line 349 of file [draw.cpp](#).

#### 7.2.4.14 `const constant BGL2::FB`

Definition at line 324 of file [draw.cpp](#).

#### 7.2.4.15 `const constant BGL2::FBs`

Definition at line 324 of file [draw.cpp](#).

#### 7.2.4.16 `const constant BGL2::FD`

Definition at line 324 of file [draw.cpp](#).

#### 7.2.4.17 `const constant BGL2::FDs`

Definition at line 324 of file [draw.cpp](#).

#### 7.2.4.18 `const constant BGL2::FK`

Definition at line 324 of file [draw.cpp](#).

#### 7.2.4.19 `const constant BGL2::Fpi`

Definition at line 324 of file [draw.cpp](#).

#### 7.2.4.20 `ex BGL2::g`

Definition at line 325 of file [draw.cpp](#).

**7.2.4.21** `const possymbol BGL2::GF`

Definition at line 322 of file [draw.cpp](#).

**7.2.4.22** `int BGL2::iBD2taunu`

Definition at line 343 of file [draw.cpp](#).

**7.2.4.23** `int BGL2::iBDtaunu`

Definition at line 343 of file [draw.cpp](#).

**7.2.4.24** `int BGL2::iBtaunu`

Definition at line 343 of file [draw.cpp](#).

**7.2.4.25** `ROOT::Math::Interpolator BGL2::inter1`

Definition at line 345 of file [draw.cpp](#).

**7.2.4.26** `ROOT::Math::Interpolator BGL2::inter2`

Definition at line 345 of file [draw.cpp](#).

**7.2.4.27** `const constant BGL2::MB0`

Definition at line 323 of file [draw.cpp](#).

**7.2.4.28** `const constant BGL2::MBp`

Definition at line 323 of file [draw.cpp](#).

**7.2.4.29** `const constant BGL2::MBs0`

Definition at line 323 of file [draw.cpp](#).

**7.2.4.30** `const possymbol BGL2::McH`

Definition at line 326 of file [draw.cpp](#).

#### 7.2.4.31 possymbol BGL2::Md[3]

Definition at line 328 of file [draw.cpp](#).

#### 7.2.4.32 const constant BGL2::MD0

Definition at line 323 of file [draw.cpp](#).

#### 7.2.4.33 ROOT::Math::Interpolator BGL2::Md\_[3]

Definition at line 346 of file [draw.cpp](#).

#### 7.2.4.34 const constant BGL2::MDp

Definition at line 323 of file [draw.cpp](#).

#### 7.2.4.35 const constant BGL2::MDs0

Definition at line 323 of file [draw.cpp](#).

#### 7.2.4.36 const constant BGL2::MDsp

Definition at line 323 of file [draw.cpp](#).

#### 7.2.4.37 const possymbol BGL2::Mh

Definition at line 322 of file [draw.cpp](#).

#### 7.2.4.38 const possymbol BGL2::MI

Definition at line 326 of file [draw.cpp](#).

#### 7.2.4.39 const Mixes BGL2::mixes

Definition at line 339 of file [draw.cpp](#).

#### 7.2.4.40 const constant BGL2::MK0

Definition at line 323 of file [draw.cpp](#).



**7.2.4.41** `const constant BGL2::MKp`

Definition at line 323 of file [draw.cpp](#).

**7.2.4.42** `double BGL2::mmax`

Definition at line 347 of file [draw.cpp](#).

**7.2.4.43** `const constant BGL2::Mpi0`

Definition at line 323 of file [draw.cpp](#).

**7.2.4.44** `const constant BGL2::Mpip`

Definition at line 323 of file [draw.cpp](#).

**7.2.4.45** `const possymbol BGL2::MR`

Definition at line 326 of file [draw.cpp](#).

**7.2.4.46** `possymbol BGL2::Mu[3]`

Definition at line 328 of file [draw.cpp](#).

**7.2.4.47** `realsymbol BGL2::mu`

Definition at line 341 of file [draw.cpp](#).

**7.2.4.48** `ROOT::Math::Interpolator BGL2::Mu_[3]`

Definition at line 346 of file [draw.cpp](#).

**7.2.4.49** `const possymbol BGL2::MW`

Definition at line 322 of file [draw.cpp](#).

**7.2.4.50** `const possymbol BGL2::MZ`

Definition at line 322 of file [draw.cpp](#).

**7.2.4.51** `const double BGL2::planck`

Definition at line [321](#) of file [draw.cpp](#).

**7.2.4.52** `const realsymbol BGL2::QCD1`

Definition at line [327](#) of file [draw.cpp](#).

**7.2.4.53** `const realsymbol BGL2::QCD2`

Definition at line [327](#) of file [draw.cpp](#).

**7.2.4.54** `lst BGL2::replacements`

Definition at line [331](#) of file [draw.cpp](#).

**7.2.4.55** `const possymbol BGL2::rho`

Definition at line [326](#) of file [draw.cpp](#).

**7.2.4.56** `const realsymbol BGL2::Sparam`

Definition at line [327](#) of file [draw.cpp](#).

**7.2.4.57** `double BGL2::stepsize`

Definition at line [347](#) of file [draw.cpp](#).

**7.2.4.58** `const possymbol BGL2::tanb`

Definition at line [326](#) of file [draw.cpp](#).

**7.2.4.59** `const realsymbol BGL2::Tparam`

Definition at line [327](#) of file [draw.cpp](#).

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

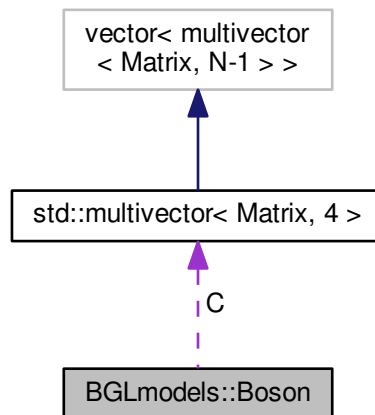
- [draw.cpp](#)

## 7.3 BGLmodels::Boson Class Reference

Gauge boson.

```
#include <BGL.h>
```

Collaboration diagram for BGLmodels::Boson:



### Public Member Functions

- [Boson](#) ()
- ex [couplingL](#) (const [Fermion](#) &f2, const [Fermion](#) &f1) const
- ex [couplingR](#) (const [Fermion](#) &f2, const [Fermion](#) &f1) const
- ex [couplingdaggerL](#) (const [Fermion](#) &f2, const [Fermion](#) &f1) const
- ex [couplingdaggerR](#) (const [Fermion](#) &f2, const [Fermion](#) &f1) const
- ex [coupling](#) (const [Fermion](#) &f2, const [Fermion](#) &f1, ex mu)
- void [reset](#) ()

### Public Attributes

- [BSpin](#) s
- ex [mass](#)
- [multivector](#)< [Matrix](#), 4 > C

### 7.3.1 Detailed Description

Gauge boson.

Definition at line 21 of file [BGL.h](#).

### 7.3.2 Constructor & Destructor Documentation

#### 7.3.2.1 BGLmodels::Boson::Boson ( ) [inline]

Definition at line 24 of file [BGL.h](#).

```
00024 : C(Matrix(),2,2,2,2){}
```

### 7.3.3 Member Function Documentation

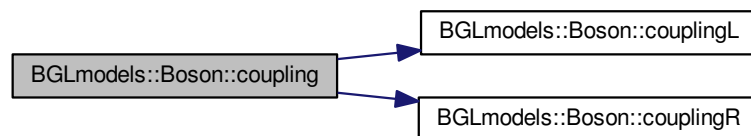
#### 7.3.3.1 ex BGLmodels::Boson::coupling ( const Fermion & f2, const Fermion & f1, ex mu ) [inline]

Definition at line 59 of file [BGL.h](#).

References [couplingL\(\)](#), [couplingR\(\)](#), [s](#), and [BGLmodels::sScalar](#).

```
00059                                     {
00060         if(s==sScalar) return couplingL(f2,f1)*dirac_gammaL()+
couplingR(f2,f1)*dirac_gammaR();
00061         else return couplingL(f2,f1)*dirac_gammaL()+couplingR(f2,f1)*dirac_gammaR
();
00062     }
```

Here is the call graph for this function:



#### 7.3.3.2 ex BGLmodels::Boson::couplingdaggerL ( const Fermion & f2, const Fermion & f1 ) const [inline]

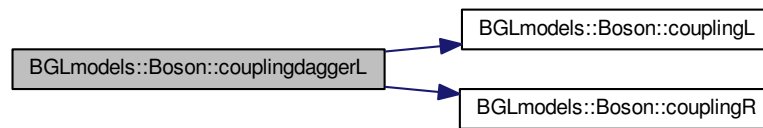
Definition at line 51 of file [BGL.h](#).

References [couplingL\(\)](#), [couplingR\(\)](#), [s](#), and [BGLmodels::sScalar](#).

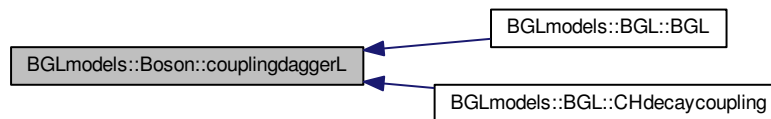
Referenced by [BGLmodels::BGL::BGL\(\)](#), and [BGLmodels::BGL::CHdecaycoupling\(\)](#).

```
00051                                     {
00052         if(s==sScalar) return couplingR(f1,f2).conjugate();
00053         return couplingL(f1,f2).conjugate();
00054     }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.3.3.3 `ex BGLmodels::Boson::couplingdaggerR ( const Fermion & f2, const Fermion & f1 ) const` `[inline]`

Definition at line 55 of file [BGL.h](#).

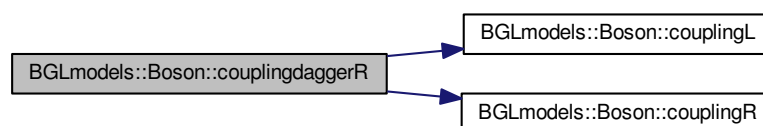
References [couplingL\(\)](#), [couplingR\(\)](#), [s](#), and [BGLmodels::sScalar](#).

Referenced by [BGLmodels::BGL::CHdecaycoupling\(\)](#).

```

00055                                     {
00056         if(s==sScalar) return couplingL(f1,f2).conjugate();
00057         return couplingR(f1,f2).conjugate();
00058     }
  
```

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.3.3.4 ex BGLmodels::Boson::couplingL ( const Fermion & f2, const Fermion & f1 ) const [inline]

Definition at line 26 of file [BGL.h](#).

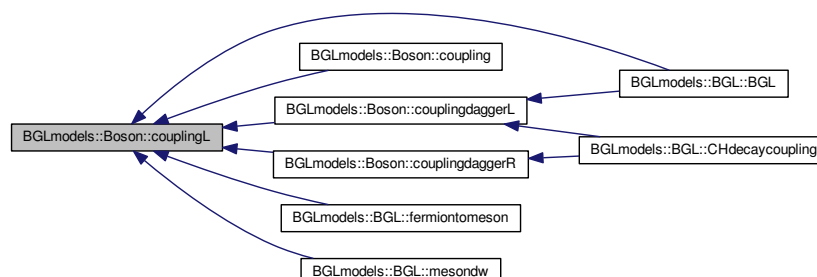
References [C](#), [BGLmodels::Fermion::flavour](#), [BGLmodels::Fermion::helicity](#), [BGLmodels::hLeft](#), [BGLmodels::hRight](#), [BGLmodels::Fermion::isospin](#), [s](#), [BGLmodels::sScalar](#), [BGLmodels::sVector](#), and [BGLmodels::Fermion::type](#).

Referenced by [BGLmodels::BGL::BGL\(\)](#), [coupling\(\)](#), [couplingdaggerL\(\)](#), [couplingdaggerR\(\)](#), [BGLmodels::BGL::fermionmeson\(\)](#), and [BGLmodels::BGL::mesondw\(\)](#).

```

00026                                     {
00027     bool quiralfilter=0;
00028     if(f1.type!=f2.type) return 0;
00029     if(s==sScalar){
00030         if(f1.helicity!=hRight && f2.helicity!=hLeft) quiralfilter=1;
00031     }
00032     else if(s==sVector){
00033         if(f1.helicity!=hRight && f2.helicity!=hRight) quiralfilter=1;
00034     }
00035     if(quiralfilter) return C[f2.type][f2.isospin][f1.isospin][
00036 hLeft][f2.flavour][f1.flavour];
00037     return 0;
00038 }
  
```

Here is the caller graph for this function:



7.3.3.5 `ex BGLmodels::Boson::couplingR ( const Fermion & f2, const Fermion & f1 ) const` `[inline]`

Definition at line 39 of file [BGL.h](#).

References [C](#), [BGLmodels::Fermion::flavour](#), [BGLmodels::Fermion::helicity](#), [BGLmodels::hLeft](#), [BGLmodels::hRight](#), [BGLmodels::Fermion::isospin](#), [s](#), [BGLmodels::sScalar](#), [BGLmodels::sVector](#), and [BGLmodels::Fermion::type](#).

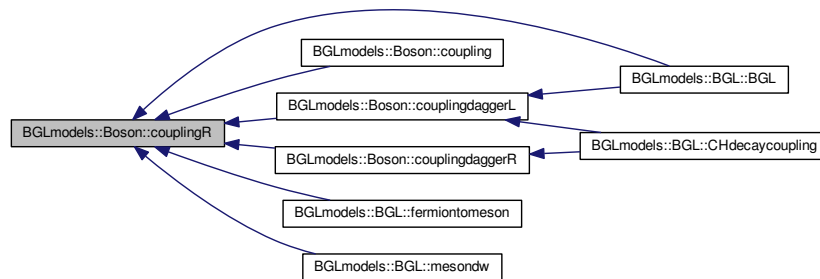
Referenced by [BGLmodels::BGL::BGL\(\)](#), [coupling\(\)](#), [couplingdaggerL\(\)](#), [couplingdaggerR\(\)](#), [BGLmodels::BGL::fermionmeson\(\)](#), and [BGLmodels::BGL::mesondw\(\)](#).

```

00039                                     {
00040         bool quiralfilter=0;
00041         if(f1.type!=f2.type) return 0;
00042         if(s==sScalar){
00043             if(f2.helicity!=hRight && f1.helicity!=hLeft) quiralfilter=1;
00044         }
00045         else if(s==sVector){
00046             if(f1.helicity!=hLeft && f2.helicity!=hLeft) quiralfilter=1;
00047         }
00048         if(quiralfilter) return C[f2.type][f2.isospin][f1.isospin][
00049             hRight][f2.flavour][f1.flavour];
00050         return 0;
00051     }

```

Here is the caller graph for this function:



7.3.3.6 `void BGLmodels::Boson::reset ( )` `[inline]`

Definition at line 63 of file [BGL.h](#).

References [C](#).

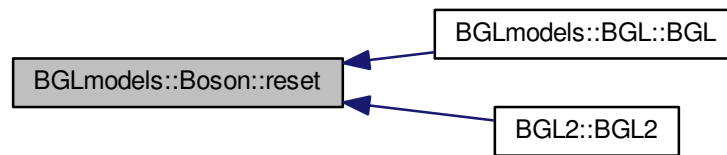
Referenced by [BGLmodels::BGL::BGL\(\)](#), and [BGL2::BGL2\(\)](#).

```

00063         {
00064             C=multivector<Matrix,4>(Matrix(),2,2,2,2);
00065         }

```

Here is the caller graph for this function:



### 7.3.4 Member Data Documentation

#### 7.3.4.1 `multivector<Matrix,4> BGLmodels::Boson::C`

Definition at line 68 of file [BGL.h](#).

Referenced by [BGLmodels::BGL::BGL\(\)](#), [BGL2::BGL2\(\)](#), [couplingL\(\)](#), [couplingR\(\)](#), and [reset\(\)](#).

#### 7.3.4.2 `ex BGLmodels::Boson::mass`

Definition at line 67 of file [BGL.h](#).

Referenced by [BGLmodels::BGL::BGL\(\)](#), [BGL2::BGL2\(\)](#), [BGLmodels::BGL::decaywidth\(\)](#), [BGLmodels::BGL::fermionmeson\(\)](#), [BGLmodels::BGL::gRR2\(\)](#), [BGLmodels::BGL::mesondw\(\)](#), [BGLmodels::BGL::mesonmixing\(\)](#), and [BGLmodels::BGL::tautomu\\_tautoe\(\)](#).

#### 7.3.4.3 `BSpin BGLmodels::Boson::s`

Definition at line 66 of file [BGL.h](#).

Referenced by [BGLmodels::BGL::BGL\(\)](#), [BGL2::BGL2\(\)](#), [coupling\(\)](#), [couplingdaggerL\(\)](#), [couplingdaggerR\(\)](#), [couplingL\(\)](#), [couplingR\(\)](#), [BGLmodels::BGL::decaywidth\(\)](#), [BGLmodels::BGL::fermionmeson\(\)](#), [BGLmodels::BGL::get\\_integral\\_symb\(\)](#), [BGLmodels::BGL::gRR2\(\)](#), [BGLmodels::BGL::mesondw\(\)](#), [BGLmodels::BGL::mesonmixing\(\)](#), and [BGLmodels::BGL::tautomu\\_tautoe\(\)](#).

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

- [BGL.h](#)

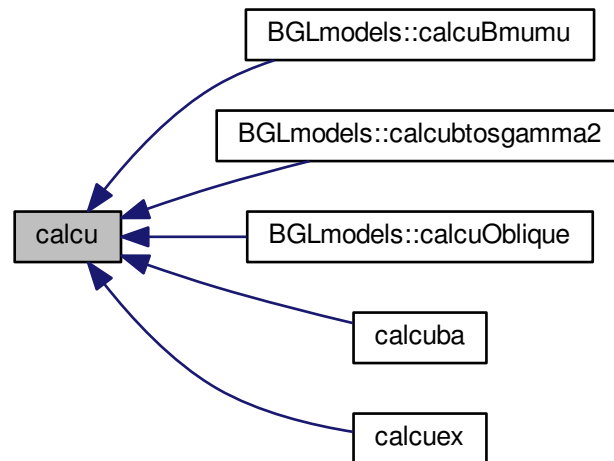


## 7.4 calcu Class Reference

Base class to do the calculus of a constraint to the model.

```
#include <model.h>
```

Inheritance diagram for calcu:



### Public Member Functions

- virtual double [operator\(\)](#) (const [parameters](#) &p) const =0

#### 7.4.1 Detailed Description

Base class to do the calculus of a constraint to the model.

Definition at line [237](#) of file [model.h](#).

#### 7.4.2 Member Function Documentation

**7.4.2.1** virtual double [calcu::operator\(\)](#) ( const [parameters](#) & *p* ) const [pure virtual]

Parameters

<i>hypothesis</i>	the theoretical hypothesis
-------------------	----------------------------

### Returns

the logarithm of the probability of measuring what was measured, assuming that the hypothesis is true

Implemented in [BGLmodels::calcuBmumu](#), [BGLmodels::calcutosgamma2](#), [calcuex](#), [calcuba](#), and [BGLmodels::calcuOblique](#).

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

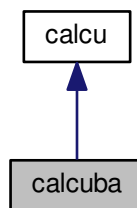
- [model.h](#)

## 7.5 calcuba Class Reference

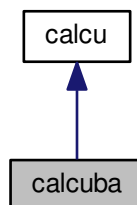
class to do the calculus of a constraint based on a GiNaC compiled expression

```
#include <model.h>
```

Inheritance diagram for calcuba:



Collaboration diagram for calcuba:



### Public Member Functions

- [calcuba](#) ([observable](#) \*ob, const FUNCP\_CUBA &e0)
- double [operator\(\)](#) (const [parameters](#) &p) const

## Public Attributes

- `shared_ptr< observable > o`
- `FUNCP_CUBA e`

### 7.5.1 Detailed Description

class to do the calculus of a constraint based on a GiNaC compiled expression

Definition at line 248 of file [model.h](#).

### 7.5.2 Constructor & Destructor Documentation

**7.5.2.1** `calcuba::calcuba( observable * ob, const FUNCP_CUBA & e0 )` `[inline]`

Definition at line 250 of file [model.h](#).

```
00250 : calcu(), o(ob), e(e0){}
```

### 7.5.3 Member Function Documentation

**7.5.3.1** `double calcuba::operator()( const parameters & p ) const` `[inline],[virtual]`

#### Parameters

<i>hypothesis</i>	the theoretical hypothesis
-------------------	----------------------------

#### Returns

the logarithm of the probability of measuring what was measured, assuming that the hypothesis is true

Implements [calcu](#).

Definition at line 252 of file [model.h](#).

References [parameters::values](#).

```
00252                                     {
00253         double ret=1000;
00254         int pass=1;
00255
00256         /* try{
00257             ret=ex_to<numeric>(e.subs(p.p,subs_options::no_pattern).evalf()).to_double();
00258         }
00259         catch(GiNaC::pole_error e){
00260             pass=0;
00261             cout<<"Pole error"<<endl;
00262         }
00263         catch(...){
00264             cout<<"Other exception"<<endl;
00265             exit(1);
00266         }
00267         */
00268         int n=p.values.size(), m=1;
00269         e(&n,&(p.values[0]),&m,&ret);
00270         if(pass) ret=o->loglikelihood(ret);
00271         else ret=1000;
00272
00273         return ret;
00274     }
```

## 7.5.4 Member Data Documentation

### 7.5.4.1 FUNCP\_CUBA `calcuba::e`

Definition at line 277 of file [model.h](#).

### 7.5.4.2 `shared_ptr<observable>` `calcuba::o`

Definition at line 276 of file [model.h](#).

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

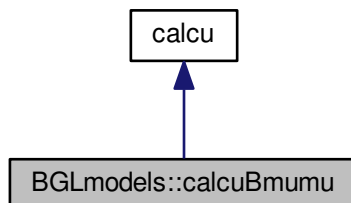
- [model.h](#)

## 7.6 BGLmodels::calcuBmumu Class Reference

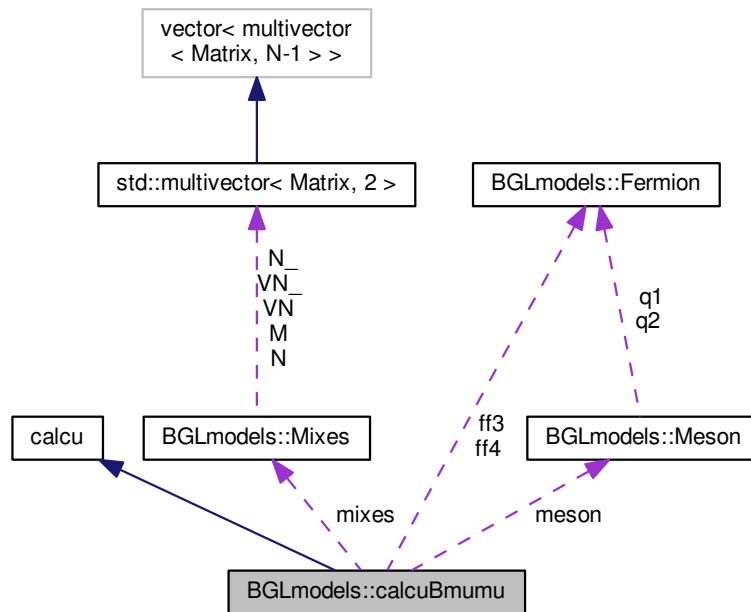
calculus of the constraints coming from the  $B \rightarrow \mu \mu$  decay

```
#include <Formulas.h>
```

Inheritance diagram for BGLmodels::calcuBmumu:



Collaboration diagram for BGLmodels::calcuBmumu:



## Public Member Functions

- `calcuBmumu` (const `Mixes` &mix, const `Meson` &m, const `Fermion` &f3, const `Fermion` &f4, `observable` \*ob, const char \*name)
- double `operator()` (const `parameters` &p) const
- double `obsvalue` (const `parameters` &p) const
- double `Y` (double x) const
- ex `mesondwtest` () const

## Public Attributes

- const `Meson` meson
- const `Fermion` & ff3
- const `Fermion` ff4
- `shared_ptr< observable >` o
- const realsymbol `gSr`
- const realsymbol `gSi`
- const realsymbol `gPr`
- const realsymbol `gPi`
- const realsymbol `gAr`
- const realsymbol `gAi`
- const `Mixes` mixes
- `FUNCP_CUBA` fp

### 7.6.1 Detailed Description

calculus of the constraints coming from the  $B \rightarrow \mu \mu$  decay

Definition at line 672 of file [Formulas.h](#).

### 7.6.2 Constructor & Destructor Documentation

**7.6.2.1** `BGLmodels::calcuBmumu::calcuBmumu ( const Mixes & mix, const Meson & m, const Fermion & f3, const Fermion & f4, observable * ob, const char * name ) [inline]`

Definition at line 674 of file [Formulas.h](#).

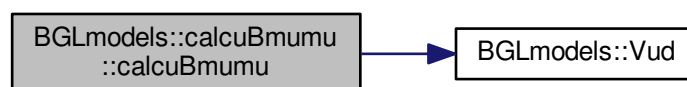
References [BGLmodels::Fermion::flavour](#), [BGLmodels::Fermion::isospin](#), [BGLmodels::Meson::q1](#), [BGLmodels::Meson::q2](#), [BGLmodels::tLepton](#), [BGLmodels::tQuark](#), and [BGLmodels::Vud\(\)](#).

```

00674
                                :
00675         meson(m), ff3(f3), ff4(f4),
00676         o(ob),
00677         gSr("gSr"), gSi("gSi"), gPr("gPr"), gPi("gPi"), gAr("gAr"),
         gAi("gAi"), mixes(mix) {
00678             const ex Nq=mixes.N[tQuark][m.q2.isospin][m.q1.flavour][m.q2.flavour];
00679             const ex Nq_=mixes.N[tQuark][m.q2.isospin][m.q1.flavour][m.q2.flavour].
         conjugate();
00680             const ex Nl=mixes.N[tLepton][f3.isospin][f3.flavour][f4.flavour];
00681             const ex Nl_=mixes.N[tLepton][f3.isospin][f4.flavour][f3.flavour].
         conjugate();
00682             possymbol MR("MR"), MI("MI"), Mch("Mch");
00683             ex MR2=MR*MR, MI2=MI*MI, Mch2=Mch*Mch;
00684
00685             ex cLL=Nq_*Nl_*(1/MR2-1/MI2);
00686             ex cLR=Nq_*Nl_*(1/MR2+1/MI2);
00687             ex cRL=Nq_*Nl_*(1/MR2+1/MI2);
00688             ex cRR=Nq_*Nl_*(1/MR2-1/MI2);
00689
00690             ex ggS=-(2*M_GF/sqrt(2))*(-cRL-cRR+cLL+cLR)/4).subs(
         mixes.replacements).evalf();
00691             ex ggP=-(2*M_GF/sqrt(2))*(cRL-cRR-cLL+cLR)/4).subs(
         mixes.replacements).evalf();
00692             CD ggA=0;
00693             if(m.q2.isospin==iDown && m.q2.flavour==2 && f3.flavour==1 && f4.flavour==1){
00694                 ggA=-conj(Vud[2][m.q2.flavour])*Vud[2][m.q1.flavour]*
         Y(std::pow(M_Mu[2]/M_MW,2));
00695                 ggA+=-conj(Vud[1][m.q2.flavour])*Vud[1][m.q1.flavour]*
         Y(std::pow(M_Mu[1]/M_MW,2));
00696                 ggA*=M_GF*M_GF*M_MW*M_MW/M_PI/M_PI/2;
00697
00698             }
00699             //ex ggA=0;
00700             //ex ggA=ggA.real()+I*ggA.imag();
00701
00702             ex width=collect_common_factors(mesondwtest()).subs(1st(
         gAr==ggA.real(), gAi==ggA.imag(), gSr==ggS.real_part(), gSi==ggS.imag_part(),
         gPr==ggP.real_part(), gPi==ggP.imag_part()))).subs(mixes.replacements).evalf().
         real_part();
00703
00704             compile_ex(1st(width), 1st(mixes.tanb, Mch, MR, MI),
         fp);
00705
00706         }

```

Here is the call graph for this function:



### 7.6.3 Member Function Documentation

#### 7.6.3.1 ex BGLmodels::calcuBmumu::mesondwtest( ) const [inline]

Definition at line 729 of file [Formulas.h](#).

```

00729     {
00730         const Fermion& f1(meson.q2), f2(meson.q1);
00731         ex mesonmass=meson.mass;
00732
00733         Fermion f3=ff3, f4=ff4;
00734         realsymbol q3("q3"), q4("q4");
00735         ex s2=pow(mesonmass,2);
00736
00737         ex v1=0, v2=0;
00738         ex mq1=mixes.mass(f1),mq2=mixes.mass(f2);
00739         ex mq3=mixes.mass(f3),mq4=mixes.mass(f4);
00740
00741         ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;
00742         scalar_products sp;
00743         sp.add(q4, q3, (s2-m2q4-m2q3)/2);
00744         sp.add(q3, q3, m2q3);
00745         sp.add(q4, q4, m2q4);
00746         ex q10=(s2+mq1*mq1-mq2*mq2)/(2*sqrt(s2)), lq1l=sqrt(q10*q10-mq1*mq1);
00747         ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);
00748
00749         ex a;
00750         a=-(gSr+I*gSi)*s2/(mq1+mq2);
00751         v1=v1+a*dirac_ONE();
00752         v2=v2+a.conjugate()*dirac_ONE();
00753         a=-(gPr+I*gPi)*s2/(mq1+mq2);
00754         v1=v1+a*dirac_gamma5();
00755         v2=v2-a.conjugate()*dirac_gamma5();
00756         ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
00757         a=(gAr+I*gAi);
00758         v1=v1+a*sl*dirac_gamma5();
00759         v2=v2+a.conjugate()*sl*dirac_gamma5();
00760
00761         ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)-mq4*dirac_ONE();
00762         ex dt=dirac_trace(vq3*v1*vq4*v2).simplify_indexed(sp);
00763         ex result=expand(dt*4*lq3l/s2/Pi/32);
00764
00765         lst ltest;
00766         //ltest.append(conjugate(gL)==pow(abs(gL),2)/gL);
00767         //ltest.append(conjugate(gR)==pow(abs(gR),2)/gR);
00768         //ltest.append(conjugate(gS)==pow(abs(gS),2)/gS);
00769         //ltest.append(conjugate(gP)==pow(abs(gP),2)/gP);
00770         //ltest.append(conjugate(gA)==pow(abs(gA),2)/gA);
00771
00772         return pow(meson.decay_factor,2)*collect_common_factors(result.subs(ltest));
00773         //return
00774         expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
00774     }

```

#### 7.6.3.2 double BGLmodels::calcuBmumu::obsvalue( const parameters & p ) const [inline]

Definition at line 716 of file [Formulas.h](#).

References [parameters::values](#).

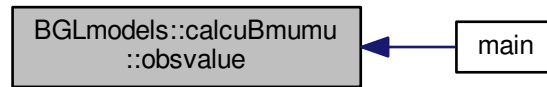
Referenced by [main\(\)](#).

```

00716     {
00717         //double
00718         factor=std::pow(M_GF*M_MW,4)/8/std::pow(M_PI,5)*std::sqrt(MM*MM-4*M_M1[1]*M_M1[1])*M_M1[1]*M_M1[1];
00719         int n=4,m=1;
00720         double ret=0;
00721         fp(&n,&(p.values[0]),&m,&ret);
00722         return ret;
00723     }

```

Here is the caller graph for this function:



**7.6.3.3** `double BGLmodels::calcuBmumu::operator() ( const parameters & p ) const` `[inline], [virtual]`

#### Parameters

<i>hypothesis</i>	the theoretical hypothesis
-------------------	----------------------------

#### Returns

the logarithm of the probability of measuring what was measured, assuming that the hypothesis is true

Implements [calcu](#).

Definition at line [707](#) of file [Formulas.h](#).

References [parameters::values](#).

```

00707                                     {
00708         //double
00709         factor=std::pow(M_GF*M_MW,4)/8/std::pow(M_PI,5)*std::sqrt(MM*MM-4*M_M1[1]*M_M1[1])*M_M1[1]*M_M1[1];
00710         int n=4,m=1;
00711         double ret=0;
00712         fp(&n,&(p.values[0]),&m,&ret);
00713         return o->loglikelihood(ret);
00714     }
  
```

**7.6.3.4** `double BGLmodels::calcuBmumu::Y ( double x ) const` `[inline]`

Definition at line [725](#) of file [Formulas.h](#).

```

00725                                     {
00726         return 1.0113*x/8/(1-x)*(4-x+3*x*log(x)/(1-x));
00727     }
  
```

## 7.6.4 Member Data Documentation

**7.6.4.1** `const Fermion& BGLmodels::calcuBmumu::ff3`

Definition at line [776](#) of file [Formulas.h](#).



#### 7.6.4.2 const Fermion BGLmodels::calcuBmumu::ff4

Definition at line 776 of file [Formulas.h](#).

#### 7.6.4.3 FUNCP\_CUBA BGLmodels::calcuBmumu::fp

Definition at line 780 of file [Formulas.h](#).

#### 7.6.4.4 const realsymbol BGLmodels::calcuBmumu::gAi

Definition at line 778 of file [Formulas.h](#).

#### 7.6.4.5 const realsymbol BGLmodels::calcuBmumu::gAr

Definition at line 778 of file [Formulas.h](#).

#### 7.6.4.6 const realsymbol BGLmodels::calcuBmumu::gPi

Definition at line 778 of file [Formulas.h](#).

#### 7.6.4.7 const realsymbol BGLmodels::calcuBmumu::gPr

Definition at line 778 of file [Formulas.h](#).

#### 7.6.4.8 const realsymbol BGLmodels::calcuBmumu::gSi

Definition at line 778 of file [Formulas.h](#).

#### 7.6.4.9 const realsymbol BGLmodels::calcuBmumu::gSr

Definition at line 778 of file [Formulas.h](#).

#### 7.6.4.10 const Meson BGLmodels::calcuBmumu::meson

Definition at line 775 of file [Formulas.h](#).

#### 7.6.4.11 const Mixes BGLmodels::calcuBmumu::mixes

Definition at line 779 of file [Formulas.h](#).

#### 7.6.4.12 `shared_ptr<observable> BGLmodels::calcuBmumu::o`

Definition at line 777 of file [Formulas.h](#).

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

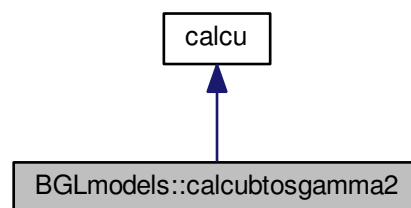
- [Formulas.h](#)

## 7.7 BGLmodels::calcubtosgamma2 Class Reference

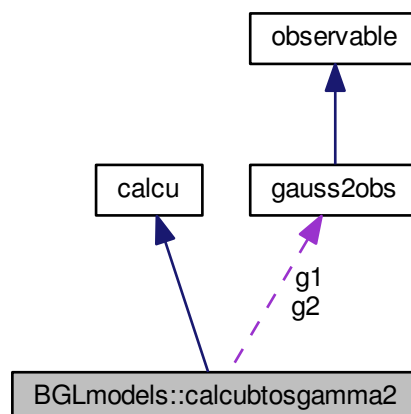
calculus of the constraints coming from the b->s gamma decay

```
#include <Formulas.h>
```

Inheritance diagram for BGLmodels::calcubtosgamma2:



Collaboration diagram for BGLmodels::calcubtosgamma2:



## Public Member Functions

- [calcubtogsamma2](#) (const [Mixes](#) &mixes)
- double [operator\(\)](#) (const [parameters](#) &p) const
- double [width](#) (const [parameters](#) &p, int option=0) const
- double [A0](#) (double x) const
- double [A1](#) (double x) const
- double [A2](#) (double x) const
- double [A3](#) (double x) const

## Public Attributes

- ROOT::Math::Interpolator [inter1](#)
- ROOT::Math::Interpolator [inter2](#)
- ROOT::Math::Interpolator [inter3](#)
- ROOT::Math::Interpolator [inter4](#)
- ROOT::Math::Interpolator [Mu\\_](#) [3]
- ROOT::Math::Interpolator [Md\\_](#) [3]
- const [uint](#) [ii](#)
- FUNCP\_CUBA [fp](#)
- const [gauss2obs](#) [g1](#)
- const [gauss2obs](#) [g2](#)
- double [ratio](#)

## Static Public Attributes

- static constexpr double [calN](#) =2.567e-3
- static constexpr double [a](#) =7.8221
- static constexpr double [aee](#) =0.4384
- static constexpr double [aer](#) =-1.6981
- static constexpr double [a77](#) =0.8161
- static constexpr double [a7r](#) =4.8802
- static constexpr double [a7er](#) =-0.7827
- static constexpr double [a88](#) =0.0197
- static constexpr double [a8r](#) =0.5680
- static constexpr double [a8er](#) =-0.0601
- static constexpr double [a87r](#) =0.1923
- static constexpr double [a7i](#) =0.3546
- static constexpr double [a8i](#) =-0.0987
- static constexpr double [aei](#) =2.4997
- static constexpr double [a87i](#) =-0.0487
- static constexpr double [a7ei](#) =-0.9067
- static constexpr double [a8ei](#) =-0.0661

### 7.7.1 Detailed Description

calculus of the constraints coming from the  $b \rightarrow s$  gamma decay

Definition at line 279 of file [Formulas.h](#).

## 7.7.2 Constructor & Destructor Documentation

### 7.7.2.1 BGLmodels::calcubtosgamma2::calcubtosgamma2 ( const Mixes & mixes ) [inline]

Definition at line 286 of file [Formulas.h](#).

References [BGLmodels::C7SM\\_Mt](#), [BGLmodels::C8SM\\_Mt](#), [BGLmodels::iDown](#), [BGLmodels::iUp](#), [BGLmodels::Mixes::N\\_](#), [BGLmodels::Mixes::replacements](#), [BGLmodels::Mixes::tanb](#), [BGLmodels::iQuark](#), [BGLmodels::Mixes::VN\\_](#), and [BGLmodels::Vud\(\)](#).

```

00286                                     :
00287         ii(2),
00288         g1(3.43e-4,sqrt(2)*0.23e-4),
00289         g2(9.2e-6,4e-6),
00290         ratio(0){
00291             //cout<<"C7 "<<C7SM_Mt<<" "<<C7SM_MW<<" "<<C7SM(std::pow(261.8/M_MW,2))<<endl;
00292             double res[2];
00293             constexpr double C7SM_[2]={C7SM_Mt,C7SM_Mt};
00294             constexpr double C8SM_[2]={C8SM_Mt,C8SM_Mt};
00295             for(uint j=0; j<2; j++){
00296                 const uint i=2;
00297                 const CD epsilon=conj(Vud[0][j])*Vud[0][i]/conj(Vud[2][j])/
Vud[2][i];
00298                 const double upsilon=norm(conj(Vud[2][j])*Vud[2][i]/Vud[1][i]);
00299                 const CD R7=(C7SM_Mt)/C7SM_MW;
00300                 const CD R8=(C8SM_Mt)/C8SM_MW;
00301                 const CD R7_=0;
00302                 const CD R8_=0;
00303
00304                 res[j]=a+ae*norm(epsilon)+aer*epsilon.real()+aei*epsilon.imag();
00305                 res[j]+=a77*(norm(R7)+norm(R7_))+a7r*R7.real()+a7i*R7.imag();
00306                 res[j]+=a88*(norm(R8)+norm(R8_))+a8r*R8.real()+a8i*R8.imag();
00307                 res[j]+=a87r*(R8*conj(R7)+R8_*conj(R7_)).real()+a7er*(R7*conj(epsilon)).real()+
a8er*(R8*conj(epsilon)).real();
00308                 res[j]+=a87i*(R8*conj(R7)+R8_*conj(R7_)).imag()+a7ei*(R7*conj(epsilon)).imag()+
a8ei*(R8*conj(epsilon)).imag();
00309                 res[j]*=calN/100*upsilon;
00310             }
00311             //cout<<"Btosgamma "<<res[0]/9.2e-6<<" "<<res[1]/3.15e-4<<endl;
00312             ifstream finter("interpolation.dat");
00313
00314             if(!finter.is_open()){
00315                 cout<<"ERROR: interpolation.dat not found"<<endl;
00316                 exit(1);
00317             }
00318             vector<double> vinter0, vinter1, vinter2;
00319             while(!finter.eof()){
00320                 double a=0,b=0,c=0;
00321                 finter>>a>>b>>c;
00322                 if(a!=0){
00323                     // cout<<a<<" "<<b<<" "<<c<<endl;
00324                     vinter0.push_back(a);
00325                     vinter1.push_back(b);
00326                     vinter2.push_back(c);
00327                 }
00328             }
00329
00330             inter1.SetData(vinter0,vinter1);
00331             inter2.SetData(vinter0,vinter2);
00332
00333             finter.close();
00334
00335             ifstream finter2("masses.dat");
00336
00337             if(!finter2.is_open()){
00338                 cout<<"ERROR: masses.dat not found"<<endl;
00339                 exit(1);
00340             }
00341             vector<vector<double>> m_(7);
00342             while(!finter2.eof()){
00343                 for(uint i=0; i<7;i++){
00344                     double a=0;
00345                     finter2>>a;
00346
00347                     if(a!=0){
00348                         if(i==0) a=log(a);
00349                         else if(i<4) a*=1e-3;
00350                         m_[i].push_back(a);
00351                     }
00352                     // cout<<a<<" ";

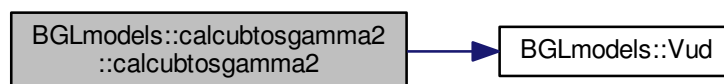
```

```

00353         } //cout<<endl;
00354     }
00355     for(uint i=0; i<3;i++) {
00356         Md_[i].SetData(m_[0],m_[2*i+1]);
00357         Mu_[i].SetData(m_[0],m_[2*i+2]);
00358     }
00359     // cout<<"Eval " <<Mu_[2].Eval(log(100.0))<<endl;
00360     // cout<<"Eval " <<Md_[2].Eval(log(100.0))<<endl;
00361
00362     finter2.close();
00363
00364     ifstream finter3("interpolation2.dat");
00365
00366     if(!finter3.is_open()){
00367         cout<<"ERROR: interpolation2.dat not found"<<endl;
00368         exit(1);
00369     }
00370     vector<double> vinter20, vinter21, vinter22;
00371     while(!finter3.eof()){
00372         double a=0,b=0,c=0;
00373         finter3>>a>>b>>c;
00374         if(a!=0){
00375             // cout<<a<<" " <<b<<" " <<c<<endl;
00376             vinter20.push_back(a);
00377             vinter21.push_back(b);
00378             vinter22.push_back(c);
00379         }
00380     }
00381
00382     inter3.SetData(vinter20,vinter21);
00383     inter4.SetData(vinter20,vinter22);
00384
00385     finter3.close();
00386
00387     vector<ex> vex(24);
00388
00389     const uint i=ii;
00390     for(uint j=0;j<2;j++){
00391         for(uint k=0;k<3;k++){
00392             vex[j*6+k*2+0]=mixes.VN_[tQuark][iUp][k][j].conjugate()*mixes.VN_[
tQuark][iUp][k][i];
00393             vex[j*6+k*2+1]=mixes.N_[tQuark][iDown][j][k]*mixes.N_[
tQuark][iDown][i][k].conjugate();
00394             vex[j*6+k*2+12]=-mixes.VN_[tQuark][iUp][k][j].conjugate()*mixes.VN_[
tQuark][iDown][k][i];
00395             vex[j*6+k*2+18]=mixes.VN_[tQuark][iDown][k][j].conjugate()*mixes.VN_[
tQuark][iDown][k][i];
00396         }
00397         lst l;
00398         for(uint k=0;k<vex.size();k++){
00399             vex[k]=vex[k].subs(mixes.replacements).evalf();
00400             l.append(vex[k].real_part());
00401             l.append(vex[k].imag_part());
00402         }
00403         compile_ex(l, lst(mixes.tanb), fp);
00404     }

```

Here is the call graph for this function:



## 7.7.3 Member Function Documentation

### 7.7.3.1 double BGLmodels::calcubtosgamma2::A0 ( double x ) const [inline]

Definition at line 640 of file [Formulas.h](#).

```

00640         {
00641         return x*(2+3*x-6*x*x+ x*x*x+6*x*std::log(x))/(24*std::pow(1-x,4));
00642         }

```

### 7.7.3.2 double BGLmodels::calcubtosgamma2::A1 ( double x ) const [inline]

Definition at line 644 of file [Formulas.h](#).

```

00644         {
00645         return x*(-3+4*x-x*x-2*std::log(x))/(4*std::pow(1-x,3));
00646         }

```

### 7.7.3.3 double BGLmodels::calcubtosgamma2::A2 ( double x ) const [inline]

Definition at line 648 of file [Formulas.h](#).

```

00648         {
00649         return x/(6*std::pow(1-x,3))*((-7+5*x+8*x*x)/6.0+x*std::log(x)/(1-x)*(-2+3*x));
00650         }

```

### 7.7.3.4 double BGLmodels::calcubtosgamma2::A3 ( double x ) const [inline]

Definition at line 652 of file [Formulas.h](#).

```

00652         {
00653         return (-3+8*x-5*x*x+(6*x-4)*std::log(x))*x/(6*std::pow(1-x,3));
00654         }

```

### 7.7.3.5 double BGLmodels::calcubtosgamma2::operator() ( const parameters & p ) const [inline],[virtual]

#### Parameters

<i>hypothesis</i>	the theoretical hypothesis
-------------------	----------------------------

#### Returns

the logarithm of the probability of measuring what was measured, assuming that the hypothesis is true

Implements [calcu](#).

Definition at line 406 of file [Formulas.h](#).

References [BGLmodels::C7SM\\_Mt](#), [BGLmodels::C8SM\\_Mt](#), [BGLmodels::mt\\_mt](#), and [BGLmodels::Vud\(\)](#).

```

00406         {
00407         double tanb=p[0].value;
00408         double y=p[1].value;

```

```

00409         double z=p[2].value;
00410         double w=p[3].value;
00411         double McH=y, MR=z, MI=w;
00412
00413         double y0=y;
00414         if(y<mt_mt) y0=mt_mt;
00415         double QCD1[2]={inter3.Eval(y0), inter1.Eval(y)};
00416         double QCD2[2]={inter4.Eval(y0), inter2.Eval(y)};
00417
00418         double Mu[3],Md[3];
00419
00420         for(uint i=0;i<3;i++){
00421             Mu[i]=Mu_[i].Eval(log(y));
00422             Md[i]=Md_[i].Eval(log(z));
00423         }
00424         const uint i=ii;
00425         CD CC7[2],DD7[2],CC8[2],DD8[2];
00426         double res[2];
00427         // constexpr double C7SM_[2]={C7SM_Mt,C7SM_Mb};
00428         // constexpr double C8SM_[2]={C8SM_Mt,C8SM_Mb};
00429
00430         std::array<double,48> ret;
00431         const int n=1,m=48;
00432         fp(&n,&(tanb),&m,&(ret[0]));
00433         for(uint j=0;j<2;j++){
00434             const double mbottom=Md[i];
00435             const double mstrange=Md[j];
00436             //ex mbottom=mixes.M[tQuark][iDown][i][i];
00437             //ex mstrange=mixes.M[tQuark][iDown][j][j];
00438
00439             CD C7,D7,C8,D8;
00440             for(uint k=0;k<3;k++){
00441                 double mup=Mu[k];
00442                 double mdown=Md[k];
00443                 //ex mup=mixes.M[tQuark][iUp][k][k];
00444                 //ex mdown=mixes.M[tQuark][iDown][k][k];
00445                 //f1+=
00446                 double mmu=std::pow(mup/McH,2);
00447                 double mmdR=std::pow(mdown/MR,2);
00448                 double mmdI=std::pow(mdown/MI,2);
00449
00450                 double A0u=A0(mmu);
00451                 double A1u=A1(mmu);
00452                 double A2u=A2(mmu);
00453                 double A3u=A3(mmu);
00454                 double A0d=(A0(mmdR)+A0(mmdI));
00455                 double A1d=(A1(mmdR)-A1(mmdI));
00456
00457                 CD f1(ret[j*12+4*k+0],ret[j*12+4*k+1]);
00458                 C7+=f1*A2u;
00459                 C8+=-2.0*f1*A0u;
00460
00461                 CD f2=CD(ret[36+j*2+4*k+0],ret[36+j*2+4*k+1])*mstrange*mbottom/mup/mup;
00462                 //CD f2=f1*mstrange*mbottom/mup/mup;
00463                 D7+=f2*A2u;
00464                 D8+=-2.0*f2*A0u;
00465
00466                 CD f12(ret[24+j*2+4*k+0],ret[24+j*2+4*k+1]);
00467                 C7+=f12*A3u;
00468                 C8+=2.0*f12*A1u;
00469
00470                 CD f4(ret[j*12+4*k+2],ret[j*12+4*k+3]);
00471                 C7+=f4*A0d/3.0;
00472                 C8+=-f4*A0d;
00473
00474                 C7+=f4*A1d/3.0;
00475                 C8+=-f4*A1d;
00476
00477                 CD f6=f4*mstrange*mbottom/mdown/mdown;
00478                 D7+=f6*A0d/3.0;
00479                 D8+=-f6*A0d;
00480             }
00481             uint j0=j;
00482             CC7[j]=(QCD1[j]*C7+QCD2[j]*C8)/2.0/conj(Vud[2][j])/Vud[2][i];
00483             DD7[j]=(QCD1[j]*D7+QCD2[j]*D8)/2.0/conj(Vud[2][j])/Vud[2][i];
00484             const double QCD3=(3*QCD2[j]/8+QCD1[j]);
00485             CC8[j]=QCD3*C8/2.0/conj(Vud[2][j])/Vud[2][i];
00486             DD8[j]=QCD3*D8/2.0/conj(Vud[2][j])/Vud[2][i];
00487             const CD epsilon=conj(Vud[0][j])*Vud[0][i]/conj(Vud[2][j])/
Vud[2][i];
00488             const double upsilon=norm(conj(Vud[2][j])*Vud[2][i]/Vud[1][i]);
00489             const CD R7=(C7SM_Mt+CC7[j])/C7SM_MW;
00490             const CD R8=(C8SM_Mt+CD(0)*CC8[j])/C8SM_MW;
00491             const CD R7_=(DD7[j])/C7SM_MW;
00492             const CD R8_=CD(0)*(DD8[j])/C8SM_MW;
00493
00494

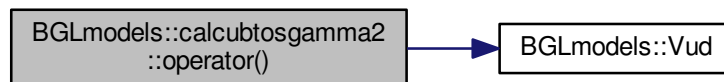
```

```

00495         res[j]=a+ae*norm(epsilon)+aer*epsilon.real()+aei*epsilon.imag();
00496         res[j]=a77*(norm(R7)+norm(R7_))+a7r*R7.real()+a7i*R7.imag();
00497         res[j]=a88*(norm(R8)+norm(R8_))+a8r*R8.real()+a8i*R8.imag();
00498         res[j]=a87r*(R8*conj(R7)+R8_*conj(R7_)).real()+a7er*(R7*conj(epsilon)).real()+
a8er*(R8*conj(epsilon)).real();
00499         res[j]=a87i*(R8*conj(R7)+R8_*conj(R7_)).imag()+a7ei*(R7*conj(epsilon)).imag()+
a8ei*(R8*conj(epsilon)).imag();
00500         res[j]=calN/100*upsilon;
00501
00502         /*res[j]=a+ae*norm(epsilon)+aer*epsilon.real()+aei*epsilon.imag();
00503         res[j]=a77*(norm(R7)+norm(R7_))+a7r*1+a7i*0;
00504         res[j]=a88*(norm(R8)+norm(R8_))+a8r*R8.real()+a8i*R8.imag();
00505         res[j]=a87r*1+a7er*(conj(epsilon)).real()+a8er*(R8*conj(epsilon)).real();
00506         res[j]=a87i*0+a7ei*(conj(epsilon)).imag()+a8ei*(R8*conj(epsilon)).imag();
00507         res[j]=calN/100*upsilon;
00508         */
00509     }
00510     double r1=3.15e-4+0.00247*(norm(CC7[1])+norm(DD7[1])-0.706*CC7[1].real());
00511
00512     //ratio=res[0]/9.2e-6;
00513     //cout<<"RATIO " <<ratio<<endl;
00514     return g1.loglikelihood(r1)+0*g2.loglikelihood(res[0]);
00515 }

```

Here is the call graph for this function:



**7.7.3.6** `double BGLmodels::calcubtosgamma2::width ( const parameters & p, int option = 0 ) const` [inline]

Definition at line 517 of file [Formulas.h](#).

References [BGLmodels::C7SM\\_Mt](#), [BGLmodels::C8SM\\_Mt](#), [BGLmodels::mt\\_mt](#), and [BGLmodels::Vud\(\)](#).

Referenced by [BGL2::bsgammawidth\(\)](#), and [BGLmodels::BGL::bsgammawidth\(\)](#).

```

00517     {
00518         double tanb=p[0].value;
00519         double y=p[1].value;
00520         double z=p[2].value;
00521         double w=p[3].value;
00522         double Mch=y, MR=z, MI=w;
00523
00524         double y0=y;
00525         if (y<mt_mt) y0=mt_mt;
00526         double QCD1[2]={inter3.Eval(y0),inter1.Eval(y)};
00527         double QCD2[2]={inter4.Eval(y0),inter2.Eval(y)};
00528
00529         double Mu[3],Md[3];
00530
00531         for(uint i=0;i<3;i++){
00532             Mu[i]=Mu_[i].Eval(log(y));
00533             Md[i]=Md_[i].Eval(log(z));
00534         }
00535         const uint i=ii;
00536         CD CC7[2],DD7[2],CC8[2],DD8[2];
00537         double res[2];
00538         // constexpr double C7SM_[2]={C7SM_Mt,C7SM_Mb};
00539         // constexpr double C8SM_[2]={C8SM_Mt,C8SM_Mb};
00540
00541         std::array<double,24> ret;

```



```

00542     const int n=1,m=24;
00543     fp(&n,&(tanb),&m,&(ret[0]));
00544     for(uint j=0;j<2;j++){
00545         const double mbottom=Md[i];
00546         const double mstrange=Md[j];
00547         //ex mbottom=mixes.M[tQuark][iDown][i][i];
00548         //ex mstrange=mixes.M[tQuark][iDown][j][j];
00549
00550         CD C7,D7,C8,D8;
00551         for(uint k=0;k<3;k++){
00552             double mup=Mu[k];
00553             double mdown=Md[k];
00554             //ex mup=mixes.M[tQuark][iUp][k][k];
00555             //ex mdown=mixes.M[tQuark][iDown][k][k];
00556             //f1+=
00557             double mmu=std::pow(mup/McH,2);
00558             double mmdR=std::pow(mdown/MR,2);
00559             double mmdI=std::pow(mdown/MI,2);
00560             double A0u=0,A1u=0,A2u=0,A3u=0,A0d=0,A1d=0;
00561
00562             if(option==0 || option==1){
00563                 A0u=A0(mmu);
00564                 A1u=A1(mmu);
00565                 A2u=A2(mmu);
00566                 A3u=A3(mmu);
00567             }
00568             if(option==0 || option==2){
00569                 A0d=(A0(mmdR)+A0(mmdI));
00570                 A1d=(A1(mmdR)-A1(mmdI));
00571             }
00572             if(option==3){
00573                 A0d=(A0(mmdR));
00574                 A1d=(A1(mmdR));
00575             }
00576             if(option==4){
00577                 A0d=(A0(mmdI));
00578                 A1d=(-A1(mmdI));
00579             }
00580
00581             CD f1(ret[j*12+4*k+0],ret[j*12+4*k+1]);
00582             C7+=f1*A2u;
00583             C8+=-2.0*f1*A0u;
00584
00585             CD f2=f1*mstrange*mbottom/mup/mup;
00586             D7+=f2*A2u;
00587             D8+=-2.0*f2*A0u;
00588
00589             C7+=-f1*A3u;
00590             C8+=-2.0*f1*A1u;
00591
00592             CD f4(ret[j*12+4*k+2],ret[j*12+4*k+3]);
00593             C7+=f4*A0d/3.0;
00594             C8+=-f4*A0d;
00595
00596             C7+=f4*A1d/3.0;
00597             C8+=-f4*A1d;
00598
00599             CD f6=f4*mstrange*mbottom/mdown/mdown;
00600             D7+=f6*A0d/3.0;
00601             D8+=-f6*A0d;
00602
00603         }
00604         uint j0=j;
00605         CC7[j]=(QCD1[j]*C7+QCD2[j]*C8)/2.0/conj(Vud[2][j])/Vud[2][i];
00606         DD7[j]=(QCD1[j]*D7+QCD2[j]*D8)/2.0/conj(Vud[2][j])/Vud[2][i];
00607         const double QCD3=(3*QCD2[j]/8+QCD1[j]);
00608         CC8[j]=QCD3*C8/2.0/conj(Vud[2][j])/Vud[2][i];
00609         DD8[j]=QCD3*D8/2.0/conj(Vud[2][j])/Vud[2][i];
00610         const CD epsilon=conj(Vud[0][j])*Vud[0][i]/conj(Vud[2][j])/
Vud[2][i];
00611         const double upsi=norm(conj(Vud[2][j])*Vud[2][i]/Vud[1][i]);
00612         const CD R7=(C7SM_Mt+CC7[j])/C7SM_MW;
00613         const CD R8=(C8SM_Mt+CD(0)*CC8[j])/C8SM_MW;
00614         const CD R7_=(DD7[j])/C7SM_MW;
00615         const CD R8_=(DD8[j])/C8SM_MW;
00616
00617         res[j]=a+aee*norm(epsilon)+aer*epsilon.real()+aei*epsilon.imag();
00618         res[j]+=a77*(norm(R7)+norm(R7_))+a7r*R7.real()+a7i*R7.imag();
00619         res[j]+=a88*(norm(R8)+norm(R8_))+a8r*R8.real()+a8i*R8.imag();
00620         res[j]+=a87r*(R8*conj(R7)+R8_*conj(R7_)).real()+a7er*(R7*conj(epsilon)).real()+
a8er*(R8*conj(epsilon)).real();
00621         res[j]+=a87i*(R8*conj(R7)+R8_*conj(R7_)).imag()+a7ei*(R7*conj(epsilon)).imag()+
a8er*(R8*conj(epsilon)).imag();
00622         res[j]*=calN/100*upsi;
00623
00624         //res[j]=a+aee*norm(epsilon)+aer*epsilon.real()+aei*epsilon.imag();
00625

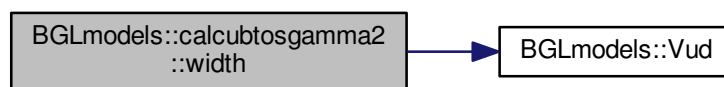
```

```

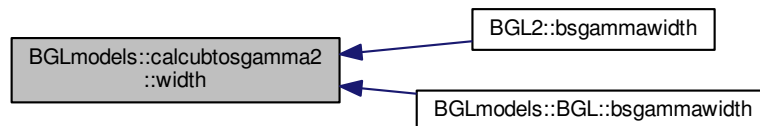
00626         res[j]=a77*(norm(R7)+norm(R7_))+a7r*1+a7i*0;
00627         res[j]=a88*(norm(R8)+norm(R8_))+a8r*R8.real()+a8i*R8.imag();
00628         res[j]=a87r*1+a7er*(conj(epsilon)).real()+a8er*(R8*conj(epsilon)).real();
00629         res[j]=a87i*0+a7ei*(conj(epsilon)).imag()+a8er*(R8*conj(epsilon)).imag();
00630         res[j]=calN/100*upsilon;
00631         */
00632     }
00633     double r1=3.15e-4+0.00247*(norm(CC7[1])+norm(DD7[1])-0.706*CC7[1].real());
00634
00635     //ratio=res[0]/9.2e-6;
00636     //cout<<"RATIO " <<ratio<<endl;
00637     return gl.error(r1);
00638 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



## 7.7.4 Member Data Documentation

**7.7.4.1** `constexpr double BGLmodels::calcubtosgamma2::a = 7.8221` `[static]`

Definition at line 283 of file [Formulas.h](#).

**7.7.4.2** `constexpr double BGLmodels::calcubtosgamma2::a77 = 0.8161` `[static]`

Definition at line 283 of file [Formulas.h](#).

**7.7.4.3** `constexpr double BGLmodels::calcubtosgamma2::a7ei = -0.9067` `[static]`

Definition at line 284 of file [Formulas.h](#).

7.7.4.4 constexpr double BGLmodels::calcubtosgamma2::a7e  $\approx -0.7827$  [static]

Definition at line 283 of file [Formulas.h](#).

7.7.4.5 constexpr double BGLmodels::calcubtosgamma2::a7i  $\approx 0.3546$  [static]

Definition at line 284 of file [Formulas.h](#).

7.7.4.6 constexpr double BGLmodels::calcubtosgamma2::a7r  $\approx 4.8802$  [static]

Definition at line 283 of file [Formulas.h](#).

7.7.4.7 constexpr double BGLmodels::calcubtosgamma2::a87i  $\approx -0.0487$  [static]

Definition at line 284 of file [Formulas.h](#).

7.7.4.8 constexpr double BGLmodels::calcubtosgamma2::a87r  $\approx 0.1923$  [static]

Definition at line 284 of file [Formulas.h](#).

7.7.4.9 constexpr double BGLmodels::calcubtosgamma2::a88  $\approx 0.0197$  [static]

Definition at line 283 of file [Formulas.h](#).

7.7.4.10 constexpr double BGLmodels::calcubtosgamma2::a8ei  $\approx -0.0661$  [static]

Definition at line 284 of file [Formulas.h](#).

7.7.4.11 constexpr double BGLmodels::calcubtosgamma2::a8er  $\approx -0.0601$  [static]

Definition at line 284 of file [Formulas.h](#).

7.7.4.12 constexpr double BGLmodels::calcubtosgamma2::a8i  $\approx -0.0987$  [static]

Definition at line 284 of file [Formulas.h](#).

7.7.4.13 constexpr double BGLmodels::calcubtosgamma2::a8r  $\approx 0.5680$  [static]

Definition at line 283 of file [Formulas.h](#).

7.7.4.14 `constexpr double BGLmodels::calcubtosgamma2::aee = 0.4384` `[static]`

Definition at line 283 of file [Formulas.h](#).

7.7.4.15 `constexpr double BGLmodels::calcubtosgamma2::aei = 2.4997` `[static]`

Definition at line 284 of file [Formulas.h](#).

7.7.4.16 `constexpr double BGLmodels::calcubtosgamma2::aer = -1.6981` `[static]`

Definition at line 283 of file [Formulas.h](#).

7.7.4.17 `constexpr double BGLmodels::calcubtosgamma2::calN = 2.567e-3` `[static]`

Definition at line 282 of file [Formulas.h](#).

7.7.4.18 `FUNCP_CUBA BGLmodels::calcubtosgamma2::fp`

Definition at line 660 of file [Formulas.h](#).

7.7.4.19 `const gauss2obs BGLmodels::calcubtosgamma2::g1`

Definition at line 661 of file [Formulas.h](#).

7.7.4.20 `const gauss2obs BGLmodels::calcubtosgamma2::g2`

Definition at line 661 of file [Formulas.h](#).

7.7.4.21 `const uint BGLmodels::calcubtosgamma2::ii`

Definition at line 659 of file [Formulas.h](#).

7.7.4.22 `ROOT::Math::Interpolator BGLmodels::calcubtosgamma2::inter1`

Definition at line 656 of file [Formulas.h](#).

7.7.4.23 `ROOT::Math::Interpolator BGLmodels::calcubtosgamma2::inter2`

Definition at line 656 of file [Formulas.h](#).

#### 7.7.4.24 ROOT::Math::Interpolator BGLmodels::calcubtosgamma2::inter3

Definition at line 656 of file [Formulas.h](#).

#### 7.7.4.25 ROOT::Math::Interpolator BGLmodels::calcubtosgamma2::inter4

Definition at line 656 of file [Formulas.h](#).

#### 7.7.4.26 ROOT::Math::Interpolator BGLmodels::calcubtosgamma2::Md\_[3]

Definition at line 657 of file [Formulas.h](#).

#### 7.7.4.27 ROOT::Math::Interpolator BGLmodels::calcubtosgamma2::Mu\_[3]

Definition at line 657 of file [Formulas.h](#).

#### 7.7.4.28 double BGLmodels::calcubtosgamma2::ratio [mutable]

Definition at line 662 of file [Formulas.h](#).

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

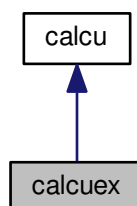
- [Formulas.h](#)

## 7.8 calcuex Class Reference

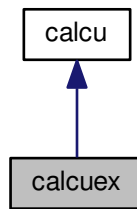
class to do the calculus of a constraint based on a GiNaC symbolic expression

```
#include <model.h>
```

Inheritance diagram for calcuex:



Collaboration diagram for `calcuex`:



## Public Member Functions

- `calcuex (observable *ob, const ex &e0)`
- `~calcuex ()`
- `double operator() (const parameters &p) const`
- `double error (const parameters &p) const`

## Public Attributes

- `shared_ptr< observable > o`
- `ex e`

### 7.8.1 Detailed Description

class to do the calculus of a constraint based on a GiNaC symbolic expression

Definition at line 282 of file [model.h](#).

### 7.8.2 Constructor & Destructor Documentation

#### 7.8.2.1 `calcuex::calcuex ( observable * ob, const ex & e0 )` `[inline]`

Definition at line 284 of file [model.h](#).

```
00284 :   calculu(), o(ob), e(e0) {}
```

#### 7.8.2.2 `calcuex::~calcuex ( )` `[inline]`

Definition at line 285 of file [model.h](#).

```
00285 {}
```

### 7.8.3 Member Function Documentation

#### 7.8.3.1 double calcuex::error ( const parameters & p ) const [inline]

Definition at line 310 of file [model.h](#).

References [parameters::p](#).

```

00310                                     {
00311             double ret=1000;
00312             int pass=1;
00313             try{
00314                 cout<<e<<endl;
00315                 cout<<e.subs(p.p)<<endl;
00316
00317                 ret=ex_to<numeric>(e.subs(p.p,subs_options::no_pattern).evalf()).to_double();
00318             }
00319             catch(GiNaC::pole_error er){
00320                 pass=0;
00321                 cout<<"Pole error"<<endl;
00322             }
00323             catch(exception er){
00324                 pass=0;
00325
00326                 cout<<er.what()<<endl;
00327                 cout<<e.subs(p.p,subs_options::no_pattern).evalf()<<endl;
00328             }
00329             catch(...) {
00330                 cout<<"Other exception"<<endl;
00331                 exit(1);
00332             }
00333             if(pass) ret=o->error(ret);
00334             else ret=1000;
00335
00336             return ret;
00337     }

```

#### 7.8.3.2 double calcuex::operator() ( const parameters & p ) const [inline],[virtual]

##### Parameters

<i>hypothesis</i>	the theoretical hypothesis
-------------------	----------------------------

##### Returns

the logarithm of the probability of measuring what was measured, assuming that the hypothesis is true

Implements [calcu](#).

Definition at line 287 of file [model.h](#).

References [parameters::p](#).

```

00287                                     {
00288             double ret=1000;
00289             int pass=1;
00290             try{
00291                 ret=ex_to<numeric>(e.subs(p.p,subs_options::no_pattern).evalf()).to_double();
00292             }
00293             catch(GiNaC::pole_error e){
00294                 pass=0;
00295                 cout<<"Pole error"<<endl;
00296             }
00297             catch(exception e){
00298                 cout<<e.what()<<endl;

```

```

00299         }
00300         catch(...) {
00301             cout<<"Other exception"<<endl;
00302             exit(1);
00303         }
00304         if (pass) ret=o->loglikelihood(ret);
00305         else ret=1000;
00306
00307         return ret;
00308     }

```

## 7.8.4 Member Data Documentation

### 7.8.4.1 ex calcuex::e

Definition at line 340 of file [model.h](#).

### 7.8.4.2 shared\_ptr<observable> calcuex::o

Definition at line 339 of file [model.h](#).

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

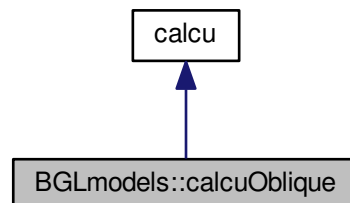
- [model.h](#)

## 7.9 BGLmodels::calcuOblique Class Reference

calculus of the constraints coming from the oblique parameters

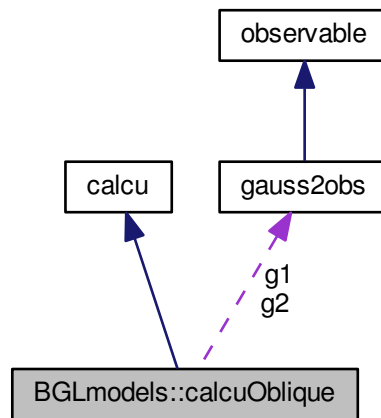
```
#include <Formulas.h>
```

Inheritance diagram for BGLmodels::calcuOblique:





Collaboration diagram for BGLmodels::calcuOblique:



## Public Member Functions

- [calcuOblique](#) ()
- double [operator\(\)](#) (const [parameters](#) &p) const
- double [F](#) (double x, double y) const
- double [f](#) (double t, double r) const
- double [lnxy\\_xy](#) (double x, double y) const
- double [G](#) (double x, double y, double z) const

## Public Attributes

- const double [c1](#)
- const double [c2](#)
- const [gauss2obs](#) [g1](#)
- const [gauss2obs](#) [g2](#)

### 7.9.1 Detailed Description

calculus of the constraints coming from the oblique parameters

Definition at line 221 of file [Formulas.h](#).

### 7.9.2 Constructor & Destructor Documentation

#### 7.9.2.1 BGLmodels::calcuOblique::calcuOblique ( ) [inline]

Definition at line 224 of file [Formulas.h](#).

```
00224 : c1(0.741), c2(0.671), g1(c1+0.02,0.0397), g2(c2+0.02,0.1579) {}
```

### 7.9.3 Member Function Documentation

#### 7.9.3.1 double BGLmodels::calcuOblique::F ( double x, double y ) const [inline]

Definition at line 238 of file [Formulas.h](#).

```
00238                                     {
00239     if(x==y) return 0;
00240     return (x+y)/2-x*y*log(x/y)/(x-y);
00241 }
```

#### 7.9.3.2 double BGLmodels::calcuOblique::f ( double t, double r ) const [inline]

Definition at line 242 of file [Formulas.h](#).

```
00242                                     {
00243     if(r==0) return 0;
00244     if(r<0) return 2*sqrt(-r)*atan(sqrt(-r)/t);
00245     return sqrt(r)*log(fabs((t-sqrt(r))/(t+sqrt(r))));
00246 }
```

#### 7.9.3.3 double BGLmodels::calcuOblique::G ( double x, double y, double z ) const [inline]

Definition at line 252 of file [Formulas.h](#).

```
00252                                     {
00253     double t=x+y-z;
00254     double r=std::pow(z,2)-2*z*(x+y)+std::pow(x-y,2);
00255     return -16.0/3+5*(x+y)/z-2*std::pow((x-y)/z,2)+r/std::pow(z,3)*f(t,r)+\
00256         3/z*lnxy_xy(x,y)*(std::pow(x,2)+std::pow(y,2)+(x-y)/std::pow(z,2)*(-std::pow
00257 (x,2)+std::pow(y,2)+std::pow(x-y,3)/3));
00257 }
```

#### 7.9.3.4 double BGLmodels::calcuOblique::lnxy\_xy ( double x, double y ) const [inline]

Definition at line 248 of file [Formulas.h](#).

```
00248                                     {
00249     if(x==y) return 1/y;
00250     return log(x/y)/(x-y);
00251 }
```

#### 7.9.3.5 double BGLmodels::calcuOblique::operator() ( const parameters & p ) const [inline],[virtual]

##### Parameters

<i>hypothesis</i>	the theoretical hypothesis
-------------------	----------------------------

**Returns**

the logarithm of the probability of measuring what was measured, assuming that the hypothesis is true

Implements [calcu](#).

Definition at line 225 of file [Formulas.h](#).

```

00225                                     {
00226         double y=p[1].value;
00227         double z=p[2].value;
00228         double w=p[3].value;
00229
00230         double TT=(F(y*y,z*z)-F(w*w,z*z)+F(y*y,w*w))/(16*M_PI*M_MW*
M_MW*(1-M_cos2));
00231         double Sparam=(std::pow(1-2*M_cos2,2)*G(y*y,y*y,M_MZ*M_MZ)+
G(z*z,w*w,M_MZ*M_MZ)+2*log(z*w/y/y))/24/M_PI;
00232
00233         double T1=c1*TT-c2*Sparam;
00234         double T2=c2*TT+c1*Sparam;
00235
00236         return g1.loglikelihood(T1)+g2.loglikelihood(T2);
00237     }
```

**7.9.4 Member Data Documentation****7.9.4.1 const double BGLmodels::calcuOblique::c1**

Definition at line 258 of file [Formulas.h](#).

**7.9.4.2 const double BGLmodels::calcuOblique::c2**

Definition at line 258 of file [Formulas.h](#).

**7.9.4.3 const gauss2obs BGLmodels::calcuOblique::g1**

Definition at line 259 of file [Formulas.h](#).

**7.9.4.4 const gauss2obs BGLmodels::calcuOblique::g2**

Definition at line 259 of file [Formulas.h](#).

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

- [Formulas.h](#)

**7.10 discreteparameter Class Reference**

A parameter which will be fitted in the simulation.

```
#include <model.h>
```

## Public Member Functions

- [discreteparameter](#) (int *mi*, int *ma*, TRandom3 \**r*)

## Public Attributes

- double [min](#)  
*minimum possible value for the parameter*
- double [max](#)  
*maximum possible value for the parameter*
- double [value](#)  
*value of the parameter*

### 7.10.1 Detailed Description

A parameter which will be fitted in the simulation.

Definition at line [161](#) of file [model.h](#).

### 7.10.2 Constructor & Destructor Documentation

7.10.2.1 `discreteparameter::discreteparameter ( int mi, int ma, TRandom3 * r )` `[inline]`

#### Parameters

<i>mi</i>	minimum possible value for the parameter
<i>ma</i>	maximum possible value for the parameter
<i>r</i>	random number generator

Definition at line [167](#) of file [model.h](#).

```
00167 : min(mi), max(ma), value(mi+r->Integer(ma-mi+1)) {}
```

### 7.10.3 Member Data Documentation

7.10.3.1 `double discreteparameter::max`

maximum possible value for the parameter

Definition at line [171](#) of file [model.h](#).

7.10.3.2 `double discreteparameter::min`

minimum possible value for the parameter

Definition at line [169](#) of file [model.h](#).

## 7.10.3.3 double discreteparameter::value

value of the parameter

Definition at line 173 of file [model.h](#).

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

- [model.h](#)

## 7.11 BGLmodels::Fermion Class Reference

a fermion properties

```
#include <Formulas.h>
```

### Public Member Functions

- [Fermion](#) (FType t, FIsospin i, FFlavour f=fAny, FCharge p=cParticle, FHelicity h=hAny)

### Public Attributes

- [FType](#) type
- [FIsospin](#) isospin
- [FFlavour](#) flavour
- [FCharge](#) particle
- [FHelicity](#) helicity

### 7.11.1 Detailed Description

a fermion properties

Definition at line 32 of file [Formulas.h](#).

### 7.11.2 Constructor & Destructor Documentation

7.11.2.1 BGLmodels::Fermion::Fermion ( FType t, FIsospin i, FFlavour f=fAny, FCharge p=cParticle, FHelicity h=hAny ) [inline]

Definition at line 35 of file [Formulas.h](#).

```
00035 : type(t), isospin(i), flavour(f), particle(p),
      helicity(h) {}
```

### 7.11.3 Member Data Documentation

#### 7.11.3.1 FFlavour BGLmodels::Fermion::flavour

Definition at line 39 of file [Formulas.h](#).

Referenced by [BGLmodels::calcuBmumu::calcuBmumu\(\)](#), [BGLmodels::BGL::CHdecaycoupling\(\)](#), [BGLmodels::BGL::Boson::couplingL\(\)](#), [BGLmodels::Boson::couplingR\(\)](#), [BGLmodels::BGL::decaywidth\(\)](#), [BGLmodels::BGL::fermiontomeson\(\)](#), [BGLmodels::BGL::fermiontomesontest\(\)](#), [BGLmodels::BGL::gRR2\(\)](#), [BGLmodels::Mixes::mass\(\)](#), [BGLmodels::Mixes::massnum\(\)](#), [BGLmodels::BGL::mesondw\(\)](#), [BGLmodels::BGL::mesondwtest\(\)](#), and [BGLmodels::BGL::tautomu\\_tautoe\(\)](#).

#### 7.11.3.2 FHelicity BGLmodels::Fermion::helicity

Definition at line 41 of file [Formulas.h](#).

Referenced by [BGLmodels::Boson::couplingL\(\)](#), and [BGLmodels::Boson::couplingR\(\)](#).

#### 7.11.3.3 FIsospin BGLmodels::Fermion::isospin

Definition at line 38 of file [Formulas.h](#).

Referenced by [BGLmodels::calcuBmumu::calcuBmumu\(\)](#), [BGLmodels::Boson::couplingL\(\)](#), [BGLmodels::Boson::couplingR\(\)](#), [BGLmodels::Mixes::mass\(\)](#), and [BGLmodels::Mixes::massnum\(\)](#).

#### 7.11.3.4 FCharge BGLmodels::Fermion::particle

Definition at line 40 of file [Formulas.h](#).

#### 7.11.3.5 FType BGLmodels::Fermion::type

Definition at line 37 of file [Formulas.h](#).

Referenced by [BGLmodels::Boson::couplingL\(\)](#), [BGLmodels::Boson::couplingR\(\)](#), [BGLmodels::Mixes::mass\(\)](#), and [BGLmodels::Mixes::massnum\(\)](#).

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

- [Formulas.h](#)

## 7.12 freeparameter Class Reference

A parameter which will be fitted in the simulation.

```
#include <model.h>
```

## Public Member Functions

- [freeparameter](#) (double *mi*, double *ma*, TRandom3 \**r*, double *ss*=1e-2)
- void [next](#) (TRandom3 \**r*, double *f*=1)  
*changes randomly the ::value of the parameter, the standard deviation is ::step*
- bool [isvalid](#) () const  
*checks if the value of the parameter is between ::min and ::max*
- double [dist](#) (double *x*) const  
*probability distribution, to be used by the Markov Chain Monte Carlo simulation*

## Public Attributes

- double [min](#)  
*minimum possible value for the parameter*
- double [max](#)  
*maximum possible value for the parameter*
- double [value](#)  
*value of the parameter*
- double [step](#)  
*standard deviation of the random changes of ::value in next(TRandom3 \*)*

### 7.12.1 Detailed Description

A parameter which will be fitted in the simulation.

Definition at line 124 of file [model.h](#).

### 7.12.2 Constructor & Destructor Documentation

7.12.2.1 `freeparameter::freeparameter ( double mi, double ma, TRandom3 * r, double ss = 1e-2 )` `[inline]`

#### Parameters

<i>mi</i>	minimum possible value for the parameter
<i>ma</i>	maximum possible value for the parameter
<i>r</i>	random number generator

Definition at line 130 of file [model.h](#).

```
00130 : min(mi), max(ma), value(mi+(ma-mi)*r->Rndm()), step((ma-mi)*ss) {}
```

### 7.12.3 Member Function Documentation

7.12.3.1 `double freeparameter::dist ( double x ) const` `[inline]`

probability distribution, to be used by the Markov Chain Monte Carlo simulation

**Returns**

$$\left( \frac{x - \text{::value}}{\text{::step}} \right)^2$$

Definition at line 146 of file [model.h](#).

```
00146      {
00147      return std::pow( (x-value) / step, 2);
00148      }
```

**7.12.3.2 bool freeparameter::isvalid ( ) const [inline]**

checks if the value of the parameter is between ::min and ::max

Definition at line 140 of file [model.h](#).

```
00140      {
00141      return min<=value && value<=max;
00142      }
```

**7.12.3.3 void freeparameter::next ( TRandom3 \* r, double f = 1 ) [inline]**

changes randomly the ::value of the parameter, the standard deviation is ::step

**Parameters**

<i>r</i>	random number generator
----------	-------------------------

Definition at line 135 of file [model.h](#).

```
00135      {
00136      //double x=r->Gaus()*step;
00137      value+=r->Gaus()*step*f;
00138      }
```

**7.12.4 Member Data Documentation****7.12.4.1 double freeparameter::max**

maximum possible value for the parameter

Definition at line 152 of file [model.h](#).

**7.12.4.2 double freeparameter::min**

minimum possible value for the parameter

Definition at line 150 of file [model.h](#).



## 7.12.4.3 double freeparameter::step

standard deviation of the random changes of ::value in next(TRandom3 \*)

Definition at line 156 of file [model.h](#).

## 7.12.4.4 double freeparameter::value

value of the parameter

Definition at line 154 of file [model.h](#).

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

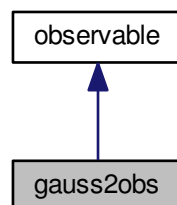
- [model.h](#)

## 7.13 gauss2obs Class Reference

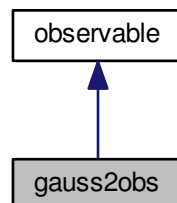
the same as gaussobs but with a different initializer, such that the uncertainty sigma is absolute

```
#include <model.h>
```

Inheritance diagram for gauss2obs:



Collaboration diagram for gauss2obs:



## Public Member Functions

- [gauss2obs](#) ([measure](#) *v*)
- [gauss2obs](#) (double *mean*, double *sigma*)
- [~gauss2obs](#) ()
- double [loglikelihood](#) (double *hipothesis*) const
- double [error](#) (double *hipothesis*) const
- double [expected](#) () const

## Public Attributes

- const double [m](#)
- const double [s](#)

### 7.13.1 Detailed Description

the same as gaussobs but with a different initializer, such that the uncertainty sigma is absolute

Definition at line 100 of file [model.h](#).

### 7.13.2 Constructor & Destructor Documentation

#### 7.13.2.1 `gauss2obs::gauss2obs ( measure v )` `[inline]`

##### Parameters

<i>mean</i>	mean value of the measure
<i>sigma</i>	standard deviation of the measure

Definition at line 106 of file [model.h](#).

```
00106 : m(v.value), s(v.error) {}
```

#### 7.13.2.2 `gauss2obs::gauss2obs ( double mean, double sigma )` `[inline]`

Definition at line 107 of file [model.h](#).

```
00107 : m(mean), s(sigma) {}
```

#### 7.13.2.3 `gauss2obs::~~gauss2obs ( )` `[inline]`

Definition at line 108 of file [model.h](#).

```
00108 {}
```

### 7.13.3 Member Function Documentation

#### 7.13.3.1 double gauss2obs::error ( double *hipothesis* ) const [inline],[virtual]

Implements [observable](#).

Definition at line 113 of file [model.h](#).

```
00113         {
00114             double diff=(hipothesis-m)/s;
00115             return diff;
00116         }
```

#### 7.13.3.2 double gauss2obs::expected ( ) const [inline]

Definition at line 118 of file [model.h](#).

```
00118 {return m;}
```

#### 7.13.3.3 double gauss2obs::loglikelihood ( double *hipothesis* ) const [inline],[virtual]

##### Parameters

<i>hipothesis</i>	the theoretical hypothesis
-------------------	----------------------------

##### Returns

the logarithm of the probability of measuring what was measured, assuming that the hypothesis is true

Implements [observable](#).

Definition at line 109 of file [model.h](#).

```
00109         {
00110             double diff=(m-hipothesis)/s;
00111             return diff*diff/2;
00112         }
```

### 7.13.4 Member Data Documentation

#### 7.13.4.1 const double gauss2obs::m

Definition at line 120 of file [model.h](#).

#### 7.13.4.2 const double gauss2obs::s

Definition at line 120 of file [model.h](#).

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

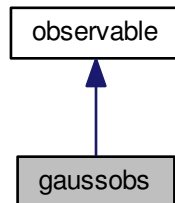
- [model.h](#)

## 7.14 gaussobs Class Reference

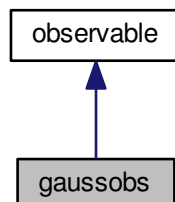
An experimental measure of a parameter which is a mean value and a standard deviation.

```
#include <model.h>
```

Inheritance diagram for gaussobs:



Collaboration diagram for gaussobs:



### Public Member Functions

- [gaussobs](#) ([measure](#) v)
- [gaussobs](#) (double mean, double sigma)
- [~gaussobs](#) ()
- double [loglikelihood](#) (double hypothesis) const
- double [error](#) (double hypothesis) const

### Public Attributes

- const double [m](#)
- const double [s](#)

### 7.14.1 Detailed Description

An experimental measure of a parameter which is a mean value and a standard deviation.

Definition at line 78 of file [model.h](#).

### 7.14.2 Constructor & Destructor Documentation

#### 7.14.2.1 gaussobs::gaussobs ( *measure v* ) [inline]

Parameters

<i>mean</i>	mean value of the measure
<i>sigma</i>	standard deviation of the measure

Definition at line 84 of file [model.h](#).

```
00084 : m(v.value), s(v.error) {}
```

#### 7.14.2.2 gaussobs::gaussobs ( *double mean, double sigma* ) [inline]

Definition at line 85 of file [model.h](#).

```
00085 : m(mean), s(mean*sigma) {}
```

#### 7.14.2.3 gaussobs::~~gaussobs ( ) [inline]

Definition at line 86 of file [model.h](#).

```
00086 {}
```

### 7.14.3 Member Function Documentation

#### 7.14.3.1 double gaussobs::error ( *double hypothesis* ) const [inline],[virtual]

Implements [observable](#).

Definition at line 91 of file [model.h](#).

```
00091         {
00092             double diff=(hypothesis-m)/s;
00093             return diff;
00094         }
```

#### 7.14.3.2 double gaussobs::loglikelihood ( *double hypothesis* ) const [inline],[virtual]

## Parameters

<i>hipothesis</i>	the theoretical hypothesis
-------------------	----------------------------

## Returns

the logarithm of the probability of measuring what was measured, assuming that the hypothesis is true

Implements [observable](#).

Definition at line 87 of file [model.h](#).

```

00087                                     {
00088         double diff=(m-hipothesis)/s;
00089         return diff*diff/2;
00090     }
```

## 7.14.4 Member Data Documentation

### 7.14.4.1 const double gaussobs::m

Definition at line 95 of file [model.h](#).

### 7.14.4.2 const double gaussobs::s

Definition at line 95 of file [model.h](#).

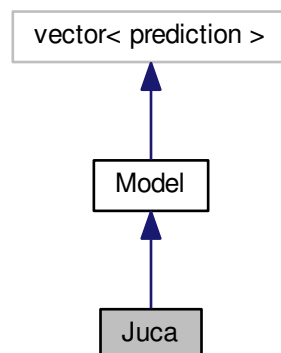
The documentation for this class was generated from the following file:

- [model.h](#)

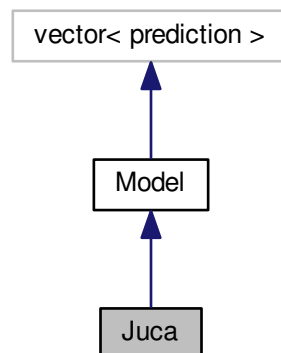
## 7.15 Juca Class Reference

```
#include <Juca.h>
```

Inheritance diagram for Juca:



Collaboration diagram for Juca:



## Public Member Functions

- [Juca](#) ()
- [~Juca](#) ()
- void [add](#) (const char \*s, ex pred, [observable](#) \*ob)
- [parameters generateparameters](#) () const
- 1st [getlist](#) (const [parameters](#) &p) const

## Public Attributes

- const possymbol [Mu](#)
- const possymbol [Md](#)
- const possymbol [Mc](#)
- const possymbol [Ms](#)
- const possymbol [Mt](#)
- const possymbol [Mb](#)
- const possymbol [lambda](#)
- const possymbol [A](#)
- const realsymbol [rho](#)
- const realsymbol [eta](#)
- 1st [replacements](#)

### 7.15.1 Detailed Description

Definition at line 15 of file [Juca.h](#).

## 7.15.2 Constructor & Destructor Documentation

### 7.15.2.1 Juca::Juca ( ) [inline]

Definition at line 18 of file [Juca.h](#).

```
00018      :Mu("Mu"), Md("Md"), Mc("Mc"), Ms("Ms"), Mt("Mt"), Mb("Mb"),
      lambda("lambda"), A("A"), rho("rho"), eta("eta"){
00019
00020      add("",lambda,new gauss2obs(0.22535,0.00065));
00021      add("",A,new gauss2obs(0.811, 0.017));
00022      add("",rho,new gauss2obs(0.131, 0.02));
00023      add("",eta,new gauss2obs(0.345, 0.014));
00024      add("",Mu,new gauss2obs(1.27e-3, 0.46e-3));
00025      add("",Md,new gauss2obs(2.9e-3, 1.22e-3));
00026      add("",Ms,new gauss2obs(55e-3, 16e-3));
00027      add("",Mc,new gauss2obs(0.619, 0.084));
00028      add("",Mb,new gauss2obs(2.89, 0.09));
00029      add("",Mt,new gauss2obs(171.7, 3.0));
00030 }
```

### 7.15.2.2 Juca::~~Juca ( ) [inline]

Definition at line 32 of file [Juca.h](#).

```
00032 {}
```

## 7.15.3 Member Function Documentation

### 7.15.3.1 void Juca::add ( const char \* s, ex pred, observable \* ob ) [inline]

Definition at line 36 of file [Juca.h](#).

```
00036
00037      ex p=collect_common_factors(expand(pred.subs(replacements)));
00038      push_back(prediction(ob,p));
00039 }
```

### 7.15.3.2 parameters Juca::generateparameters ( ) const [inline]

Definition at line 41 of file [Juca.h](#).

```
00041
00042      parameters p;
00043
00044      //buu
00045      p.push_back(freeparameter(-5,0,r));
00046      //auu
00047      p.push_back(freeparameter(-5,0,r));
00048      //add
00049      p.push_back(freeparameter(-5,0,r));
00050      //bdd
00051      p.push_back(freeparameter(-5,0,r));
00052      //eu
00053      p.push_back(freeparameter(-2,2,r));
00054      //gu
00055      p.push_back(freeparameter(-2,2,r));
00056      //ed
00057      p.push_back(freeparameter(-2,2,r));
```



```

00058         //gd
00059         p.push_back(freeparameter(-2,2,r));
00060         //nu
00061         p.push_back(freeparameter(-2,3,r));
00062         //nd
00063         p.push_back(freeparameter(-2,3,r));
00064         //cuu
00065         //p.push_back(freeparameter(-5,0,r));
00066         //cdd
00067         p.push_back(freeparameter(-5,0,r));
00068         //hu
00069         //p.push_back(freeparameter(-2,2,r));
00070         //hd
00071         p.push_back(freeparameter(-2,2,r));
00072
00073         return p;
00074     }

```

### 7.15.3.3 Ist Juca::getlist ( const parameters & p ) const [inline],[virtual]

Implements [Model](#).

Definition at line 77 of file [Juca.h](#).

Referenced by [main\(\)](#).

```

00077         {
00078
00079
00080         double buu = pow(10.0,p[0].value), auu = pow(10.0,p[1].value);
00081         double add = pow(10.0,p[2].value), bdd = pow(10.0,p[3].value);
00082         double eu = p[4].value, gu = p[5].value;
00083         double ed = p[6].value, gd = p[7].value;
00084         //double cuu = pow(10.0,p[10].value);
00085         double cdd = pow(10.0,p[10].value);
00086         //double hu = p[12].value;
00087         double hd= p[11].value;
00088
00089         complex<double> bu = buu*exp(complex<double>(0,gu*M_PI_2)), au = auu*exp(complex<double>(0,eu*M_PI_2));
00090         complex<double> ad = add*exp(complex<double>(0,ed*M_PI_2)), bd = bdd*exp(complex<double>(0,gd*M_PI_2));
00091         //complex<double> cu = cuu*exp(complex<double>(0,hu*M_PI_2));
00092         complex<double> cu = bu;
00093         complex<double> cd = cdd*exp(complex<double>(0,hd*M_PI_2));
00094         //complex<double> cd = bd;
00095
00096         //Matrix3cd X = Matrix3cd::Random(3,3);
00097         Matrix3cd mu,md;
00098         mu<<1,1.0,1.0+au+bu,1.0,1.0,1.0+bu,1.0+bu+au,1.0+bu,1.0+cu;
00099         md<<1,1.0,1.0+ad+bd,1.0,1.0,1.0+bd,1.0+bd+ad,1.0+bd,1.0+cd;
00100         mu=mu*pow(10.0,p[8].value);
00101         md=md*pow(10.0,p[9].value);
00102
00103         Matrix3cd Hu = mu*mu.adjoint(), Hd = md*md.adjoint();
00104         SelfAdjointEigenSolver<Matrix3cd> usolver(Hu), dsolver(Hd);
00105         Vector3d Du=usolver.eigenvalues(); //asDiagonal();
00106         Vector3d Dd=dsolver.eigenvalues(); //asDiagonal();
00107         Matrix3cd VLd=dsolver.eigenvectors().adjoint();
00108         Matrix3cd VLu=usolver.eigenvectors().adjoint();
00109         Matrix3cd Vckm=VLu*VLd.adjoint();
00110         double lambda0=sqrt(norm(Vckm(0,1))/(norm(Vckm(0,0))+norm(Vckm(0,1))));
00111         double A0=abs(Vckm(1,2))/abs(Vckm(0,1))/lambda0;
00112         complex<double> rhoeta=-Vckm(0,0)*conj(Vckm(0,2))/Vckm(1,0)/conj(Vckm(1,2));
00113         double rho0=real(rhoeta), eta0=imag(rhoeta);
00114
00115         return lst(lambda==lambda0,A==A0,rho==rho0,eta==eta0,Mu==sqrt(abs(Du[0])),
00116                 Md==sqrt(abs(Dd[0])),Mc==sqrt(abs(Du[1])),Ms==sqrt(abs(Dd[1])),Mt==sqrt(abs(Du[2])),
00117                 Mb==sqrt(abs(Dd[2])));
00116     }

```

Here is the caller graph for this function:



## 7.15.4 Member Data Documentation

### 7.15.4.1 `const possymbol Juca::A`

Definition at line 118 of file [Juca.h](#).

### 7.15.4.2 `const realsymbol Juca::eta`

Definition at line 119 of file [Juca.h](#).

### 7.15.4.3 `const possymbol Juca::lambda`

Definition at line 118 of file [Juca.h](#).

### 7.15.4.4 `const possymbol Juca::Mb`

Definition at line 118 of file [Juca.h](#).

### 7.15.4.5 `const possymbol Juca::Mc`

Definition at line 118 of file [Juca.h](#).

### 7.15.4.6 `const possymbol Juca::Md`

Definition at line 118 of file [Juca.h](#).

### 7.15.4.7 `const possymbol Juca::Ms`

Definition at line 118 of file [Juca.h](#).

#### 7.15.4.8 const possymbol Juca::Mt

Definition at line 118 of file [Juca.h](#).

#### 7.15.4.9 const possymbol Juca::Mu

Definition at line 118 of file [Juca.h](#).

#### 7.15.4.10 Ist Juca::replacements

Definition at line 121 of file [Juca.h](#).

#### 7.15.4.11 const realsymbol Juca::rho

Definition at line 119 of file [Juca.h](#).

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

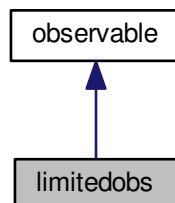
- [Juca.h](#)

## 7.16 limitedobs Class Reference

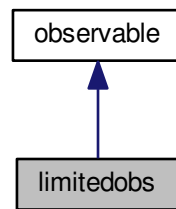
An experimental measure which is an upper limit on a parameter with a given Confidence Level.

```
#include <model.h>
```

Inheritance diagram for limitedobs:



Collaboration diagram for limitedobs:



## Public Member Functions

- `limitedobs` (double limit, double cl=0.9, double m=0)
- `~limitedobs` ()
- double `loglikelihood` (double hypothesis) const
- double `error` (double hypothesis) const

## Public Attributes

- double `s`
- double `min`
- double `lim`

### 7.16.1 Detailed Description

An experimental measure which is an upper limit on a parameter with a given Confidence Level.

Definition at line 52 of file [model.h](#).

### 7.16.2 Constructor & Destructor Documentation

7.16.2.1 `limitedobs::limitedobs ( double limit, double cl = 0.9, double m = 0 )` [inline]

#### Parameters

<i>limit</i>	upper limit on the parameter
<i>m</i>	minimum possible value of the parameter
<i>p</i>	1-Confidence Level

Definition at line 60 of file [model.h](#).

```
00060
```

```
: s (fabs (limit-m) / (1.282+sqrt (M_PI_2))),
```

```

        min(m), lim(limit) {
00061         if(c1==0.95) s*=(1.282+sqrt(M_PI_2))/(1.645+sqrt(M_PI_2));
00062         }

```

### 7.16.2.2 limitedobs::~limitedobs ( ) [inline]

Definition at line 63 of file [model.h](#).

```
00063 {}
```

## 7.16.3 Member Function Documentation

### 7.16.3.1 double limitedobs::error ( double *hypothesis* ) const [inline],[virtual]

Implements [observable](#).

Definition at line 69 of file [model.h](#).

```

00069         {
00070         double diff=(hypothesis-min)/s;
00071         return diff;
00072         }

```

### 7.16.3.2 double limitedobs::loglikelihood ( double *hypothesis* ) const [inline],[virtual]

#### Parameters

<i>hypothesis</i>	the theoretical hypothesis
-------------------	----------------------------

#### Returns

the logarithm of the probability of measuring what was measured, assuming that the hypothesis is true

Implements [observable](#).

Definition at line 64 of file [model.h](#).

```

00064         {
00065         double diff=(hypothesis-min-sqrt(M_PI_2)*s)/s;
00066         if(diff<0) diff=0;
00067         return diff*diff/2;
00068         }

```

## 7.16.4 Member Data Documentation

### 7.16.4.1 double limitedobs::lim

Definition at line 74 of file [model.h](#).

#### 7.16.4.2 double limitedobs::min

Definition at line 74 of file [model.h](#).

#### 7.16.4.3 double limitedobs::s

Definition at line 74 of file [model.h](#).

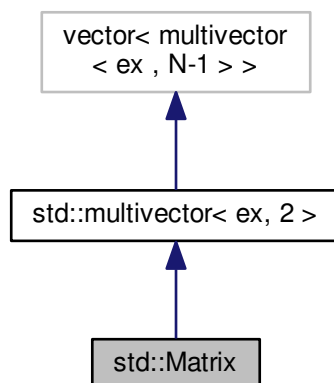
The documentation for this class was generated from the following file:

- [model.h](#)

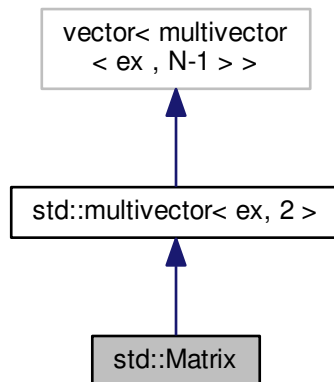
## 7.17 std::Matrix Class Reference

```
#include <multivector.h>
```

Inheritance diagram for std::Matrix:



Collaboration diagram for std::Matrix:



## Public Member Functions

- [Matrix](#) ()
- [Matrix](#) (const [Matrix](#) &m)
- [Matrix](#) (const char \*m[3][3])  
*constructs a symbolic matrix with the symbols names given by the argument*
- [Matrix](#) (const char \*name, const char \*\*index1, const char \*\*index2)  
*constructs a symbolic matrix with the symbols names given by the arguments*
- [Matrix](#) (ex m1, ex m2, ex m3)  
*constructs a diagonal matrix*
- [Matrix](#) (ex m1)  
*constructs a diagonal matrix with all diagonal elements equal*
- [Matrix](#) (ex t12, ex t13, ex t23, ex d13)  
*constructs a unitary matrix in the standard form*
- ex [cs](#) (ex t12)  
*used in the unitary constructor*
- ex [sn](#) (ex t12)  
*used in the unitary constructor*
- [Matrix conjugate](#) () const  
*computes the hermitian conjugate of the matrix*

## Additional Inherited Members

### 7.17.1 Detailed Description

Definition at line 67 of file [multivector.h](#).

## 7.17.2 Constructor & Destructor Documentation

### 7.17.2.1 `std::Matrix::Matrix ( )` [inline]

Definition at line 70 of file [multivector.h](#).

```
00070 : multivector< ex,2>(0,3,3) {}
```

### 7.17.2.2 `std::Matrix::Matrix ( const Matrix & m )` [inline]

Definition at line 72 of file [multivector.h](#).

```
00072 : multivector< ex,2>(m) {}
```

### 7.17.2.3 `std::Matrix::Matrix ( const char * m[3][3] )` [inline]

constructs a symbolic matrix with the symbols names given by the argument

Definition at line 74 of file [multivector.h](#).

```
00074 : multivector< ex,2>(0,3,3){
00075     for(uint i=0;i<3;i++){
00076         for(uint j=0;j<3;j++) at(i)[j]=symbol(m[i][j]);
00077     }
```

### 7.17.2.4 `std::Matrix::Matrix ( const char * name, const char ** index1, const char ** index2 )` [inline]

constructs a symbolic matrix with the symbols names given by the arguments

Definition at line 79 of file [multivector.h](#).

```
00079 : multivector< ex,2>(0,3,3){
00080     for(uint i=0;i<3;i++){
00081         for(uint j=0;j<3;j++){
00082             string res=string(name)+"_"+string(index1[i])+" "+string(index2[j])+" ";
00083             //cout<<res<<endl;
00084             at(i)[j]=symbol(res.c_str());
00085         }
00086     }
```

### 7.17.2.5 `std::Matrix::Matrix ( ex m1, ex m2, ex m3 )` [inline]

constructs a diagonal matrix

Definition at line 88 of file [multivector.h](#).

```
00088 : multivector< ex,2>(0,3,3) {
00089     at(0)[0]=m1;
00090     at(1)[1]=m2;
00091     at(2)[2]=m3;
00092 }
```



**7.17.2.6** std::Matrix::Matrix ( ex m1 ) [inline]

constructs a diagonal matrix with all diagonal elements equal

Definition at line 95 of file [multivector.h](#).

```
00095         : multivector< ex,2>(0,3,3){
00096     Matrix();
00097     at(0)[0]=m1;
00098     at(1)[1]=m1;
00099     at(2)[2]=m1;
00100 }
```

**7.17.2.7** std::Matrix::Matrix ( ex t12, ex t13, ex t23, ex d13 ) [inline]

constructs a unitary matrix in the standard form

Definition at line 102 of file [multivector.h](#).

```
00102         : multivector< ex,2>(0,3,3) {
00103     Matrix();
00104     ex c12=cos(t12), c13=cos(t13), c23=cos(t23);
00105     ex s12=sin(t12), s13=sin(t13), s23=sin(t23);
00106     ex e13=exp(I*d13);
00107     ex e13t=ex(1)/e13;
00108
00109     ex aux[3][3]={
00110         {c12*c13,s12*c13,s13*e13t},
00111         {-s12*c23-c12*s23*s13*e13,c12*c23-s12*s23*s13*e13,s23*c13},
00112         {s12*s23-c12*c23*s13*e13,-c12*s23-s12*c23*s13*e13,c13*c23}
00113     };
00114     for(uint i=0;i<3; i++) at(i).assign(aux[i],aux[i]+3);
00115 }
```

**7.17.3** Member Function Documentation**7.17.3.1** Matrix std::Matrix::conjugate ( ) const [inline]

computes the hermitian conjugate of the matrix

Definition at line 125 of file [multivector.h](#).

References [conjugate\(\)](#).

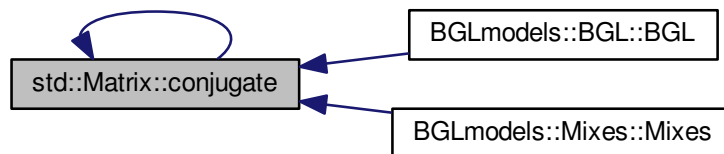
Referenced by [BGLmodels::BGL::BGL\(\)](#), [conjugate\(\)](#), and [BGLmodels::Mixes::Mixes\(\)](#).

```
00125     {
00126     Matrix res;
00127     for(uint i=0;i<3;i++)
00128         for(uint j=0;j<3;j++)
00129             res[i][j]=at(j)[i].conjugate();
00130
00131     return res;
00132 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 7.17.3.2 `ex std::Matrix::cs ( ex t12 ) [inline]`

used in the unitary constructor

Definition at line 117 of file [multivector.h](#).

```

00117     {
00118         return (exp (I*t12)+1/exp (I*t12)) /2;
00119     }
  
```

#### 7.17.3.3 `ex std::Matrix::sn ( ex t12 ) [inline]`

used in the unitary constructor

Definition at line 121 of file [multivector.h](#).

```

00121     {
00122         return -I*(exp (I*t12)-1/exp (I*t12)) /2;
00123     }
  
```

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

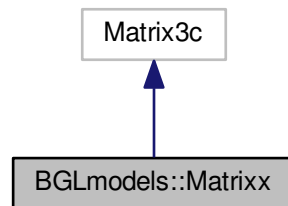
- [multivector.h](#)

## 7.18 BGLmodels::Matrixx Class Reference

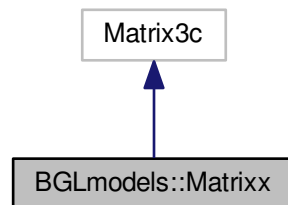
a class to represent the mixing matrices VCKM and VPMNS

```
#include <Formulas.h>
```

Inheritance diagram for BGLmodels::Matrixx:



Collaboration diagram for BGLmodels::Matrixx:



### Public Member Functions

- [Matrixx](#) ()
- [Matrixx](#) (const [Matrix3c](#) &a)
- [Matrixx](#) (double c12, double c13, double c23, double s12, double s13, double s23, [CD](#) e13, [CD](#) e13t)  
*constructs a unitary [Matrixx](#) in the standard form*
- [Matrixx](#) (double t12, double t13, double t23, double d13)
- const [Matrixx conjugate](#) () const  
*computes the hermitian conjugate of the [Matrixx](#)*

### 7.18.1 Detailed Description

a class to represent the mixing matrices VCKM and VPMNS

Definition at line 72 of file [Formulas.h](#).

## 7.18.2 Constructor & Destructor Documentation

### 7.18.2.1 BGLmodels::Matrixx::Matrixx ( ) [inline]

Definition at line 74 of file [Formulas.h](#).

```
00074 : Matrix3c() {}
```

### 7.18.2.2 BGLmodels::Matrixx::Matrixx ( const Matrix3c & a ) [inline]

Definition at line 75 of file [Formulas.h](#).

```
00075 : Matrix3c(a) {}
```

### 7.18.2.3 BGLmodels::Matrixx::Matrixx ( double c12, double c13, double c23, double s12, double s13, double s23, CD e13, CD e13t ) [inline]

constructs a unitary [Matrixx](#) in the standard form

Definition at line 79 of file [Formulas.h](#).

```
00079 :
00080     Matrix3c({
00081         Vector3c({c12*c13,s12*c13,s13*e13t}),
00082         Vector3c({-s12*c23-c12*s23*s13*e13,c12*c23-s12*s23*s13*e13,s23*c13}),
00083         Vector3c({s12*s23-c12*c23*s13*e13,-c12*s23-s12*c23*s13*e13,c13*c23})
00084     }) {}
```

### 7.18.2.4 BGLmodels::Matrixx::Matrixx ( double t12, double t13, double t23, double d13 ) [inline]

Definition at line 86 of file [Formulas.h](#).

```
00086 :
00087     Matrixx(std::cos(t12),std::cos(t13),std::cos(t23),std::sin(t12),std::sin(t13),std::sin(t23),
std::exp(CD(0,d13)),std::exp(CD(0,-d13))) {}
```

## 7.18.3 Member Function Documentation

### 7.18.3.1 const Matrixx BGLmodels::Matrixx::conjugate ( ) const [inline]

computes the hermitian conjugate of the [Matrixx](#)

Definition at line 89 of file [Formulas.h](#).

```
00089 {
00090     Matrixx res;
00091     for(uint i=0;i<3;i++)
00092         for(uint j=0;j<3;j++)
00093             res[i][j]=std::conj((*this)[j][i]);
00094     return res;
00095 }
```

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

- [Formulas.h](#)

## 7.19 measure Class Reference

A class containing the value and uncertainty of an experimental measure.

```
#include <model.h>
```

### Public Member Functions

- [measure](#) (double *v*=0, double *e*=0)
- [measure operator\\*](#) ([measure](#) *m2*) const
- [measure operator/](#) ([measure](#) *m2*) const

### Public Attributes

- double [value](#)
- double [error](#)

#### 7.19.1 Detailed Description

A class containing the value and uncertainty of an experimental measure.

Definition at line 20 of file [model.h](#).

#### 7.19.2 Constructor & Destructor Documentation

**7.19.2.1** `measure::measure ( double v = 0, double e = 0 )` `[inline]`

Definition at line 22 of file [model.h](#).

```
00022 :value(v),error(e){}
```

#### 7.19.3 Member Function Documentation

**7.19.3.1** `measure measure::operator* ( measure m2 ) const` `[inline]`

Definition at line 23 of file [model.h](#).

References [error](#), and [value](#).

```
00023                                     {
00024         const measure & m1=*this;
00025         return measure(m1.value*m2.value,sqrt(std::pow(m1.
value*m2.error,2)+std::pow(m2.value*m1.error,2)));
00026     }
```

### 7.19.3.2 `measure` `measure::operator/ ( measure m2 ) const` `[inline]`

Definition at line 27 of file [model.h](#).

References [error](#), and [value](#).

```
00027         {
00028             const measure & m1=*this;
00029             return measure(m1.value/m2.value, sqrt(std::pow(m1.
value/std::pow(m2.value, 2)*m2.error, 2)+std::pow(m1.error/m2.
value, 2)));
00030         }
```

## 7.19.4 Member Data Documentation

### 7.19.4.1 `double` `measure::error`

Definition at line 32 of file [model.h](#).

Referenced by [operator\\*\(\)](#), and [operator/\(\)](#).

### 7.19.4.2 `double` `measure::value`

Definition at line 31 of file [model.h](#).

Referenced by [operator\\*\(\)](#), and [operator/\(\)](#).

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

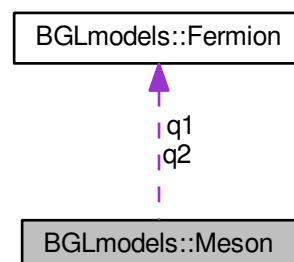
- [model.h](#)

## 7.20 BGLmodels::Meson Class Reference

a meson properties

```
#include <Formulas.h>
```

Collaboration diagram for BGLmodels::Meson:



## Public Member Functions

- [Meson](#) (const [Fermion](#) &qq1, const [Fermion](#) &qq2, ex m, ex d)

## Public Attributes

- [Fermion](#) q1
- [Fermion](#) q2
- ex [mass](#)
- ex [decay\\_factor](#)

### 7.20.1 Detailed Description

a meson properties

Definition at line 47 of file [Formulas.h](#).

### 7.20.2 Constructor & Destructor Documentation

7.20.2.1 [BGLmodels::Meson::Meson](#) ( const [Fermion](#) & qq1, const [Fermion](#) & qq2, ex m, ex d ) [inline]

Definition at line 50 of file [Formulas.h](#).

```
00050 : q1(qq1), q2(qq2), mass(m), decay_factor(d){}
```

### 7.20.3 Member Data Documentation

7.20.3.1 ex [BGLmodels::Meson::decay\\_factor](#)

Definition at line 54 of file [Formulas.h](#).

Referenced by [BGLmodels::BGL::fermiontomeson\(\)](#), [BGLmodels::BGL::fermiontomesontest\(\)](#), [BGLmodels::BGL::mesondw\(\)](#), and [BGLmodels::BGL::mesondwtest\(\)](#).

7.20.3.2 ex [BGLmodels::Meson::mass](#)

Definition at line 53 of file [Formulas.h](#).

Referenced by [BGLmodels::BGL::fermiontomeson\(\)](#), [BGLmodels::BGL::fermiontomesontest\(\)](#), [BGLmodels::BGL::mesondw\(\)](#), and [BGLmodels::BGL::mesondwtest\(\)](#).

7.20.3.3 [Fermion](#) [BGLmodels::Meson::q1](#)

Definition at line 52 of file [Formulas.h](#).

Referenced by [BGLmodels::calcuBmumu::calcuBmumu\(\)](#), [BGLmodels::BGL::fermiontomeson\(\)](#), [BGLmodels::BGL::fermiontomesontest\(\)](#), [BGLmodels::BGL::mesondw\(\)](#), and [BGLmodels::BGL::mesondwtest\(\)](#).

### 7.20.3.4 Fermion BGLmodels::Meson::q2

Definition at line 52 of file [Formulas.h](#).

Referenced by [BGLmodels::calcuBmumu::calcuBmumu\(\)](#), [BGLmodels::BGL::fermiontomeson\(\)](#), [BGLmodels::BGL::fermiontomesontest\(\)](#), [BGLmodels::BGL::mesondw\(\)](#), and [BGLmodels::BGL::mesondwtest\(\)](#).

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

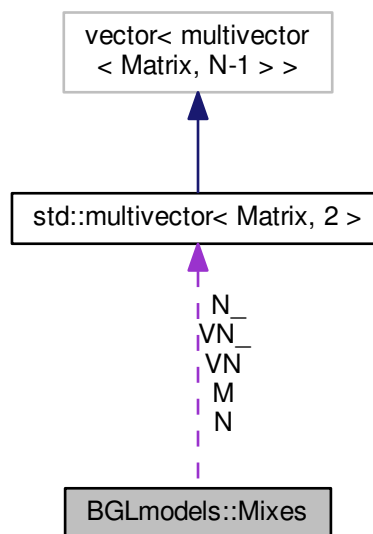
- [Formulas.h](#)

## 7.21 BGLmodels::Mixes Class Reference

definition of the couplings for the different [BGL](#) models

```
#include <Formulas.h>
```

Collaboration diagram for BGLmodels::Mixes:



### Public Member Functions

- [Mixes](#) (ex tanb0, ex cp0, int genL=2, int genQ=2, int lup=0, int qup=0, int mssm=0)
- ex [mass](#) (const [Fermion](#) &f) const
- double [massnum](#) (const [Fermion](#) &f) const
- void [appendtolst](#) (lst &reps) const



## Public Attributes

- [lst reps](#)
- [vector< Matrix > V](#)
- [multivector< Matrix, 2 > M](#)
- [multivector< Matrix, 2 > N](#)
- [multivector< Matrix, 2 > VN](#)
- [multivector< Matrix, 2 > N\\_](#)
- [multivector< Matrix, 2 > VN\\_](#)
- [lst replacements](#)
- [ex cp](#)
- [ex tanb](#)

## 7.21.1 Detailed Description

definition of the couplings for the different [BGL](#) models

Definition at line 105 of file [Formulas.h](#).

## 7.21.2 Constructor &amp; Destructor Documentation

**7.21.2.1** [BGLmodels::Mixes::Mixes](#) ( [ex tanb0](#), [ex cp0](#), [int genL](#) = 2, [int genQ](#) = 2, [int lup](#) = 0, [int qup](#) = 0, [int mssm](#) = 0 )  
[inline]

Definition at line 108 of file [Formulas.h](#).

References [std::Matrix::conjugate\(\)](#), [BGLmodels::iDown](#), [BGLmodels::iUp](#), [BGLmodels::tLepton](#), and [BGLmodels::tQuark](#).

```

00108                                     :M(
00109     Matrix(), 2, 2), N(Matrix(), 2, 2), VN(Matrix(), 2, 2), N_(Matrix(), 2, 2),
00110     VN_(Matrix(), 2, 2), cp(cp0)
00111     {
00112         tanb=tanb0;
00113         M[tLepton][iDown]=Matrix(possymbol("m_e"),possymbol("m_\mu"),possymbol("m_\tau"));
00114         M[tQuark][iUp]=Matrix(possymbol("m_u"),possymbol("m_c"),possymbol("m_t"));
00115         M[tQuark][iDown]=Matrix(possymbol("m_d"),possymbol("m_s"),possymbol("m_b"));
00116         const char * ln[3]={ "l", "2", "3" };
00117         const char * ll[3]={ "e", "\mu", "\tau" };
00118         const char * lu[3]={ "u", "c", "t" };
00119         const char * ld[3]={ "d", "s", "b" };
00120         V.push_back(Matrix("U", ln, ll));
00121         V.push_back(Matrix("V", lu, ld));
00122
00123         int up[2];
00124         up[0]=lup;
00125         up[1]=qup;
00126
00127         vector< Matrix > delta;
00128
00129         vector<int> gL(3,0);
00130         gL[genL]=1;
00131         //Leptons
00132         delta.push_back(Matrix(gL[0],gL[1],gL[2]));
00133         //Quarks
00134         vector<int> gQ(3,0);
00135         gQ[genQ]=1;
00136         delta.push_back(Matrix(gQ[0],gQ[1],gQ[2]));
00137
00138         for(uint i=0;i<2;i++){
00139             if(mssm){
00140                 //Nu

```

```

00141         N_[i][0]=0;
00142         //Nd
00143         N_[i][1]=0;
00144         //VNd
00145         VN_[i][1]=Matrix(tanb)*V[i];
00146         //Nu*V
00147         VN_[i][0]=Matrix(1/tanb)*V[i];
00148     }
00149     else if(up[i]){
00150         //Nu
00151         N_[i][0]=Matrix(tanb)+Matrix(-tanb-1/tanb)*
V[i]*delta[i]*V[i].conjugate();
00152         //Nd
00153         N_[i][1]=Matrix(tanb)+Matrix(-tanb-1/tanb)*delta[i];
00154         //VNd
00155         VN_[i][1]=Matrix(tanb)*V[i]+Matrix(-tanb-1/tanb)*
V[i]*delta[i];
00156         //Nu*V
00157         VN_[i][0]=Matrix(-1)*(Matrix(tanb)*V[i]+Matrix(-
tanb-1/tanb)*V[i]*delta[i]);
00158     }else{
00159         //Nu
00160         N_[i][0]=Matrix(tanb)+Matrix(-tanb-1/tanb)*delta[i];
00161         //Nd
00162         N_[i][1]=Matrix(tanb)+Matrix(-tanb-1/tanb)*V[i].conjugate()*delta[i]*
V[i];
00163         //VNd
00164         VN_[i][1]=Matrix(tanb)*V[i]+Matrix(-tanb-1/tanb)*delta[i]*V[i];
00165         //Nu*V
00166         VN_[i][0]=Matrix(-1)*(Matrix(tanb)*V[i]+Matrix(-
tanb-1/tanb)*delta[i]*V[i]);
00167     }
00168
00169         N[i][0]=N_[i][0]*M[i][0];
00170         N[i][1]=N_[i][1]*M[i][1];
00171         VN[i][1]=VN_[i][1]*M[i][1];
00172         VN[i][0]=M[i][0]*VN_[i][0];
00173     }
00174     appendtolst(replacements);
00175
00176
00177 }

```

Here is the call graph for this function:



## 7.21.3 Member Function Documentation

### 7.21.3.1 void BGLmodels::Mixes::appendtolst ( lst & reps ) const [inline]

Definition at line 181 of file [Formulas.h](#).

```

00181                                     {///vector<ex>& var_errors
00182         reps.append(M[0][1][0][0]==0.510998910e-3);
00183         reps.append(M[0][1][1][1]==105.6583715e-3);
00184         reps.append(M[0][1][2][2]==1776.82e-3);
00185
00186         reps.append(M[1][0][0][0]==2.4e-3);
00187         reps.append(M[1][0][1][1]==1.29);
00188         reps.append(M[1][0][2][2]==172.9);

```

```

00189
00190     reps.append(M[1][1][0][0]==5.3e-3);
00191     reps.append(M[1][1][1][1]==95e-3);
00192     reps.append(M[1][1][2][2]==4.2);
00193
00194     vector< Matrix > Vn;
00195     Vn.push_back(Matrix(33.6*M_PI/180,9.11*M_PI/180,40.4*M_PI/180,M_PI/4).conjugate());
00196     Vn.push_back(Matrix(13.04*M_PI/180,0.201*M_PI/180,2.38*M_PI/180,1.2));
00197     for(uint i=0; i<2;i++)
00198         for(uint j=0; j<3;j++)
00199             for(uint k=0; k<3;k++)
00200                 reps.append(V[i][j][k]==Vn[i][j][k]);
00201     }

```

### 7.21.3.2 ex BGLmodels::Mixes::mass ( const Fermion & f ) const [inline]

Definition at line 178 of file [Formulas.h](#).

References [BGLmodels::Fermion::flavour](#), [BGLmodels::Fermion::isospin](#), and [BGLmodels::Fermion::type](#).

```

00178 {return M[f.type][f.isospin][f.flavour][f.flavour];}

```

### 7.21.3.3 double BGLmodels::Mixes::massnum ( const Fermion & f ) const [inline]

Definition at line 179 of file [Formulas.h](#).

References [BGLmodels::Fermion::flavour](#), [BGLmodels::Fermion::isospin](#), and [BGLmodels::Fermion::type](#).

```

00179 {return ex_to<numeric>(M[f.type][f.isospin][f.flavour][f.flavour].subs(
    replacements)).to_double();}

```

## 7.21.4 Member Data Documentation

### 7.21.4.1 ex BGLmodels::Mixes::cp

Definition at line 213 of file [Formulas.h](#).

### 7.21.4.2 multivector<Matrix,2> BGLmodels::Mixes::M

Definition at line 205 of file [Formulas.h](#).

### 7.21.4.3 multivector<Matrix,2> BGLmodels::Mixes::N

Definition at line 206 of file [Formulas.h](#).

### 7.21.4.4 multivector<Matrix,2> BGLmodels::Mixes::N\_

Definition at line 209 of file [Formulas.h](#).

Referenced by [BGLmodels::calcubtosgamma2::calcubtosgamma2\(\)](#).

#### 7.21.4.5 `lst BGLmodels::Mixes::replacements`

Definition at line 212 of file [Formulas.h](#).

Referenced by [BGLmodels::calcubtosgamma2::calcubtosgamma2\(\)](#).

#### 7.21.4.6 `lst BGLmodels::Mixes::reps`

Definition at line 203 of file [Formulas.h](#).

#### 7.21.4.7 `ex BGLmodels::Mixes::tanb`

Definition at line 214 of file [Formulas.h](#).

Referenced by [BGLmodels::calcubtosgamma2::calcubtosgamma2\(\)](#).

#### 7.21.4.8 `vector< Matrix > BGLmodels::Mixes::V`

Definition at line 204 of file [Formulas.h](#).

#### 7.21.4.9 `multivector<Matrix,2> BGLmodels::Mixes::VN`

Definition at line 207 of file [Formulas.h](#).

#### 7.21.4.10 `multivector<Matrix,2> BGLmodels::Mixes::VN_`

Definition at line 210 of file [Formulas.h](#).

Referenced by [BGLmodels::calcubtosgamma2::calcubtosgamma2\(\)](#).

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

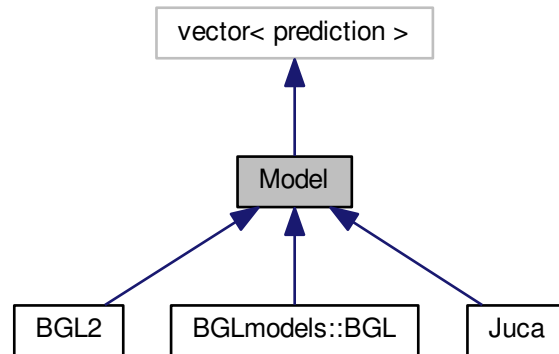
- [Formulas.h](#)

## 7.22 Model Class Reference

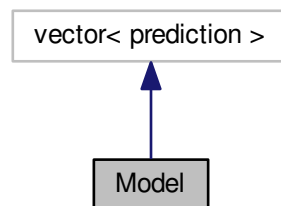
Abstract class for a model.

```
#include <model.h>
```

Inheritance diagram for Model:



Collaboration diagram for Model:



### Public Member Functions

- [Model](#) ()
- virtual [~Model](#) ()
- virtual [parameters getlist](#) (const [parameters](#) &p) const =0
- virtual [parameters generateparameters](#) (int max=0) const =0
- virtual int [veto](#) (const [parameters](#) &p, int max=0) const
- double [likelihood](#) (const [parameters](#) &p, bool check=1, int max=0) const  
*calculates the probability of getting all the experimental measures if the model describes the reality*
- double [loglike](#) (const [parameters](#) &p, bool check=1, int max=0) const

## Public Attributes

- `TRandom3 * r`

### 7.22.1 Detailed Description

Abstract class for a model.

Definition at line 361 of file [model.h](#).

### 7.22.2 Constructor & Destructor Documentation

#### 7.22.2.1 `Model::Model ( )` `[inline]`

Definition at line 364 of file [model.h](#).

```
00364 : r(new TRandom3(0)) {}
```

#### 7.22.2.2 `virtual Model::~~Model ( )` `[inline]`, `[virtual]`

Definition at line 365 of file [model.h](#).

```
00365 {delete r};
```

### 7.22.3 Member Function Documentation

#### 7.22.3.1 `virtual parameters Model::generateparameters ( int max = 0 ) const` `[pure virtual]`

Implemented in [BGLmodels::BGL](#), and [BGL2](#).

#### 7.22.3.2 `virtual parameters Model::getlist ( const parameters & p ) const` `[pure virtual]`

Implemented in [BGLmodels::BGL](#), [BGL2](#), and [Juca](#).

#### 7.22.3.3 `double Model::likelihood ( const parameters & p, bool check = 1, int max = 0 ) const` `[inline]`

calculates the probability of getting all the experimental measures if the model describes the reality

#### Parameters

<i>p</i>	vector with the values of the free parameters
----------	---

Definition at line 374 of file [model.h](#).

```
00374                                     {
00375         if(veto(p,max) && check) return 0;
00376         double total=loglike(p,0);
00377         if(total<-1000) return 0;
00378         return exp(total);
00379     }
```

#### 7.22.3.4 double Model::loglike ( const parameters & p, bool check = 1, int max = 0 ) const [inline]

Definition at line 381 of file [model.h](#).

Referenced by [main\(\)](#).

```
00381                                     {
00382         if(veto(p,max) && check) return -1000;
00383         parameters pp(getlist(p));
00384
00385         double total=0;
00386         int n=0;
00387         for(const_iterator i=begin();i!=end();i++) {
00388             try{
00389                 n++;
00390                 total+=i->loglikelihood(pp);
00391             }
00392             catch(exception e){
00393                 cout<<n<<e.what()<<endl;
00394                 exit(1);
00395             }
00396             //catch(...) {
00397                 //cout<<"DD "<<n<<endl;
00398                 //exit(1);
00399                 //}
00400         }
00401
00402         return -total;
00403     }
```

Here is the caller graph for this function:



#### 7.22.3.5 virtual int Model::veto ( const parameters & p, int max = 0 ) const [inline], [virtual]

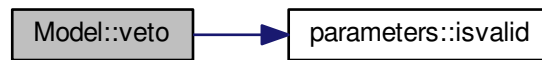
Reimplemented in [BGLmodels::BGL](#).

Definition at line 368 of file [model.h](#).

References [parameters::isvalid\(\)](#).

```
00368 {return !p.isvalid();}
```

Here is the call graph for this function:



## 7.22.4 Member Data Documentation

### 7.22.4.1 TRandom3\* Model::r

Definition at line 405 of file [model.h](#).

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

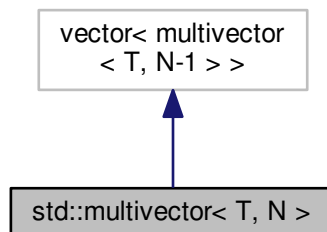
- [model.h](#)

## 7.23 std::multivector< T, N > Class Template Reference

A vector of vectors of vectors of... (N times) of class T objects.

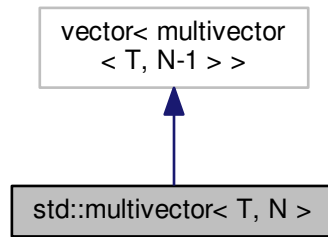
```
#include <multivector.h>
```

Inheritance diagram for std::multivector< T, N >:





Collaboration diagram for std::multivector< T, N >:



## Public Types

- typedef vector< [multivector](#)< T, N-1 > > [v](#)

## Public Member Functions

- [multivector](#) ()  
*Default constructor.*
- [multivector](#) (const [multivector](#) &m)  
*Copy constructor.*
- [multivector](#) (const T &value,...)  
*Recommended constructor.*
- [multivector](#) (const T &value, va\_list &listPointer)  
*Auxiliary constructor (recursive)*

### 7.23.1 Detailed Description

```
template<class T, int N>
class std::multivector< T, N >
```

A vector of vectors of vectors of... (N times) of class T objects.

Definition at line 8 of file [multivector.h](#).

### 7.23.2 Member Typedef Documentation

7.23.2.1 template<class T, int N> typedef vector< [multivector](#)< T,N-1> > [std::multivector](#)< T, N >::v

Definition at line 10 of file [multivector.h](#).

### 7.23.3 Constructor & Destructor Documentation

**7.23.3.1** `template<class T, int N> std::multivector< T, N >::multivector ( ) [inline]`

Default constructor.

Definition at line 13 of file [multivector.h](#).

```
00013 : v() {}
```

**7.23.3.2** `template<class T, int N> std::multivector< T, N >::multivector ( const multivector< T, N > & m ) [inline]`

Copy constructor.

Definition at line 15 of file [multivector.h](#).

```
00015 : v(m) {}
```

**7.23.3.3** `template<class T, int N> std::multivector< T, N >::multivector ( const T & value, ... ) [inline]`

Recommended constructor.

Example: `multivector<double, 2> m(1.5,4,6)`, `m` is a matrix of doubles with dimensions 4x6, with all doubles initialized to 1.5

#### Parameters

<i>value</i>	the value with which every objects are initialized
...	list with the number of dimensions of each vector

#### See also

[multivector\(const T&, va\\_list &\)](#)

Definition at line 22 of file [multivector.h](#).

```
00022             {
00023         va_list listPointer;
00024         va_start(listPointer, value);
00025         int n=va_arg(listPointer, int);
00026         v::insert(v::begin(), n, multivector<T, N-1>(value, listPointer));
00027         va_end(listPointer);
00028     }
```

**7.23.3.4** `template<class T, int N> std::multivector< T, N >::multivector ( const T & value, va_list & listPointer ) [inline]`

Auxiliary constructor (recursive)

See also

[multivector\(const T&, ...\)](#)  
[multivector<T,1>](#)

Definition at line 34 of file [multivector.h](#).

```
00035         {  
00036             int n=va_arg(listPointer,int);  
00037             v::insert(v::begin(),n,multivector<T,N-1>(value,listPointer));  
00038         }
```

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

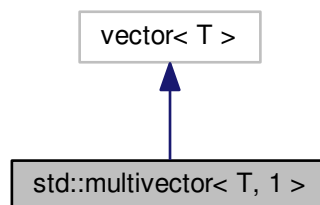
- [multivector.h](#)

## 7.24 `std::multivector< T, 1 >` Class Template Reference

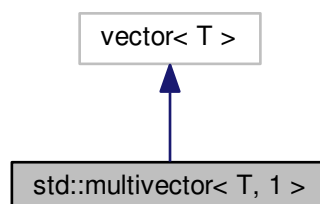
Specialization template class of [multivector<T,N>](#) for N=1.

```
#include <multivector.h>
```

Inheritance diagram for `std::multivector< T, 1 >`:



Collaboration diagram for `std::multivector< T, 1 >`:



## Public Types

- `typedef vector< T > v`

## Public Member Functions

- `multivector ()`  
*Default constructor.*
- `multivector (const multivector &m)`  
*Copy constructor.*
- `multivector (const T &value, int x)`  
*Recommended constructor.*
- `multivector (const T &value, va_list &listPointer)`  
*Auxiliary constructor.*

### 7.24.1 Detailed Description

```
template<class T>
class std::multivector< T, 1 >
```

Specialization template class of `multivector<T,N>` for N=1.

See also

`multivector<T,N>`

Definition at line 46 of file `multivector.h`.

### 7.24.2 Member Typedef Documentation

7.24.2.1 `template<class T> typedef vector< T > std::multivector< T, 1 >::v`

Definition at line 48 of file `multivector.h`.

### 7.24.3 Constructor & Destructor Documentation

7.24.3.1 `template<class T> std::multivector< T, 1 >::multivector ( ) [inline]`

Default constructor.

Definition at line 50 of file `multivector.h`.

```
00050 : v() {}
```

7.24.3.2 `template<class T> std::multivector< T, 1 >::multivector ( const multivector< T, 1 > &m ) [inline]`

Copy constructor.

Definition at line 52 of file `multivector.h`.

```
00052 : v(m) {}
```

7.24.3.3 `template<class T> std::multivector< T, 1 >::multivector ( const T &value, int x ) [inline]`

Recommended constructor.

## Parameters

<i>value</i>	the value with which every objects are initialized
<i>x</i>	number of dimensions of the vector

Definition at line 57 of file [multivector.h](#).

```
00057 : v(x,value){}
```

7.24.3.4 `template<class T > std::multivector< T, 1 >::multivector ( const T & value, va_list & listPointer )`  
[inline]

Auxiliary constructor.

It is the last constructor to be called in the recursive constructor [multivector<T,N>::multivector\(const T&,va\\_list &\)](#).

See also

[multivector<T,N>::multivector\(const T&,va\\_list &\)](#)

Definition at line 63 of file [multivector.h](#).

```
00063 :  
00064 v(va_arg(listPointer,int), value){}
```

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

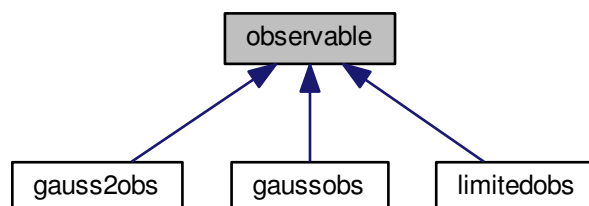
- [multivector.h](#)

## 7.25 observable Class Reference

A base class representing an experimental measure.

```
#include <model.h>
```

Inheritance diagram for observable:



## Public Member Functions

- [observable](#) ()
- virtual [~observable](#) ()
- virtual double [loglikelihood](#) (double *hipothesis*) const =0
- virtual double [error](#) (double *hipothesis*) const =0

## Public Attributes

- [uint copies](#)

### 7.25.1 Detailed Description

A base class representing an experimental measure.

Definition at line 35 of file [model.h](#).

### 7.25.2 Constructor & Destructor Documentation

#### 7.25.2.1 `observable::observable ( )` `[inline]`

Definition at line 37 of file [model.h](#).

```
00037 : copies (1) {}
```

#### 7.25.2.2 `virtual observable::~~observable ( )` `[inline]`, `[virtual]`

Definition at line 38 of file [model.h](#).

```
00038 {}
```

### 7.25.3 Member Function Documentation

#### 7.25.3.1 `virtual double observable::error ( double hipothesis ) const` `[pure virtual]`

Implemented in [gauss2obs](#), [gaussobs](#), and [limitedobs](#).

#### 7.25.3.2 `virtual double observable::loglikelihood ( double hipothesis ) const` `[pure virtual]`

#### Parameters

<i>hipothesis</i>	the theoretical hypothesis
-------------------	----------------------------

**Returns**

the logarithm of the probability of measuring what was measured, assuming that the hypothesis is true

Implemented in [gauss2obs](#), [gaussobs](#), and [limitedobs](#).

**7.25.4 Member Data Documentation****7.25.4.1** `uint observable::copies` [mutable]

Definition at line 48 of file [model.h](#).

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

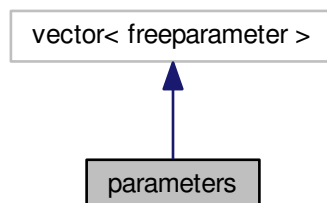
- [model.h](#)

**7.26 parameters Class Reference**

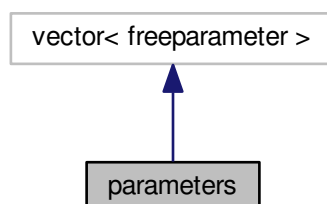
vector of parameters

```
#include <model.h>
```

Inheritance diagram for parameters:



Collaboration diagram for parameters:



## Public Member Functions

- void [next](#) (TRandom3 \*r, double f=1)  
*changes randomly the value of the parameters*
- bool [isvalid](#) () const  
*checks if all the values are between their minimums and maximums*
- double [dist](#) (const [parameters](#) &p) const  
*checks if this and another vector of parameters are within 1sigma of distance*
- void [setvalues](#) (const [parameters](#) &p)
- double [area](#) () const
- double [gausslikelihood](#) (const [parameters](#) &p2) const

## Public Attributes

- [lst](#) [p](#)
- vector< double > [values](#)

### 7.26.1 Detailed Description

vector of parameters

Definition at line 177 of file [model.h](#).

### 7.26.2 Member Function Documentation

#### 7.26.2.1 double [parameters::area](#) ( ) const [\[inline\]](#)

Definition at line 215 of file [model.h](#).

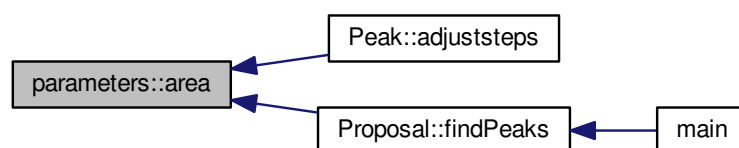
Referenced by [Peak::adjuststeps\(\)](#), and [Proposal::findPeaks\(\)](#).

```

00215     {
00216         float a=1;
00217         for(const_iterator i=begin();i!=end();i++){
00218             a*=i->step;
00219         }
00220         return a;
00221     }

```

Here is the caller graph for this function:





**7.26.2.2** `double parameters::dist ( const parameters & p ) const` `[inline]`

checks if this and another vector of parameters are within 1sigma of distance

Definition at line 200 of file [model.h](#).

```
00200                                     {
00201     double total=0;
00202     for(uint i=0;i<size();i++){
00203         total+=at(i).dist(p[i].value);
00204     }
00205     return sqrt(total/size());
00206 }
```

**7.26.2.3** `double parameters::gausslikelihood ( const parameters & p2 ) const` `[inline]`

Definition at line 223 of file [model.h](#).

```
00223                                     {
00224     double l=1;
00225     for(uint i=0;i<size();i++){
00226         l*=TMath::Gaus((p2[i].value-at(i).value)/at(i).step);
00227     }
00228     return l;
00229 }
```

**7.26.2.4** `bool parameters::isvalid ( ) const` `[inline]`

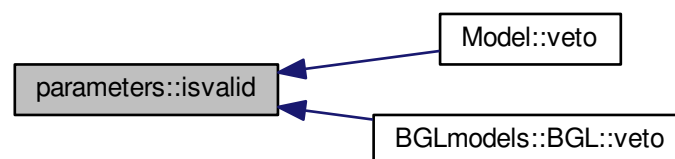
checks if all the values are between their minimums and maximums

Definition at line 193 of file [model.h](#).

Referenced by [Model::veto\(\)](#), and [BGLmodels::BGL::veto\(\)](#).

```
00193     {
00194     for(const_iterator i=begin();i!=end();i++){
00195         if(!i->isvalid()) return 0;
00196     }
00197     return 1;
00198 }
```

Here is the caller graph for this function:



#### 7.26.2.5 void parameters::next ( TRandom3 \* r, double f = 1 ) [inline]

changes randomly the value of the parameters

Definition at line 182 of file [model.h](#).

Referenced by [Peak::findPeak\(\)](#).

```

00182                                     {
00183     for(iterator i=begin(); i!=end(); i++) {
00184         i->next(r, f);
00185     }
00186
00187     //for(uint i=0; i<discrete.size(); i++) {
00188         //discrete[i].next(r);
00189     //}
00190 }
```

Here is the caller graph for this function:



#### 7.26.2.6 void parameters::setvalues ( const parameters & p ) [inline]

Definition at line 208 of file [model.h](#).

```

00208                                     {
00209
00210     for(uint i=0; i<size(); i++) {
00211         at(i).value=p[i].value;
00212     }
00213 }
```

### 7.26.3 Member Data Documentation

#### 7.26.3.1 Ist parameters::p

Definition at line 231 of file [model.h](#).

Referenced by [BGL2::epsK\(\)](#), [calcuex::error\(\)](#), [BGL2::getlist\(\)](#), [BGLmodels::BGL::getlist\(\)](#), [main\(\)](#), and [calcuex::operator\(\)](#).

#### 7.26.3.2 vector<double> parameters::values

Definition at line 232 of file [model.h](#).

Referenced by [BGL2::getlist\(\)](#), [BGLmodels::BGL::getlist\(\)](#), [BGLmodels::calcuBmumu::obsvalue\(\)](#), [calcuBmumu::operator\(\)](#), and [BGLmodels::calcuBmumu::operator\(\)](#).

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

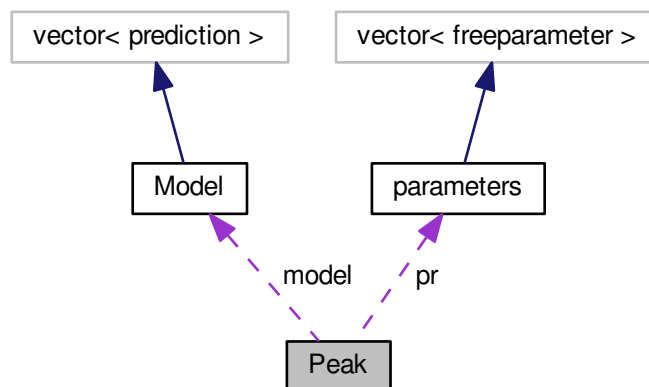
- [model.h](#)

## 7.27 Peak Class Reference

A class containing the parameters of a maximum of the likelihood function.

```
#include <MCMC.h>
```

Collaboration diagram for Peak:



### Public Member Functions

- [Peak](#) (const [Model](#) \*m, int maxx=0)
- void [findPeak](#) ()
- bool [adjuststeps](#) ()

### Public Attributes

- const [Model](#) \* [model](#)
- [parameters](#) [pr](#)
- double [lmax](#)
- double [llmax](#)
- double [area](#)
- double [larea](#)
- bool [max](#)

#### 7.27.1 Detailed Description

A class containing the parameters of a maximum of the likelihood function.

Definition at line 11 of file [MCMC.h](#).

## 7.27.2 Constructor & Destructor Documentation

### 7.27.2.1 Peak::Peak ( const Model \* m, int maxx = 0 ) [inline]

Definition at line 13 of file [MCMC.h](#).

```
00013                                     : model(m), pr(m->
generateparameters(maxx)), max(maxx){
00014         lmax=model->likelihood(pr);
00015     }
```

## 7.27.3 Member Function Documentation

### 7.27.3.1 bool Peak::adjuststeps ( ) [inline]

Definition at line 136 of file [MCMC.h](#).

References [parameters::area\(\)](#).

```
00136     {
00137         parameters p1(pr);
00138         for(uint i=0;i<pr.size(); i++){
00139             double s=p1[i].step;
00140             p1[i].value+=s;
00141             double x=(lmax-model->likelihood(p1))*2/
lmax/s/s;
00142             double x0=std::pow(2/(pr[i].max-pr[i].min),2);
00143             if(x<x0) return 0;
00144             // cout<<"X "<<x<<endl;
00145             pr[i].step=1/sqrt(x);
00146             p1[i].value-=s;
00147         }
00148         area=pr.area();
00149         larea=area*lmax;
00150         return 1;
00151     }
```

Here is the call graph for this function:



## 7.27.3.2 void Peak::findPeak ( ) [inline]

Definition at line 17 of file [MCMC.h](#).

References [parameters::next\(\)](#).

Referenced by [Proposal::findPeaks\(\)](#).

```

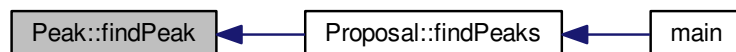
00017         {
00018             llmax=model->loglike(pr, l, max);
00019             area=1;
00020             uint fixed=1e2;
00021             uint f=fixed;
00022             double d=1;
00023             //cout<<"f "<<f<<"llmax "<<llmax<<endl;
00024             for(uint i=1e5; i; i--) {
00025                 parameters p1(pr);
00026                 p1.next(model->r, d);
00027
00028                 double ll=llmax;
00029                 if(!model->veto(p1, max)) {ll=model->
loglike(p1, l, max);
00030             }
00031             if(ll>llmax){pr=p1; llmax=ll; f=fixed;}
00032             else {f--; if(!f) {d/=100; f=fixed; if(d<1e-2) break;}}
00033         }
00034         cout<<"d "<<d<<"llmax "<<llmax<<endl;
00035         if(llmax<-1000) lmax=0;
00036         else{lmax=exp(llmax);
00037             area=pr.area();
00038             larea=area*lmax;
00039         }
00040     }
00041 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



## 7.27.4 Member Data Documentation

## 7.27.4.1 double Peak::area

Definition at line 156 of file [MCMC.h](#).

#### 7.27.4.2 double Peak::larea

Definition at line 157 of file [MCMC.h](#).

#### 7.27.4.3 double Peak::lmax

Definition at line 155 of file [MCMC.h](#).

Referenced by [Proposal::findPeaks\(\)](#).

#### 7.27.4.4 double Peak::lmax

Definition at line 155 of file [MCMC.h](#).

Referenced by [Proposal::findPeaks\(\)](#), and [main\(\)](#).

#### 7.27.4.5 bool Peak::max

Definition at line 158 of file [MCMC.h](#).

#### 7.27.4.6 const Model\* Peak::model

Definition at line 153 of file [MCMC.h](#).

#### 7.27.4.7 parameters Peak::pr

Definition at line 154 of file [MCMC.h](#).

Referenced by [Proposal::findPeaks\(\)](#), and [main\(\)](#).

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

- [MCMC.h](#)

## 7.28 points Class Reference

### Public Member Functions

- [points](#) (double p, double m, double e)

### Public Attributes

- double [prediction](#)
- double [measure](#)
- double [error](#)

### 7.28.1 Detailed Description

Definition at line 7 of file [eigen.cpp](#).

### 7.28.2 Constructor & Destructor Documentation

#### 7.28.2.1 `points::points ( double p, double m, double e )` `[inline]`

Definition at line 9 of file [eigen.cpp](#).

```
00009 :prediction(p),measure(m),error(e){}
```

### 7.28.3 Member Data Documentation

#### 7.28.3.1 `double points::error`

Definition at line 10 of file [eigen.cpp](#).

#### 7.28.3.2 `double points::measure`

Definition at line 10 of file [eigen.cpp](#).

#### 7.28.3.3 `double points::prediction`

Definition at line 10 of file [eigen.cpp](#).

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

- [eigen.cpp](#)

## 7.29 prediction Class Reference

theoretical expression for an experimental measure

```
#include <model.h>
```

### Public Member Functions

- [prediction](#) ([observable](#) \*ob, const FUNCP\_CUBA &e0)
- [prediction](#) ([observable](#) \*ob, const ex &e0)
- [prediction](#) ([calcu](#) \*c)
- [prediction](#) (const [prediction](#) &p)
- [~prediction](#) ()
- double [loglikelihood](#) (const [parameters](#) &p) const

## Public Attributes

- `shared_ptr< calcu > calculate`  
*theoretical expression for the experimental measure*

### 7.29.1 Detailed Description

theoretical expression for an experimental measure

Definition at line [344](#) of file [model.h](#).

### 7.29.2 Constructor & Destructor Documentation

#### 7.29.2.1 `prediction::prediction ( observable * ob, const FUNCP_CUBA & e0 )` `[inline]`

Definition at line [346](#) of file [model.h](#).

```
00346 : calculate(new calcuba(ob,e0)) {}
```

#### 7.29.2.2 `prediction::prediction ( observable * ob, const ex & e0 )` `[inline]`

Definition at line [347](#) of file [model.h](#).

```
00347 : calculate(new calcuex(ob,e0)) {}
```

#### 7.29.2.3 `prediction::prediction ( calcu * c )` `[inline]`

Definition at line [348](#) of file [model.h](#).

```
00348 : calculate(c) {}
```

#### 7.29.2.4 `prediction::prediction ( const prediction & p )` `[inline]`

Definition at line [349](#) of file [model.h](#).

```
00349 : calculate(p.calculate) {}
```

#### 7.29.2.5 `prediction::~~prediction ( )` `[inline]`

Definition at line [350](#) of file [model.h](#).

```
00350 {}
```



### 7.29.3 Member Function Documentation

#### 7.29.3.1 `double prediction::loglikelihood ( const parameters & p ) const` `[inline]`

Definition at line 352 of file [model.h](#).

```
00352 { return (*calculate) (p); }
```

### 7.29.4 Member Data Documentation

#### 7.29.4.1 `shared_ptr<calcu> prediction::calculate`

theoretical expression for the experimental measure

Definition at line 355 of file [model.h](#).

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

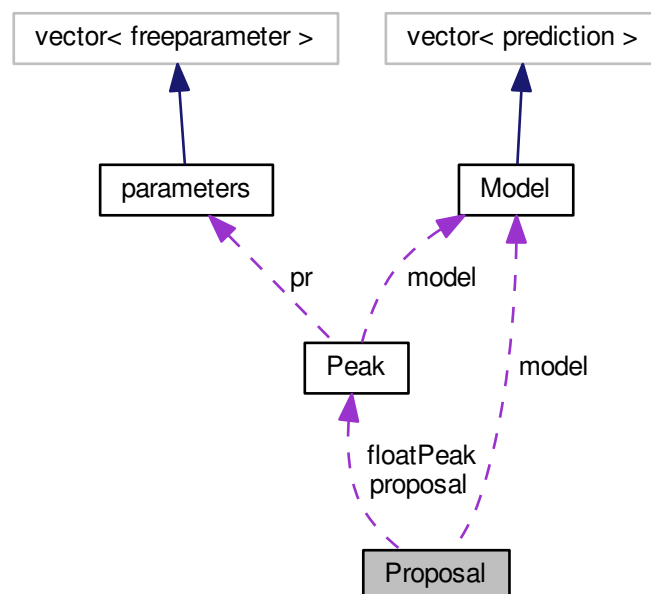
- [model.h](#)

## 7.30 Proposal Class Reference

A class containing the parameters of a proposal for the next step in the Markov Chain.

```
#include <MCMC.h>
```

Collaboration diagram for Proposal:



## Public Member Functions

- [Proposal](#) (const [Model](#) \*m)
- void [findPeaks](#) (uint ns=1, int max=0)
- void [getProposal](#) ()
- void [getNextPoint](#) ()

## Public Attributes

- const [Model](#) \* [model](#)
- vector< [Peak](#) > [vPeak](#)
- [Peak](#) [floatPeak](#)
- [Peak](#) [proposal](#)
- double [total](#)

### 7.30.1 Detailed Description

A class containing the parameters of a proposal for the next step in the Markov Chain.

Definition at line 162 of file [MCMC.h](#).

### 7.30.2 Constructor & Destructor Documentation

#### 7.30.2.1 [Proposal::Proposal](#) ( const [Model](#) \* *m* ) [inline]

Definition at line 165 of file [MCMC.h](#).

```
00165 : model (m), floatPeak (m), proposal (m) {}
```

### 7.30.3 Member Function Documentation

#### 7.30.3.1 void [Proposal::findPeaks](#) ( uint *ns* = 1, int *max* = 0 ) [inline]

Definition at line 167 of file [MCMC.h](#).

References [parameters::area\(\)](#), [Peak::findPeak\(\)](#), [Peak::lmax](#), [Peak::lmax](#), and [Peak::pr](#).

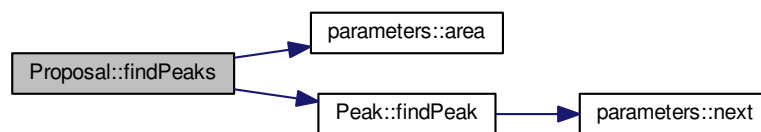
Referenced by [main\(\)](#).

```

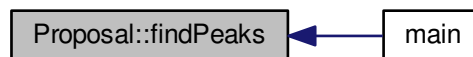
00167         {
00168             //float pmin=-100, pmax=100, s=0.1;
00169             //floatPeak.s=s;
00170             //floatPeak.lmax=0;
00171             //int imax=-1;
00172             floatPeak=Peak(model,max);
00173             floatPeak.lmax=0;
00174             floatPeak.llmax=-1000;
00175             cout<<"started"<<endl;
00176             //for(uint i=5e1;i;i--){
00177             for(uint i=ns;i;i--){
00178                 Peak pp(model,max);
00179                 pp.findPeak();
00180                 if(pp.llmax>-15){
00181                     //for(uint j=0; j< pp.pr.size();j++){
00182                     //cout<<j<<" "<<pp.pr[j].value<<endl;
00183                     //}
00184                     //lst l=model->getlist(pp.pr);
00185                     //for(uint j=0; j< model->size();j++){
00186                     //    double mean=model->at(j).calculate(1);
00187                     //cout<<j<<" "<<mean<<" "<<sqrt(2*model->at(j).o->loglikelihood(mean))<<endl;
00188                     //}
00189                 }
00190                 if(pp.lmax>floatPeak.lmax){
00191                     cout<<i<<" "<<pp.lmax<<endl;
00192                     floatPeak.lmax=pp.lmax;
00193                     floatPeak.pr=pp.pr;
00194                 }
00195             }
00196             floatPeak.area=floatPeak.pr.area();
00197         }
00198     }

```

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.30.3.2 void Proposal::getNextPoint ( ) [inline]

Definition at line 210 of file [MCMC.h](#).

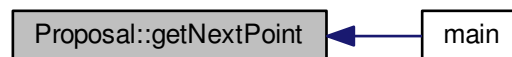
Referenced by [main\(\)](#).

```

00210         {
00211         getProposal();
00212         double l1=0;
00213         l1=model->likelihood(proposal.pr);
00214         if(model->r->Rndm() <=l1/floatPeak.lmax){
00215             floatPeak.lmax=l1;
00216             floatPeak.pr.setvalues(proposal.pr);
00217         }
00218     }

```

Here is the caller graph for this function:



#### 7.30.3.3 void Proposal::getProposal ( ) [inline]

Definition at line 200 of file [MCMC.h](#).

```

00200         {
00201         if(model->r->Rndm() <=0.9) {
00202             proposal.pr=floatPeak.pr;
00203             proposal.pr.next(model->r);
00204             return;
00205         }
00206         proposal.pr=model->generateparameters();
00207     }
00208 }

```

### 7.30.4 Member Data Documentation

#### 7.30.4.1 Peak Proposal::floatPeak

Definition at line 223 of file [MCMC.h](#).

Referenced by [main\(\)](#).

#### 7.30.4.2 const Model\* Proposal::model

Definition at line 220 of file [MCMC.h](#).

#### 7.30.4.3 Peak Proposal::proposal

Definition at line 223 of file [MCMC.h](#).

## 7.30.4.4 double Proposal::total

Definition at line 224 of file [MCMC.h](#).

## 7.30.4.5 vector&lt;Peak&gt; Proposal::vPeak

Definition at line 222 of file [MCMC.h](#).

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

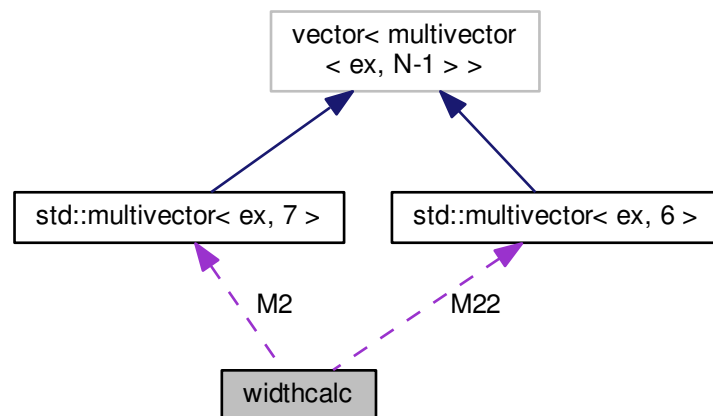
- [MCMC.h](#)

## 7.31 widthcalc Class Reference

this class calculates decay widths of one lepton to 3 leptons

```
#include <widthcalc.h>
```

Collaboration diagram for widthcalc:



### Public Member Functions

- [widthcalc](#) ()
- void [genM22](#) ()
- void [genM2](#) ()
- ex [get\\_integral](#) (const [multivector](#)< ex, 4 > &a, const vector< ex > &mass, const vector< int > &op, double m1, double m2, double m3, double m4) const
- ex [get\\_integral\\_symb](#) (const [multivector](#)< ex, 4 > &a, const vector< ex > &mass, const vector< int > &op, ex m1) const
- ex [get\\_integral\\_meson](#) (const [multivector](#)< ex, 4 > &a, const vector< ex > &mass, const vector< int > &op, ex mm, ex m1, ex m2, ex m3, ex m4) const
- ex [get\\_integral\\_meson2](#) (const [multivector](#)< ex, 4 > &a, const vector< ex > &mass, const vector< int > &op, ex mm, ex m1, ex m2, ex m3, ex m4) const

## Public Attributes

- [multivector](#)< ex, 7 > [M2](#)
- [multivector](#)< ex, 6 > [M22](#)
- realsymbol [s2](#)
- realsymbol [s3](#)
- realsymbol [mq1](#)
- realsymbol [mq2](#)
- realsymbol [mq3](#)
- realsymbol [mq4](#)

### 7.31.1 Detailed Description

this class calculates decay widths of one lepton to 3 leptons

Definition at line 21 of file [widthcalc.h](#).

### 7.31.2 Constructor & Destructor Documentation

#### 7.31.2.1 widthcalc::widthcalc ( ) [inline]

Definition at line 25 of file [widthcalc.h](#).

```
00025         : M2(0,2,2,2,2,2,2,2), M22(0,2,2,2,2,2,2,2), s2("s2"), s3("s3"),
mq1("mq1"), mq2("mq2"), mq3("mq3"), mq4("mq4"){
00026
00027         integral::max_integration_level=100;
00028         integral::relative_integration_error=1e-3;
00029
00030         genM2();
00031         genM22();
00032 }
```

### 7.31.3 Member Function Documentation

#### 7.31.3.1 void widthcalc::genM2 ( ) [inline]

Definition at line 113 of file [widthcalc.h](#).

```
00113         {
00114         cout<<"Generating M2.dat"<<endl;
00115
00116         realsymbol mu("mu"), nu("nu"), alpha("alpha"), beta("beta"), rho("rho"), zeta("zeta");
00117         realsymbol q1("q1"), q2("q2"), q3("q3"), q4("q4");
00118         realsymbol h1("h1"), h2("h2"), h3("h3"), h4("h4");
00119
00120         varidx imu(mu,4,0), inu(nu,4,0), irho(rho,4,0);
00121         varidx ialpha(alpha,4,0), ibeta(beta,4,0), izeta(zeta,4,0);
00122
00123         varidx jmu(mu,4,1), jnu(nu,4,1), jrho(rho,4,1);
00124
00125         ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*
mq4;
00126
00127         ex q2mu=indexed(q2,imu);
00128         ex q3mu=indexed(q3,imu);
00129         ex q4mu=indexed(q4,imu);
00130         ex q1mu=q2mu+q3mu+q4mu;
00131
00132 }
```

```

00133
00134         ex vq1=dirac_slash(q2,4)+dirac_slash(q3,4)+dirac_slash(q4,4)+mq1*dirac_ONE(), vq2=dirac_slash(q2,4)
+mq2*dirac_ONE();
00135         ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)-mq4*dirac_ONE();
00136
00137         ex s4=m2q1+m2q2+m2q3+m2q4-s2-s3;
00138         scalar_products sp;
00139         sp.add(q2, q3, (s4-m2q2-m2q3)/2);
00140         sp.add(q4, q3, (s2-m2q4-m2q3)/2);
00141         sp.add(q2, q4, (s3-m2q2-m2q4)/2);
00142
00143         sp.add(q2, q2, m2q2);
00144         sp.add(q3, q3, m2q3);
00145         sp.add(q4, q4, m2q4);
00146
00147         sp.add(h1,h1,-1);
00148         sp.add(h2,h2,-1);
00149         sp.add(h3,h3,-1);
00150         sp.add(h4,h4,-1);
00151
00152         sp.add(h2,q2,0);
00153         sp.add(h3,q3,0);
00154         sp.add(h4,q4,0);
00155
00156         multivector<ex,3> v(0,2,2,2);
00157         v[0][0][0]=dirac_gammaL(); v[0][0][1]=dirac_gammaR();
00158         v[0][1][0]=dirac_gammaR(); v[0][1][1]=dirac_gammaL();
00159         v[1][0][0]=dirac_gamma(jmu)*dirac_gammaL(); v[1][0][1]=dirac_gamma(jmu)*dirac_gammaL();
00160         v[1][1][0]=dirac_gamma(jmu)*dirac_gammaR(); v[1][1][1]=dirac_gamma(jmu)*dirac_gammaR();
00161
00162         multivector<ex,7> traces(0,2,2,2,2,2,2);
00163         for(uint i=0;i<2;i++)
00164             for(uint j=0;j<2;j++)
00165                 for(uint k=0;k<2;k++)
00166                     for(uint l=0;l<2;l++)
00167                         for(uint m=0;m<2;m++)
00168                             for(uint n=0;n<2;n++){
00169                                 ex vik=v[i][k][0];
00170                                 ex vim=v[i][m][0].subs(mu==nu);
00171                                 ex vjl=v[j][l][1].subs(mu==alpha);
00172                                 ex vjn=v[j][n][1].subs(mu==beta);
00173
00174                                 traces[i][j][k][l][m][n][0]=dirac_trace(vq2*vik*vq1*vjl)*dirac_trace(vq3*
vim*vq4*vjn);
00175                                 traces[i][j][k][l][m][n][1]=-dirac_trace(vq2*vik*vq1*vjl*vq3*vim*vq4*vjn);
00176                             }
00177
00178         vector<ex> prop(2,0);
00179         prop[0]=1;
00180         prop[1]=lorentz_g(imu,inu);
00181
00182         multivector<ex,2> prop2(0,2,2);
00183         for(uint i=0;i<2;i++)
00184             for(uint j=0;j<2;j++){
00185                 prop2[i][j]=prop[i]*prop[j].subs(1st(mu==alpha,nu==beta));
00186             }
00187
00188         //ofstream f("M2.dat");
00189         for(uint i=0;i<2;i++)
00190             for(uint j=0;j<2;j++)
00191                 for(uint k=0;k<2;k++)
00192                     for(uint l=0;l<2;l++)
00193                         for(uint m=0;m<2;m++)
00194                             for(uint n=0;n<2;n++)
00195                                 for(uint o=0;o<2;o++){
00196                                     {
00197                                         //cout<<i<<" "<<j<<" "<<k<<" "<<l<<" "<<m<<endl;
00198                                         M2[i][j][k][l][m][n][o]=(traces[i][j][k][l][m][n]
[o]*prop2[i][j]).simplify_indexed(sp);
00199                                         //cout<<M2[i][j][k][l][m][n][o]<<endl<<endl;
00200                                         //f<<M2[i][j][k][l][m][n][o]<<endl;
00201                                     }
00202             }

```

### 7.31.3.2 void widthcalc::genM22( ) [inline]

Definition at line 34 of file [widthcalc.h](#).

```

00034         {
00035             cout<<"Generating M22.dat"<<endl;

```

```

00036
00037     realsymbol mu("mu"), nu("nu"), alpha("alpha"), beta("beta"), rho("rho"), zeta("zeta");
00038     realsymbol q1("q1"), q2("q2"), q3("q3"), q4("q4");
00039     realsymbol h1("h1"), h2("h2"), h3("h3"), h4("h4");
00040
00041     varidx imu(mu,4,0), inu(nu,4,0), irho(rho,4,0);
00042     varidx ialpha(alpha,4,0), ibeta(beta,4,0), izeta(zeta,4,0);
00043
00044     varidx jmu(mu,4,1), jnu(nu,4,1), jrho(rho,4,1);
00045
00046     ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*
mq4;
00047
00048     ex q2mu=indexed(q2,imu);
00049     ex q3mu=indexed(q3,imu);
00050     ex q4mu=indexed(q4,imu);
00051     ex q1mu=q2mu+q3mu+q4mu;
00052
00053     ex s4=m2q1+m2q2+m2q3+m2q4-s2-s3;
00054
00055     ex vq1=dirac_slash(q2,4)+dirac_slash(q3,4)+dirac_slash(q4,4)+mq1*dirac_ONE(), vq2=dirac_slash(q2,4)
+mq2*dirac_ONE();
00056     ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)-mq4*dirac_ONE();
00057
00058     scalar_products sp;
00059     //sp.add(q2, q3, (s4-m2q2-m2q3)/2);
00060     sp.add(q4, q3, (s2-m2q4-m2q3)/2);
00061     //sp.add(q2, q4, (s3-m2q2-m2q4)/2);
00062
00063     //sp.add(q2, q2, m2q2);
00064     sp.add(q3, q3, m2q3);
00065     sp.add(q4, q4, m2q4);
00066
00067     //sp.add(h1,h1,-1);
00068     //sp.add(h2,h2,-1);
00069     //sp.add(h3,h3,-1);
00070     //sp.add(h4,h4,-1);
00071     //sp.add(h1,h1,-1);
00072
00073     multivector<ex,3> v(0,2,2,2);
00074     v[0][0][0]=dirac_gammaL(); v[0][0][1]=dirac_gammaR();
00075     v[0][1][0]=dirac_gammaR(); v[0][1][1]=dirac_gammaL();
00076     v[1][0][0]=dirac_gammaL(); v[1][0][1]=dirac_gammaL();
00077     v[1][1][0]=dirac_gammaR(); v[1][1][1]=dirac_gammaR();
00078
00079     for(uint i=0;i<2;i++){
00080         for(uint j=0;j<2;j++){
00081             v[0][i][j]=-v[0][i][j]*s2/(mq1+mq2);
00082             v[1][i][j]=(dirac_slash(q3,4)+dirac_slash(q4,4))*v[1][i][j];
00083         }
00084
00085         //vector<ex> prop(2,0);
00086         //prop[0]=-s2/(mq1+mq2);
00087         //prop[1]=indexed(q3,imu.toggle_variance())+indexed(q4,imu.toggle_variance());
00088         /*
00089         multivector<ex,2> prop2(0,2,2);
00090         for(uint i=0;i<2;i++){
00091             for(uint j=0;j<2;j++){
00092                 prop2[i][j]=prop[i]*prop[j].subs(mu==nu);
00093             }
00094         */
00095
00096         //ofstream f("M22.dat");
00097         for(uint i=0;i<2;i++){
00098             for(uint j=0;j<2;j++){
00099                 for(uint k=0;k<2;k++){
00100                     for(uint l=0;l<2;l++){
00101                         for(uint m=0;m<2;m++){
00102                             for(uint n=0;n<2;n++){
00103                                 //cout<<i<<" "<<j<<" "<<k<<" "<<l<<" "<<m<<" "<<n<<endl;
00104                                 //cout<<dirac_trace(vq3*v[i][m][1]*vq4*v[j][n][0].subs(mu==nu))<<endl;
00105
00106                                 ex tmp=dirac_trace(vq3*v[i][m][0]*vq4*v[j][n][1])*int(pow(-1.0,double(k+1+l
))) );
00107                                 M22[i][j][k][l][m][n]=tmp.simplify_indexed(sp);
00108                                 //cout<<M22[i][j][k][l][m][n]<<endl<<endl;
00109                                 //f<<M22[i][j][k][l][m][n]<<endl;
00110                             }
00111     }

```



**7.31.3.3** `ex widthcalc::get_integral ( const multivector< ex, 4 > & a, const vector< ex > & mass, const vector< int > & op, double m1, double m2, double m3, double m4 ) const` `[inline]`

Definition at line 205 of file [widthcalc.h](#).

```
00205
00206
00207
00208
00209
00210
00211
00212
00213
00214
00215
00216
00217
00218
00219
00220
00221
00222
00223
00224
00225
00226
00227
00228
00229
00230
00231
00232
00233
00234
00235
00236 }

{
    ex q10=(s2+m1*m1-m2*m2)/(2*sqrt(s2)), lq11=sqrt(q10*q10-
    m1*m1);
    ex q30=(s2+m3*m3-m4*m4)/(2*sqrt(s2)), lq31=sqrt(q30*q30-
    m3*m3);
    ex q20=(m1*m1+m2*m2-s2)/(2*m1), lq21=sqrt(q20*q20-m2*m2);
    ex total=0;
    for(uint i=0;i<a.size();i++) if(!mass[i].is_zero())
        for(uint j=0;j<a.size();j++)
            for(uint k=0;k<2;k++)
            for(uint l=0;l<2;l++)
            for(uint m=0;m<2;m++)
            for(uint n=0;n<2;n++)
            for(uint r=0;r<2;r++)
            for(uint s=0;s<2;s++){
                ex coup=a[i][r][k][m]*a[j][s][l][n].conjugate();
                if(!coup.is_zero()){
                    //cout<<i<<" "<<j<<" "<<k<<" "<<l<<" "<<n<<" "<<endl;
                    ex integrand=M2[op[i]][op[j]][k][l][m][n][(r+s)%2];
                    integrand=expand(integral(s3, m1*m1+m3*m3-2*q10*q30-2*lq11*
                    lq31, m1*m1+m3*m3-2*q10*q30+2*lq11*lq31, integrand)\
                    eval_integ()/lq11/sqrt(s2)*lq21/m1/m1);
                    double mm2=m2, mm3=m3;
                    if(l) {mm2=m3; mm3=m2;}
                    double result=ex_to_numeric>(integral(s2, std::pow(mm3+m4,2),
                    std::pow(m1-mm2,2), integrand.subs(1st(m1 == m1, mq2 == mm2, mq3 == mm3, mq4 == m4))).evalf()).to_double()/
                    std::pow(M_PI,3)/512;
                    ex partial=result*coup/(pow(mass[i],2)*pow(mass[j],2));
                    //cout<<partial<<endl;
                    total=total+partial;
                }
            }
    return total;
}
```

**7.31.3.4** `ex widthcalc::get_integral_meson ( const multivector< ex, 4 > & a, const vector< ex > & mass, const vector< int > & op, ex mm, ex m1, ex m2, ex m3, ex m4 ) const` `[inline]`

Definition at line 272 of file [widthcalc.h](#).

```
00272
00273
00274
00275
00276
00277
00278
00279
00280
00281
00282
00283
00284
00285
00286
00287
00288
00289
00290

{
    ex q10=(s2+m1*m1-m2*m2)/(2*sqrt(s2)), lq11=sqrt(q10*q10-
    m1*m1);
    ex q30=(s2+m3*m3-m4*m4)/(2*sqrt(s2)), lq31=sqrt(q30*q30-
    m3*m3);
    ex total=0;
    for(uint i=0;i<a.size();i++)
        for(uint j=0;j<a.size();j++)
            for(uint k=0;k<2;k++)
            for(uint l=0;l<2;l++)
            for(uint m=0;m<2;m++)
            for(uint n=0;n<2;n++){
                ex coup=a[i][0][k][m]*a[j][0][l][n].conjugate();
                if(!coup.is_zero()){
                    //cout<<i<<" "<<j<<" "<<k<<" "<<l<<" "<<m<<" "<<n<<" "<<endl;
                    ex integrand=M22[op[i]][op[j]][k][l][m][n];
                    //cout<<collect_common_factors(expand(a[i][0][k][m]))<<endl;
                    //
                    cout<<collect_common_factors(expand(a[j][0][l][n].conjugate()))<<endl;
                    integrand=expand(integral(s3, m1*m1+m3*m3-2*q10*q30-2*lq11*
                    lq31, m1*m1+m3*m3-2*q10*q30+2*lq11*lq31, integrand)/lq11/s2);
                    ex result=integrand.subs(1st(sqrt(s2) == mm,
                    s2==mm*mm, m1 == m1, mq2 == m2, mq3 == m3, mq4 == m4))/Pi/128;
```

```

00291         ex mi=mass[i];
00292         if(mi.is_zero()) mi=mm;
00293         ex mj=mass[j];
00294         if(mj.is_zero()) mj=mm;
00295         ex partial=result*coup/(pow(mi,2)*pow(mj,2));
00296         //cout<<i<<" "<<op[i]<<" "<<j<<" "<<op[j]<<"
" <<a[i]*a[j].conjugate()/(pow(mass[i],2)*pow(mass[j],2))<<endl<<endl;
00297
00298         total=total+partial;
00299     }
00300 }
00301
00302     return total;
00303 }

```

**7.31.3.5** `ex widthcalc::get_integral_meson2 ( const multivector< ex, 4 > & a, const vector< ex > & mass, const vector< int > & op, ex mm, ex m1, ex m2, ex m3, ex m4 ) const [inline]`

Definition at line 305 of file [widthcalc.h](#).

```

00305     {
00306
00307         ex q10=(s2+m1*m1-m2*m2)/(2*sqrt(s2)), lq11=sqrt(q10*q10-
m1*m1);
00308         ex q30=(s2+m3*m3-m4*m4)/(2*sqrt(s2)), lq31=sqrt(q30*q30-
m3*m3);
00309         ex total=0;
00310         for(uint i=0;i<a.size();i++)
00311             for(uint j=0;j<a.size();j++)
00312                 for(uint k=0;k<2;k++)
00313                     for(uint l=0;l<2;l++)
00314                         for(uint m=0;m<2;m++)
00315                             for(uint n=0;n<2;n++){
00316                                 ex coup=a[i][0][k][m]*a[j][0][l][n].conjugate();
00317                                 if(!coup.is_zero()){
00318                                     //cout<<i<<" "<<j<<" "<<k<<" "<<l<<" "<<m<<" "<<n<<" "<<endl;
00319                                     ex integrand=M22[op[i]][op[j]][k][l][m][n];
00320                                     //cout<<collect_common_factors(expand(a[i][0][k][m]))<<endl;
00321                                     //
00322                                     cout<<collect_common_factors(expand(a[j][0][l][n].conjugate()))<<endl;
00323                                     integrand=expand(integral(s3, m1*m1+m3*m3-2*q10*q30-2*lq11*
lq31, m1*m1+m3*m3-2*q10*q30+2*lq11*lq31, integrand)/lq11/s2);
00324                                     ex result=integrand.subs(lst(sqrt(s2) == mm,
s2==mm*mm, m1 == m1, m2 == m2, m3 == m3, m4 == m4))/Pi/128;
00325                                     ex mi=mass[i];
00326                                     if(mi.is_zero()) mi=mm;
00327                                     ex mj=mass[j];
00328                                     if(mj.is_zero()) mj=mm;
00329                                     ex partial=result*coup/(pow(mi,2)*pow(mj,2));
00330                                     //cout<<i<<" "<<op[i]<<" "<<j<<" "<<op[j]<<"
" <<a[i]*a[j].conjugate()/(pow(mass[i],2)*pow(mass[j],2))<<endl<<endl;
00331
00332                                     total=total+partial;
00333             }
00334 }
00335     return total;
00336 }

```

**7.31.3.6** `ex widthcalc::get_integral_symb ( const multivector< ex, 4 > & a, const vector< ex > & mass, const vector< int > & op, ex m1 ) const [inline]`

Definition at line 237 of file [widthcalc.h](#).

```

00237     {
00238
00239         ex q10=(s2+m1*m1)/(2*sqrt(s2)), lq11=(m1*m1-s2)/(2*sqrt(s2));
00240         ex q30=sqrt(s2)/2, lq31=q30;
00241         ex q20=(m1*m1-s2)/(2*m1), lq21=q20;

```

```

00242
00243     ex total=0;
00244     for(uint i=0;i<a.size();i++) if(!mass[i].is_zero())
00245         for(uint j=0;j<a.size();j++)
00246             for(uint k=0;k<2;k++)
00247                 for(uint l=0;l<2;l++)
00248                     for(uint m=0;m<2;m++)
00249                         for(uint n=0;n<2;n++)
00250                             for(uint r=0;r<2;r++)
00251                                 for(uint s=0;s<2;s++){
00252                                     ex coup=a[i][r][k][m]*a[j][s][l][n].conjugate();
00253                                     if(!coup.is_zero()){
00254                                         //cout<<i<<" "<<j<<" "<<k<<" "<<l<<endl;
00255                                         ex integrand=M2{op[i]][op[j]][k][l][m][n][(r+s)%2].subs(lst(
mq1 == m1, mq2 == 0, mq3 == 0, mq4 == 0));
00256                                         integrand=expand(integral(s3, m1*m1-2*q10*q30-2*lq11*lq31, m1*m1-
2*q10*q30+2*lq11*lq31, integrand)\
00257                                         .
eval_integ()/lq11/sqrt(s2)*lq21/m1/m1);
00258                                         //integrand=integrand.subs(lst(mq1 == m1, mq2 == 0, mq3 == 0, mq4
== 0));
00259                                         //
integrand=integrand.subs(pow(m1,2)/4-s2/2+pow(s2/m1,2)/4==pow((m1-s2/m1)/2,2));
00260
00261                                         double mm2=0, mm3=0, m4=0;
00262                                         //if (1) {mm2=m3; mm3=m2;}
00263                                         ex result=integral(s2, std::pow(mm3+m4,2), pow(m1-mm2,2), integrand)
00264                                         .eval_integ()/pow(Pi,3)/512;
00265
00266                                         ex partial=result*coup/(pow(mass[i],2)*pow(mass[j],2));
00267                                         total=total+partial;
00268                                     }
00269                                 }
00270     return total;
00271 }

```

## 7.31.4 Member Data Documentation

### 7.31.4.1 multivector<ex,7> widthcalc::M2

Definition at line 365 of file [widthcalc.h](#).

### 7.31.4.2 multivector<ex,6> widthcalc::M22

Definition at line 366 of file [widthcalc.h](#).

### 7.31.4.3 realsymbol widthcalc::mq1

Definition at line 368 of file [widthcalc.h](#).

### 7.31.4.4 realsymbol widthcalc::mq2

Definition at line 368 of file [widthcalc.h](#).

### 7.31.4.5 realsymbol widthcalc::mq3

Definition at line 368 of file [widthcalc.h](#).

#### 7.31.4.6 `realsymbol widthcalc::mq4`

Definition at line 368 of file [widthcalc.h](#).

#### 7.31.4.7 `realsymbol widthcalc::s2`

Definition at line 367 of file [widthcalc.h](#).

#### 7.31.4.8 `realsymbol widthcalc::s3`

Definition at line 367 of file [widthcalc.h](#).

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

- [widthcalc.h](#)

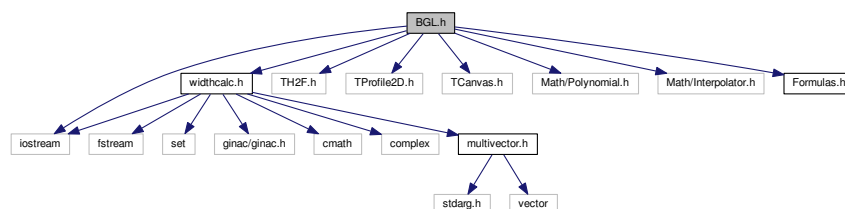
## Chapter 8

# File Documentation

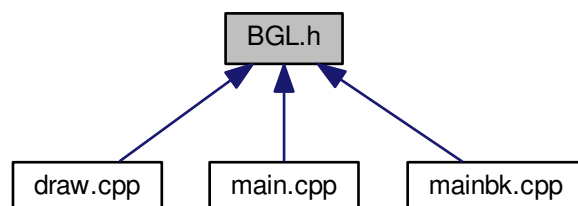
### 8.1 BGL.h File Reference

```
#include "widthcalc.h"  
#include "TH2F.h"  
#include "TProfile2D.h"  
#include "TCanvas.h"  
#include <iostream>  
#include "Math/Polynomial.h"  
#include "Math/Interpolator.h"  
#include "Formulas.h"
```

Include dependency graph for BGL.h:



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



## Classes

- class [BGLmodels::Boson](#)  
*Gauge boson.*
- class [BGLmodels::BGL](#)  
*Implementation of the [BGL](#) model.*

## Namespaces

- [BGLmodels](#)

## 8.2 BGL.h

```

00001 #ifndef BGL_H
00002 #define BGL_H
00003
00004 #include "widthcalc.h"
00005
00006 #include "TH2F.h"
00007 #include "TProfile2D.h"
00008 #include "TCanvas.h"
00009 #include <iostream>
00010
00011 #include "Math/Polynomial.h"
00012 #include "Math/Interpolator.h"
00013 #include "Formulas.h"
00014
00015
00016 namespace BGLmodels{
00017
00018 /**
00019  * @brief Gauge boson
00020  */
00021 class Boson {
00022 public:
00023
00024     Boson(): C(Matrix(),2,2,2,2){}
00025
00026     ex couplingL(const Fermion& f2,const Fermion& f1) const {
00027         bool quiralfilter=0;
00028         if(f1.type!=f2.type) return 0;
00029         if(s==sScalar){
00030             if(f1.helicity!=hRight && f2.helicity!=
00031 hLeft) quiralfilter=1;
00032             }
00033             else if(s==sVector){
00034                 if(f1.helicity!=hRight && f2.helicity!=
00035 hRight) quiralfilter=1;
00036             }
00037             if(quiralfilter) return C[f2.type][f2.isospin][f1.
00038 isospin][hLeft][f2.flavour][f1.flavour];
00039             return 0;
00040         }
00041     }
00042     ex couplingR(const Fermion& f2,const Fermion& f1) const {
00043         bool quiralfilter=0;
00044         if(f1.type!=f2.type) return 0;
00045         if(s==sScalar){
00046             if(f2.helicity!=hRight && f1.helicity!=
00047 hLeft) quiralfilter=1;
00048             }
00049             else if(s==sVector){
00050                 if(f1.helicity!=hLeft && f2.helicity!=
00051 hLeft) quiralfilter=1;
00052             }
00053             if(quiralfilter) return C[f2.type][f2.isospin][f1.
00054 isospin][hRight][f2.flavour][f1.flavour];
00055             return 0;
00056         }
00057     }
00058     ex couplingdaggerL(const Fermion& f2,const Fermion& f1) const {
00059         if(s==sScalar) return couplingR(f1,f2).conjugate();
00060         return couplingL(f1,f2).conjugate();
00061     }
00062     ex couplingdaggerR(const Fermion& f2,const Fermion& f1) const {

```

```

00056         if(s==sScalar) return couplingL(f1,f2).conjugate();
00057         return couplingR(f1,f2).conjugate();
00058     }
00059     ex coupling(const Fermion& f2,const Fermion& f1, ex mu){
00060         if(s==sScalar) return couplingL(f2,f1)*dirac_gammaL()+
couplingR(f2,f1)*dirac_gammaR();
00061         else return couplingL(f2,f1)*dirac_gammaL()+couplingR(f2,f1)*dirac_gammaR
();
00062     }
00063     void reset(){
00064         C=multivector<Matrix,4>(Matrix(),2,2,2,2);
00065     }
00066     BSpin s;
00067     ex mass;
00068     multivector<Matrix,4> C;
00069 };
00070
00071
00072
00073 //Boson::Type
Boson::dagger[2][2]={Boson::scalarright,Boson::scalarleft,Boson::vectorleft,Boson::vectorright};
00074
00075 /**
00076  * @brief Implementation of the BGL model
00077  */
00078 class BGL: public Model{
00079 public:
00080
00081     BGL(int genL=2,int genQ=2, int lup=0, int qup=0, int mssm=0):
00082         planck(6.58211928e-25),
00083         GF("G_F"),
00084         MZ("M_Z"),
00085         MW("M_W"),
00086         Mpip("Mpip",0.1396,"M_{\\pi^+}",domain::real),
00087         Mpi0("Mpi0",0.1349766,"M_{\\pi^0}",domain::real),
00088         MBp("MBp",5.279,"M_{B^+}",domain::real),
00089         MB0("MB0",5.2795,"M_{B^0}",domain::real),
00090         MBs0("MBs0",5.3663,"M_{B_s^0}",domain::real),
00091         MKp("MKp",0.493677,"MKp",domain::real),
00092         MK0("MK0",0.497614,"MK0",domain::real),
00093         MDp("MDp",1.86957,"MDp",domain::real),
00094         MD0("MD0",1.86480,"MD0",domain::real),
00095         MDsp("MDsp",1.96845,"MDsp",domain::real),
00096         MDs0("MDs0",0),
00097         Fpi("Fpi",0.132,"Fpi",domain::real),
00098         FB("FB",0.189,"FB",domain::real),
00099         FBs("FBs",0.225,"FBs",domain::real),
00100         FK("FK",0.159,"FK",domain::real),
00101         FD("FD",0.208,"FD",domain::real),
00102         FDs("FDs",0.248,"FDs",domain::real),
00103         //alpha(7.297352e-3*4*M_PI),
00104         cos2(pow(MW/MZ,2)),
00105         g(sqrt(GF*8/sqrt(ex(2)))*MW),
00106         //g(sqrt(4*Pi*alpha/(1-cos2))),
00107         tanb("tg\\beta"),
00108         cp("cp"),
00109         McH("M_{H^+}"),
00110         MR("M_{R}"),
00111         MI("M_{I}"),
00112         mixes(tanb,cp, genL,genQ, lup, qup, mssm),
00113         mu("\\mu"),
00114         BGLtype(4,0),
00115         mmmmax(1000),
00116         stepsize(1e-2)
00117         //muwidth(planck/2.197034e-6)
00118         {
00119             alpha=pow(g,2)*(1-cos2)/(4*Pi);
00120             replacements.append(GF==1.166371e-5);
00121             replacements.append(MZ==M_MZ);
00122             replacements.append(MW==M_MW);
00123
00124             mixes.appendtolst(replacements);
00125
00126             replacements.append(Pi==M_PI);
00127             replacements.append(sqrt(ex(2))==sqrt(2));
00128
00129             //cout<<pow(sqrt(2)/8*pow(g/MW,2),2)<<endl;
00130             //cout<<pow(1.166,2)<<endl;
00131
00132             Boson boson;
00133
00134             realsymbol q3("q3");
00135             ex vq3=dirac_slash(q3,4);
00136             varidx jmu(mu,4,1);
00137
00138             for(uint i=0;i<2;i++)
00139                 for(uint j=0;j<3;j++)

```

```

00140                                     for(uint k=0;k<3;k++){
00141                                     conjtoabs.append(conjugate(mixes.V[i][j][k])==pow(abs(mixes.V[i][j][k]),2)/
mixes.V[i][j][k]);
00142                                     }
00143                                     /*
00144                                     //Gamma boson
00145                                     boson.mass=0;
00146                                     boson.s=Boson::vector;
00147
00148                                     boson.coupsL[0][0]=Matrix(g*sqrt(1-cos2)*0);
00149                                     boson.coupsL[1][1]=Matrix(g*sqrt(1-cos2)*(-1));
00150                                     boson.coupsL[2][2]=Matrix(g*sqrt(1-cos2)*ex(2)/3);
00151                                     boson.coupsL[3][3]=Matrix(g*sqrt(1-cos2)*ex(-1)/3);
00152
00153                                     boson.coupsR[0][0]=Matrix(g*sqrt(1-cos2)*0);
00154                                     boson.coupsR[1][1]=Matrix(g*sqrt(1-cos2)*(-1));
00155                                     boson.coupsR[2][2]=Matrix(g*sqrt(1-cos2)*ex(2)/3);
00156                                     boson.coupsR[3][3]=Matrix(g*sqrt(1-cos2)*ex(-1)/3);
00157
00158                                     bosons.push_back(boson);
00159                                     boson.reset();
00160                                     */
00161                                     //W+ boson
00162                                     boson.mass=MW;
00163                                     boson.s=sVector;
00164
00165                                     for(uint t=tLepton;t<=tQuark;t++) boson.C[t][iUp][
iDown][hLeft]=mixes.V[t]*Matrix(g/sqrt(ex(2)));
00166                                     Boson wboson=boson;
00167                                     bosons.push_back(boson);
00168                                     boson.reset();
00169
00170                                     //H+ boson
00171                                     boson.mass=MCH;
00172                                     boson.s=sScalar;
00173
00174                                     for(uint t=tLepton;t<=tQuark;t++)
00175                                     for(uint i=iUp;i<=iDown;i++) boson.C[t][iUp][iDown][i]=mixes.VN[t][i]*
Matrix(g/MW/sqrt(ex(2)));
00176                                     Boson chiggs=boson;
00177                                     bosons.push_back(boson);
00178                                     boson.reset();
00179
00180                                     for(int b=bosons.size()-1;b>=0;b--){
00181                                     boson.mass=bosons[b].mass;
00182                                     boson.s=bosons[b].s;
00183                                     if(boson.s==sVector)
00184                                     for(uint t=tLepton;t<=tQuark;t++)
00185                                     for(uint i=iUp;i<=iDown;i++)
00186                                     for(uint j=iUp;j<=iDown;j++)
00187                                     for(uint h=hLeft;h<=hRight;h++){
00188                                     boson.C[t][i][j][h]=bosons[b].C[t][j][i][h].conjugate();
00189                                     }
00190                                     else for(uint t=tLepton;t<=tQuark;t++)
00191                                     for(uint i=iUp;i<=iDown;i++)
00192                                     for(uint j=iUp;j<=iDown;j++)
00193                                     for(uint h=hLeft;h<=hRight;h++){
00194                                     boson.C[t][i][j][hLeft]=bosons[b].C[t][j][i][
hRight].conjugate();
00195                                     boson.C[t][i][j][hRight]=bosons[b].C[t][j][i][
hLeft].conjugate();
00196                                     }
00197                                     bosons.push_back(boson);
00198                                     boson.reset();
00199                                     }
00200
00201                                     //(R+iI)/sqrt(2) boson
00202                                     boson.mass=MR;
00203                                     boson.s=sScalar;
00204
00205                                     for(uint t=tLepton;t<=tQuark;t++){
00206                                     boson.C[t][iDown][iDown][hRight]=mixes.N[t][
iDown]*Matrix(g/MW/ex(2));
00207                                     boson.C[t][iUp][iUp][hLeft]=mixes.N[t][iUp].conjugate()*
Matrix(g/MW/ex(2));
00208                                     boson.C[t][iDown][iDown][hLeft]=mixes.N[t][
iDown].conjugate()*Matrix(g/MW/ex(2));
00209                                     boson.C[t][iUp][iUp][hRight]=mixes.N[t][iUp]*
Matrix(g/MW/ex(2));
00210                                     }
00211                                     bosons.push_back(boson);
00212                                     boson.reset();
00213
00214                                     //(R+iI)/sqrt(2) boson
00215                                     boson.mass=MI;
00216                                     boson.s=sScalar;
00217

```



```

00218         for(uint t=tLepton;t<=tQuark;t++){
00219             boson.C[t][iDown][iDown][hRight]=mixes.N[t][
00220 iDown]*Matrix(I*g/MW/ex(2));
00221             boson.C[t][iUp][iUp][hLeft]=mixes.N[t][iUp].conjugate()*
00222 Matrix(I*g/MW/ex(2));
00223             boson.C[t][iDown][iDown][hLeft]=mixes.N[t][
00224 iDown].conjugate()*Matrix(-I*g/MW/ex(2));
00225             boson.C[t][iUp][iUp][hRight]=mixes.N[t][iUp]*
00226 Matrix(-I*g/MW/ex(2));
00227         }
00228         bosons.push_back(boson);
00229         boson.reset();
00230         Fermion electron(tLepton,iDown,fElectron);
00231         Fermion electronR(tLepton,iDown,fElectron,
00232 cParticle,hRight);
00233         Fermion muon(tLepton,iDown,fMuon);
00234         Fermion muonR(tLepton,iDown,fMuon,cParticle,
00235 hRight);
00236         Fermion tau(tLepton,iDown,fTau);
00237         Fermion tauR(tLepton,iDown,fTau,cParticle,
00238 hRight);
00239         Fermion neutrino(tLepton,iUp);
00240         Fermion neutrinotau(tLepton,iUp,fTau);
00241         Fermion neutrinomuon(tLepton,iUp,fMuon);
00242         Fermion neutrinoe(tLepton,iUp,fElectron);
00243         Fermion up(tQuark,iUp,fElectron);
00244         Fermion down(tQuark,iDown,fElectron);
00245         Fermion bottom(tQuark,iDown,fTau);
00246         Fermion strange(tQuark,iDown,fMuon);
00247         Fermion charm(tQuark,iUp,fMuon);
00248         Fermion top(tQuark,iUp,fTau);
00249         Meson Pi0d(down,down,Mpi0,Fpi);
00250         Meson Pi0u(down,down,Mpi0,Fpi);
00251         Meson Pip(up,down,Mpip,Fpi);
00252         Meson Pim(down,up,Mpip,Fpi);
00253         Meson K0(down,strange,MK0,FK);
00254         Meson Kp(up,strange,MKp,FK);
00255         Meson D0(charm,up,MD0,FD);
00256         Meson Dp(charm,down,MDp,FD);
00257         Meson Dsp(charm,strange,MDsp,FDs);
00258         Meson B0(down,bottom,MB0,FB);
00259         Meson Bp(up,bottom,MBp,FB);
00260         Meson Bs0(strange,bottom,MBs0,FBs);
00261         1st sb;
00262         //sb.append(mixes.M[tQuark][iUp][0][0]==0);
00263         sb.append(pow(abs(mixes.V[0][2][2]),2)==1-pow(abs(mixes.V[0][1][2]),2)-pow(abs(mixes.V[0][0][2]),2),2);
00264         sb.append(pow(abs(mixes.V[0][2][1]),2)==1-pow(abs(mixes.V[0][1][1]),2)-pow(abs(mixes.V[0][0][1]),2),2);
00265         //cout<<"Btaunu ";<<collect_common_factors(expand(Btaunu.subs(sb).subs(conjtoabs)))<<endl;
00266         cout<<latex;
00267         ex mutoenunu=decaywidth(muon,neutrino,electron,neutrino);
00268         //cout<<"mutoenunu ";<<mutoenunu<<endl;
00269         //add("mutoenunu",decaywidth(muon,neutrino,electron,neutrino),new
00270 gaussobs(planck/2.197034e-6,0.03));
00271         add("muRtoeRnnu",gRR2(muon,electron),new limitedobs(std::pow(0.035,2),0.95));
00272         //add("tautoenunu",decaywidth(tau,neutrino,electron,neutrino),new
00273 gaussobs(planck/290.6e-15*0.1782,0.03));
00274         add("tauRtoeRnnu",gRR2(tau,electron),new limitedobs(std::pow(0.7,2),0.95));
00275         //add("tautomununu",decaywidth(tau,neutrino,muon,neutrino),new
00276 gaussobs(planck/290.6e-15*0.1739,0.03));
00277         add("tauRtomuRnnu",gRR2(tau,muon),new limitedobs(std::pow(0.72,2),0.95));
00278         add("tautomu_tautoe",tautomu_tautoe(),new gaussobs(1.0018,0.0014/1.0018)); //PROBLEM!!!
00279         cout<<"tautomu_tautoe: "<<1/1.0018<< "ERROR: "<<0.0014/1.0018<<endl;
00280         cout<<"ratio1 "<<tautomu_tautoe().subs(replacements)<<endl;
00281         cout<<"ratio2 "<<(decaywidth(tau,neutrino,muon,neutrino,sVector)/decaywidth(tau,neutrino,
00282 electron,neutrino,sVector)).subs(replacements)<<endl;
00283         //muto3e
00284         ex mu3e=decaywidth(muon,electron,electron,electron);
00285         add("muto3e", mu3e,new limitedobs(planck/2.197034e-6*1e-12));

```

```

00292         cout<<"mu3e "<<decaywidthtest2(muon)<<endl;
00293
00294         //tauto3e
00295         add("tauto3e", decaywidth(tau,electron,electron,electron),new limitedobs(planck/290.6e-15
*2.7e-8));
00296         //tauto2elmu+
00297         add("tauto2elmu+", decaywidth(tau,electron,electron,muon), new
limitedobs(planck/290.6e-15*1.5e-8));
00298         //tauto2elmu-
00299         add("tauto2elmu-", decaywidth(tau,electron,muon,electron), new limitedobs(planck/290.6e-15
*1.8e-8));
00300         //tauto2mule+
00301
00302         add("tauto2mulep", decaywidth(tau,muon,muon,electron), new limitedobs(planck/290.6e-15*1.
7e-8));
00303         cout<<"tauto2mulep "<<decaywidthtest2(tau)<<endl;
00304         //tauto2mule-
00305         add("tauto2mulep", decaywidth(tau,muon,electron,muon), new limitedobs(planck/290.6e-15*2.
7e-8));
00306         cout<<"tauto2mule "<<decaywidthtest2(tau)<<endl;
00307
00308         //tauto3mu
00309         add("tauto3mu", decaywidth(tau,muon,muon,muon),new limitedobs(planck/290.6e-15*2.1e-8));
00310         //cout<<"tauto3mu "<<collect_common_factors(expand(decaywidth(tau,muon,muon,muon)))<<endl;
00311
00312
00313         ex piratio=1.2352e-4/(mesondw(Pip,neutrino,electron,sVector)/mesondw(Pip,neutrino,muon,
sVector));
00314         ex picorrection=piratio.subs(replacements);
00315         ex pierror=picorrection*0.0001/1.2352;
00316         cout<<"PiRatio "<<picorrection-1<<" +/- "<<pierror<<endl;
00317         piratio*=mesondw(Pip,neutrino,electron)/mesondw(Pip,neutrino,muon);
00318         add("piontoenu_munu",piratio,new gaussobs(1.230e-4,0.003)); //PROBLEM!!!
00319         cout<<"piontoenu_munu: "<<1.2352e-4/1.230e-4<<" ERROR: "<<0.003<<endl;
00320
00321
00322         add("tautopinu_pitomunu", (1+0.16e-2)*fermiontomeson(tau,neutrino,Pip)/mesondw(Pip,neutrino,muon),
new gaussobs((10.83e-2/290.6e-15/(0.9998770/2.6033e-8)),0.06/10.83));
00323         cout<<"tautopinu/pitomunu: "<<(1+0.16e-2)*(fermiontomeson(tau,neutrino,Pip,
sVector)/mesondw(Pip,neutrino,muon,sVector)).subs(replacements)/((10.83e-2/290.6e-15/(0.99987
70/2.6033e-8)))<<" ERROR: "<<0.06/10.83<<endl;
00324         cout<<"tautopinu: "<<fermiontomesontest(tau,neutrino,Pip)<<endl;
00325         cout<<"tautopinu_pitomunu: "<<(10.83e-2/290.6e-15/(0.9998770/2.6033e-8)<<" +/- "<<0.06e-2/290.6e-15/
(0.9998770/2.6033e-8)<<endl;
00326
00327         add("tautoKnu_Ktomunu", (1+0.9e-2)*fermiontomeson(tau,neutrino,Kp)/mesondw(Kp,neutrino,muon),new
gaussobs((7e-3/290.6e-15)/(0.6355/1.238e-8),0.1/7));
00328         cout<<"tautoKnu/Ktomunu: "<<(1+0.9e-2)*(fermiontomeson(tau,neutrino,Kp,
sVector)/mesondw(Kp,neutrino,muon,sVector)).subs(replacements)/((7e-3/290.6e-15)/(0.6355/1.23
8e-8))<<" ERROR: "<<0.1/7<<endl;
00329         cout<<"tautoKnu/Ktomunu: "<<(7e-3/290.6e-15)/(0.6355/1.238e-8)<<" +/- "<<(0.1e-3/290.6e-15)/(0.6355
/1.238e-8)<<endl;
00330
00331         //ex
00332         pi0toemu=(mesondecaywidth(Mpi0,down,down,electron,muon)+mesondecaywidth(Mpi0,up,up,electron,muon)+mesondecaywidth(Mpi0,
mesondw(Pi0u,muon,electron))/2;
00333         add("pi0toemu",pi0toemu,new limitedobs(3.6e-10*planck/8.52e-17));
00334
00335         ex Kratio=2.477e-5/(mesondw(Kp,neutrino,electron,sVector)/mesondw(Kp,neutrino,muon,
sVector));
00336         ex Kcorrection=Kratio.subs(replacements);
00337         ex Kerror=Kcorrection*0.001/2.477;
00338         cout<<"KRatio "<<Kcorrection-1<<" +/- "<<Kerror<<endl;
00339         Kratio*=mesondw(Kp,neutrino,electron)/mesondw(Kp,neutrino,muon);
00340         add("Ktoenu_munu",Kratio,new gaussobs(2.488e-5,0.005));
00341         cout<<"Ktoenu_munu: "<<2.477e-5/2.488e-5<<" ERROR: "<<0.005<<endl;
00342
00343         ex k0Ltoemu=mesondw(K0,electron,muon)+mesondw(K0,muon,electron);
00344         add("K0Ltoemu",k0Ltoemu,new limitedobs((4.7e-12*planck/5.116e-8));
00345         add("K0Ltoee",mesondw(K0,electron,electron),new limitedobs((9e-12*planck/5.116e-8));
00346
00347         //add("K0Ltomumu",mesondw(K0,muon,muon),new limitedobs((6.84e-9*planck/5.116e-8));
00348
00349         add("Dtoenu",mesondw(Dp,neutrino,electron),new limitedobs(8.8e-6*planck/1040e-15));
00350         add("Dtomunu",mesondw(Dp,neutrino,muon),new gaussobs(3.82e-4*planck/1040e-15,0.1)); //
PROBLEM!!!
00351         cout<<"Dtomunu: "<<mesondw(Dp,neutrino,muon,sVector).subs(replacements)/(3.82e-4*planck/1040
e-15)<<" ERROR: "<<0.1<<endl;
00352
00353         add("Dtotaunu",mesondw(Dp,neutrino,tau),new limitedobs(1.2e-3*planck/1040e-15)); //
PROBLEM!!!
00354         cout<<"Dtotaunu: "<<mesondw(Dp,neutrino,tau,sVector).subs(replacements)/(1.2e-3*planck/1040e
-15)<<" LIMIT"<<endl;
00355
00356         //D0 2.6e-7/410.1e-15
00357

```

```

00358         ex D0toemu=mesondw(D0,electron,muon)+mesondw(D0,muon,electron);
00359         add("D0toemu",D0toemu,new limitedobs((2.6e-7*planck/410.1e-15)));
00360     ex D0toee=mesondw(D0,electron,electron);
00361         add("D0toee",D0toee,new limitedobs((7.9e-8*planck/410.1e-15)));
00362     ex D0tomumu=mesondw(D0,muon,muon);
00363         add("D0tomumu",D0tomumu,new limitedobs((1.4e-7*planck/410.1e-15)));
00364
00365         //ex Dstomunu=mesondecaywidth(MDsp, strange, charm, muon, neutrino); //500e-15
00366         add("Dstomunu",mesondw(Dsp,neutrino,muon),new gaussobs(5.9e-3*planck/500e-15,0.33/5.9)); //
PROBLEM!!!
00367         cout<<"Dstomunu: "<<mesondw(Dsp,neutrino,muon,sVector).subs(replacements)/(5.9e-3*planck/500
e-15)<<" ERROR: "<<0.33/5.9<<endl;
00368
00369         add("Dstoenue",mesondw(Dsp,neutrino,electron),new limitedobs(1.2e-4*planck/500e-15));
00370         add("Dstotaunu",mesondw(Dsp,neutrino,tau),new gaussobs(5.43e-2*planck/500e-15,0.31/5.43));
//PROBLEM!!!
00371         cout<<"Dstotaunu: "<<mesondw(Dsp,neutrino,tau,sVector).subs(replacements)/(5.43e-2*planck/
500e-15)<<" ERROR: "<<0.31/5.43<<endl;
00372
00373         add("Btomunu",mesondw(Bp,neutrino,muon),new limitedobs(9.8e-7*planck/1.641e-12));
00374         add("Btoenu",mesondw(Bp,neutrino,electron),new limitedobs(1e-6*planck/1.641e-12));
00375
00376         add("Btotaunu",mesondw(Bp,neutrino,tau),new gaussobs(1.15e-4*planck/1.641e-12,0.23/1.15));
//add("Btotaunu",mesondw(Bp,neutrino,tau),new gaussobs(0.79e-4*planck/1.641e-12,0.23/1.15));
00377
00378
00379         //calcuBmumu
00380         calculatest(mixes,Bs0,muon,muon,2.9e-9*planck/1.516e-12,0.7e-9*planck/1.516e-12,"Bs_to_mumu");
//calcuBmumu
00381         calculatest2(mixes,B0,muon,muon,3.6e-10*planck/1.519e-12,1.6e-10*planck/1.516e-19,"B_to_mumu");
//cout<<"TESTE "<<endl;
00382         //double ps[4]={1,1e16,1e16,1e16};
00383         //double resteste=0,resteste2=0;
00384         //int nt=4,mt=1;
00385         //calculatest.fp(&nt,ps,&mt,&resteste);
00386         //calculatest2.fp(&nt,ps,&mt,&resteste2);
00387         //cout<<"TESTE "<<resteste/(2.9e-9*planck/1.516e-12)<<"
"<<resteste2/(3.6e-10*planck/1.519e-12)<<endl;
00388         //ex B0tomumu=mesondw(B0,muon,muon);
00389         //cout<<"B0tomumu "<<collect_common_factors(B0tomumu)<<endl;
00390         //1.65e-4
00391         //add("B0tomumu",B0tomumu,new limitedobs((8e-10*planck/1.519e-12)));
00392
00393         push_back(prediction(new calcuBmumu(mixes,B0,muon,muon,new
limitedobs(6.3e-10*planck/1.519e-12),"B_to_mumu")));
00394         //push_back(prediction(new calcuBmumu(mixes,B0,muon,muon,new
gauss2obs(3.6e-10*planck/1.519e-12,1.6e-10*planck/1.519e-12),"B_to_mumu")));
00395         push_back(prediction(new calcuBmumu(mixes,Bs0,muon,muon,new
gauss2obs(2.9e-9*planck/1.516e-12,0.7e-9*planck/1.516e-12),"Bs_to_mumu")));
00396         push_back(prediction(new calcuBmumu(mixes,K0,muon,muon,new
limitedobs(2.3e-9*planck/5.116e-8),"K0L_to_mumu")));
00397
00398         cBmumu=new calcuBmumu(mixes,B0,muon,muon,new limitedobs(6.3e-10*planck/1.519e-12),"
B_to_mumu");
00399         cBsmumu=new calcuBmumu(mixes,Bs0,muon,muon,new gauss2obs(2.9e-9*planck/1.516e-12,0.7
e-9*planck/1.516e-12),"Bs_to_mumu");
00400
00401     ex B0toetau=mesondw(B0,electron,tau)+mesondw(B0,tau,electron);
00402         add("B0toetau",B0toetau,new limitedobs((2.8e-5*planck/1.519e-12)));
00403     ex B0tomutau=mesondw(B0,muon,tau)+mesondw(B0,tau,muon);
00404         add("B0tomutau",B0tomutau,new limitedobs((2.2e-5*planck/1.519e-12)));
00405     ex B0toee=mesondw(B0,electron,electron);
00406         add("B0toee",B0toee,new limitedobs((8.3e-8*planck/1.519e-12)));
00407     ex B0totautau=mesondw(B0,tau,tau);
00408         add("B0totautau",B0totautau,new limitedobs((4.1e-3*planck/1.519e-12)));
00409
00410         //Bs m=5.3663, life=1.472e-12 emu=2e-7, ee=2.8e-7 mumu=4.2e-8
00411     ex Bs0toemu=mesondw(Bs0,electron,muon)+mesondw(Bs0,muon,electron);
00412         add("Bs0toemu",Bs0toemu,new limitedobs((2e-7*planck/1.516e-12)));
00413     ex Bs0toee=mesondw(Bs0,electron,electron);
00414         add("Bs0toee",Bs0toee,new limitedobs((2.8e-7*planck/1.516e-12)));
00415     // ex Bs0tomumu=mesondw(Bs0,muon,muon);
00416     // add("Bs0tomumu",Bs0tomumu,new limitedobs((3.2e-9*planck/1.516e-12)));
00417
00418     // add("chargedHiggs",pow(McH,-2),new limitedobs(std::pow(80.0,-2),0.9));
00419
00420     cout<<"Bs0tomumu: "<<mesondwtest(Bs0,muon,muon)<<endl;
00421     //add("chargedHiggs",1/McH,new limitedobs(1/80.0,0));
00422
00423     /*
00424     Matrix llgamma2loop=Matrix(sqrt(ex(2))*mixes.N[tQuark][iUp][2][2]*mixes.M[tQuark][iUp][fTau][fTau]*
pow(1/McH*log(mixes.M[tQuark][iUp][fTau][fTau]/McH,2))*mixes.N[tLepton][iDown];
00425     for(uint i=0;i<3;i++)
00426         for(uint j=0;j<3;j++)
00427             if(j<i) llgamma2loop[i][j]=ex(3)*pow(g*g*(1-cos2)/4/Pi/Pi,3)*llgamma2loop[j][i]*
llgamma2loop[j][i].conjugate()/pow(mixes.M[tLepton][iDown][i][i],2);
00428             else llgamma2loop[i][j]=0;
00429         add("mutoegamma",llgamma2loop[1][0],new limitedobs(1.2e-11));

```

```

00430         add("tautoegamma",llgamma2loop[2][0],new limitedobs(3.3e-8));
00431         add("tautomugamma",llgamma2loop[2][1],new limitedobs(4.4e-8));
00432         //add("mutoegamma",llgamma2loop[1][0],new limitedobs(planck/2.197034e-6*1.2e-11));
00433         //add("tautoegamma",llgamma2loop[2][0],new limitedobs(planck/290.6e-15*3.3e-8));
00434         //add("tautomugamma",llgamma2loop[2][1],new limitedobs(planck/290.6e-15*4.4e-8));
00435     */
00436
00437     Matrix llgammaCH;
00438     //Matrix
00439     llgammaCH=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH*12))*mixes.M[tLepton][iDown]*mixes.VN[tLepton][1].conjugate()*mixes.VN[1]
00440     Matrix llgammaHOM,llgammaH0E;
00441     //for(uint i=0;i<3;i++)
00442         for(uint j=0;j<3;j++)
00443             for(uint k=0;k<3;k++){
00444                 ex z=pow(fmasses[1][i][i]/McH,2); mixes.M[tQuark][iUp][i]/MR,2);
00445                 llgammaHOM[j][k]=llgammaHOM[j][k]+(mixes.VN[0][1][j][i].conjugate()*
00446                 mixes.VN[0][1][k][i]+mixes.VN[0][1][i][j]*mixes.VN[0][1][i][k].conjugate())/pow(mixes.M[tLepton][iDown][i][i],2)*(2
00447                 *z+6*z*log(z))/6;
00448                 llgammaHOM[j][k]=llgammaHOM[j][k]+(mixes.VN[0][1][i][j]*
00449                 mixes.VN[0][1][k][i])/mixes.M[tLepton][iDown][i][i]/mixes.M[tLepton][iDown][j][j]*(3*z+2*z*log(z));
00450                 llgammaH0E[j][k]=llgammaH0E[j][k]+(mixes.VN[0][1][j][i].conjugate()*
00451                 mixes.VN[0][1][k][i]-mixes.VN[0][1][i][j]*mixes.VN[0][1][i][k].conjugate())/pow(mixes.M[tLepton][iDown][i][i],2)*(2
00452                 *z+6*z*log(z))/6;
00453                 llgammaH0E[j][k]=llgammaH0E[j][k]+(mixes.VN[0][1][i][j]*
00454                 mixes.VN[0][1][k][i])/mixes.M[tLepton][iDown][i][i]/mixes.M[tLepton][iDown][j][j]*(3*z+2*z*log(z));
00455             }
00456     */
00457     llgammaHOM=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH))*mixes.M[tLepton][
00458     iDown]*llgammaHOM;
00459     llgammaH0E=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH))*mixes.M[tLepton][
00460     iDown]*llgammaH0E;
00461
00462     Matrix llgamma, llgamma2;
00463
00464     for(uint i=0;i<3;i++)
00465         for(uint j=0;j<3;j++){
00466             //if(j<i)
00467                 llgamma[i][j]=(llgammaCH[i][j]*llgammaCH[i][j].conjugate()+llgammaH0E[i][j]*llgamma
00468                 aH0E[i][j].conjugate()+llgammaHOM[i][j]*llgammaHOM[i][j].conjugate())*g*g*(1-cos2)*pow((pow(mixes.M[tLepton][iDown][i][i],
00469                 ex mmuon=mixes.M[tLepton][iDown][i][i];
00470                 ex A,B;
00471
00472                 if(j<i){ for(uint k=0;k<3;k++){
00473                     ex mtau=mixes.M[tLepton][iDown][k][k];
00474                     B+=-mixes.VN[tLepton][1][k][j].conjugate()*mixes.VN[
00475                     tLepton][1][k][i]/(12*pow(McH,2));
00476                     B+=mixes.N[tLepton][1][k][j].conjugate()*mixes.N[
00477                     tLepton][1][k][i]/12*(pow(MR,-2)+pow(MI,-2));
00478                     A+=mixes.N[tLepton][1][j][k]*mixes.N[tLepton][1][i][k].
00479                     conjugate()*(pow(MR,-2)+pow(MI,-2))/12;
00480                     A+=mixes.N[tLepton][1][j][k]*mixes.N[tLepton][1][k][i]/mtau/mmuon*(
00481                     Fh2(pow(mtau/MR,2))-Fh2(pow(mtau/MI,2)))/4;
00482                 }
00483                 llgamma[i][j]=(A*A.conjugate()+B*B.conjugate())*alpha*pow(mmuon,5)*GF*GF/(128*pow(
00484                 Pi,4));
00485             }
00486             else if(j==i){
00487                 for(uint k=0;k<3;k++){
00488                     ex mtau=mixes.M[tLepton][iDown][k][k];
00489                     B+=-mixes.VN[tLepton][1][k][j].conjugate()*mixes.VN[
00490                     tLepton][1][k][i]/(12*pow(McH,2));
00491                     B+=mixes.N[tLepton][1][k][j].conjugate()*mixes.N[
00492                     tLepton][1][k][i]/12*(pow(MR,-2)+pow(MI,-2));
00493                     B+=mixes.N[tLepton][1][j][k].conjugate()*mixes.N[
00494                     tLepton][1][i][k]/12*(pow(MR,-2)+pow(MI,-2));
00495                 }
00496                 llgamma[i][j]=-B*GF*sqrt(1/2)/(8*pow(Pi,2))*2*mmuon; //e (GeV)^-1=1/(51e6) (e cm) where
00497                 e=sqrt(alpha*4*Pi)
00498             }
00499         }
00500     add("mutoegamma",llgamma[1][0],new limitedobs(planck/2.197034e-6*2.4e-12));
00501     add("tautoegamma",llgamma[2][0],new limitedobs(planck/290.6e-15*3.3e-8));
00502     add("tautomugamma",llgamma[2][1],new limitedobs(planck/290.6e-15*4.4e-8));
00503
00504     //add("d_e",abs(llgamma[0][0].imag_part()),new limitedobs(10.5e-28*51e6));
00505     add("d_mu",abs(llgamma[1][1].imag_part()),new limitedobs(1.9e-19*51e6));
00506     add("d_tau",llgamma[2][2].imag_part(),new gaussobs(-0.85e-17*51e6,0.825/0.85));
00507     cout<<"EDM: "<<llgamma[0][0].subs(conjtoabs).subs(replacements).imag_part()<<endl;
00508     add("a_mu",-llgamma[1][1].real_part()*2*mixes.M[tLepton][iDown][1][1],new gaussobs(3e-9,1.0/3.0));
00509     */
00510     //llgammaCH=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH*12))*mixes.M[tQuark][iDown]*
00511     mixes.VN[1][1].conjugate()*mixes.VN[1][1]; //4+1
00512     //Matrix llgammaHOM,llgammaH0E;
00513     for(uint i=0;i<3;i++)
00514         for(uint j=0;j<3;j++)

```

```

00496         for(uint k=0;k<3;k++){
00497             ex z=pow(mixes.M[tQuark][iUp][i][i]/MR,2);
00498             llgammaHOM[j][k]=llgammaHOM[j][k]+(mixes.VN[1][1][j][i].conjugate()*
mixes.VN[1][1][k][i]+mixes.VN[1][1][i][j]*mixes.VN[1][1][i][k].conjugate())/pow(mixes.M[tQuark][iDown][i][i],2)*(2*
z+6*z*z*log(z))/6;
00499             llgammaHOM[j][k]=llgammaHOM[j][k]+(mixes.VN[1][1][i][j]*
mixes.VN[1][1][k][i])/mixes.M[tQuark][iDown][i][i]/mixes.M[tQuark][iDown][j][j]*(3*z+2*z*log(z));
00500             llgammaHOM[j][k]=llgammaHOM[j][k]+(mixes.VN[1][1][j][i].conjugate()*
mixes.VN[1][1][k][i]-mixes.VN[1][1][i][j]*mixes.VN[1][1][i][k].conjugate())/pow(mixes.M[tQuark][iDown][i][i],2)*(2*
z+6*z*z*log(z))/6;
00501             llgammaHOM[j][k]=llgammaHOM[j][k]+(mixes.VN[1][1][i][j]*
mixes.VN[1][1][k][i])/mixes.M[tQuark][iDown][i][i]/mixes.M[tQuark][iDown][j][j]*(3*z+2*z*log(z));
00502             llgammaHOM[j][k]=llgammaHOM[j][k]+(mixes.VN[1][1][i][j]*
mixes.VN[1][1][k][i])/mixes.M[tQuark][iDown][i][i]/mixes.M[tQuark][iDown][j][j]*(3*z+2*z*log(z));
00503         }
00504         llgammaHOM=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH))*mixes.M[tQuark][iDown]*llgammaHOM;
00505         llgammaHOM=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH))*mixes.M[tQuark][iDown]*llgammaHOM;
00506         //Matrix llgamma;
00507         for(uint i=0;i<3;i++){
00508             for(uint j=0;j<3;j++){
00509                 if(j<i) {llgamma[i][j]=(llgammaCH[i][j]*llgammaCH[i][j].conjugate()+llgammaHOM[i][j]*
llgammaHOM[i][j].conjugate()+llgammaHOM[i][j]*llgammaHOM[i][j].conjugate())*g*g*(1-cos2)*
pow((pow(mixes.M[tQuark][iDown][i][i],2)-pow(mixes.M[tQuark][iDown][j][j],2))/mixes.M[tQuark][iDown][i][i],3)/(4*Pi);
00510                 //llgamma[i][j]=llgamma[i][j].subs(1st(abs(wild()*pow(MH0,-2))==abs(wild())
00511                 *pow(MH0,-2)))};
00512             else llgamma[i][j]=0;
00513         }
00514         */
00515
00516
00517         push_back(prediction(new calculbtosgamma2(mixes)));
00518
00519         //add("btosgamma",llgamma[2][1],new gaussobs(3.55e-4,sqrt(2)*0.25/3.55),1);
00520         //cout<<csrc<<llgamma[2][1]<<endl;
00521         //cout<<latex;
00522
00523
00524         BR_Htotaunu=(CHdecaycoupling(chiggs,tau,neutrino)+3*CHdecaycoupling(chiggs,strange,charm))/factor(
CHdecaycoupling(chiggs,Fermion(tLepton,iDown),neutrino)+3*CHdecaycoupling(chiggs,
Fermion(tQuark,iDown),charm)+3*CHdecaycoupling(chiggs,Fermion(
tQuark,iDown),up));
00525         BR_Htotaunu=BR_Htotaunu.subs(replacements);
00526
00527         //BR_toptoHq=decaywidth(top,bottom,chiggs);
00528         //ex toptoWb=decaywidth(top,bottom,wboson);
00529         //BR_toptoHq=BR_toptoHq/(BR_toptoHq+toptoWb);
00530         //BR_toptoHq=BR_toptoHq.subs(replacements);
00531
00532         //cout<<"toptoWb "<<toptoWb.subs(replacements).evalf()<<endl;
00533
00534         //b to c tau- nu/b to c e- nu
00535         //ex
        btocR=decaywidth(bottom,charm,tau,neutrino,sVector)/(decaywidth(bottom,charm,electron,neutrino,sVector)+decaywidth(bot
00536         //cout<<btocR.subs(replacements)<<endl;
00537
00538         ex BtoDtaunu,BtoD2taunu, BtoDtaunuSM, KtoPi;
00539         for(uint i=0; i<3; i++){
00540             ex Wcoup=wboson.couplingL(charm,bottom)*wboson.
couplingdaggerL(tau,Fermion(tLepton,iUp,Fflavour(i)));
00541             if(Wcoup.subs(replacements)==ex(0)) continue;
00542             ex chcoup_Wcoup=-pow(MW/McH,2)*(chiggs.couplingR(charm,bottom)+chiggs.
couplingL(charm,bottom))*chiggs.couplingdaggerL(tau,
Fermion(tLepton,iUp,Fflavour(i)))/Wcoup;
00543             ex chcoup2_Wcoup=-pow(MW/McH,2)*(chiggs.couplingR(charm,bottom)-chiggs.
couplingL(charm,bottom))*chiggs.couplingdaggerL(tau,
Fermion(tLepton,iUp,Fflavour(i)))/Wcoup;
00544             BtoDtaunuSM+=Wcoup*Wcoup.conjugate();
00545             BtoDtaunu+=Wcoup*Wcoup.conjugate()*(1+1.5*chcoup_Wcoup.real_part()+chcoup_Wcoup.conjugate()
*chcoup_Wcoup);
00546             BtoD2taunu+=Wcoup*Wcoup.conjugate()*(1+0.12*chcoup2_Wcoup.real_part()+0.05*chcoup2_Wcoup.
conjugate()*chcoup2_Wcoup);
00547         }
00548         1st r2(pow(mixes.V[1][1][2].imag_part(),2)==pow(abs(mixes.V[1][1][2]),2)-pow(mixes.V[1][1][2].
real_part(),2));
00549         r2.append(pow(mixes.V[0][2][2].imag_part(),2)==pow(abs(mixes.V[0][2][2]),2)-pow(mixes.V[0][2][2].
real_part(),2));
00550         r2.append(mixes.M[1][0][1][1]==0);
00551         r2.append(pow(abs(mixes.V[0][2][2]),2)==1-pow(abs(mixes.V[0][1][2]),2)-pow(abs(mixes.V[0][0][2]),2)
);
00552         r2.append(pow(abs(mixes.V[0][2][1]),2)==1-pow(abs(mixes.V[0][1][1]),2)-pow(abs(mixes.V[0][0][1]),2)
);
00553         r2.append(abs(sqrt(ex(2))*GF)==sqrt(ex(2))*GF);
00554
00555         BtoDtaunuSM=collect_common_factors(BtoDtaunuSM.subs(conjtoabs).subs(r2));
00556         BtoDtaunu=collect_common_factors(BtoDtaunu.subs(conjtoabs).subs(r2));

```

```

00559
00560     BtoDtaunuR=(BtoDtaunu/BtoDtaunuSM).subs(replacements).real_part();
00561
00562     BtoD2taunu=BtoD2taunu.subs(conjtoabs).subs(r2);
00563     BtoD2taunuR=(BtoD2taunu/BtoDtaunuSM).subs(replacements).real_part();
00564
00565
00566     //cout<<"BtoDtaunu/BtoDtaunuSM "<<expand(BtoDtaunu/BtoDtaunuSM)<<endl;
00567     iBDtaunu=size();
00568     add("BtoDtaunu_BtoDtaunuSM",BtoDtaunu/BtoDtaunuSM,new gaussobs(440.0/296, 1.4*58.0/440));
00569
00570     iBD2taunu=size();
00571     //cout<<"BtoD2taunu/BtoD2taunuSM
00572     "<<1+collect_common_factors(expand(BtoD2taunu/BtoDtaunuSM-1))<<endl;
00573     add("BtoD2taunu_BtoD2taunuSM",BtoD2taunu/BtoDtaunuSM,new gaussobs(332.0/252, 1.4*24.0/332.0
00574 ));
00575
00576     for(uint j=0; j<2; j++){
00577         ex KtoPimunu, KtoPimunuSM;
00578         for(uint i=0; i<3; i++){
00579             ex Wcoup=wboson.couplingL(up, strange)*wboson.
00580             couplingdaggerL(Fermion(tLepton,iDown,FFlavour(j)),
00581             Fermion(tLepton,iUp,FFlavour(i)));
00582             if(Wcoup.subs(replacements)==ex(0)) continue;
00583             ex chcoup_Wcoup=-pow(MW/McH,2)*(chiggs.couplingR(up, strange)+chiggs.
00584             couplingL(up, strange))\
00585             *chiggs.couplingdaggerL(muon,
00586             Fermion(tLepton,iUp,FFlavour(i)))/Wcoup*(pow(MKp,2)-pow(Mpip,2))\
00587             /(mixes.mass(Fermion(tLepton,iDown,
00588             FFlavour(j)))*(mixes.mass(strange)-mixes.mass(up)));
00589             chcoup_Wcoup=collect_common_factors(expand(chcoup_Wcoup));
00590             KtoPimunuSM+=collect_common_factors(expand(Wcoup*Wcoup.conjugate()));
00591             KtoPimunu+=collect_common_factors(expand(Wcoup*Wcoup.conjugate()*pow(1+chcoup_Wcoup,2)));
00592         }
00593         KtoPimunuSM=collect_common_factors(expand(KtoPimunuSM.subs(conjtoabs).subs(r2)));
00594         KtoPimunu=collect_common_factors(expand(KtoPimunu.subs(conjtoabs).subs(r2)));
00595         KtoPimunu=expand(KtoPimunu.subs(replacements).real_part().subs(1st(abs(wild()*pow(MR,-2))==abs(
00596         wild()*pow(MR,-2))).subs(1st(log(wild()*pow(MR,-2))==log(wild())-2*log(MR))));
00597         KtoPimunu=expand(KtoPimunu.evalf());
00598         KtoPimunuSM=expand(KtoPimunuSM.subs(replacements).real_part().subs(1st(abs(wild()*pow(MR,-2))==
00599         abs(wild()*pow(MR,-2))).subs(1st(log(wild()*pow(MR,-2))==log(wild())-2*log(MR))));
00600         KtoPimunuSM=expand(KtoPimunuSM.evalf());
00601         KtoPi+=0.5*log(KtoPimunu/KtoPimunuSM);
00602     }
00603     add("KtoPi",KtoPi/(pow(MKp,2)-pow(Mpip,2)),new gaussobs(0.08, 0.11/0.08));
00604
00605     //add("b to c tau- nu/b to c e- nu", decaywidth(bottom,charm,electron,neutrino), new
00606     limitedobs(planck/290.6e-15*2.7e-8));
00607
00608     double fD=0.207;
00609     ex DDbbar=ex(std::pow(fD,2))*mesonmixing(MD0,charm,up);
00610     DDbbar=expand(DDbbar.subs(replacements).subs(1st(abs(wild()*pow(MR,-2))==abs(wild()*pow(MR,-2))).subs(
00611     1st(log(wild()*pow(MR,-2))==log(wild())-2*log(MR))));
00612     DDbbar=expand(DDbbar.evalf());
00613     ex aDDbbar=sqrt(DDbbar.real_part()*DDbbar.real_part()+DDbbar.imag_part()*DDbbar.imag_part());
00614     add("DDbbar",aDDbbar,new limitedobs(9.47e-15));
00615     cout<<DDbbar<<endl;
00616     //2|M12|<6.6e-15GeV
00617
00618     double fK=0.156;
00619     ex KKbar=ex(std::pow(fK,2))*mesonmixing(MK0,strange,down);
00620     KKbar=expand(KKbar.subs(replacements).subs(1st(abs(wild()*pow(MR,-2))==abs(wild()*pow(MR,-2))).subs(
00621     1st(log(wild()*pow(MR,-2))==log(wild())-2*log(MR))));
00622     KKbar=expand(KKbar.evalf());
00623     ex aKKbar=sqrt(KKbar.real_part()*KKbar.real_part()+KKbar.imag_part()*KKbar.imag_part());
00624     add("KKbar",aKKbar,new limitedobs(3.5e-15));
00625     ex eK=0.94*imag_part(KKbar)/3.5e-15/sqrt(2);
00626     //add("a_eK",abs(eK),new limitedobs(2.2e-3));
00627     add("a_eK",abs(eK),new limitedobs(20*0.0114e-3));
00628     cout<<abs(KKbar)<<endl;
00629
00630     double fB=0.189;
00631     ex Vtb=mixes.V[tQuark][2][2]/mixes.V[tQuark][2][2].conjugate();
00632     ex Vtd=mixes.V[tQuark][2][0]/mixes.V[tQuark][2][0].conjugate();
00633     ex Vts=mixes.V[tQuark][2][1]/mixes.V[tQuark][2][1].conjugate();
00634
00635     ex BBbar=1+ex(std::pow(fB,2))*mesonmixing(MB0,bottom,down)/(3.337e-13*Vtb*Vtd.conjugate());
00636     add("BBbarimag",imag_part(BBbar),new gauss2obs(-0.199,0.062));
00637     add("BBbarreal",real_part(BBbar),new gauss2obs(0.823,0.143));
00638     cout<<BBbar<<endl;
00639     BBbar=3.337e-13*Vtb*Vtd.conjugate();
00640     cout<<"Bbar "<<(abs(imag_part(BBbar))/abs(BBbar)).subs(replacements)<<endl;
00641     double fBs=0.225;

```

```

00634     ex BsBsbar=1+ex(std::pow(fBs,2))*mesonmixing(MBs0,bottom,strange)/(1.186e-11*Vtb*Vts.conjugate());
00635     add("BsBsbarimag",imag_part(BsBsbar),new gauss2obs(0,0.1));
00636     add("BsBsbarreal",real_part(BsBsbar),new gauss2obs(0.965,0.133));
00637     cout<<BsBsbar<<endl;
00638     BsBsbar=1.186e-11*Vtb*Vts.conjugate();
00639     cout<<"Bbar " <<(abs(imag_part(BsBsbar))/abs(BsBsbar)).subs(replacements)<<endl;
00640
00641     ex McH2=McH*McH;
00642     ex MR2=MR*MR;
00643     ex MI2=MI*MI;
00644
00645     ex cu=collect_common_factors(expand(chiggs.couplingL(top,bottom)))/mixes.mass(top)/(g/MW/sqrt(
ex(2)))/mixes.V[1][2][2];
00646     cout<<"cu " <<cu<<endl;
00647     ex Zbb=(cu-0.72)/McH;
00648     add("Zbb",Zbb,new limitedobs(0.0024));
00649     cout<<"Zbb " <<Zbb<<endl;
00650     cout<<"SIZE " <<size()<<endl;
00651
00652     push_back(prediction(new calcuOblique()));
00653 }
00654
00655 ~BGL(){
00656     delete cBmumu;
00657     delete cBsmumu;
00658 }
00659
00660 ex Y(ex x) const{
00661     return 1.0113*x/8/(1-x)*(4-x+3*x*log(x)/(1-x));
00662 }
00663
00664 ex GW(ex x) const{
00665     return (94-x*(179+x*(-55+12*x)))/(36*pow(1-x,3))+x*(16+x*(-32+9*x))*log(x)/(6*pow(1-x,4));
00666 }
00667
00668 ex GH1(ex x) const{
00669     return x*(x*((39-14*x)*x-6)+6*x*(3*x-8)*log(x)-19)/(36*pow(x-1,4));
00670     //return -x/12;
00671 }
00672
00673 ex GH2(ex x) const{
00674     return x*((x-1)*(11*x-21)+(16-6*x)*log(x))/(6*pow(x-1,3));
00675     //return x/2;
00676 }
00677
00678 ex FW(ex x) const{
00679     return (94-x*(179+x*(-55+12*x)))/(36*pow(1-x,3))+x*(16+x*(-32+9*x))*log(x)/(6*pow(1-x,4));
00680 }
00681
00682 ex FH1(ex x) const{
00683     return -x/12;
00684 }
00685
00686 ex FH2(ex x) const{
00687     return x/2;
00688 }
00689
00690 ex Fh1(ex x) const{
00691     //return (2*x+3*pow(x,2)-6*pow(x,3)+pow(x,4)+6*pow(x,2)*log(x))/(6*pow(1-x,4));
00692     return x/3;
00693 }
00694
00695 ex Fh2(ex x) const{
00696     //return (-3*x+4*pow(x,2)-pow(x,3)-2*x*log(x))/pow(1-x,3);
00697     return -2*(3/2+log(x))*x;
00698 }
00699
00700 ex A0(ex x) const{
00701     return x*(2+3*x-6*x*x+ x*x*x+6*x*log(x))/(24*pow(1-x,4));
00702 }
00703
00704 ex A1(ex x) const{
00705     return x*(-3+4*x-x*x-2*log(x))/(4*pow(1-x,3));
00706 }
00707
00708 ex A2(ex x) const{
00709     return x/(6*pow(1-x,3))*((-7+5*x-8*x*x)/6.0+x*log(x)/(1-x)*(-2+3*x));
00710 }
00711
00712 ex A3(ex x) const{
00713     return (-3+8*x-5*x*x+(6*x-4)*log(x))*x/(6*pow(1-x,3));
00714 }
00715
00716 void add(const char * s, ex pred, observable * ob, bool sb=0){
00717     //cout<<s<<endl;
00718     //cout<<"prediction symb"<<pred<<endl;
00719     //,pow(sin(wild()), 2) == 1-pow(cos(wild()), 2)

```



```

00720         //ex
00721         p=expand(pred.subs(replacements).real_part()).subs(1st(abs(wild())*pow(MR,-2))==abs(wild())*pow(MR,-2))).subs(1st(log(wi.
00722         ex=pred.subs(replacements).real_part());
00723         p=collect_common_factors(expand(p.evalf()));
00724         FUNCP_CUBA fp;
00725
00726         lst l(tanb,McH,MR,MI);
00727
00728         for(uint i=0;i<3;i++){
00729             l.append(Mu[i]);
00730             l.append(Md[i]);
00731         }
00732         if(sb) push_back(prediction(ob,p));
00733         else {
00734             compile_ex(lst(p), l, fp);
00735             //cout<<"prediction numeric"<<p<<endl;
00736             //cout<<"exp "<<ob->expected()<<endl<<endl;
00737             push_back(prediction(ob,fp));
00738         }
00739     }
00740
00741 int veto(const parameters & p, int max=0) const{
00742     if(!p.isvalid()) return 1;
00743     if(max==1){
00744         double mr=p[1].value+p[2].value;
00745         if(mr<10 || mr>10000) return 1;
00746         mr+=p[3].value;
00747         if(mr<10 || mr>10000) return 1;
00748         return 0;
00749     }
00750     else{
00751         double mr=p[1].value+p[2].value;
00752         if(mr<10 || mr>mmmax) return 1;
00753         mr+=p[3].value;
00754         if(mr<10 || mr>mmmax) return 1;
00755         return 0;
00756     }
00757 }
00758
00759 parameters generateparameters(int max=0) const{
00760     parameters p;
00761     //x=log_10(tanb)
00762     p.push_back(freeparameter(-3,3,r,stepsize));
00763     //y=log_10(McH)
00764     if(max==1) p.push_back(freeparameter(10,10000,r,stepsize));
00765     else p.push_back(freeparameter(10,mmmax,r,stepsize));
00766     //log_10(massR)
00767     p.push_back(freeparameter(-200,200,r,stepsize));
00768     //log_10(massI)
00769     p.push_back(freeparameter(-50,50,r,stepsize));
00770
00771     return p;
00772 }
00773
00774 parameters getlist(const parameters & p) const{
00775     //cout<<aux<<endl;
00776     //double
00777     c2=(1+sqrt(1-4*sqrt(ex_to<numeric>(mudecay.subs(1st(tanb==exp(p[0].value),McH==p[1].value))).to_double())))/2;
00778
00779     double x=pow(10.0,p[0].value);
00780     //double y=pow(10.0,p[1].value);
00781     //double z=pow(10.0,p[2].value);
00782     //double w=pow(10.0,p[3].value);
00783
00784     double y=p[1].value;
00785     double z=y+p[2].value;
00786     double w=z+p[3].value;
00787
00788     parameters pp(p);
00789     pp[0].value=x;
00790     pp[2].value+=pp[1].value;
00791     pp[3].value+=pp[2].value;
00792     pp.values=vector<double>();
00793     for(uint i=0; i<4; i++) pp.values.push_back(pp[i].value);
00794     lst &l=pp.p;
00795     l=lst(tanb==x,McH==y,MR==z,MI==w);
00796
00797     return pp;
00798 }
00799
00799 double bsgammawidth(double tanb,double McH,double MR,double MI, int option=0){
00800     parameters p=generateparameters();
00801     p[0].value=pow(10.0,tanb);
00802     p[1].value=McH;
00803     p[2].value=MR;
00804     p[3].value=MI;

```



```

00805
00806         calcubtosgamma2 cal(mixes);
00807
00808         return cal.width(p,option);
00809     }
00810     /*
00811     ex decaywidth2(const Fermion& f1, const Fermion& ff2, const Fermion& ff3, const Fermion& ff4, BSpin s=sAny)
    const{
00812
00813         Fermion f2=ff2,f3=ff3, f4=ff4;
00814
00815         ex ret=0;
00816
00817         realsymbol q1("q1"), q2("q2"), q3("q3"), q4("q2"), s3("s3");
00818         ex s2=pow(mixes.mass(f1),2);
00819
00820         for(uint j=fElectron;j<=fTau;j++)
00821             if(ff2.flavour==fAny || ff2.flavour==j){
00822                 f2.flavour=(Fflavour)j;
00823             for(uint k=fElectron;k<=fTau;k++)
00824                 if(ff3.flavour==fAny || ff3.flavour==k){
00825                     f3.flavour=(Fflavour)k;
00826                 for(uint l=fElectron;l<=fTau;l++)
00827                     if(ff4.flavour==fAny || ff4.flavour==l){
00828                         f4.flavour=(Fflavour)l;
00829                         ex v1=0, v2=0;
00830                         ex mq1=mixes.mass(f1),mq2=mixes.mass(f2),mq3=mixes.mass(f3),mq4=mixes.mass(f4);
00831                         ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;
00832                         ex s4=m2q1+m2q2+m2q3+m2q4-s2-s3;
00833
00834                         scalar_products sp;
00835                         sp.add(q2, q3, (s4-m2q2-m2q3)/2);
00836                         sp.add(q4, q3, (s2-m2q4-m2q3)/2);
00837                         sp.add(q2, q4, (s3-m2q2-m2q4)/2);
00838
00839                         sp.add(q2, q2, m2q2);
00840                         sp.add(q3, q3, m2q3);
00841                         sp.add(q4, q4, m2q4);
00842
00843                         ex q10=(s2+mq1*mq1-mq2*mq2)/(2*sqrt(s2)), lq1l=sqrt(q10*q10-mq1*mq1);
00844                         ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);
00845
00846                         for(uint i=0;i<bosons.size();i++)if(bosons[i].s==s || s==sAny){
00847                             if(bosons[i].s==0){
00848                                 ex a=- (bosons[i].couplingR(f2,f1)-bosons[i].couplingL(f2,f1))*
    bosons[i].couplingdaggerL(f3,f4)*s2/(mq1+mq2)/pow(bosons[i].mass,2);
00849                                 v1=v1+a*dirac_gammaL();
00850                                 v2=v2+a.conjugate()*dirac_gammaR();
00851                                 a=- (bosons[i].couplingR(f2,f1)-bosons[i].couplingL(f2,f1))*
    bosons[i].couplingdaggerR(f3,f4)*s2/(mq1+mq2)/pow(bosons[i].mass,2);
00852                                 v1=v1+a*dirac_gammaR();
00853                                 v2=v2+a.conjugate()*dirac_gammaL();
00854                             }
00855                             else{
00856                                 ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
00857                                 ex a=(bosons[i].couplingR(f2,f1)-bosons[i].couplingL(f2,f1))*
    bosons[i].couplingdaggerL(f3,f4)/pow(bosons[i].mass,2);
00858                                 v1=v1+a*sl*dirac_gammaL();
00859                                 v2=v2+a.conjugate()*sl*dirac_gammaL();
00860                                 a=(bosons[i].couplingR(f2,f1)-bosons[i].couplingL(f2,f1))*
    bosons[i].couplingdaggerR(f3,f4)/pow(bosons[i].mass,2);
00861                                 v1=v1+a*sl*dirac_gammaR();
00862                                 v2=v2+a.conjugate()*sl*dirac_gammaR();
00863                             }
00864                         }
00865                         ex vq3=dirac_slash(q3,4)-mq3*dirac_ONE(), vq4=dirac_slash(q4,4)+mq4*dirac_ONE();
00866                         ex dt=dirac_trace(vq3*v1*vq4*v2).simplify_indexed(sp);
00867                         cout<<"dt: " <<dt<<endl;
00868                         ex result=expand(dt*4*lq3l/s2/Pi/128);
00869
00870                         ret+=result;
00871                     }
00872                 }
00873
00874         return collect_common_factors(ret.subs(conjtoabs));
00875         //return expand(ret.subs(lst(exp(-I*wild())==1/exp(I*
    wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
00876     }
00877     */
00878
00879     ex decaywidth(const Fermion& ff1, const Fermion& ff2, const
    Fermion& ff3, const Fermion& ff4, BSpin s=sAny) const{
00880         multivector<ex,4> a(0,bosons.size(),2,2,2);
00881         vector<ex> mass(bosons.size(),0);
00882         vector<int> op(bosons.size(),0);
00883         ex ret=0;
00884         Fermion f1=ff1, f2=ff2,f3=ff3, f4=ff4;

```

```

00885
00886
00887     for(uint i=fElectron;i<=fTau;i=i+1)
00888     if(ff1.flavour==fAny || ff1.flavour==i){
00889         f1.flavour=(FFlavour)i;
00890     for(uint j=fElectron;j<=fTau;j++)
00891     if(ff2.flavour==fAny || ff2.flavour==j){
00892         f2.flavour=(FFlavour)j;
00893     for(uint k=fElectron;k<=fTau;k++)
00894     if(ff3.flavour==fAny || ff3.flavour==k){
00895         f3.flavour=(FFlavour)k;
00896     for(uint l=fElectron;l<=fTau;l++)
00897     if(ff4.flavour==fAny || ff4.flavour==l){
00898         f4.flavour=(FFlavour)l;
00899     for(uint i=0;i<bosons.size();i++) if(bosons[i].s==s || s==sAny){
00900         op[i]=bosons[i].s;
00901         mass[i]=bosons[i].mass;
00902         a[i][0][0][0]=bosons[i].couplingdaggerL(f2,f1)*bosons[i].couplingL(f3,f4);
00903         a[i][0][0][1]=bosons[i].couplingdaggerL(f2,f1)*bosons[i].couplingR(f3,f4);
00904         a[i][0][1][0]=bosons[i].couplingdaggerR(f2,f1)*bosons[i].couplingL(f3,f4);
00905         a[i][0][1][1]=bosons[i].couplingdaggerR(f2,f1)*bosons[i].couplingR(f3,f4);
00906
00907         a[i][1][0][0]=bosons[i].couplingdaggerL(f3,f1)*bosons[i].couplingL(f2,f4);
00908         a[i][1][0][1]=bosons[i].couplingdaggerL(f3,f1)*bosons[i].couplingR(f2,f4);
00909         a[i][1][1][0]=bosons[i].couplingdaggerR(f3,f1)*bosons[i].couplingL(f2,f4);
00910         a[i][1][1][1]=bosons[i].couplingdaggerR(f3,f1)*bosons[i].couplingR(f2,f4);
00911     }
00912
00913     ret+=wc.get_integral_symb(a,mass,op,mixes.mass(f1));
00914     //
00915     ret+=wc.get_integral(a,mass,op,mixes.massnum(f1),mixes.massnum(f2),mixes.massnum(f3),mixes.massnum(f4))/pow(mixes.massnum(f1),mixes.massnum(f2),mixes.massnum(f3),mixes.massnum(f4));
00916     if(ff2.flavour==ff4.flavour) ret=ret/2;
00917     return collect_common_factors(ret.subs(conjtoabs));
00918     //return
00919     expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
00920 }
00921 ex get_integral_symb(const multivector<ex,3>& a, ex m1) const{
00922     realsymbol s2("s2"), s3("s3");
00923     realsymbol q1("q1"), q2("q2"), q3("q3"), q4("q4");
00924
00925     ex m2q1=m1*m1;
00926
00927     ex vq1=dirac_slash(q2,4)+dirac_slash(q3,4)+dirac_slash(q4,4)+m1*dirac_ONE(), vq2=dirac_slash(q2,4);
00928     ex vq3=dirac_slash(q3,4), vq4=dirac_slash(q4,4);
00929
00930     ex s4=m2q1-s2-s3;
00931     scalar_products sp;
00932     sp.add(q2, q3, (s4)/2);
00933     sp.add(q4, q3, (s2)/2);
00934     sp.add(q2, q4, (s3)/2);
00935
00936     sp.add(q2, q2, 0);
00937     sp.add(q3, q3, 0);
00938     sp.add(q4, q4, 0);
00939
00940     multivector<ex,2> v(0,2,2);
00941     v[0][0]=dirac_gammaL(); v[0][1]=dirac_gammaR();
00942     v[1][0]=dirac_gammaR(); v[1][1]=dirac_gammaL();
00943
00944     multivector<ex,5> traces(0,2,2,2,2);
00945     for(uint k=0;k<2;k++)
00946     for(uint l=0;l<2;l++)
00947     for(uint m=0;m<2;m++)
00948     for(uint n=0;n<2;n++){
00949         ex vk=v[k][0];
00950         ex vm=v[m][0];
00951         ex vl=v[l][1];
00952         ex vn=v[n][1];
00953
00954         traces[k][l][m][n][0]=dirac_trace(vq2*vk*vq1*vl)*dirac_trace(vq3*vm*vq4*vn);
00955     }
00956     traces[k][l][m][n][1]=-dirac_trace(vq2*vk*vq1*vl*vq3*vm*vq4*vn);
00957
00958     for(uint k=0;k<2;k++)
00959     for(uint l=0;l<2;l++)
00960     for(uint m=0;m<2;m++)
00961     for(uint n=0;n<2;n++)
00962     for(uint o=0;o<2;o++){
00963         traces[k][l][m][n][o]=(traces[k][l][m][n][o]).
00964     simplify_indexed(sp);
00965     }
00966
00967     ex q10=(s2+m1*m1)/(2*sqrt(s2)), lq1l=(m1*m1-s2)/(2*sqrt(s2));

```

```

00968     ex q30=sqrt(s2)/2, lq3l=q30;
00969     ex q20=(m1*m1-s2)/(2*m1), lq2l=q20;
00970
00971     ex total=0;
00972     for(uint k=0;k<2;k++){
00973         for(uint l=0;l<2;l++){
00974             for(uint m=0;m<2;m++){
00975                 for(uint n=0;n<2;n++){
00976                     for(uint r=0;r<2;r++){
00977                         for(uint s=0;s<2;s++){
00978                             ex coup=a[r][k][m]*a[s][l][n].conjugate();
00979                             ex integrand=traces[k][l][m][n][(r+s)%2];
00980                             integrand=expand(integral(s3, 0, m1*m1-s2, integrand).eval_integ())/lq1l/sqrt(s2)*lq2l/m1/m1
00981                         );
00982                         //double mm2=0, mm3=0, m4=0;
00983                         ex result=integral(s2,0,m1*m1,integrand).eval_integ()/pow(Pi,3)/512;
00984                         ex partial=result*coup;
00985                         total=total+partial;
00986                     }
00987                 }
00988             }
00989         }
00990     }
00991     ex decaywidthtest2(const Fermion& ff1) const{
00992         multivector<ex,3> a(0,2,2);
00993         symbol gLL("g_{LL}"), gLR("g_{LR}"), gRL("g_{RL}"), gRR("g_{RR}"), cLL("c_{LL}"), cLR("c_{LR}"), cRL("c_{RL}"), cRR("c_{RR}");
00994         a[0][0][0]=gLL;
00995         a[0][0][1]=gLR;
00996         a[0][1][0]=gRL;
00997         a[0][1][1]=gRR;
00998         a[1][0][0]=cLL;
00999         a[1][0][1]=cLR;
01000         a[1][1][0]=cRL;
01001         a[1][1][1]=cRR;
01002
01003         ex ret=get_integral_symb(a,mixes.mass(ff1));
01004         //
01005         ret+=wc.get_integral(a,mass,op,mixes.massnum(f1),mixes.massnum(f2),mixes.massnum(f3),mixes.massnum(f4))/pow(mixes.massnum(f1),2);
01006         return collect_common_factors(ret.subs(conjtoabs));
01007         //return
01008     }
01009     expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
01010 }
01011 /*
01012 ex decaywidthtest(const Fermion& f1, const Fermion& f2, const Fermion& ff3, const Fermion& ff4, BSpin
01013 s=sAny) const{
01014     Fermion f1=ff1, f2=ff2,f3=ff3, f4=ff4;
01015     ex ret=0;
01016     realsymbol q1("q1"), q2("q2"), q3("q3"), q4("q4");
01017     symbol gL("gL"), gR("gR");
01018     ex s2("s2"), s3("s3");
01019     for(uint i=fElectron;i<=fTau;i++){
01020         if(ff1.flavour==fAny || ff1.flavour==i){
01021             f1.flavour=(FFlavour)i;
01022         }
01023         for(uint j=fElectron;j<=fTau;j++){
01024             if(ff2.flavour==fAny || ff2.flavour==j){
01025                 f2.flavour=(FFlavour)j;
01026             }
01027         }
01028         for(uint k=fElectron;k<=fTau;k++){
01029             if(ff3.flavour==fAny || ff3.flavour==k){
01030                 f3.flavour=(FFlavour)k;
01031             }
01032         }
01033         for(uint l=fElectron;l<=fTau;l++){
01034             if(ff4.flavour==fAny || ff4.flavour==l){
01035                 f4.flavour=(FFlavour)l;
01036             }
01037         }
01038         ex v1=0, v2=0;
01039         ex mq1=mixes.mass(f1),mq2=mixes.mass(f2),mq3=mixes.mass(f3),mq4=mixes.mass(f4);
01040         ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;
01041         ex s4=m2q1+m2q2+m2q3+m2q4-s2-s3;
01042         scalar_products sp;
01043         sp.add(q4, q3, (s2-m2q4-m2q3)/2);
01044         sp.add(q3, q3, m2q3);
01045         sp.add(q4, q4, m2q4);
01046         ex q10=(s2+mq1*mq1-mq2*mq2)/(2*sqrt(s2)), lq1l=sqrt(q10*q10-mq1*mq1);
01047         ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);
01048         ex vq1=dirac_slash(q2,4)+dirac_slash(q3,4)+dirac_slash(q4,4)+mq1*dirac_ONE(),
01049             vq2=dirac_slash(q2,4)+mq2*dirac_ONE(),
01050             vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)-mq4*dirac_ONE();

```

```

01049
01050         ex a;
01051
01052         a=gL;
01053         v1=v1+a*dirac_gammaL();
01054         v2=v2+a.conjugate()*dirac_gammaR();
01055         a=gR;
01056         v1=v1+a*dirac_gammaL();
01057         v2=v2+a.conjugate()*dirac_gammaL();
01058
01059         / * }
01060     else{
01061         ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
01062         ex a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*
bosons[i].couplingL(f3,f4)/pow(bosons[i].mass,2);
01063         v1=v1+a*sl*dirac_gammaL();
01064         v2=v2+a.conjugate()*sl*dirac_gammaL();
01065         a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*
bosons[i].couplingR(f3,f4)/pow(bosons[i].mass,2);
01066         v1=v1+a*sl*dirac_gammaR();
01067         v2=v2+a.conjugate()*sl*dirac_gammaR();
01068     } * /
01069
01070     ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)-mq4*dirac_ONE();
01071     ex dt=dirac_trace(vq3*v1+vq4*v2).simplify_indexed(sp);
01072     ex result=expand(dt*4*lq3l/s2/Pi/128);
01073
01074     ret+=result;
01075 }
01076 }
01077 lst ltest;
01078 ltest.append(conjugate(gL)==pow(abs(gL),2)/gL);
01079 ltest.append(conjugate(gR)==pow(abs(gR),2)/gR);
01080 return pow(meson.decay_factor,2)*collect_common_factors(ret.subs(conjtoabs).subs(ltest));
01081 //return expand(ret.subs(lst(exp(-I*wild())==1/exp(I*
wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
01082 }
01083 */
01084
01085 ex gRR2(const Fermion& f1, const Fermion& f3) const{
01086
01087     ex ret1=0,ret2=0;
01088     Fermion f2(tLepton,iUp);
01089     Fermion f4(tLepton,iUp);
01090
01091     for(uint k=fElectron;k<=fTau;k++){
01092         f2.flavour=(FFlavour)k;
01093         for(uint l=fElectron;l<=fTau;l++){
01094             f4.flavour=(FFlavour)l;
01095             for(uint i=0;i<bosons.size();i++){
01096                 if(bosons[i].s==sScalar){
01097                     ex x=bosons[i].couplingdaggerR(f2,f1)*bosons[i].couplingL(f3,f4)/pow(bosons[i].
mass,2);
01098                     ret1+=x*x.conjugate();
01099                 }
01100                 else if(bosons[i].s==sVector){
01101                     ex x=bosons[i].couplingdaggerL(f2,f1)*bosons[i].couplingL(f3,f4)/pow(bosons[i].
mass,2);
01102                     ret2+=x*x.conjugate();
01103                 }
01104             }
01105             //r2.append();
01106             ret2=ret2.subs(conjtoabs);
01107             ret1=ret1.subs(conjtoabs);
01108             for(uint i=0;i<3;i++){
01109                 ret1=collect_common_factors(expand(ret1.subs(pow(abs(mixes.V[0][2][i]),2)==1-pow(abs(mixes.
V[0][1][i]),2)-pow(abs(mixes.V[0][0][i]),2))));
01110                 ret2=collect_common_factors(expand(ret2.subs(pow(abs(mixes.V[0][2][i]),2)==1-pow(abs(mixes.
V[0][1][i]),2)-pow(abs(mixes.V[0][0][i]),2))));
01111             }
01112
01113             cout<<ret2<<endl;
01114             return collect_common_factors(ret1/ret2);
01115         }
01116     }
01117 ex tautomu_tautoe() const{
01118
01119     ex ret1=0,ret2=0, rety1=0, rety2=0;
01120
01121     Fermion f1(tLepton,iDown,fTau);
01122     Fermion f3l(tLepton,iDown,fMuon);
01123     Fermion f32(tLepton,iDown,fElectron);
01124
01125     Fermion f2(tLepton,iUp);
01126     Fermion f4(tLepton,iUp);
01127
01128

```

```

01129         for(uint k=fElectron;k<=fTau;k++){
01130             f2.flavour=(FFlavour)k;
01131             for(uint l=fElectron;l<=fTau;l++){
01132                 f4.flavour=(FFlavour)l;
01133                 ex x1=0, x2=0, y1=0, y2=0;
01134                 for(uint i=0;i<bosons.size();i++){
01135                     if(bosons[i].s==sScalar) {
01136                         x1+=bosons[i].couplingdaggerR(f2,f1)*bosons[i].couplingL(f31,f4)/pow(bosons[i].
01137 mass,2);
01138                         x2+=bosons[i].couplingdaggerR(f2,f1)*bosons[i].couplingL(f32,f4)/pow(bosons[i].mass
01139 ,2);
01140                     }
01141                     else if(bosons[i].s==sVector) {
01142                         y1+=bosons[i].couplingdaggerL(f2,f1)*bosons[i].couplingL(f31,f4)/pow(bosons[i].
01143 mass,2);
01144                         y2+=bosons[i].couplingdaggerL(f2,f1)*bosons[i].couplingL(f32,f4)/pow(bosons[i].mass
01145 ,2);
01146                     }
01147                     ret1+=(x1*y1.conjugate()).real_part();
01148                     ret2+=(x2*y2.conjugate()).real_part();
01149                     rety1+=y1*y1.conjugate();
01150                     rety2+=y2*y2.conjugate();
01151                 }}
01152                 ret2=(ret2/rety2*mixes.mass(f32)/mixes.mass(f1)).subs(conjtoabs);
01153                 ret1=(ret1/rety1*mixes.mass(f31)/mixes.mass(f1)).subs(conjtoabs);
01154                 for(uint i=0;i<3;i++){
01155                     ret1=collect_common_factors(expand(ret1.subs(pow(abs(mixes.V[0][2][i]),2)==1-pow(abs(mixes.
01156 V[0][1][i]),2)-pow(abs(mixes.V[0][0][i]),2)))));
01157                     ret2=collect_common_factors(expand(ret2.subs(pow(abs(mixes.V[0][2][i]),2)==1-pow(abs(mixes.
01158 V[0][1][i]),2)-pow(abs(mixes.V[0][0][i]),2)))));
01159                 }
01160                 ex x=pow(mixes.mass(f31)/mixes.mass(f1),2);
01161                 ex F1=1-8*x+8*pow(x,3)-pow(x,4)-12*pow(x,2)*log(x);
01162                 ex g1=1+9*x-9*pow(x,2)-pow(x,3)+6*x*(1+x)*log(x);
01163                 ex N1=1+gRR2(f1,f31)/4;
01164                 x=pow(mixes.mass(f32)/mixes.mass(f1),2);
01165                 ex F2=1-8*x+8*pow(x,3)-pow(x,4)-12*pow(x,2)*log(x);
01166                 ex g2=1+9*x-9*pow(x,2)-pow(x,3)+6*x*(1+x)*log(x);
01167                 ex N2=1+gRR2(f1,f32)/4;
01168                 return collect_common_factors(N1*(F1+2/N1*ret1*g1)/N2/(F2+2/N2*ret2*g2)*F2/F1);
01169             }
01170 ex mesondw(const Meson & meson, const Fermion& ff3, const
01171 Fermion& ff4, BSpin s=sAny) const{
01172     const Fermion& f1(meson.q1), f2(meson.q2);
01173     ex mesonmass=meson.mass;
01174     Fermion f3=ff3, f4=ff4;
01175     ex ret=0;
01176     realsymbol q3("q3"), q4("q4");
01177     ex s2=pow(mesonmass,2);
01178     for(uint k=fElectron;k<=fTau;k++){
01179         if(ff3.flavour==fAny || ff3.flavour==k){
01180             f3.flavour=(FFlavour)k;
01181             for(uint l=fElectron;l<=fTau;l++){
01182                 if(ff4.flavour==fAny || ff4.flavour==l){
01183                     f4.flavour=(FFlavour)l;
01184                     ex v1=0, v2=0;
01185                     ex mq1=mixes.mass(f1),mq2=mixes.mass(f2),mq3=mixes.mass(f3),mq4=mixes.mass(f4);
01186                     ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;
01187                     scalar_products sp;
01188                     sp.add(q4, q3, (s2-m2q4-m2q3)/2);
01189                     sp.add(q3, q3, m2q3);
01190                     sp.add(q4, q4, m2q4);
01191                     ex q10=(s2+mq1*mq1-mq2*mq2)/(2*sqrt(s2)), lq11=sqrt(q10*q10-mq1*mq1);
01192                     ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq31=sqrt(q30*q30-mq3*mq3);
01193                     for(uint i=0;i<bosons.size();i++) if(bosons[i].s==s || s==sAny){
01194                         if(bosons[i].s==0){
01195                             ex a=-(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*bosons[i].
01196 couplingL(f3,f4)*s2/(mq1+mq2)/pow(bosons[i].mass,2);
01197                             v1=v1+a*dirac_gammaL();
01198                             v2=v2+a.conjugate()*dirac_gammaR();
01199                             a=-(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*bosons[i].
01200 couplingR(f3,f4)*s2/(mq1+mq2)/pow(bosons[i].mass,2);
01201                             v1=v1+a*dirac_gammaL();
01202                             v2=v2+a.conjugate()*dirac_gammaL();
01203                         }
01204                     }
01205                 }
01206             }

```

```

01207         else{
01208             ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
01209             ex a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*bosons[i].
couplingL(f3,f4)/pow(bosons[i].mass,2);
01210             v1=v1+a*sl*dirac_gammaL();
01211             v2=v2+a.conjugate()*sl*dirac_gammaL();
01212             a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*bosons[i].
couplingR(f3,f4)/pow(bosons[i].mass,2);
01213             v1=v1+a*sl*dirac_gammaR();
01214             v2=v2+a.conjugate()*sl*dirac_gammaR();
01215         }
01216     }
01217     ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)-mq4*dirac_ONE();
01218     ex dt=dirac_trace(vq3*v1*vq4*v2).simplify_indexed(sp);
01219     ex result=expand(dt*4*1q3l/s2/Pi/128);
01220
01221     ret+=result;
01222 }
01223 }
01224
01225     return pow(meson.decay_factor,2)*collect_common_factors(ret.subs(conjtoabs));
01226     //return
expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
01227 }
01228
01229
01230 ex mesondwtest(const Meson & meson, const Fermion& ff3, const
Fermion& ff4, BSpin s=sAny) const{
01231
01232     const Fermion& f1(meson.q1), f2(meson.q2);
01233     ex mesonmass=meson.mass;
01234
01235     Fermion f3=ff3, f4=ff4;
01236
01237     ex ret=0;
01238
01239     realsymbol q3("q3"), q4("q4");
01240     symbol gL("gL"), gR("gR"), gVL("gVL"), gVR("gVR");
01241     symbol gS("gS"), gP("gP"), gA("gA");
01242
01243     ex s2=pow(mesonmass,2);
01244
01245     for(uint k=fElectron;k<=fTau;k++){
01246         if(ff3.flavour==fAny || ff3.flavour==k){
01247             f3.flavour=(FFlavour)k;
01248             for(uint l=fElectron;l<=fTau;l++){
01249                 if(ff4.flavour==fAny || ff4.flavour==l){
01250                     f4.flavour=(FFlavour)l;
01251                     ex v1=0, v2=0;
01252                     ex mq1=mixes.mass(f1),mq2=mixes.mass(f2),mq3=mixes.mass(f3),mq4=mixes.mass(f4);
01253                     ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;
01254                     scalar_products sp;
01255                     sp.add(q4, q3, (s2-m2q4-m2q3)/2);
01256                     sp.add(q3, q3, m2q3);
01257                     sp.add(q4, q4, m2q4);
01258                     ex q10=(s2+mq1*mq1-mq2*mq2)/(2*sqrt(s2)), lq1l=sqrt(q10*q10-mq1*mq1);
01259                     ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);
01260
01261                     ex a;
01262                     /* a=-gL*s2/(mq1+mq2);
01263                     v1=v1+a*dirac_gammaL();
01264                     v2=v2+a.conjugate()*dirac_gammaR();
01265                     a=-gR*s2/(mq1+mq2);
01266                     v1=v1+a*dirac_gammaR();
01267                     v2=v2+a.conjugate()*dirac_gammaL();
01268
01269                     ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
01270                     a=gA;
01271                     v1=v1+a*sl*dirac_gamma5();
01272                     v2=v2+a.conjugate()*sl*dirac_gamma5();
01273                     */
01274                     a=-gS*s2/(mq1+mq2);
01275                     v1=v1+a*dirac_ONE();
01276                     v2=v2+a.conjugate()*dirac_ONE();
01277                     a=-gP*s2/(mq1+mq2);
01278                     v1=v1+a*dirac_gamma5();
01279                     v2=v2-a.conjugate()*dirac_gamma5();
01280                     ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
01281                     // a=gA;
01282                     // v1=v1+a*sl*dirac_gamma5();
01283                     // v2=v2+a.conjugate()*sl*dirac_gamma5();
01284
01285                     /*}
01286                     else{
01287                         ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
01288                         ex a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*
bosons[i].couplingL(f3,f4)/pow(bosons[i].mass,2);

```

```

01289         v1=v1+a*s1*dirac_gammaL();
01290         v2=v2+a.conjugate()*s1*dirac_gammaL();
01291         a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*
bosons[i].couplingR(f3,f4)/pow(bosons[i].mass,2);
01292         v1=v1+a*s1*dirac_gammaL();
01293         v2=v2+a.conjugate()*s1*dirac_gammaR();
01294     }*/
01295
01296     ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)-mq4*dirac_ONE();
01297     ex dt=dirac_trace(vq3*v1+vq4*v2).simplify_indexed(sp);
01298     ex result=expand(dt*4*lq3l/s2/Pi/128);
01299
01300     ret+=result;
01301 }
01302 }
01303 lst ltest;
01304 ltest.append(conjugate(gL)==pow(abs(gL),2)/gL);
01305 ltest.append(conjugate(gR)==pow(abs(gR),2)/gR);
01306 ltest.append(conjugate(gS)==pow(abs(gS),2)/gS);
01307 ltest.append(conjugate(gP)==pow(abs(gP),2)/gP);
01308 ltest.append(conjugate(gA)==pow(abs(gA),2)/gA);
01309
01310     return pow(meson.decay_factor,2)*collect_common_factors(expand(ret.subs(conjtoabs).subs
(ltest)));
01311 //return
expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
01312 }
01313
01314 ex fermionmeson(const Fermion& ff4, const Fermion& ff3, const
Meson & meson, BSpin s=sAny) const{
01315
01316     const Fermion& f1(meson.q1), f2(meson.q2);
01317     ex mesonmass=meson.mass;
01318
01319     Fermion f3=ff3, f4=ff4;
01320
01321     ex ret=0;
01322
01323     realsymbol q3("q3"), q4("q4");
01324     ex s2=pow(mesonmass,2);
01325
01326     for(uint k=fElectron;k<=fTau;k++){
01327         if(ff3.flavour==fAny || ff3.flavour==k){
01328             f3.flavour=(FFlavour)k;
01329             for(uint l=fElectron;l<=fTau;l++){
01330                 if(ff4.flavour==fAny || ff4.flavour==l){
01331                     f4.flavour=(FFlavour)l;
01332                     ex v1=0, v2=0;
01333                     ex mq1=mixes.mass(f1),mq2=mixes.mass(f2),mq3=mixes.mass(f3),mq4=mixes.mass(f4);
01334                     ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;
01335                     scalar_products sp;
01336                     sp.add(q4, q3, -(s2-m2q4-m2q3)/2);
01337                     sp.add(q3, q3, m2q3);
01338                     sp.add(q4, q4, m2q4);
01339                     //ex qm0=(s2-mq3*mq3+mq4*mq4)/(2*mq4), lqml=sqrt(qm0*qm0-s2);
01340                     ex q30=(-s2+mq3*mq3+mq4*mq4)/(2*mq4), lq3l=sqrt(q30*q30-mq3*mq3);
01341                     //ex q30=-s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);
01342
01343                     for(uint i=0;i<bosons.size();i++){if(bosons[i].s==s || s==sAny){
01344                         if(bosons[i].s==0){
01345                             ex a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*bosons[i].
couplingL(f3,f4)*s2/(mq1+mq2)/pow(bosons[i].mass,2);
01346                             v1=v1+a*dirac_gammaL();
01347                             v2=v2+a.conjugate()*dirac_gammaR();
01348                             a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*bosons[i].
couplingR(f3,f4)*s2/(mq1+mq2)/pow(bosons[i].mass,2);
01349                             v1=v1+a*dirac_gammaR();
01350                             v2=v2+a.conjugate()*dirac_gammaL();
01351                         }
01352                         else{
01353                             ex s1=(dirac_slash(q3,4)+dirac_slash(q4,4));
01354                             ex a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*bosons[i].
couplingL(f3,f4)/pow(bosons[i].mass,2);
01355                             v1=v1+a*s1*dirac_gammaL();
01356                             v2=v2+a.conjugate()*s1*dirac_gammaL();
01357                             a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*bosons[i].
couplingR(f3,f4)/pow(bosons[i].mass,2);
01358                             v1=v1+a*s1*dirac_gammaR();
01359                             v2=v2+a.conjugate()*s1*dirac_gammaR();
01360                         }
01361                     }
01362                     ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)+mq4*dirac_ONE();
01363                     ex dt=dirac_trace(vq3*v1+vq4*v2).simplify_indexed(sp);
01364                     ex result=expand(dt*2*lq3l/mq4/mq4/Pi/128);
01365
01366                     ret+=result;
01367                 }

```

```

01368     }
01369
01370
01371
01372     return pow(meson.decay_factor,2)*collect_common_factors(ret.subs(conjtoabs));
01373     //return
    expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()), sin(wild())==sqrt(1-pow(cos(wild()),2)))));
01374 }
01375
01376 ex fermiontomesontest(const Fermion& ff4, const Fermion& ff3, const
    Meson & meson, BSpin s=sAny) const{
01377
01378     const Fermion& f1(meson.q1), f2(meson.q2);
01379     ex mesonmass=meson.mass;
01380
01381     Fermion f3=ff3, f4=ff4;
01382
01383     ex ret=0;
01384
01385     realsymbol q3("q3"), q4("q4");
01386
01387     symbol sL("sL"), sR("sR"), vL("vL"), vR("vR");
01388     ex s2=pow(mesonmass,2);
01389
01390     for(uint k=fElectron;k<=fTau;k++){
01391         if(ff3.flavour==fAny || ff3.flavour==k){
01392             f3.flavour=(FFlavour)k;
01393         }
01394         for(uint l=fElectron;l<=fTau;l++){
01395             if(ff4.flavour==fAny || ff4.flavour==l){
01396                 f4.flavour=(FFlavour)l;
01397                 ex v1=0, v2=0;
01398                 ex mq1=mixes.mass(f1),mq2=mixes.mass(f2),mq3=mixes.mass(f3),mq4=mixes.mass(f4);
01399                 ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;
01400                 scalar_products sp;
01401                 sp.add(q4, q3, -(s2-m2q4-m2q3)/2);
01402                 sp.add(q3, q3, m2q3);
01403                 sp.add(q4, q4, m2q4);
01404                 //ex qm0=(s2-mq3*mq3+mq4*mq4)/(2*mq4), lqm1=sqrt(qm0*qm0-s2);
01405                 ex q30=(-s2+mq3*mq3+mq4*mq4)/(2*mq4), lq3l=sqrt(q30*q30-mq3*mq3);
01406                 //ex q30=-(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);
01407
01408                 ex a=sL;
01409                 v1=v1+a*dirac_gammaL();
01410                 v2=v2+a.conjugate()*dirac_gammaR();
01411                 a=sR;
01412                 v1=v1+a*dirac_gammaR();
01413                 v2=v2+a.conjugate()*dirac_gammaL();
01414
01415                 ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
01416                 a=vL;
01417                 v1=v1+a*sl*dirac_gammaL();
01418                 v2=v2+a.conjugate()*sl*dirac_gammaR();
01419                 a=vR;
01420                 v1=v1+a*sl*dirac_gammaR();
01421                 v2=v2+a.conjugate()*sl*dirac_gammaL();
01422
01423                 ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)+mq4*dirac_ONE();
01424                 ex dt=dirac_trace(vq3*v1+vq4*v2).simplify_indexed(sp);
01425                 ex result=expand(dt*2*lq3l/mq4/mq4/Pi/128);
01426
01427                 ret+=result;
01428             }
01429         }
01430     }
01431     return pow(meson.decay_factor,2)*collect_common_factors(ret.subs(conjtoabs));
01432     //return
    expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()), sin(wild())==sqrt(1-pow(cos(wild()),2)))));
01433 }
01434
01435 ex mesonmixing(ex mesonmass, const Fermion& f1, const Fermion& f2) const{
01436
01437     ex ret=0;
01438
01439     ex v1=0, v2=0;
01440     ex mq1=mixes.mass(f1),mq2=mixes.mass(f2);
01441     ex m2q1=mq1*mq1, m2q2=mq2*mq2;
01442
01443     for(uint i=0;i<bosons.size();i++){
01444         if(bosons[i].s==0){
01445             ex a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1));
01446             v1=v1+pow(a/bosons[i].mass,2);
01447
01448             ex b=(bosons[i].couplingdaggerR(f2,f1)+bosons[i].couplingdaggerL(f2,f1));
01449             v2=v2+pow(b/bosons[i].mass,2);
01450         }
01451     }

```



```

01452         ex fc=mesonmass/(mixes.massnum(f1)+mixes.massnum(f2));
01453         fc=pow(fc,2);
01454
01455         ret=2*(-v1*(1+l1*fc)+v2*(1+fc))*mesonmass/96;
01456
01457         return collect_common_factors(ret.subs(conjtoabs));
01458         //return
01459         expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
01460     }
01461     ex CHdecaycoupling(Boson higgs, const Fermion& ff3, const
01462     Fermion& ff4) const{
01463         Fermion f3=ff3, f4=ff4;
01464         ex ret=0;
01465         for(uint k=fElectron;k<=fTau;k++){
01466             if(ff3.flavour==fAny || ff3.flavour==k){
01467                 f3.flavour=(FFlavour)k;
01468             for(uint l=fElectron;l<=fTau;l++){
01469                 if(ff4.flavour==fAny || ff4.flavour==l){
01470                     f4.flavour=(FFlavour)l;
01471                     ret+=higgs.couplingdaggerL(f3,f4)*higgs.
01472                     couplingdaggerL(f3,f4).conjugate()+higgs.couplingdaggerR(f3,f4)*higgs.
01473                     couplingdaggerR(f3,f4).conjugate();
01474                 }}
01475             return collect_common_factors(ret.subs(conjtoabs));
01476         }
01477         double BranchingRatio(double * xx,double * p){
01478             return ex_to<numeric>(BR_Htotaunu.subs(tanb==pow(10.0,xx[0])).evalf()).to_double();
01479         }
01480
01481         double topBranchingRatio(double * xx,double * p){
01482             return ex_to<numeric>(BR_toptoHq.subs(lst(tanb==pow(10.0,xx[0]),McH==xx[1])).evalf()).to_double();
01483         }
01484     }
01485
01486     widthcalc wc;
01487
01488     const double planck;
01489     const possymbol GF, MZ, MW, Mh;
01490     const constant Mpip, Mpi0, MBp,MB0,MBs0, MKp,MK0,MDp,MD0,MDsp,MDs0;
01491     const constant Fpi, FB,FBs, FK,FD,FDs;
01492     ex cos2, g, alpha;
01493     const possymbol tanb, cp, McH, MR, MI, rho;
01494     possymbol Mu[3],Md[3];
01495     vector< Boson > bosons;
01496
01497     lst replacements;
01498     ex Btaunu;
01499     ex BR_Htotaunu;
01500     ex BR_toptoHq;
01501     ex BtotaunuR;
01502     ex BtoDtaunuR;
01503     ex BtoD2taunuR;
01504
01505     const Mixes mixes;
01506     lst conjtoabs;
01507     realsymbol mu;
01508
01509     int iBtaunu, iBDtaunu, iBD2taunu;
01510     vector<int> BGLtype;
01511
01512     double mmmmax, stepsize;
01513
01514     calculBmumu * cBmumu;
01515     calculBmumu * cBsmumu;
01516
01517
01518     };
01519
01520
01521     }
01522     #endif

```

## 8.3 defs.h File Reference

### Macros

- #define [GL 2](#)

- `#define GQ 2`
- `#define UL 0`
- `#define UQ 0`

### 8.3.1 Macro Definition Documentation

#### 8.3.1.1 `#define GL 2`

Definition at line 1 of file [defs.h](#).

#### 8.3.1.2 `#define GQ 2`

Definition at line 2 of file [defs.h](#).

#### 8.3.1.3 `#define UL 0`

Definition at line 3 of file [defs.h](#).

#### 8.3.1.4 `#define UQ 0`

Definition at line 4 of file [defs.h](#).

## 8.4 `defs.h`

```
00001 #define GL 2
00002 #define GQ 2
00003 #define UL 0
00004 #define UQ 0
00005
00006
00007
```

## 8.5 `draw.cpp` File Reference

```
#include "MCMC.h"
#include "BGL.h"
#include "TF2.h"
#include "TProfile3D.h"
#include "THStack.h"
#include "TColor.h"
#include "TROOT.h"
#include "TStyle.h"
#include "TGraph.h"
#include "TLatex.h"
#include "TFile.h"
#include "TH2F.h"
#include "TVector.h"
#include "TCanvas.h"
#include "TMath.h"
#include <iostream>
#include <fstream>
#include <cln/cln.h>
#include <cln/float.h>
```

- class **BGL2**  
*A second implementation of the BGL model, for testing purposes.*

- `int main (int argc, char *argv[])`  
*the main function takes the arguments inputfile gL gQ lup qup which specify the file containing the simulation results for a BGL model and draws the plots for that model*

### 8.5.1.1 int main ( int argc, char \* argv[] )

Definition at line 357 of file draw.cpp.

```

00357
00358 // Check the number of parameters
00359
00360 if(argc<6){
00361     std::cerr<<"Usage: "<<argv[0]<<" inputfile gL gQ lup qup"<<std::endl;
00362     return 1;}
00363 CD cmu=conj(Vud[1][0])*Vud[1][1];
00364 CD umu=conj(Vud[0][0])*Vud[0][1];
00365
00366
00367 cout<<"RATIO "<<cmu*cmu<<endl;
00368 cout<<umu*umu<<endl;
00369
00370 int gL=atoi(argv[2]);
00371 int gQ=atoi(argv[3]);
00372 int lup=atoi(argv[4]);
00373 int qup=atoi(argv[5]);
00374 char name[5]="0000";
00375
00376 name[0]+=gL;
00377 name[1]+=gQ;
00378 name[2]+=lup;
00379 name[3]+=qup;
00380 string ll[2][3]={{"#nu_{1}", "#nu_{2}", "#nu_{3}"}, {"e", "#mu", "#tau"}};
00381 string qq[2][3]={{"u", "c", "t"}, {"d", "s", "b"}};
00382 //Int_t MyPalette[100];
00383 Double_t r[] = {1, 0.3};
00384 Double_t g[] = {1, 0.3};
00385 Double_t b[] = {1, 0.3};
00386 Double_t stop[] = {0., 1.0};
00387 TColor::CreateGradientColorTable(2, stop, r, g,b, 100);
00388 //TH1F * pdf1=new TH1F("pdf1","pdf1",npoints,10,500);

```

```

00389 //TGraph * chi2=new TGraph(npoints);
00390
00391 uint npoints=200;
00392 double init1=-3, final1=3;
00393 double init2=10, final2=1000;
00394 double initBsmumu=0, finalBsmumu=3;
00395 double initBsmumu=0, finalBsmumu=6;
00396
00397 double llmax=-1000, McHmax=1000, MRmax=1000, MImax=1000, tbmax=1;
00398
00399 TFile *f=new TFile(argv[1], "read");
00400 if(!f->IsOpen()) cout<<"NOFILE"<<endl;
00401 //f->ShowStreamerInfo();
00402
00403 TH2F *limits4,*Bmumu_Bsmumu,*limits_tb_MR,*limits_tb_MI;
00404 TH2F *limits_MR_MI,*limits_MR_McH,*limits_MI_McH;
00405
00406 f->GetObject("limits4;1",limits4);
00407 f->GetObject("Bmumu_Bsmumu;1",Bmumu_Bsmumu);
00408 f->GetObject("limits_tb_MR;1",limits_tb_MR);
00409 f->GetObject("limits_tb_MI;1",limits_tb_MI);
00410 f->GetObject("limits_MR_MI;1",limits_MR_MI);
00411 f->GetObject("limits_MR_McH;1",limits_MR_McH);
00412 f->GetObject("limits_MI_McH;1",limits_MI_McH);
00413
00414 TVectorD* vllmax=NULL;
00415
00416 f->GetObject("vllmax;1",vllmax);
00417 if(!vllmax) cout<<"ERROR"<<endl;
00418 llmax=(vllmax)[0];
00419 //tbmax=(vllmax)[1];
00420 //McHmax=(vllmax)[2];
00421 //MRmax=McHmax+(vllmax)[3];
00422 //MImax=MRmax+(vllmax)[4];
00423 cout<<llmax<<" "<<tbmax<<" "<<McHmax<<" "<<MRmax<<" "<<MImax<<" "<<endl;
00424
00425 /*BGL2* m=new BGL2(gL,gQ,lup,qup);
00426 double sm_(m->bsgammawidth(tbmax,McHmax,MRmax,MImax,5));
00427 double charged_(m->bsgammawidth(tbmax,McHmax,MRmax,MImax,1));
00428 double neutral_(m->bsgammawidth(tbmax,McHmax,MRmax,MImax,2));
00429 double neutralR_(m->bsgammawidth(tbmax,McHmax,MRmax,MImax,3));
00430 double neutralI_(m->bsgammawidth(tbmax,McHmax,MRmax,MImax,4));
00431 double eK_(m->epsK(tbmax,McHmax,MRmax,MImax));
00432
00433 double all_(m->bsgammawidth(tbmax,McHmax,MRmax,MImax,0));
00434 */
00435 //for(int gL=2;gL>=0;gL--)
00436 //for(int gQ=2;gQ>=0;gQ--)
00437 //for(uint lup=0;lup<2;lup++)
00438 //for(uint qup=0;qup<2;qup++)
00439 uint min1=npoints, min2=npoints, min3=npoints;
00440 uint min11=npoints, min21=npoints, min31=npoints;
00441 uint min12=npoints, min22=npoints, min32=npoints;
00442
00443 for(uint i=0;i<npoints;i++)
00444 for(uint j=0;j<npoints;j++){
00445     int binmax=limits4->GetBin(i+1,j+1);
00446     double rest=limits4->GetBinContent(binmax);
00447     if(rest>=llmax) rest=1;
00448     else rest=TMath::Prob(-2*(rest-llmax),2);
00449     if(rest>=0.05 && j<min1){min1=j;}
00450     if(rest>=0.05 && j<min11 && j>uint(180*npoints/990)){min11=j;}
00451     if(rest>=0.05 && j<min12 && j>uint(400*npoints/990)){min12=j;}
00452     limits4->SetBinContent(i+1,j+1,rest);
00453
00454     rest=Bmumu_Bsmumu->GetBinContent(binmax);
00455     if(rest>=llmax) rest=1;
00456     else rest=TMath::Prob(-2*(rest-llmax),2);
00457     //int nn=4;
00458     //int ii=(i/nn)*nn, jj=(j/nn)*nn;
00459     //for(int iii=ii;iii<ii+n;++iii)
00460     //for(int iii=ii;iii<ii+n;++iii)
00461     Bmumu_Bsmumu->SetBinContent(i+1,j+1,rest);
00462
00463     rest=limits_MR_MI->GetBinContent(binmax);
00464     if(rest>=llmax) rest=1;
00465     else rest=TMath::Prob(-2*(rest-llmax),2);
00466     limits_MR_MI->SetBinContent(i+1,j+1,rest);
00467
00468     rest=limits_MR_McH->GetBinContent(binmax);
00469     if(rest>=llmax) rest=1;
00470     else rest=TMath::Prob(-2*(rest-llmax),2);
00471     if(rest>=0.05 && i<min2){min2=i;}
00472     if(rest>=0.05 && i<min21 && j>uint(180*npoints/990)){min21=i;}
00473     if(rest>=0.05 && i<min22 && j>uint(400*npoints/990)){min22=i;}
00474     limits_MR_McH->SetBinContent(i+1,j+1,rest);
00475

```

```

00476         rest=limits_MI_McH->GetBinContent (binmax);
00477         if(rest>=llmax) rest=1;
00478         else rest=TMath::Prob(-2*(rest-llmax),2);
00479         if(rest>=0.05 && i<min3){min3=i;}
00480         if(rest>=0.05 && i<min31 && j>uint(180*npoints/990)){min31=i;}
00481         if(rest>=0.05 && i<min32 && j>uint(400*npoints/990)){min32=i;}
00482         limits_MI_McH->SetBinContent (i+1,j+1,rest);
00483
00484
00485         rest=limits_tb_MR->GetBinContent (binmax);
00486         if(rest>=llmax) rest=1;
00487         else rest=TMath::Prob(-2*(rest-llmax),2);
00488         limits_tb_MR->SetBinContent (i+1,j+1,rest);
00489     }
00490     double mmin1=init2+((min1+0.5)*(final2-init2))/npoints;
00491     double mmin2=init2+((min2+0.5)*(final2-init2))/npoints;
00492     double mmin3=init2+((min3+0.5)*(final2-init2))/npoints;
00493
00494     double mmin11=init2+((min11+0.5)*(final2-init2))/npoints;
00495     double mmin21=init2+((min21+0.5)*(final2-init2))/npoints;
00496     double mmin31=init2+((min31+0.5)*(final2-init2))/npoints;
00497
00498     double mmin12=init2+((min12+0.5)*(final2-init2))/npoints;
00499     double mmin22=init2+((min22+0.5)*(final2-init2))/npoints;
00500     double mmin32=init2+((min32+0.5)*(final2-init2))/npoints;
00501
00502     //mass<<name<<" "<<mmin1<<" "<<mmin2<<" "<<mmin3<<endl;
00503
00504     ofstream maxs((string("maxs_")+string(name)+string(".out")).c_str());
00505     maxs<<mmin1<<" "<<mmin2<<" "<<mmin3<<endl;
00506     maxs<<mmin11<<" "<<mmin21<<" "<<mmin31<<endl;
00507     maxs<<mmin12<<" "<<mmin22<<" "<<mmin32<<endl;
00508     maxs<<llmax<<" "<<tbmax<<" "<<McHmax<<" "<<MRmax<<" "<<MImax<<" "<<endl;
00509     //maxs<<eK_<<endl;
00510     //maxs<<sm_<<" "<<charged_<<" "<<neutral_<<" "<<neutralR_<<" "<<neutralI_<<" "<<all_<<endl;
00511
00512     //for(uint j=0;j<npoints;j++)
00513     //for(uint i=0;i<npoints;i++){
00514         //      int binmax=limits4->GetBin(i+1,j+1);
00515         //      maxs<<"("<<i<<","<<j<<"):"<<limits4->GetBinContent (binmax)<<endl;
00516         //      }
00517
00518     maxs.close();
00519
00520     double ma=0,me=.2, x0=1,y0=120;
00521     gStyle->SetOptTitle(0);
00522     gStyle->SetPaperSize(10.,10.);
00523     TCanvas * c21=new TCanvas("c21","",int(800*(1+ma+me)),int(600*(1+ma+me)));
00524     c21->SetMargin(me,ma,me,ma);
00525     c21->SetGrid();
00526
00527     limits4->SetStats(0);
00528     limits4->GetXaxis()->SetTitle("log_{10}(tan\\beta)");
00529     limits4->GetYaxis()->SetTitle("M_{H+} (GeV)");
00530
00531     Bmumu_Bsmumu->SetStats(0);
00532     Bmumu_Bsmumu->GetXaxis()->SetTitle("Br(B\\to\\mu\\mu)/10^{-10}");
00533     Bmumu_Bsmumu->GetYaxis()->SetTitle("Br(B_{s}\\to\\mu\\mu)/10^{-9}");
00534
00535     limits_MR_MI->SetStats(0);
00536     limits_MR_MI->GetYaxis()->SetTitle("M_{I} (GeV)");
00537     limits_MR_MI->GetXaxis()->SetTitle("M_{R} (GeV)");
00538
00539
00540     limits_MR_McH->SetStats(0);
00541     limits_MR_McH->GetYaxis()->SetTitle("M_{H+} (GeV)");
00542     limits_MR_McH->GetXaxis()->SetTitle("M_{R} (GeV)");
00543
00544     limits_MI_McH->SetStats(0);
00545     limits_MI_McH->GetYaxis()->SetTitle("M_{H+} (GeV)");
00546     limits_MI_McH->GetXaxis()->SetTitle("M_{I} (GeV)");
00547
00548     limits_tb_MR->SetStats(0);
00549     limits_tb_MR->GetXaxis()->SetTitle("log_{10}(tan\\beta)");
00550     limits_tb_MR->GetYaxis()->SetTitle("M_{R} (GeV)");
00551
00552
00553     Double_t contours[3];
00554     contours[0] = 0.003;
00555     contours[1] = 0.05;
00556     contours[2] = 0.32;
00557
00558
00559
00560
00561
00562     limits4->SetContour(3, contours);

```

```

00563 //limits4->GetYaxis()->SetLabelOffset(0.02);
00564 limits4->GetYaxis()->SetLabelSize(0.08);
00565 limits4->GetYaxis()->SetTitleSize(0.08);
00566 limits4->GetYaxis()->SetTitleOffset(1.2);
00567 limits4->GetYaxis()->SetLimits(1,999);
00568
00569
00570
00571 //limits4->GetXaxis()->SetLabelOffset(0.02);
00572 limits4->GetXaxis()->SetLabelSize(0.08);
00573 limits4->GetXaxis()->SetTitleSize(0.08);
00574 limits4->GetXaxis()->SetTitleOffset(1.2);
00575 limits4->GetXaxis()->SetLimits(-2.99,2.99);
00576
00577
00578
00579 TLatex l;
00580 l.SetTextSize(0.08);
00581 string ss=qq[qup][gQ]+"<math>"+l1[lup][gL];
00582
00583
00584 limits4->Draw("CONT Z LIST");
00585 //limits4->Draw("CONT LIST");
00586 //limits4->Draw("colz");
00587
00588 l.DrawLatex(x0,y0,ss.c_str());
00589
00590 c21->SaveAs((string("pdf_")+string(name)+string(".png")).c_str());
00591
00592 delete c21;
00593 //Bmumu_Bsmumu->SetBit(TH1::kCanRebin);
00594 Bmumu_Bsmumu->Rebin2D(2,2);
00595
00596 //Bmumu_Bsmumu->SetContour(3, contours);
00597 //limits4->GetYaxis()->SetLabelOffset(0.02);
00598 Bmumu_Bsmumu->GetYaxis()->SetLabelSize(0.08);
00599 Bmumu_Bsmumu->GetYaxis()->SetTitleSize(0.08);
00600 Bmumu_Bsmumu->GetYaxis()->SetTitleOffset(0.8);
00601 Bmumu_Bsmumu->GetYaxis()->SetLimits(0.01,4.99);
00602 //Bmumu_Bsmumu->GetYaxis()->SetRangeUser(0.01, 3.49);
00603 // Bmumu_Bsmumu->GetYaxis()->SetLimits(0.01,3.49);
00604 //limits4->GetXaxis()->SetLabelOffset(0.02);
00605 Bmumu_Bsmumu->GetYaxis()->SetNddivisions(5, kTRUE);
00606 Bmumu_Bsmumu->GetXaxis()->SetNddivisions(5, kTRUE);
00607
00608 Bmumu_Bsmumu->GetXaxis()->SetLabelSize(0.08);
00609 Bmumu_Bsmumu->GetXaxis()->SetTitleSize(0.08);
00610 Bmumu_Bsmumu->GetXaxis()->SetTitleOffset(1.2);
00611 Bmumu_Bsmumu->GetXaxis()->SetLimits(0.01,1.99);
00612 //Bmumu_Bsmumu->GetXaxis()->SetRangeUser(0., 2);
00613
00614 TCanvas * cB=new TCanvas("cB","",int(800*(1+ma+me)),int(600*(1+ma+me)));
00615 cB->SetMargin(.14,ma,me,ma);
00616 cB->SetGrid();
00617 //limits4->Draw("CONT Z LIST");
00618 Bmumu_Bsmumu->Draw("COLZ");
00619
00620 l.DrawLatex(1.5,1,ss.c_str());
00621
00622 cB->SaveAs((string("Bmumu_Bsmumu_")+string(name)+string(".png")).c_str());
00623
00624 delete cB;
00625
00626 limits_MR_MI->SetContour(3, contours);
00627 //limits4->GetYaxis()->SetLabelOffset(0.02);
00628 limits_MR_MI->GetYaxis()->SetLabelSize(0.06);
00629 limits_MR_MI->GetYaxis()->SetTitleSize(0.06);
00630 limits_MR_MI->GetYaxis()->SetTitleOffset(1.1);
00631 limits_MR_MI->GetYaxis()->SetLimits(1,999);
00632 //limits4->GetXaxis()->SetLabelOffset(0.02);
00633 limits_MR_MI->GetXaxis()->SetLabelSize(0.06);
00634 limits_MR_MI->GetXaxis()->SetTitleSize(0.06);
00635 limits_MR_MI->GetXaxis()->SetTitleOffset(1.1);
00636 limits_MR_MI->GetXaxis()->SetLimits(1,999);
00637
00638 TCanvas * c3=new TCanvas("c3","",800,600);
00639 c3->SetMargin(me,ma,me,ma);
00640 c3->SetGrid();
00641
00642 limits_MR_MI->Draw("CONT LIST");
00643
00644 c3->SaveAs((string("pdf_")+string(name)+string("_MRMI.png")).c_str());
00645
00646 limits_MR_McH->SetContour(3, contours);
00647 //limits4->GetYaxis()->SetLabelOffset(0.02);
00648 limits_MR_McH->GetYaxis()->SetLabelSize(0.06);
00649 limits_MR_McH->GetYaxis()->SetTitleSize(0.06);

```

```

00650     limits_MR_McH->GetYaxis()->SetTitleOffset(1.1);
00651     limits_MR_McH->GetYaxis()->SetLimits(1,999);
00652     //limits4->GetXaxis()->SetLabelOffset(0.02);
00653     limits_MR_McH->GetXaxis()->SetLabelSize(0.06);
00654     limits_MR_McH->GetXaxis()->SetTitleSize(0.06);
00655     limits_MR_McH->GetXaxis()->SetTitleOffset(1.1);
00656     limits_MR_McH->GetXaxis()->SetLimits(1,999);
00657
00658     TCanvas * c4=new TCanvas("c4","",800,600);
00659     limits_MR_McH->Draw("CONT LIST");
00660     c4->SetMargin(me,ma,me,ma);
00661     c4->SetGrid();
00662     c4->SaveAs((string("pdf_")+string(name)+string("_MRMcH.png")).c_str());
00663
00664     limits_MI_McH->SetContour(3, contours);
00665     //limits4->GetYaxis()->SetLabelOffset(0.02);
00666     limits_MI_McH->GetYaxis()->SetLabelSize(0.06);
00667     limits_MI_McH->GetYaxis()->SetTitleSize(0.06);
00668     limits_MI_McH->GetYaxis()->SetTitleOffset(1.1);
00669     limits_MI_McH->GetYaxis()->SetLimits(1,999);
00670     //limits4->GetXaxis()->SetLabelOffset(0.02);
00671     limits_MI_McH->GetXaxis()->SetLabelSize(0.06);
00672     limits_MI_McH->GetXaxis()->SetTitleSize(0.06);
00673     limits_MI_McH->GetXaxis()->SetTitleOffset(1.1);
00674     limits_MI_McH->GetXaxis()->SetLimits(1,999);
00675
00676     TCanvas * c6=new TCanvas("c6","",800,600);
00677     limits_MI_McH->Draw("CONT LIST");
00678     c6->SetMargin(me,ma,me,ma);
00679     c6->SetGrid();
00680     c6->SaveAs((string("pdf_")+string(name)+string("_MIMcH.png")).c_str());
00681
00682     TCanvas * c5=new TCanvas("c5","",800,600);
00683     limits_tb_MR->Draw("colz");
00684
00685     c5->SaveAs((string("pdf_")+string(name)+string("_tbMR.png")).c_str());
00686
00687     //delete m;
00688     //mass.close();
00689     f->Close();
00690     delete f;
00691     return 0;
00692
00693 }

```

Here is the call graph for this function:



## 8.6 draw.cpp

```

00001 #include "MCMC.h"
00002 #include "BGL.h"
00003 #include "TF2.h"
00004 #include "TProfile3D.h"
00005 #include "THStack.h"
00006 #include "TColor.h"
00007 #include "TROOT.h"
00008 #include "TStyle.h"
00009 #include "TGraph.h"
00010 #include "TLatex.h"
00011 #include "TFile.h"
00012 #include "TH2F.h"
00013 #include "TVector.h"
00014 #include "TCanvas.h"
00015 #include "TMath.h"

```

```

00016 #include <iostream>
00017 #include <fstream>
00018
00019
00020 #include <cln/cln.h>
00021 #include <cln/float.h>
00022
00023 using namespace BGLmodels;
00024
00025 /**
00026  * @brief A second implementation of the BGL model, for testing purposes
00027  */
00028 class BGL2: public Model{
00029 public:
00030
00031 BGL2(int genL=2,int genQ=2, int lup=0, int qup=0, int mssm=0):
00032     planck(6.58211928e-25),
00033     GF("G_F"),
00034     MZ("M_Z"),
00035     MW("M_W"),
00036     Mpip("Mpip",0.1396,"M_{\\pi^+}",domain::real),
00037     Mpi0("Mpi0",0.1349766,"M_{\\pi^0}",domain::real),
00038     MBp("MBp",5.279,"M_{B^+}",domain::real),
00039     MB0("MB0",5.2795,"M_{B^0}",domain::real),
00040     MBs0("MBs0",5.3663,"M_{B_s^0}",domain::real),
00041     MKp("MKp",0.493677,"MKp",domain::real),
00042     MK0("MK0",0.497614,"MK0",domain::real),
00043     MDp("MDp",1.86957,"MDp",domain::real),
00044     MD0("MD0",1.86480,"MD0",domain::real),
00045     MDsp("MDsp",1.96845,"MDsp",domain::real),
00046     MDs0("MDs0",0),
00047     Fpi("Fpi",0.132,"Fpi",domain::real),
00048     FB("FB",0.189,"FB",domain::real),
00049     FBs("FBs",0.225,"FBs",domain::real),
00050     FK("FK",0.159,"FK",domain::real),
00051     FD("FD",0.208,"FD",domain::real),
00052     FDs("FDs",0.248,"FDs",domain::real),
00053     //alpha(7.297352e-3*4*M_PI),
00054     cos2(pow(MW/MZ,2)),
00055     g(sqrt(GF*8/sqrt(ex(2)))*MW),
00056     //g(sqrt(4*Pi*alpha/(1-cos2))),
00057     tanb("tg\\beta"),
00058     cp("cp"),
00059     McH("M_{H^+}"),
00060     MR("M_{R}"),
00061     MI("M_{I}"),
00062     Tparam("T_param"),
00063     Sparam("S_param"),
00064     QCD1("QCD_1"),
00065     QCD2("QCD_2"),
00066     mixes(tanb,cp, genL,genQ, lup, qup, mssm),
00067     mu("\\mu"),
00068     BGLtype(4,0),
00069     mmmmax(1000),
00070     stepsize(1e-2)
00071     {
00072         alpha=pow(g,2)*(1-cos2)/(4*Pi);
00073         replacements.append(GF==1.166371e-5);
00074         replacements.append(MZ==M_MZ);
00075         replacements.append(MW==M_MW);
00076
00077         mixes.appenddtolst(replacements);
00078
00079         replacements.append(Pi==M_PI);
00080         replacements.append(sqrt(ex(2))==sqrt(2));
00081         replacements.append(Pi==M_PI);
00082         replacements.append(sqrt(ex(2))==sqrt(2));
00083
00084         Boson boson;
00085
00086         realsymbol q3("q3");
00087         ex vq3=dirac_slash(q3,4);
00088         varidx jmu(mu,4,1);
00089
00090         for(uint i=0;i<2;i++)
00091             for(uint j=0;j<3;j++)
00092                 for(uint k=0;k<3;k++){
00093                     conjtoabs.append(conjugate(mixes.V[i][j][k])==pow(abs(mixes.V[i][j][k]),2)/
00094                     mixes.V[i][j][k]);
00095                 }
00096
00097         //W+ boson
00098         boson.mass=MW;
00099         boson.s=sVector;
00100
00101         for(uint t=tLepton;t<=tQuark;t++) boson.C[t][iUp][
00102             iDown][hLeft]=mixes.V[t]*Matrix(g/sqrt(ex(2)));

```



```

00101 Boson wboson=boson;
00102 bosons.push_back(boson);
00103 boson.reset();
00104
00105 //H+ boson
00106 boson.mass=MCH;
00107 boson.s=sScalar;
00108
00109 for(uint t=tLepton;t<=tQuark;t++)
00110 for(uint i=iUp;i<=iDown;i++) boson.C[t][iUp][iDown][i]=mixes.VN[t][i]*
Matrix(g/MW/sqrt(ex(2)));
00111 Boson chiggs=boson;
00112 bosons.push_back(boson);
00113 boson.reset();
00114
00115 for(int b=bosons.size()-1;b>=0;b--) {
00116 boson.mass=bosons[b].mass;
00117 boson.s=bosons[b].s;
00118 if(boson.s==sVector)
00119 for(uint t=tLepton;t<=tQuark;t++)
00120 for(uint i=iUp;i<=iDown;i++)
00121 for(uint j=iUp;j<=iDown;j++)
00122 for(uint h=hLeft;h<=hRight;h++) {
00123 boson.C[t][i][j][h]=bosons[b].C[t][j][i][h].conjugate();
00124 }
00125 else for(uint t=tLepton;t<=tQuark;t++)
00126 for(uint i=iUp;i<=iDown;i++)
00127 for(uint j=iUp;j<=iDown;j++)
00128 for(uint h=hLeft;h<=hRight;h++) {
00129 boson.C[t][i][j][hLeft]=bosons[b].C[t][j][i][
hRight].conjugate();
00130 boson.C[t][i][j][hRight]=bosons[b].C[t][j][i][
hLeft].conjugate();
00131 }
00132 bosons.push_back(boson);
00133 boson.reset();
00134 }
00135
00136 // (R+iI)/sqrt(2) boson
00137 boson.mass=MR;
00138 boson.s=sScalar;
00139
00140 for(uint t=tLepton;t<=tQuark;t++) {
00141 boson.C[t][iDown][iDown][hRight]=mixes.N[t][
iDown]*Matrix(g/MW/ex(2));
00142 boson.C[t][iUp][iUp][hLeft]=mixes.N[t][iUp].conjugate()*
Matrix(g/MW/ex(2));
00143 boson.C[t][iDown][iDown][hLeft]=mixes.N[t][
iDown].conjugate()*Matrix(g/MW/ex(2));
00144 boson.C[t][iUp][iUp][hRight]=mixes.N[t][iUp]*
Matrix(g/MW/ex(2));
00145 }
00146 bosons.push_back(boson);
00147 boson.reset();
00148
00149 // (R+iI)/sqrt(2) boson
00150 boson.mass=MI;
00151 boson.s=sScalar;
00152
00153 for(uint t=tLepton;t<=tQuark;t++) {
00154 boson.C[t][iDown][iDown][hRight]=mixes.N[t][
iDown]*Matrix(I*g/MW/ex(2));
00155 boson.C[t][iUp][iUp][hLeft]=mixes.N[t][iUp].conjugate()*
Matrix(I*g/MW/ex(2));
00156 boson.C[t][iDown][iDown][hLeft]=mixes.N[t][
iDown].conjugate()*Matrix(-I*g/MW/ex(2));
00157 boson.C[t][iUp][iUp][hRight]=mixes.N[t][iUp]*
Matrix(-I*g/MW/ex(2));
00158 }
00159 bosons.push_back(boson);
00160 boson.reset();
00161
00162 Fermion electron(tLepton,iDown,fElectron);
00163 Fermion electronR(tLepton,iDown,fElectron,
cParticle,hRight);
00164
00165 Fermion muon(tLepton,iDown,fMuon);
00166 Fermion muonR(tLepton,iDown,fMuon,cParticle,
hRight);
00167
00168 Fermion tau(tLepton,iDown,fTau);
00169 Fermion tauR(tLepton,iDown,fTau,cParticle,
hRight);
00170 Fermion neutrino(tLepton,iUp);
00171 Fermion neutrinotau(tLepton,iUp,fTau);
00172 Fermion neutrinomuon(tLepton,iUp,fMuon);
00173 Fermion neutrinoe(tLepton,iUp,fElectron);

```

```

00174
00175     Fermion up(tQuark,iUp,fElectron);
00176     Fermion down(tQuark,iDown,fElectron);
00177     Fermion bottom(tQuark,iDown,fTau);
00178     Fermion strange(tQuark,iDown,fMuon);
00179     Fermion charm(tQuark,iUp,fMuon);
00180     Fermion top(tQuark,iUp,fTau);
00181
00182     Meson Pi0d(down,down,Mpi0,Fpi);
00183     Meson Pi0u(down,down,Mpi0,Fpi);
00184     Meson Pip(up,down,Mpip,Fpi);
00185     Meson Pim(down,up,Mpip,Fpi);
00186
00187     Meson K0(down,strange,MK0,FK);
00188     Meson Kp(up,strange,MKp,FK);
00189
00190     Meson D0(charm,up,MD0,FD);
00191     Meson Dp(charm,down,MDp,FD);
00192     Meson Dsp(charm,strange,MDsp,FDs);
00193
00194     Meson B0(down,bottom,MB0,FB);
00195     Meson Bp(up,bottom,MBp,FB);
00196     Meson Bs0(strange,bottom,MBs0,FBs);
00197
00198     1st sb;
00199     //sb.append(mixes.M[tQuark][iUp][0][0]==0);
00200     sb.append(pow(abs(mixes.V[0][2][2]),2)==1-pow(abs(mixes.V[0][1][2]),2)-pow(abs(mixes.V[0][0][2]),2)
);
00201     sb.append(pow(abs(mixes.V[0][2][1]),2)==1-pow(abs(mixes.V[0][1][1]),2)-pow(abs(mixes.V[0][0][1]),2)
);
00202
00203     //cout<<pow(sqrt(2)/8*pow(g/MW,2),2)<<endl;
00204     //cout<<pow(1.166,2)<<endl;
00205     double fK=0.156;
00206     ex KKbar=ex(std::pow(fK,2))*mesonmixing(MK0,strange,down);
00207     ex eK=0.94*imag_part(KKbar)/3.5e-15/sqrt(2);
00208     cout<<"KKbar "<<KKbar<<endl;
00209     KKbar=expand(KKbar.subs(replacements).subs(1st(abs(wild()*pow(MR,-2))==abs(wild()*pow(MR,-2))).subs(
1st(log(wild()*pow(MR,-2))==log(wild()-2*log(MR)))));
00210     KKbar=expand(KKbar.evalf());
00211     ex aKKbar=sqrt(KKbar.real_part()+KKbar.real_part()+KKbar.imag_part()*KKbar.imag_part());
00212
00213     eK=0.94*imag_part(KKbar)/3.5e-15/sqrt(2);
00214     eK=eK.subs(replacements).real_part();
00215     eK=collect_common_factors(expand(eK.evalf()));
00216     cout<<"eK"<<eK<<endl;
00217     //add("a_eK",abs(eK),new limitedobs(2.2e-3));
00218     epsilonK=new calcuex(new limitedobs(2*0.011e-3),abs(eK));
00219
00220
00221 }
00222
00223 ~BGL2() {epsilonK->~calcuex();}
00224
00225 parameters generateparameters(int max=0) const{
00226     parameters p;
00227     //x=log_10(tanb)
00228     p.push_back(freeparameter(-3,3,r,stepsize));
00229     //y=log_10(McH)
00230     if(max==1) p.push_back(freeparameter(10,10000,r,stepsize));
00231     else p.push_back(freeparameter(10,mmmax,r,stepsize));
00232     //log_10(massR)
00233     p.push_back(freeparameter(-200,200,r,stepsize));
00234     //log_10(massI)
00235     p.push_back(freeparameter(-50,50,r,stepsize));
00236
00237     return p;
00238 }
00239
00240
00241 parameters getlist(const parameters & p) const{
00242     //cout<<aux<<endl;
00243     //double
00244     c2=(1+sqrt(1-4*sqrt(ex_to<numeric>(mudecay.subs(1st(tanb==exp(p[0].value),McH==p[1].value))).to_double()))/2;
00245
00246     double x=pow(10.0,p[0].value);
00247     //double y=pow(10.0,p[1].value);
00248     //double z=pow(10.0,p[2].value);
00249     //double w=pow(10.0,p[3].value);
00250
00251     double y=p[1].value;
00252     double z=y+p[2].value;
00253     double w=z+p[3].value;
00254
00255     parameters pp(p);
00256     pp[0].value=x;
00257     pp[2].value+=pp[1].value;

```

```

00257         pp[3].value+=pp[2].value;
00258         pp.values=vector<double>();
00259         for(uint i=0; i<4; i++) pp.values.push_back(pp[i].value);
00260         lst &l=pp.p;
00261         l=lst (tanb==x,McH==y,MR==z,MI==w);
00262         l.append(QCD1==inter1.Eval(y));
00263         l.append(QCD2==inter2.Eval(y));
00264
00265         for(uint i=0;i<3;i++){
00266             l.append(Mu[i]==Mu_[i].Eval(log(y)));
00267             l.append(Md[i]==Md_[i].Eval(log(y)));
00268         }
00269         return pp;
00270     }
00271
00272     ex mesonmixing(ex mesonmass, const Fermion& f1, const Fermion& f2) const{
00273
00274         ex ret=0;
00275
00276         ex v1=0, v2=0;
00277         ex mq1=mixes.mass(f1),mq2=mixes.mass(f2);
00278         ex m2q1=mq1*mq1, m2q2=mq2*mq2;
00279
00280         for(uint i=0;i<bosons.size();i++){
00281             if(bosons[i].s==0){
00282                 ex a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1));
00283                 v1=v1+pow(a/bosons[i].mass,2);
00284
00285                 ex b=(bosons[i].couplingdaggerR(f2,f1)+bosons[i].couplingdaggerL(f2,f1));
00286                 v2=v2+pow(b/bosons[i].mass,2);
00287             }
00288
00289             ex fc=mesonmass/(mixes.massnum(f1)+mixes.massnum(f2));
00290             fc=pow(fc,2);
00291
00292             ret=2*(-v1*(1+l1*fc)+v2*(1+fc))*mesonmass/96;
00293
00294             return collect_common_factors(ret.subs(conjtoabs));
00295             //return
00296             expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
00297         }
00298
00299     double bsgammawidth(double tanb_,double McH_,double MR_,double MI_, int option=0){
00300         parameters p=generateparameters();
00301         p[0].value=pow(10.0,tanb_);
00302         p[1].value=McH_;
00303         p[2].value=MR_;
00304         p[3].value=MI_;
00305         calcubtosgamma2 cal(mixes);
00306         return cal.width(p,option);
00307     }
00308
00309     double epsK(double tanb_,double McH_,double MR_,double MI_, int option=0){
00310         parameters p=generateparameters();
00311         p[0].value=pow(10.0,tanb_);
00312         p[1].value=McH_;
00313         p[2].value=MR_;
00314         p[3].value=MI_;
00315         p.p=lst(tanb==p[0].value,McH==p[1].value,MR==p[2].value,MI==p[3].value);
00316
00317         return epsilonK->error(p);
00318     }
00319 }
00320
00321 const double planck;
00322 const possymbol GF, MZ, MW, Mh;
00323 const constant Mpip, Mpi0, MBp, MB0, MBs0, MKp, MK0, MDp, MD0, MDsp, MDs0;
00324 const constant Fpi, FB, FBs, FK, FD, FDs;
00325 ex cos2, g, alpha;
00326 const possymbol tanb, cp, McH, MR, MI, rho;
00327 const realsymbol Tparam, Sparam, QCD1, QCD2;
00328 possymbol Mu[3],Md[3];
00329 vector< Boson > bosons;
00330
00331 lst replacements;
00332 ex Btaunu;
00333 ex BR_Htotaunu;
00334 ex BR_toptoHg;
00335 ex BtotaunuR;
00336 ex BtoDtaunuR;
00337 ex BtoD2taunuR;
00338
00339 const Mixes mixes;
00340 lst conjtoabs;
00341 realsymbol mu;
00342

```

```

00343 int iBtaunu, iBDtaunu, iBD2taunu;
00344 vector<int> BGLtype;
00345 ROOT::Math::Interpolator inter1, inter2;
00346 ROOT::Math::Interpolator Mu_[3], Md_[3];
00347 double mmmmax, stepsize;
00348
00349 calcuex * epsilonK;
00350
00351 };
00352
00353
00354 /**
00355  * @brief the main function takes the arguments inputfile gL gQ lup qup which specify the file containing
00356  * the simulation results for a BGL model and draws the plots for that model
00357 */
00357 int main(int argc, char* argv[]){
00358     // Check the number of parameters
00359
00360     if(argc<6){
00361         std::cerr<<"Usage: "<<argv[0]<<" inputfile gL gQ lup qup"<<std::endl;
00362         return 1;
00363     }
00364     CD cmu=conj(Vud[1][0])*Vud[1][1];
00365     CD umu=conj(Vud[0][0])*Vud[0][1];
00366
00367     cout<<"RATIO " <<cmu*cmu<<endl;
00368     cout<<umu*umu<<endl;
00369
00370     int gL=atoi(argv[2]);
00371     int gQ=atoi(argv[3]);
00372     int lup=atoi(argv[4]);
00373     int qup=atoi(argv[5]);
00374     char name[5]="0000";
00375
00376     name[0]+=gL;
00377     name[1]+=gQ;
00378     name[2]+=lup;
00379     name[3]+=qup;
00380     string ll[2][3]={{"#nu_{1}", "#nu_{2}", "#nu_{3}"}, {"e", "#mu", "#tau"}};
00381     string qq[2][3]={{"u", "c", "t"}, {"d", "s", "b"}};
00382     //Int_t MyPalette[100];
00383     Double_t r[] = {1, 0.3};
00384     Double_t g[] = {1, 0.3};
00385     Double_t b[] = {1, 0.3};
00386     Double_t stop[] = {0., 1.0};
00387     TColor::CreateGradientColorTable(2, stop, r, g, b, 100);
00388     //TH1F * pdf1=new TH1F("pdf1", "pdf1", npoints, 10, 500);
00389     //TGraph * chi2=new TGraph(npoints);
00390
00391     uint npoints=200;
00392     double init1=-3, final1=3;
00393     double init2=10, final2=1000;
00394     double initBmumu=0, finalBmumu=3;
00395     double initBsmumu=0, finalBsmumu=6;
00396
00397     double llmax=-1000, McHmax=1000, MRmax=1000, MImax=1000, tbmax=1;
00398
00399     TFile *f=new TFile(argv[1], "read");
00400     if(!f->IsOpen()) cout<<"NOFILE"<<endl;
00401     //f->ShowStreamerInfo();
00402
00403     TH2F *limits4,*Bmumu_Bsmumu,*limits_tb_MR,*limits_tb_MI;
00404     TH2F *limits_MR_MI,*limits_MR_McH,*limits_MI_McH;
00405
00406     f->GetObject("limits4;1", limits4);
00407     f->GetObject("Bmumu_Bsmumu;1", Bmumu_Bsmumu);
00408     f->GetObject("limits_tb_MR;1", limits_tb_MR);
00409     f->GetObject("limits_tb_MI;1", limits_tb_MI);
00410     f->GetObject("limits_MR_MI;1", limits_MR_MI);
00411     f->GetObject("limits_MR_McH;1", limits_MR_McH);
00412     f->GetObject("limits_MI_McH;1", limits_MI_McH);
00413
00414     TVectorD* vllmax=NULL;
00415
00416     f->GetObject("vllmax;1", vllmax);
00417     if(!vllmax) cout<<"ERROR"<<endl;
00418     llmax=(*vllmax)[0];
00419     //tbmax=(*vllmax)[1];
00420     //McHmax=(*vllmax)[2];
00421     //MRmax=McHmax+(*vllmax)[3];
00422     //MImax=MRmax+(*vllmax)[4];
00423     cout<<llmax<<" "<<tbmax<<" "<<McHmax<<" "<<MRmax<<" "<<MImax<<" "<<endl;
00424
00425     //BGL2* m=new BGL2(gL, gQ, lup, qup);
00426     double sm_(m->bsgammawidth(tbmax, McHmax, MRmax, MImax, 5));
00427     double charged_(m->bsgammawidth(tbmax, McHmax, MRmax, MImax, 1));
00428     double neutral_(m->bsgammawidth(tbmax, McHmax, MRmax, MImax, 2));

```

```

00429 double neutralR_(m->bsgammawidth(tbmax,McHmax,MRmax,MImax,3));
00430 double neutralI_(m->bsgammawidth(tbmax,McHmax,MRmax,MImax,4));
00431 double eK_(m->epsK(tbmax,McHmax,MRmax,MImax));
00432
00433 double all_(m->bsgammawidth(tbmax,McHmax,MRmax,MImax,0));
00434 */
00435 //for(int gL=2;gL>=0;gL--)
00436 //for(int gQ=2;gQ>=0;gQ--)
00437 //for(uint lup=0;lup<2;lup++)
00438 //for(uint qup=0;qup<2;qup++)
00439 uint min1=npoints, min2=npoints, min3=npoints;
00440 uint min11=npoints, min21=npoints, min31=npoints;
00441 uint min12=npoints, min22=npoints, min32=npoints;
00442
00443 for(uint i=0;i<npoints;i++)
00444 for(uint j=0;j<npoints;j++){
00445     int binmax=limits4->GetBin(i+1,j+1);
00446     double rest=limits4->GetBinContent(binmax);
00447     if(rest>=llmax) rest=1;
00448     else rest=TMath::Prob(-2*(rest-llmax),2);
00449     if(rest>=0.05 && j<min1){min1=j;}
00450     if(rest>=0.05 && j<min11 && j>uint(180*npoints/990)){min11=j;}
00451     if(rest>=0.05 && j<min12 && j>uint(400*npoints/990)){min12=j;}
00452     limits4->SetBinContent(i+1,j+1,rest);
00453
00454     rest=Bmumu_Bsmumu->GetBinContent(binmax);
00455     if(rest>=llmax) rest=1;
00456     else rest=TMath::Prob(-2*(rest-llmax),2);
00457     //int nn=4;
00458     //int ii=(i/nn)*nn, jj=(j/nn)*nn;
00459     //for(int iii=ii;iii<ii+n;++iii)
00460     //for(int iii=ii;iii<ii+n;++iii)
00461     Bmumu_Bsmumu->SetBinContent(i+1,j+1,rest);
00462
00463     rest=limits_MR_MI->GetBinContent(binmax);
00464     if(rest>=llmax) rest=1;
00465     else rest=TMath::Prob(-2*(rest-llmax),2);
00466     limits_MR_MI->SetBinContent(i+1,j+1,rest);
00467
00468     rest=limits_MR_McH->GetBinContent(binmax);
00469     if(rest>=llmax) rest=1;
00470     else rest=TMath::Prob(-2*(rest-llmax),2);
00471     if(rest>=0.05 && i<min2){min2=i;}
00472     if(rest>=0.05 && i<min21 && j>uint(180*npoints/990)){min21=i;}
00473     if(rest>=0.05 && i<min22 && j>uint(400*npoints/990)){min22=i;}
00474     limits_MR_McH->SetBinContent(i+1,j+1,rest);
00475
00476     rest=limits_MI_McH->GetBinContent(binmax);
00477     if(rest>=llmax) rest=1;
00478     else rest=TMath::Prob(-2*(rest-llmax),2);
00479     if(rest>=0.05 && i<min3){min3=i;}
00480     if(rest>=0.05 && i<min31 && j>uint(180*npoints/990)){min31=i;}
00481     if(rest>=0.05 && i<min32 && j>uint(400*npoints/990)){min32=i;}
00482     limits_MI_McH->SetBinContent(i+1,j+1,rest);
00483
00484     rest=limits_tb_MR->GetBinContent(binmax);
00485     if(rest>=llmax) rest=1;
00486     else rest=TMath::Prob(-2*(rest-llmax),2);
00487     limits_tb_MR->SetBinContent(i+1,j+1,rest);
00488 }
00489 double mmin1=init2+((min1+0.5)*(final2-init2))/npoints;
00490 double mmin2=init2+((min2+0.5)*(final2-init2))/npoints;
00491 double mmin3=init2+((min3+0.5)*(final2-init2))/npoints;
00492
00493 double mmin11=init2+((min11+0.5)*(final2-init2))/npoints;
00494 double mmin21=init2+((min21+0.5)*(final2-init2))/npoints;
00495 double mmin31=init2+((min31+0.5)*(final2-init2))/npoints;
00496
00497 double mmin12=init2+((min12+0.5)*(final2-init2))/npoints;
00498 double mmin22=init2+((min22+0.5)*(final2-init2))/npoints;
00499 double mmin32=init2+((min32+0.5)*(final2-init2))/npoints;
00500
00501 //maxs<<name<<" "<<mmin1<<" "<<mmin2<<" "<<mmin3<<endl;
00502
00503 ofstream maxs((string("maxs_")+string(name)+string(".out")).c_str());
00504 maxs<<mmin1<<" "<<mmin2<<" "<<mmin3<<endl;
00505 maxs<<mmin11<<" "<<mmin21<<" "<<mmin31<<endl;
00506 maxs<<mmin12<<" "<<mmin22<<" "<<mmin32<<endl;
00507 maxs<<llmax<<" "<<tbmax<<" "<<McHmax<<" "<<MRmax<<" "<<MImax<<" "<<endl;
00508 //maxs<<eK_<<endl;
00509 //maxs<<sm_<<" "<<charged_<<" "<<neutral_<<" "<<neutralR_<<" "<<neutralI_<<" "<<all_<<endl;
00510
00511 //for(uint j=0;j<npoints;j++)
00512 //for(uint i=0;i<npoints;i++){
00513 //    int binmax=limits4->GetBin(i+1,j+1);
00514 //    maxs<<"("<<i<<","<<j<<"): "<<limits4->GetBinContent(binmax)<<endl;
00515

```

```

00516         //      }
00517
00518     maxs.close();
00519
00520     double ma=0,me=.2, x0=1,y0=120;
00521     gStyle->SetOptTitle(0);
00522     gStyle->SetPaperSize(10.,10.);
00523     TCanvas * c21=new TCanvas("c21","",int(800*(1+ma+me)),int(600*(1+ma+me)));
00524     c21->SetMargin(me,ma,me,ma);
00525     c21->SetGrid();
00526
00527     limits4->SetStats(0);
00528     limits4->GetXaxis()->SetTitle("log_{10}(tan\\beta)");
00529     limits4->GetYaxis()->SetTitle("M_{H+} (GeV)");
00530
00531     Bmumu_Bsmumu->SetStats(0);
00532     Bmumu_Bsmumu->GetXaxis()->SetTitle("Br(B\\to\\mu\\mu)/10^{-10}");
00533     Bmumu_Bsmumu->GetYaxis()->SetTitle("Br(B_{s}\\to\\mu\\mu)/10^{-9}");
00534
00535     limits_MR_MI->SetStats(0);
00536     limits_MR_MI->GetYaxis()->SetTitle("M_{I} (GeV)");
00537     limits_MR_MI->GetXaxis()->SetTitle("M_{R} (GeV)");
00538
00539
00540     limits_MR_McH->SetStats(0);
00541     limits_MR_McH->GetYaxis()->SetTitle("M_{H+} (GeV)");
00542     limits_MR_McH->GetXaxis()->SetTitle("M_{R} (GeV)");
00543
00544     limits_MI_McH->SetStats(0);
00545     limits_MI_McH->GetYaxis()->SetTitle("M_{H+} (GeV)");
00546     limits_MI_McH->GetXaxis()->SetTitle("M_{I} (GeV)");
00547
00548     limits_tb_MR->SetStats(0);
00549     limits_tb_MR->GetXaxis()->SetTitle("log_{10}(tan\\beta)");
00550     limits_tb_MR->GetYaxis()->SetTitle("M_{R} (GeV)");
00551
00552
00553
00554     Double_t contours[3];
00555     contours[0] = 0.003;
00556     contours[1] = 0.05;
00557     contours[2] = 0.32;
00558
00559
00560
00561
00562     limits4->SetContour(3, contours);
00563     //limits4->GetYaxis()->SetLabelOffset(0.02);
00564     limits4->GetYaxis()->SetLabelSize(0.08);
00565     limits4->GetYaxis()->SetTitleSize(0.08);
00566     limits4->GetYaxis()->SetTitleOffset(1.2);
00567     limits4->GetYaxis()->SetLimits(1,999);
00568
00569
00570
00571     //limits4->GetXaxis()->SetLabelOffset(0.02);
00572     limits4->GetXaxis()->SetLabelSize(0.08);
00573     limits4->GetXaxis()->SetTitleSize(0.08);
00574     limits4->GetXaxis()->SetTitleOffset(1.2);
00575     limits4->GetXaxis()->SetLimits(-2.99,2.99);
00576
00577
00578
00579     TLatex l;
00580     l.SetTextSize(0.08);
00581     string ss=qq[qup][gQ]+" "+ll[lup][gL];
00582
00583
00584     limits4->Draw("CONT Z LIST");
00585     //limits4->Draw("CONT LIST");
00586     //limits4->Draw("colz");
00587
00588     l.DrawLatex(x0,y0,ss.c_str());
00589
00590     c21->SaveAs((string("pdf_")+string(name)+string(".png")).c_str());
00591
00592     delete c21;
00593     //Bmumu_Bsmumu->SetBit(TH1::kCanRebin);
00594     Bmumu_Bsmumu->Rebin2D(2,2);
00595
00596     //Bmumu_Bsmumu->SetContour(3, contours);
00597     //limits4->GetYaxis()->SetLabelOffset(0.02);
00598     Bmumu_Bsmumu->GetYaxis()->SetLabelSize(0.08);
00599     Bmumu_Bsmumu->GetYaxis()->SetTitleSize(0.08);
00600     Bmumu_Bsmumu->GetYaxis()->SetTitleOffset(0.8);
00601     Bmumu_Bsmumu->GetYaxis()->SetLimits(0.01,4.99);
00602     //Bmumu_Bsmumu->GetYaxis()->SetRangeUser(0.01, 3.49);

```

```

00603 // Bmumu_Bsmumu->GetYaxis()->SetLimits(0.01,3.49);
00604 //limits4->GetXaxis()->SetLabelOffset(0.02);
00605 Bmumu_Bsmumu->GetYaxis()->SetNdivisions(5, kTRUE);
00606 Bmumu_Bsmumu->GetXaxis()->SetNdivisions(5, kTRUE);
00607
00608 Bmumu_Bsmumu->GetXaxis()->SetLabelSize(0.08);
00609 Bmumu_Bsmumu->GetXaxis()->SetTitleSize(0.08);
00610 Bmumu_Bsmumu->GetXaxis()->SetTitleOffset(1.2);
00611 Bmumu_Bsmumu->GetXaxis()->SetLimits(0.01,1.99);
00612 //Bmumu_Bsmumu->GetXaxis()->SetRangeUser(0., 2);
00613
00614 TCanvas * cB=new TCanvas("cB","",int(800*(1+ma+me)),int(600*(1+ma+me)));
00615 cB->SetMargin(.14,ma,me,ma);
00616 cB->SetGrid();
00617 //limits4->Draw("CONT Z LIST");
00618 Bmumu_Bsmumu->Draw("COLZ");
00619
00620 l.DrawLatex(1.5,1,ss.c_str());
00621
00622 cB->SaveAs((string("Bmumu_Bsmumu_")+string(name)+string(".png")).c_str());
00623
00624 delete cB;
00625
00626 limits_MR_MI->SetContour(3, contours);
00627 //limits4->GetYaxis()->SetLabelOffset(0.02);
00628 limits_MR_MI->GetYaxis()->SetLabelSize(0.06);
00629 limits_MR_MI->GetYaxis()->SetTitleSize(0.06);
00630 limits_MR_MI->GetYaxis()->SetTitleOffset(1.1);
00631 limits_MR_MI->GetYaxis()->SetLimits(1,999);
00632 //limits4->GetXaxis()->SetLabelOffset(0.02);
00633 limits_MR_MI->GetXaxis()->SetLabelSize(0.06);
00634 limits_MR_MI->GetXaxis()->SetTitleSize(0.06);
00635 limits_MR_MI->GetXaxis()->SetTitleOffset(1.1);
00636 limits_MR_MI->GetXaxis()->SetLimits(1,999);
00637
00638 TCanvas * c3=new TCanvas("c3","",800,600);
00639 c3->SetMargin(me,ma,me,ma);
00640 c3->SetGrid();
00641
00642 limits_MR_MI->Draw("CONT LIST");
00643
00644 c3->SaveAs((string("pdf_")+string(name)+string("_MRMI.png")).c_str());
00645
00646 limits_MR_McH->SetContour(3, contours);
00647 //limits4->GetYaxis()->SetLabelOffset(0.02);
00648 limits_MR_McH->GetYaxis()->SetLabelSize(0.06);
00649 limits_MR_McH->GetYaxis()->SetTitleSize(0.06);
00650 limits_MR_McH->GetYaxis()->SetTitleOffset(1.1);
00651 limits_MR_McH->GetYaxis()->SetLimits(1,999);
00652 //limits4->GetXaxis()->SetLabelOffset(0.02);
00653 limits_MR_McH->GetXaxis()->SetLabelSize(0.06);
00654 limits_MR_McH->GetXaxis()->SetTitleSize(0.06);
00655 limits_MR_McH->GetXaxis()->SetTitleOffset(1.1);
00656 limits_MR_McH->GetXaxis()->SetLimits(1,999);
00657
00658 TCanvas * c4=new TCanvas("c4","",800,600);
00659 limits_MR_McH->Draw("CONT LIST");
00660 c4->SetMargin(me,ma,me,ma);
00661 c4->SetGrid();
00662 c4->SaveAs((string("pdf_")+string(name)+string("_MRMcH.png")).c_str());
00663
00664 limits_MI_McH->SetContour(3, contours);
00665 //limits4->GetYaxis()->SetLabelOffset(0.02);
00666 limits_MI_McH->GetYaxis()->SetLabelSize(0.06);
00667 limits_MI_McH->GetYaxis()->SetTitleSize(0.06);
00668 limits_MI_McH->GetYaxis()->SetTitleOffset(1.1);
00669 limits_MI_McH->GetYaxis()->SetLimits(1,999);
00670 //limits4->GetXaxis()->SetLabelOffset(0.02);
00671 limits_MI_McH->GetXaxis()->SetLabelSize(0.06);
00672 limits_MI_McH->GetXaxis()->SetTitleSize(0.06);
00673 limits_MI_McH->GetXaxis()->SetTitleOffset(1.1);
00674 limits_MI_McH->GetXaxis()->SetLimits(1,999);
00675
00676 TCanvas * c6=new TCanvas("c6","",800,600);
00677 limits_MI_McH->Draw("CONT LIST");
00678 c6->SetMargin(me,ma,me,ma);
00679 c6->SetGrid();
00680 c6->SaveAs((string("pdf_")+string(name)+string("_MIMcH.png")).c_str());
00681
00682 TCanvas * c5=new TCanvas("c5","",800,600);
00683 limits_tb_MR->Draw("colz");
00684
00685 c5->SaveAs((string("pdf_")+string(name)+string("_tbMR.png")).c_str());
00686
00687 //delete m;
00688 //mass.close();
00689 f->Close();

```

```

00690         delete f;
00691         return 0;
00692
00693     }
00694

```

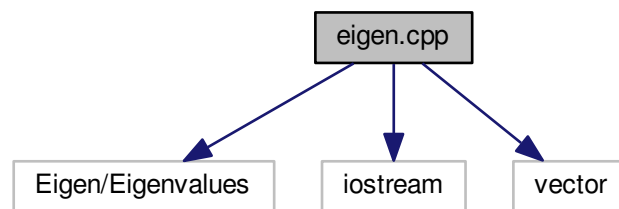
## 8.7 eigen.cpp File Reference

```

#include <Eigen/Eigenvalues>
#include <iostream>
#include <vector>

```

Include dependency graph for eigen.cpp:



### Classes

- class [points](#)

### Functions

- double [chi2](#) (double \*v)
- int [main](#) ()

### 8.7.1 Function Documentation

#### 8.7.1.1 double chi2 ( double \* v )

Definition at line 13 of file [eigen.cpp](#).

```

00013     {
00014         double buu = v[0], auu = v[1];
00015         double add = v[2], bdd = v[3];
00016         double eu = v[4], gu = v[5];
00017         double ed = v[6], gd = v[7];
00018         complex<double> bu = buu*exp(complex<double>(0,gu*M_PI_2)), au = auu*exp(complex<double>(0,eu*M_PI_2));
00019         complex<double> ad = add*exp(complex<double>(0,ed*M_PI_2)), bd = bdd*exp(complex<double>(0,gd*M_PI_2));
00020         complex<double> cd = bd - ad;
00021         complex<double> cu =bu + au;
00022
00023         //Matrix3cd X = Matrix3cd::Random(3,3);
00024         Matrix3cd mu,md;

```



```

00025 mu<<1,1,1.0+au+bu,1,1,1.0+bu,1.0+bu+au,1.0+bu,1.0+cu;
00026 md<<1,1,1.0+ad+bd,1,1,1.0+bd,1.0+bd+ad,1.0+bd,1.0+cd;
00027     mu=mu*v[8];
00028     md=md*v[9];
00029
00030 Matrix3cd Hu = mu*mu.adjoint(), Hd = md*md.adjoint();
00031 SelfAdjointEigenSolver<Matrix3cd> usolver(Hu), dsolver(Hd);
00032 Vector3d Du=usolver.eigenvalues();///.asDiagonal();
00033 Vector3d Dd=dsolver.eigenvalues();///.asDiagonal();
00034 Matrix3cd VLd=dsolver.eigenvectors().adjoint();
00035 Matrix3cd VLu=usolver.eigenvectors().adjoint();
00036 Matrix3cd Vckm=VLu*VLd.adjoint();
00037 double lambda=sqrt(norm(Vckm(0,1))/(norm(Vckm(0,0))+norm(Vckm(0,1))));
00038 double A=abs(Vckm(1,2))/abs(Vckm(0,1))/lambda;
00039 complex<double> rhoeta=-Vckm(0,0)*conj(Vckm(0,2))/Vckm(1,0)/conj(Vckm(1,2));
00040 double rho=real(rhoeta), eta=imag(rhoeta);
00041
00042 vector<points> comp;
00043 comp.push_back(points(lambda, 0.22535, 0.00065));
00044 comp.push_back(points(A, 0.811, 0.017));
00045 comp.push_back(points(rho, 0.131, 0.02));
00046 comp.push_back(points(eta, 0.345, 0.014));
00047 comp.push_back(points(eta, 0.345, 0.014));
00048 comp.push_back(points(Du[0], 1.27e-3, 0.46e-3));
00049 comp.push_back(points(Dd[0], 2.9e-3, 1.22e-3));
00050 comp.push_back(points(Dd[1], 55e-3, 16e-3));
00051 comp.push_back(points(Du[1], 0.619, 0.084));
00052 comp.push_back(points(Dd[2], 2.89, 0.09));
00053 comp.push_back(points(Du[2], 171.7, 3.0));
00054
00055 double chi=0;
00056
00057 for(uint i=0;i<comp.size();i++){
00058     chi+=pow((comp[i].prediction-comp[i].measure)/comp[i].error,2);
00059 }
00060 return chi/comp.size();
00061 }

```

### 8.7.1.2 int main ( )

Definition at line 63 of file [eigen.cpp](#).

```

00063     {
00064 double buu = 0.0162, auu = 0.0009;
00065 double add = 0.018, bdd = 0.09;
00066 //double x = 1.0/3;
00067 double eu = -1.8, gu = -1.0/2;
00068 double ed = -1.0/2, gd = -2;
00069 complex<double> bu = buu*exp(complex<double>(0,gu*M_PI_2)), au = auu*exp(complex<double>(0,eu*M_PI_2));
00070 complex<double> ad = add*exp(complex<double>(0,ed*M_PI_2)), bd = bdd*exp(complex<double>(0,gd*M_PI_2));
00071 complex<double> cd = exp(complex<double>(0,0.09/1.8));
00072
00073 //Matrix3cd X = Matrix3cd::Random(3,3);
00074 Matrix3cd mu,md;
00075 mu<<1,1,1.0+au+bu,1,1,1.0+bu,1.0+bu+au,1.0+bu,1.0+bu;
00076 //md<<1,1,1.0+ad+bd,1,1,1.0+bd,1.0+bd+ad,1.0+bd,1.0+bd;
00077 md<<1,1,1.0+ad+bd,1,1,1.0+bd,cd*(1.0+bd+ad),cd*(1.0+bd),cd*(1.0+bd);
00078 mu=mu*57.4822;
00079 md=md*1.0147;
00080
00081 Matrix3cd Hu = mu*mu.adjoint(), Hd = md*md.adjoint();
00082 SelfAdjointEigenSolver<Matrix3cd> usolver(Hu), dsolver(Hd);
00083 Vector3d Du=usolver.eigenvalues();///.asDiagonal();
00084 Vector3d Dd=dsolver.eigenvalues();///.asDiagonal();
00085 Matrix3cd VLd=dsolver.eigenvectors().adjoint();
00086 Matrix3cd VLu=usolver.eigenvectors().adjoint();
00087 Matrix3cd Vckm=VLu*VLd.adjoint();
00088 double lambda=sqrt(norm(Vckm(0,1))/(norm(Vckm(0,0))+norm(Vckm(0,1))));
00089 double A=abs(Vckm(1,2))/abs(Vckm(0,1))/lambda;
00090 complex<double> rhoeta=-Vckm(0,0)*conj(Vckm(0,2))/Vckm(1,0)/conj(Vckm(1,2));
00091 double rho=real(rhoeta), eta=imag(rhoeta);
00092 double fu=173.5/sqrt(Du[2]), fd=2.89/sqrt(Dd[2]);
00093
00094 vector<points> comp;
00095 comp.push_back(points(lambda, 0.22535, 0.00065));
00096 comp.push_back(points(A, 0.811, 0.017));
00097 comp.push_back(points(rho, 0.131, 0.02));
00098 comp.push_back(points(eta, 0.345, 0.014));
00099 comp.push_back(points(sqrt(Du[0]), 1.27e-3, 0.46e-3));
00100 comp.push_back(points(sqrt(Dd[0]), 2.9e-3, 1.22e-3));

```

```

00101 comp.push_back(points(sqrt(Dd[1]), 55e-3, 16e-3));
00102 comp.push_back(points(sqrt(Du[1]), 0.619, 0.084));
00103 comp.push_back(points(sqrt(Dd[2]), 2.89, 0.09));
00104 comp.push_back(points(sqrt(Du[2]), 171.7, 3.0));
00105
00106 double chi=0;
00107
00108 for(uint i=0;i<comp.size();i++){
00109     double aa=(comp[i].prediction-comp[i].measure)/comp[i].error;
00110     cout<<i<<" "<<aa<<endl;
00111     chi+=pow((comp[i].prediction-comp[i].measure)/comp[i].error,2);
00112 }
00113 chi/=comp.size();
00114
00115 cout<<lambda<<" "<<A<<" "<<rho<<" "<<eta<<endl;
00116 cout<<sqrt(Du[2])<<" "<<sqrt(Du[1])<<" "<<sqrt(Du[0])<<endl;
00117 cout<<sqrt(Dd[2])<<" "<<sqrt(Dd[1])<<" "<<sqrt(Dd[0])<<endl;
00118 cout<<chi<<endl;
00119 /*
00120 cout << "The eigenvalues of mu are:" << endl << es.eigenvalues() << endl;
00121 cout << "The matrix of eigenvectors, V, is:" << endl << es.eigenvectors() << endl << endl;
00122 double lambda = es.eigenvalues()[0];
00123 cout << "Consider the first eigenvalue, lambda = " << lambda << endl;
00124 Vector3cd v = es.eigenvectors().col(0);
00125 cout << "If v is the corresponding eigenvector, then lambda * v = " << endl << lambda * v << endl;
00126 cout << "... and A * v = " << endl << A * v << endl << endl;
00127 Matrix3cd D = es.eigenvalues().asDiagonal();
00128 Matrix3cd V = es.eigenvectors();
00129 cout << "Finally, V * D * V^(-1) = " << endl << V * D * V.inverse() << endl;
00130 */
00131 return 0;
00132 }

```

## 8.8 eigen.cpp

```

00001 #include <Eigen/Eigenvalues>
00002 #include <iostream>
00003 #include <vector>
00004 using namespace std;
00005 using namespace Eigen;
00006
00007 class points{
00008 public:
00009     points(double p, double m, double e):prediction(p),
00010     measure(m),error(e){}
00011 double prediction, measure, error;
00012 };
00013 double chi2(double * v){
00014     double buu = v[0], auu = v[1];
00015     double add = v[2], bdd = v[3];
00016     double eu =v[4], gu = v[5];
00017     double ed = v[6], gd = v[7];
00018 complex<double> bu = buu*exp(complex<double>(0,gu*M_PI_2)), au = auu*exp(complex<double>(0,eu*M_PI_2));
00019 complex<double> ad = add*exp(complex<double>(0,ed*M_PI_2)), bd = bdd*exp(complex<double>(0,gd*M_PI_2));
00020 complex<double> cd = bd - ad;
00021 complex<double> cu =bu + au;
00022
00023 //Matrix3cd X = Matrix3cd::Random(3,3);
00024 Matrix3cd mu,md;
00025 mu<<1,1,1.0+au+bu,1,1,1.0+bu,1.0+bu+au,1.0+bu,1.0+cu;
00026 md<<1,1,1.0+ad+bd,1,1,1.0+bd,1.0+bd+ad,1.0+bd,1.0+cd;
00027 mu=mu*v[8];
00028 md=md*v[9];
00029
00030 Matrix3cd Hu = mu*mu.adjoint(), Hd = md*md.adjoint();
00031 SelfAdjointEigenSolver<Matrix3cd> usolver(Hu), dsolver(Hd);
00032 Vector3d Du=usolver.eigenvalues();//.asDiagonal();
00033 Vector3d Dd=dsolver.eigenvalues();//.asDiagonal();
00034 Matrix3cd VLd=dsolver.eigenvectors().adjoint();
00035 Matrix3cd VLu=usolver.eigenvectors().adjoint();
00036 Matrix3cd Vckm=VLu*VLd.adjoint();
00037 double lambda=sqrt((norm(Vckm(0,1))/(norm(Vckm(0,0))+norm(Vckm(0,1)))));
00038 double A=abs(Vckm(1,2))/abs(Vckm(0,1))/lambda;
00039 complex<double> rhoeta=-Vckm(0,0)*conj(Vckm(0,2))/Vckm(1,0)/conj(Vckm(1,2));
00040 double rho=real(rhoeta), eta=imag(rhoeta);
00041
00042 vector<points> comp;
00043 comp.push_back(points(lambda, 0.22535, 0.00065));
00044 comp.push_back(points(A, 0.811, 0.017));
00045 comp.push_back(points(rho, 0.131, 0.02));
00046 comp.push_back(points(eta, 0.345, 0.014));

```

```

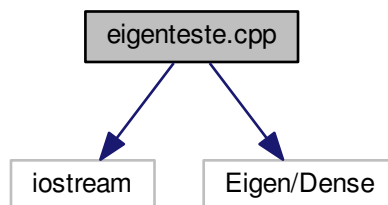
00047 comp.push_back(points(eta, 0.345, 0.014));
00048 comp.push_back(points(Du[0], 1.27e-3, 0.46e-3));
00049 comp.push_back(points(Dd[0], 2.9e-3, 1.22e-3));
00050 comp.push_back(points(Dd[1], 55e-3, 16e-3));
00051 comp.push_back(points(Du[1], 0.619, 0.084));
00052 comp.push_back(points(Dd[2], 2.89, 0.09));
00053 comp.push_back(points(Du[2], 171.7, 3.0));
00054
00055 double chi=0;
00056
00057 for(uint i=0;i<comp.size();i++){
00058     chi+=pow((comp[i].prediction-comp[i].measure)/comp[i].error,2);
00059 }
00060 return chi/comp.size();
00061 }
00062
00063 int main(){
00064     double buu = 0.0162, auu = 0.0009;
00065     double add = 0.018, bdd = 0.09;
00066     //double x = 1.0/3;
00067     double eu = -1.8, gu = -1.0/2;
00068     double ed = -1.0/2, gd = -2;
00069     complex<double> bu = buu*exp(complex<double>(0,gu*M_PI_2)), au = auu*exp(complex<double>(0,eu*M_PI_2));
00070     complex<double> ad = add*exp(complex<double>(0,ed*M_PI_2)), bd = bdd*exp(complex<double>(0,gd*M_PI_2));
00071     complex<double> cd = exp(complex<double>(0,0.09/1.8));
00072
00073     //Matrix3cd X = Matrix3cd::Random(3,3);
00074     Matrix3cd mu,md;
00075     mu<<1,1,1.0+au+bu,1,1,1.0+bu,1.0+bu+au,1.0+bu,1.0+bu;
00076     //md<<1,1,1.0+ad+bd,1,1,1.0+bd,1.0+bd+ad,1.0+bd,1.0+bd;
00077     md<<1,1,1.0+ad+bd,1,1,1.0+bd,cd*(1.0+bd+ad),cd*(1.0+bd),cd*(1.0+bd);
00078     mu=mu*57.4822;
00079     md=md*1.0147;
00080
00081     Matrix3cd Hu = mu*mu.adjoint(), Hd = md*md.adjoint();
00082     SelfAdjointEigenSolver<Matrix3cd> usolver(Hu), dsolver(Hd);
00083     Vector3d Du=usolver.eigenvalues(); //.asDiagonal();
00084     Vector3d Dd=dsolver.eigenvalues(); //.asDiagonal();
00085     Matrix3cd VLd=dsolver.eigenvectors().adjoint();
00086     Matrix3cd VLu=usolver.eigenvectors().adjoint();
00087     Matrix3cd Vckm=VLu*VLd.adjoint();
00088     double lambda=sqrt(norm(Vckm(0,1))/(norm(Vckm(0,0))+norm(Vckm(0,1))));
00089     double A=abs(Vckm(1,2))/abs(Vckm(0,1))/lambda;
00090     complex<double> rhoeta=-Vckm(0,0)*conj(Vckm(0,2))/Vckm(1,0)/conj(Vckm(1,2));
00091     double rho=real(rhoeta), eta=imag(rhoeta);
00092     double fu=173.5/sqrt(Du[2]), fd=2.89/sqrt(Dd[2]);
00093
00094     vector<points> comp;
00095     comp.push_back(points(lambda, 0.22535, 0.00065));
00096     comp.push_back(points(A, 0.811, 0.017));
00097     comp.push_back(points(rho, 0.131, 0.02));
00098     comp.push_back(points(eta, 0.345, 0.014));
00099     comp.push_back(points(sqrt(Du[0]), 1.27e-3, 0.46e-3));
00100     comp.push_back(points(sqrt(Dd[0]), 2.9e-3, 1.22e-3));
00101     comp.push_back(points(sqrt(Dd[1]), 55e-3, 16e-3));
00102     comp.push_back(points(sqrt(Du[1]), 0.619, 0.084));
00103     comp.push_back(points(sqrt(Dd[2]), 2.89, 0.09));
00104     comp.push_back(points(sqrt(Du[2]), 171.7, 3.0));
00105
00106     double chi=0;
00107
00108     for(uint i=0;i<comp.size();i++){
00109         double aa=(comp[i].prediction-comp[i].measure)/comp[i].error;
00110         cout<<i<<" "<<aa<<endl;
00111         chi+=pow((comp[i].prediction-comp[i].measure)/comp[i].error,2);
00112     }
00113     chi/=comp.size();
00114
00115     cout<<lambda<<" "<<A<<" "<<rho<<" "<<eta<<endl;
00116     cout<<sqrt(Du[2])<<" "<<sqrt(Du[1])<<" "<<sqrt(Du[0])<<endl;
00117     cout<<sqrt(Dd[2])<<" "<<sqrt(Dd[1])<<" "<<sqrt(Dd[0])<<endl;
00118     cout<<chi<<endl;
00119     /*
00120     cout << "The eigenvalues of mu are:" << endl << es.eigenvalues() << endl;
00121     cout << "The matrix of eigenvectors, V, is:" << endl << es.eigenvectors() << endl << endl;
00122     double lambda = es.eigenvalues()[0];
00123     cout << "Consider the first eigenvalue, lambda = " << lambda << endl;
00124     Vector3cd v = es.eigenvectors().col(0);
00125     cout << "If v is the corresponding eigenvector, then lambda * v = " << endl << lambda * v << endl;
00126     cout << "... and A * v = " << endl << A * v << endl << endl;
00127     Matrix3d D = es.eigenvalues().asDiagonal();
00128     Matrix3cd V = es.eigenvectors();
00129     cout << "Finally, V * D * V^(-1) = " << endl << V * D * V.inverse() << endl;
00130     */
00131     return 0;
00132 }

```

## 8.9 eigenteste.cpp File Reference

```
#include <iostream>
#include <Eigen/Dense>
```

Include dependency graph for eigenteste.cpp:



### Functions

- `int main()`

#### 8.9.1 Function Documentation

##### 8.9.1.1 `int main()`

Definition at line 4 of file `eigenteste.cpp`.

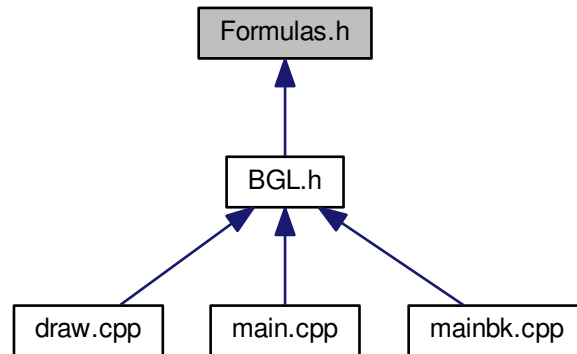
```
00005 {
00006   MatrixXd m(2,2);
00007   m(0,0) = 3;
00008   m(1,0) = 2.5;
00009   m(0,1) = -1;
00010   m(1,1) = m(1,0) + m(0,1);
00011   std::cout << m << std::endl;
00012 }
```

## 8.10 eigenteste.cpp

```
00001 #include <iostream>
00002 #include <Eigen/Dense>
00003 using Eigen::MatrixXd;
00004 int main()
00005 {
00006   MatrixXd m(2,2);
00007   m(0,0) = 3;
00008   m(1,0) = 2.5;
00009   m(0,1) = -1;
00010   m(1,1) = m(1,0) + m(0,1);
00011   std::cout << m << std::endl;
00012 }
```

## 8.11 Formulas.h File Reference

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



### Classes

- class [BGLmodels::Fermion](#)  
*a fermion properties*
- class [BGLmodels::Meson](#)  
*a meson properties*
- class [BGLmodels::Matrixx](#)  
*a class to represent the mixing matrices VCKM and VPMNS*
- class [BGLmodels::Mixes](#)  
*definition of the couplings for the different [BGL](#) models*
- class [BGLmodels::calcuOblique](#)  
*calculus of the constraints coming from the oblique parameters*
- class [BGLmodels::calcubtogsamma2](#)  
*calculus of the constraints coming from the  $b \rightarrow s$  gamma decay*
- class [BGLmodels::calcuBmumu](#)  
*calculus of the constraints coming from the  $B \rightarrow \mu \mu$  decay*

### Namespaces

- [BGLmodels](#)

### Typedefs

- typedef `std::complex< double >` [BGLmodels::CD](#)
- typedef `std::array< CD, 3 >` [BGLmodels::Vector3c](#)
- typedef `std::array< std::array< CD, 3 >, 3 >` [BGLmodels::Matrix3c](#)

## Enumerations

- enum `BGLmodels::FType` { `BGLmodels::tLepton`, `BGLmodels::tQuark` }
- enum `BGLmodels::FIsospin` { `BGLmodels::iUp`, `BGLmodels::iDown` }
- enum `BGLmodels::FFlavour` { `BGLmodels::fElectron`, `BGLmodels::fMuon`, `BGLmodels::fTau`, `BGLmodels::fAny` }
- enum `BGLmodels::FCharge` { `BGLmodels::cParticle`, `BGLmodels::cAntiParticle` }
- enum `BGLmodels::FHelicity` { `BGLmodels::hLeft`, `BGLmodels::hRight`, `BGLmodels::hAny` }
- enum `BGLmodels::BSpin` { `BGLmodels::sScalar`, `BGLmodels::sVector`, `BGLmodels::sAny` }

## Functions

- const Matrixx `BGLmodels::Vud` (13.04 \*M\_PI/180, 0.201 \*M\_PI/180, 2.38 \*M\_PI/180, 1.2)
- constexpr double `BGLmodels::C7SM` (double x)
- constexpr double `BGLmodels::C8SM` (double x)

## Variables

- constexpr double `BGLmodels::M_GF` =1.166371e-5
- constexpr double `BGLmodels::M_MZ` =91.1876
- constexpr double `BGLmodels::M_MW` =80.398
- constexpr double `BGLmodels::M_cos2` =std::pow(M\_MW/M\_MZ,2)
- constexpr double `BGLmodels::M_Mu` [3] ={2.4e-3,1.29,172.9}
- constexpr double `BGLmodels::M_Md` [3] ={5.3e-3,95e-3,4.2}
- constexpr double `BGLmodels::M_Ml` [3] ={0.510998910e-3,105.6583715e-3,1776.82e-3}
- const Matrixx `BGLmodels::Vnl` =Matrixx(33.6\*M\_PI/180,9.11\*M\_PI/180,40.4\*M\_PI/180,M\_PI/4).conjugate()
- constexpr double `BGLmodels::mt_mt` =163.3
- constexpr double `BGLmodels::mt_mW` =174.2
- constexpr double `BGLmodels::mt_mb` =261.8
- constexpr double `BGLmodels::C7SM_MW` =C7SM(std::pow(mt\_mW/M\_MW,2))
- constexpr double `BGLmodels::C7SM_Mt` =C7SM(std::pow(mt\_mt/M\_MW,2))
- constexpr double `BGLmodels::C7SM_Mb` =-0.353
- constexpr double `BGLmodels::C8SM_MW` =C8SM(std::pow(mt\_mW/M\_MW,2))
- constexpr double `BGLmodels::C8SM_Mt` =C8SM(std::pow(mt\_mt/M\_MW,2))
- constexpr double `BGLmodels::C8SM_Mb` =C8SM(std::pow(mt\_mb/M\_MW,2))

## 8.12 Formulas.h

```

00001 #ifndef Formulas_H
00002 #define Formulas_H
00003
00004 /*
00005 #include "widthcalc.h"
00006
00007 #include "TH2F.h"
00008 #include "TProfile2D.h"
00009 #include "TCanvas.h"
00010 #include <iostream>
00011
00012 #include "Math/Polynomial.h"
00013 #include "Math/Interpolator.h"
00014 #include <complex>
00015 #include <cmath>
00016 */
00017 using namespace std;
00018
00019 namespace BGLmodels{
00020

```

```

00021
00022 enum FType{tLepton,tQuark};
00023 enum FIsospin{iUp,iDown};
00024 enum FFlavour{fElectron,fMuon,fTau,fAny};
00025 enum FCharge{cParticle,cAntiParticle};
00026 enum FHelicity{hLeft,hRight,hAny};
00027 enum BSpin{sScalar, sVector,sAny};
00028
00029 /**
00030  * @brief a fermion properties
00031  */
00032 class Fermion{
00033 public:
00034
00035     Fermion(FType t, FIsospin i, FFlavour f=fAny,
00036             FCharge p=cParticle, FHelicity h=hAny): type(t), isospin(i), flavour(f), particle(
00037             p), helicity(h){}
00038
00039     FType type;
00040     FIsospin isospin;
00041     FFlavour flavour;
00042     FCharge particle;
00043     FHelicity helicity;
00044 };
00045
00046 /**
00047  * @brief a meson properties
00048  */
00049 class Meson{
00050 public:
00051
00052     Meson(const Fermion& qq1, const Fermion& qq2, ex m, ex d): q1(qq1), q2(qq2), mass(m)
00053     , decay_factor(d){}
00054
00055     Fermion q1, q2;
00056     ex mass;
00057     ex decay_factor;
00058 };
00059
00060 constexpr double M_GF=1.166371e-5;
00061 constexpr double M_MZ=91.1876;
00062 constexpr double M_MW=80.398;
00063 constexpr double M_cos2=std::pow(M_MW/M_MZ,2);
00064 constexpr double M_Mu[3]={2.4e-3,1.29,172.9};
00065 constexpr double M_Md[3]={5.3e-3,95e-3,4.2};
00066 constexpr double M_Ml[3]={0.510998910e-3,105.6583715e-3,1776.82e-3};
00067
00068 typedef std::complex<double> CD;
00069 typedef std::array<CD,3> Vector3c;
00070 typedef std::array<std::array<CD,3>,3> Matrix3c;
00071
00072 /**
00073  * @brief a class to represent the mixing matrices VCKM and VPMNS
00074  */
00075 class Matrixx: public Matrix3c{
00076 public:
00077     Matrixx(): Matrix3c({}){}
00078     Matrixx(const Matrix3c& a): Matrix3c(a){}
00079     ///constructs a unitary Matrixx in the standard form
00080     Matrixx(std::initializer_list<Vector3c> l):
00081     {}{}
00082     Matrixx(double c12, double c13, double c23,double s12, double s13, double s23, CD e13,CD e13t):
00083     Matrix3c({
00084         Vector3c({c12*c13,s12*c13,s13*e13t}),
00085         Vector3c({-s12*c23-c12*s23*s13*e13,c12*c23-s12*s23*s13*e13,s23*c13}),
00086         Vector3c({s12*s23-c12*c23*s13*e13,-c12*s23-s12*c23*s13*e13,c13*c23})
00087     }){}
00088     Matrixx(double t12, double t13, double t23, double d13):
00089     Matrixx(std::cos(t12),std::cos(t13),std::cos(t23),std::sin(t12),
00090             std::sin(t13),std::sin(t23),std::exp(CD(0,d13)),std::exp(CD(0,-d13))){}
00091     ///computes the hermitian conjugate of the Matrixx
00092     const Matrixx conjugate() const{
00093         Matrixx res;
00094         for(uint i=0;i<3;i++)
00095             for(uint j=0;j<3;j++){
00096                 res[i][j]=std::conj((*this)[j][i]);
00097             }
00098         return res;
00099     }
00100 };
00101
00102 constexpr Matrixx Vnl=Matrixx(33.6*M_PI/180,9.11*M_PI/180,40.4*M_PI/180,M_PI/4).
00103     conjugate();
00104 constexpr Matrixx Vud(13.04*M_PI/180,0.201*M_PI/180,2.38*M_PI/180,1.2);
00105
00106 /**

```

```

00103 * @brief definition of the couplings for the different BGL models
00104 */
00105 class Mixes{
00106 public:
00107
00108     Mixes(ex tanb0,ex cp0,int genL=2,int genQ=2,int lup=0,int qup=0,int mssm=0):M(
Matrix(),2,2),N(Matrix(),2,2),VN(Matrix(),2,2),N_(Matrix(),2,2),VN_(
Matrix(),2,2),cp(cp0)
00109     {
00110         tanb=tanb0;
00111         M[tLepton][iDown]=Matrix(possymbol("m_e"),possymbol("m_\mu"),possymbol("m_\tau"));
00112         M[tQuark][iUp]=Matrix(possymbol("m_u"),possymbol("m_c"),possymbol("m_t"));
00113         M[tQuark][iDown]=Matrix(possymbol("m_d"),possymbol("m_s"),possymbol("m_b"));
00114         const char * ln[3]={"1","2","3"};
00115         const char * ll[3]={"e","\mu","\tau"};
00116         const char * lu[3]={"u","c","t"};
00117         const char * ld[3]={"d","s","b"};
00118
00119         V.push_back(Matrix("U",ln,ll));
00120         V.push_back(Matrix("V",lu,ld));
00121
00122
00123         int up[2];
00124         up[0]=lup;
00125         up[1]=qup;
00126
00127         vector< Matrix > delta;
00128
00129         vector<int> gL(3,0);
00130         gL[genL]=1;
00131         //Leptons
00132         delta.push_back(Matrix(gL[0],gL[1],gL[2]));
00133         //Quarks
00134         vector<int> gQ(3,0);
00135         gQ[genQ]=1;
00136         delta.push_back(Matrix(gQ[0],gQ[1],gQ[2]));
00137
00138         for(uint i=0;i<2;i++){
00139             if(mssm){
00140                 //Nu
00141                 N_[i][0]=0;
00142                 //Nd
00143                 N_[i][1]=0;
00144                 //VNd
00145                 VN_[i][1]=Matrix(tanb)*V[i];
00146                 //Nu*V
00147                 VN_[i][0]=Matrix(1/tanb)*V[i];
00148             }
00149             else if(up[i]){
00150                 //Nu
00151                 N_[i][0]=Matrix(tanb)+Matrix(-tanb-1/tanb)*V[i]*delta[i]*V[i].
conjugate();
00152                 //Nd
00153                 N_[i][1]=Matrix(tanb)+Matrix(-tanb-1/tanb)*delta[i];
00154                 //VNd
00155                 VN_[i][1]=Matrix(tanb)*V[i]+Matrix(-tanb-1/tanb)*V[i]*delta[i];
00156                 //Nu*V
00157                 VN_[i][0]=Matrix(-1)*(Matrix(tanb)*V[i]+Matrix(-tanb-1/tanb)*V[i]*delta[i]);
00158                 }else{
00159                     //Nu
00160                     N_[i][0]=Matrix(tanb)+Matrix(-tanb-1/tanb)*delta[i];
00161                     //Nd
00162                     N_[i][1]=Matrix(tanb)+Matrix(-tanb-1/tanb)*V[i].conjugate()*delta[i]*V[i];
00163                     //VNd
00164                     VN_[i][1]=Matrix(tanb)*V[i]+Matrix(-tanb-1/tanb)*delta[i]*V[i];
00165                     //Nu*V
00166                     VN_[i][0]=Matrix(-1)*(Matrix(tanb)*V[i]+Matrix(-tanb-1/tanb)*delta[i]*V[i]);
00167                 }
00168
00169                 N[i][0]=N_[i][0]*M[i][0];
00170                 N[i][1]=N_[i][1]*M[i][1];
00171                 VN[i][1]=VN_[i][1]*M[i][1];
00172                 VN[i][0]=M[i][0]*VN_[i][0];
00173             }
00174             appendtolst(replacements);
00175
00176         }
00177     }
00178     ex mass(const Fermion& f) const{return M[f.type][f.isospin][f.
flavour][f.flavour];}
00179     double massnum(const Fermion& f) const{return ex_to<numeric>(M[f.
type][f.isospin][f.flavour][f.flavour].subs(replacements)).to_double();}
00180
00181     void appendtolst(lst & reps) const{//,vector<ex>& var_errors
00182         reps.append(M[0][1][0][0]==0.510998910e-3);
00183         reps.append(M[0][1][1][1]==105.6583715e-3);

```



```

00184         reps.append(M[0][1][2][2]==1776.82e-3);
00185
00186         reps.append(M[1][0][0][0]==2.4e-3);
00187         reps.append(M[1][0][1][1]==1.29);
00188         reps.append(M[1][0][2][2]==172.9);
00189
00190         reps.append(M[1][1][0][0]==5.3e-3);
00191         reps.append(M[1][1][1][1]==95e-3);
00192         reps.append(M[1][1][2][2]==4.2);
00193
00194         vector< Matrix > Vn;
00195         Vn.push_back(Matrix(33.6*M_PI/180,9.11*M_PI/180,40.4*M_PI/180,M_PI/4).conjugate());
00196         Vn.push_back(Matrix(13.04*M_PI/180,0.201*M_PI/180,2.38*M_PI/180,1.2));
00197         for(uint i=0; i<2;i++)
00198             for(uint j=0; j<3;j++)
00199                 for(uint k=0; k<3;k++)
00200                     reps.append(V[i][j][k]==Vn[i][j][k]);
00201     }
00202
00203     lst reps;
00204     vector< Matrix > V;
00205     multivector<Matrix,2> M;
00206     multivector<Matrix,2> N;
00207     multivector<Matrix,2> VN;
00208
00209     multivector<Matrix,2> N_;
00210     multivector<Matrix,2> VN_;
00211
00212     lst replacements;
00213     ex cp;
00214     ex tanb;
00215 };
00216
00217
00218 /**
00219  * @brief calculus of the constraints coming from the oblique parameters
00220  */
00221 class calcuOblique:public calcu{
00222     public:
00223
00224         calcuOblique(): c1(0.741),c2(0.671), g1(c1*0.02,0.0397), g2(c2*0.02,0.1579) {}
00225         double operator()(const parameters & p) const{
00226             double y=p[1].value;
00227             double z=p[2].value;
00228             double w=p[3].value;
00229
00230             double TT=(F(y*y,z*z)-F(w*w,z*z)+F(y*y,w*w))/(16*M_PI*M_MW*M_MW*(1-M_cos2));
00231             double Sparam=(std::pow(1-2*M_cos2,2)*G(y*y,y*y,M_MZ*M_MZ)+G(z*z,w*w,M_MZ*M_MZ)+2*log(z*w/y/y))/
24/M_PI;
00232
00233             double T1=c1*TT-c2*Sparam;
00234             double T2=c2*TT+c1*Sparam;
00235
00236             return g1.loglikelihood(T1)+g2.loglikelihood(T2);
00237         }
00238         double F(double x,double y) const{
00239             if(x==y) return 0;
00240             return (x+y)/2-x*y*log(x/y)/(x-y);
00241         }
00242         double f(double t,double r) const{
00243             if(r==0) return 0;
00244             if(r<0) return 2*sqrt(-r)*atan(sqrt(-r)/t);
00245             return sqrt(r)*log(fabs((t-sqrt(r))/(t+sqrt(r))));
00246         }
00247
00248         double lnxy_xy(double x, double y) const{
00249             if(x==y) return 1/y;
00250             return log(x/y)/(x-y);
00251         }
00252         double G(double x,double y,double z) const{
00253             double t=x+y-z;
00254             double r=std::pow(z,2)-2*z*(x+y)+std::pow(x-y,2);
00255             return -16.0/3+5*(x+y)/z-2*std::pow((x-y)/z,2)+r/std::pow(z,3)*f(t,r)+\
00256                 3/z*lnxy_xy(x,y)*(std::pow(x,2)+std::pow(y,2)+(x-y)/std::pow(z,2)*(-std::pow(x,2)+
std::pow(y,2)+std::pow(x-y,3)/3));
00257         }
00258         const double c1,c2;
00259         const gauss2obs g1,g2;
00260 };
00261
00262 constexpr double mt_mt=163.3,mt_mW=174.2,mt_mb=261.8;
00263
00264 constexpr double C7SM(double x){
00265     return ((1/(x-1)+3)*x*std::log(x)+(-8*x*x-5*x+7)/6)*x/4/std::pow(x-1,3);
00266 }
00267
00268 constexpr double C8SM(double x){

```

```

00269         return (-3/(x-1)*x*std::log(x)+(-x*x+5*x+2)/2)*x/4/std::pow(x-1,3);
00270     }
00271
00272     constexpr double C7SM_MW=C7SM(std::pow(mt_mW/M_MW,2)),C7SM_Mt=
00273     C7SM(std::pow(mt_mt/M_MW,2)),C7SM_Mb=-0.353;
00274     constexpr double C8SM_MW=C8SM(std::pow(mt_mW/M_MW,2)),C8SM_Mt=
00275     C8SM(std::pow(mt_mt/M_MW,2)),C8SM_Mb=C8SM(std::pow(mt_mb/M_MW,2));
00276
00277     /**
00278     * @brief calculus of the constraints coming from the b->s gamma decay
00279     */
00280     class calcubtosgamma2:public calcu{
00281     public:
00282     constexpr static double calN=2.567e-3;
00283     constexpr static double a=7.8221,aee=0.4384,aer=-1.6981,a77=0.8161,a7r=4.8802,a7er=-0.7827,a88=0.0197,a8r=0
00284     .5680;
00285     constexpr static double a8er=-0.0601,a87r=0.1923,a7i=0.3546,a8i=-0.0987,aei=2.4997,a87i=-0.0487,a7ei=-0.906
00286     7,a8ei=-0.0661;
00287
00288     calcubtosgamma2(const Mixes& mixes):
00289     {
00290         ii(2),
00291         g1(3.43e-4,sqrt(2)*0.23e-4),
00292         g2(9.2e-6,4e-6),
00293         ratio(0){
00294             //cout<<"C7 "<<C7SM_Mt<<" "<<C7SM_MW<<" "<<C7SM(std::pow(261.8/M_MW,2))<<endl;
00295             double res[2];
00296             constexpr double C7SM_[2]={C7SM_Mt,C7SM_Mt};
00297             constexpr double C8SM_[2]={C8SM_Mt,C8SM_Mt};
00298             for(uint j=0; j<2; j++){
00299                 const uint i=2;
00300                 const CD epsilon=conj(Vud[0][j])*Vud[0][i]/conj(Vud[2][j])/
00301                 Vud[2][i];
00302                 const double upsilon=norm(conj(Vud[2][j])*Vud[2][i]/Vud[1][i]);
00303                 const CD R7=(C7SM_Mt)/C7SM_MW;
00304                 const CD R8=(C8SM_Mt)/C8SM_MW;
00305                 const CD R7_=0;
00306                 const CD R8_=0;
00307                 res[j]=a+aee*norm(epsilon)+aer*epsilon.real()+aei*epsilon.imag();
00308                 res[j]+=a77*(norm(R7)+norm(R7_))+a7r*R7.real()+a7i*R7.imag();
00309                 res[j]+=a88*(norm(R8)+norm(R8_))+a8r*R8.real()+a8i*R8.imag();
00310                 res[j]+=a87r*(R8*conj(R7)+R8_*conj(R7_)).real()+a7er*(R7*conj(epsilon)).real()+a8er*(R8*
00311                 conj(epsilon)).real();
00312                 res[j]+=a87i*(R8*conj(R7)+R8_*conj(R7_)).imag()+a7ei*(R7*conj(epsilon)).imag()+a8ei*(R8*
00313                 conj(epsilon)).imag();
00314                 res[j]*=calN/100*upsilon;
00315             }
00316             //cout<<"Btosgamma "<<res[0]/9.2e-6<<" "<<res[1]/3.15e-4<<endl;
00317             ifstream finter("interpolation.dat");
00318
00319             if(!finter.is_open()){
00320                 cout<<"ERROR: interpolation.dat not found"<<endl;
00321                 exit(1);
00322             }
00323             vector<double> vinter0, vinter1, vinter2;
00324             while(!finter.eof()){
00325                 double a=0,b=0,c=0;
00326                 finter>>a>>b>>c;
00327                 if(a!=0){
00328                     // cout<<a<<" "<<b<<" "<<c<<endl;
00329                     vinter0.push_back(a);
00330                     vinter1.push_back(b);
00331                     vinter2.push_back(c);
00332                 }
00333             }
00334             inter1.SetData(vinter0,vinter1);
00335             inter2.SetData(vinter0,vinter2);
00336
00337             finter.close();
00338
00339             ifstream finter2("masses.dat");
00340
00341             if(!finter2.is_open()){
00342                 cout<<"ERROR: masses.dat not found"<<endl;
00343                 exit(1);
00344             }
00345             vector<vector<double> > m_(7);
00346             while(!finter2.eof()){
00347                 for(uint i=0; i<7;i++){
00348                     double a=0;
00349                     finter2>>a;
00350
00351                     if(a!=0){
00352                         if(i==0) a=log(a);

```

```

00349                                     else if(i<4) a*=1e-3;
00350                                     m_[i].push_back(a);
00351                                     //      cout<<a<<" ";
00352                                     }
00353                                     } //cout<<endl;
00354                                     }
00355                                     for(uint i=0; i<3;i++) {
00356                                         Md_[i].SetData(m_[0],m_[2*i+1]);
00357                                         Mu_[i].SetData(m_[0],m_[2*i+2]);
00358                                     }
00359                                     //      cout<<"Eval "<<Mu_[2].Eval(log(100.0))<<endl;
00360                                     //      cout<<"Eval "<<Md_[2].Eval(log(100.0))<<endl;
00361
00362                                     finter2.close();
00363
00364                                     ifstream finter3("interpolation2.dat");
00365
00366                                     if(!finter3.is_open()){
00367                                         cout<<"ERROR: interpolation2.dat not found"<<endl;
00368                                         exit(1);
00369                                     }
00370                                     vector<double> vinter20, vinter21, vinter22;
00371                                     while(!finter3.eof()){
00372                                         double a=0,b=0,c=0;
00373                                         finter3>>a>>b>>c;
00374                                         if(a!=0){
00375                                             //      cout<<a<<" "<<b<<" "<<c<<endl;
00376                                             vinter20.push_back(a);
00377                                             vinter21.push_back(b);
00378                                             vinter22.push_back(c);
00379                                         }
00380                                     }
00381
00382                                     inter3.SetData(vinter20,vinter21);
00383                                     inter4.SetData(vinter20,vinter22);
00384
00385                                     finter3.close();
00386
00387                                     vector<ex> vex(24);
00388
00389                                     const uint i=ii;
00390                                     for(uint j=0;j<2;j++){
00391                                         for(uint k=0;k<3;k++){
00392                                             vex[j*6+k*2+0]=mixes.VN_[tQuark][iUp][k][j].conjugate()*mixes.
00393                                             VN_[tQuark][iUp][k][i];
00394                                             vex[j*6+k*2+1]=mixes.N_[tQuark][iDown][j][k]*mixes.N_[
00395                                             tQuark][iDown][i][k].conjugate();
00396                                             vex[j*6+k*2+12]=-mixes.VN_[tQuark][iUp][k][j].conjugate()*mixes.
00397                                             VN_[tQuark][iDown][k][i];
00398                                             vex[j*6+k*2+18]=mixes.VN_[tQuark][iDown][k][j].conjugate()*mixes.
00399                                             VN_[tQuark][iDown][k][i];
00400                                         }
00401                                         lst l;
00402                                         for(uint k=0;k<vex.size();k++){
00403                                             vex[k]=vex[k].subs(mixes.replacements).evalf();
00404                                             l.append(vex[k].real_part());
00405                                             l.append(vex[k].imag_part());
00406                                         }
00407                                         compile_ex(l, lst(mixes.tanb), fp);
00408                                     }
00409
00410                                     double operator()(const parameters & p) const{
00411                                         double tanb=p[0].value;
00412                                         double y=p[1].value;
00413                                         double z=p[2].value;
00414                                         double w=p[3].value;
00415                                         double Mch=y, MR=z, MI=w;
00416
00417                                         double y0=y;
00418                                         if(y<mt_mt) y0=mt_mt;
00419                                         double QCD1[2]={inter3.Eval(y0),inter1.Eval(y)};
00420                                         double QCD2[2]={inter4.Eval(y0),inter2.Eval(y)};
00421
00422                                         double Mu[3],Md[3];
00423
00424                                         for(uint i=0;i<3;i++){
00425                                             Mu[i]=Mu_[i].Eval(log(y));
00426                                             Md[i]=Md_[i].Eval(log(z));
00427                                         }
00428                                         const uint i=ii;
00429                                         CD CC7[2],DD7[2],CC8[2],DD8[2];
00430                                         double res[2];
00431                                         // constexpr double C7SM_[2]={C7SM_Mt,C7SM_Mb};
00432                                         // constexpr double C8SM_[2]={C8SM_Mt,C8SM_Mb};
00433
00434                                         std::array<double,48> ret;
00435                                         const int n=1,m=48;

```

```

00432         fp(&n, &(tanb), &m, &(ret[0]));
00433     for(uint j=0; j<2; j++){
00434         const double mbottom=Md[i];
00435         const double mstrange=Md[j];
00436         //ex mbottom=mixes.M[tQuark][iDown][i][i];
00437         //ex mstrange=mixes.M[tQuark][iDown][j][j];
00438
00439         CD C7,D7,C8,D8;
00440         for(uint k=0; k<3; k++){
00441             double mup=Mu[k];
00442             double mdown=Md[k];
00443             //ex mup=mixes.M[tQuark][iUp][k][k];
00444             //ex mdown=mixes.M[tQuark][iDown][k][k];
00445             //f1+=
00446             double mmu=std::pow(mup/McH, 2);
00447             double mmdR=std::pow(mdown/MR, 2);
00448             double mmdI=std::pow(mdown/MI, 2);
00449
00450             double A0u=A0(mmu);
00451             double A1u=A1(mmu);
00452             double A2u=A2(mmu);
00453             double A3u=A3(mmu);
00454             double A0d=(A0(mmdR)+A0(mmdI));
00455             double A1d=(A1(mmdR)-A1(mmdI));
00456
00457             CD f1(ret[j*12+4*k+0], ret[j*12+4*k+1]);
00458             C7+=f1*A2u;
00459             C8+=-2.0*f1*A0u;
00460
00461             CD f2=CD(ret[36+j*2+4*k+0], ret[36+j*2+4*k+1])*mstrange*mbottom/mup/mup;
00462             //CD f2=f1*mstrange*mbottom/mup/mup;
00463             D7+=f2*A2u;
00464             D8+=-2.0*f2*A0u;
00465
00466             CD f12(ret[24+j*2+4*k+0], ret[24+j*2+4*k+1]);
00467             C7+=f12*A3u;
00468             C8+=2.0*f12*A1u;
00469
00470             CD f4(ret[j*12+4*k+2], ret[j*12+4*k+3]);
00471             C7+=f4*A0d/3.0;
00472             C8+=-f4*A0d;
00473
00474             C7+=f4*A1d/3.0;
00475             C8+=-f4*A1d;
00476
00477             CD f6=f4*mstrange*mbottom/mdown/mdown;
00478             D7+=f6*A0d/3.0;
00479             D8+=-f6*A0d;
00480         }
00481         uint j0=j;
00482         CC7[j]=(QCD1[j]*C7+QCD2[j]*C8)/2.0/conj(Vud[2][j])/Vud[2][i];
00483         DD7[j]=(QCD1[j]*D7+QCD2[j]*D8)/2.0/conj(Vud[2][j])/Vud[2][i];
00484         const double QCD3=(3*QCD2[j]/8+QCD1[j]);
00485         CC8[j]=QCD3*C8/2.0/conj(Vud[2][j])/Vud[2][i];
00486         DD8[j]=QCD3*D8/2.0/conj(Vud[2][j])/Vud[2][i];
00487         const CD epsilon=conj(Vud[0][j])*Vud[0][i]/conj(Vud[2][j])/
Vud[2][i];
00488         const double upsilon=norm(conj(Vud[2][j])*Vud[2][i]/Vud[1][i]);
00489         const CD R7=(C7SM_Mt+CC7[j])/C7SM_MW;
00490         const CD R8=(C8SM_Mt+CD(0)*CC8[j])/C8SM_MW;
00491         const CD R7_=(DD7[j])/C7SM_MW;
00492         const CD R8_=(DD8[j])/C8SM_MW;
00493
00494         res[j]=a+ae*norm(epsilon)+aer*epsilon.real()+aei*epsilon.imag();
00495         res[j]=a77*(norm(R7)+norm(R7_))+a7r*R7.real()+a7i*R7.imag();
00496         res[j]=a88*(norm(R8)+norm(R8_))+a8r*R8.real()+a8i*R8.imag();
00497         res[j]=a87r*(R8*conj(R7)+R8_*conj(R7_)).real()+a7er*(R7*conj(epsilon)).real()+a8er*(R8*
conj(epsilon)).real();
00498         res[j]=a87i*(R8*conj(R7)+R8_*conj(R7_)).imag()+a7ei*(R7*conj(epsilon)).imag()+a8ei*(R8*
conj(epsilon)).imag();
00499         res[j]=calN/100*upsilon;
00500
00501         /*
00502         res[j]=a+ae*norm(epsilon)+aer*epsilon.real()+aei*epsilon.imag();
00503         res[j]=a77*(norm(R7)+norm(R7_))+a7r*1+a7i*0;
00504         res[j]=a88*(norm(R8)+norm(R8_))+a8r*R8.real()+a8i*R8.imag();
00505         res[j]=a87r*1+a7er*(conj(epsilon)).real()+a8er*(R8*conj(epsilon)).real();
00506         res[j]=a87i*0+a7ei*(conj(epsilon)).imag()+a8ei*(R8*conj(epsilon)).imag();
00507         res[j]=calN/100*upsilon;
00508         */
00509     }
00510     double r1=3.15e-4+0.00247*(norm(CC7[1])+norm(DD7[1])-0.706*CC7[1].real());
00511
00512     //ratio=res[0]/9.2e-6;
00513     //cout<<"RATIO " <<ratio<<endl;
00514     return g1.loglikelihood(r1)+0*g2.loglikelihood(res[0]);
00515 }

```

```

00516
00517 double width(const parameters & p, int option=0) const{
00518     double tanb=p[0].value;
00519     double y=p[1].value;
00520     double z=p[2].value;
00521     double w=p[3].value;
00522     double McH=y, MR=z, MI=w;
00523
00524     double y0=y;
00525     if(y<mt_mt) y0=mt_mt;
00526     double QCD1[2]={inter3.Eval(y0),inter1.Eval(y)};
00527     double QCD2[2]={inter4.Eval(y0),inter2.Eval(y)};
00528
00529     double Mu[3],Md[3];
00530
00531     for(uint i=0;i<3;i++){
00532         Mu[i]=Mu_[i].Eval(log(y));
00533         Md[i]=Md_[i].Eval(log(z));
00534     }
00535     const uint i=ii;
00536     CD CC7[2],DD7[2],CC8[2],DD8[2];
00537     double res[2];
00538     // constexpr double C7SM_[2]={C7SM_Mt,C7SM_Mb};
00539     // constexpr double C8SM_[2]={C8SM_Mt,C8SM_Mb};
00540
00541     std::array<double,24> ret;
00542     const int n=1,m=24;
00543     fp(&n,&(tanb),&m,&(ret[0]));
00544     for(uint j=0;j<2;j++){
00545         const double mbottom=Md[i];
00546         const double mstrange=Md[j];
00547         //ex mbottom=mixes.M[tQuark][iDown][i][i];
00548         //ex mstrange=mixes.M[tQuark][iDown][j][j];
00549
00550         CD C7,D7,C8,D8;
00551         for(uint k=0;k<3;k++){
00552             double mup=Mu[k];
00553             double mdown=Md[k];
00554             //ex mup=mixes.M[tQuark][iUp][k][k];
00555             //ex mdown=mixes.M[tQuark][iDown][k][k];
00556             //f1+=
00557             double mmu=std::pow(mup/McH,2);
00558             double mmdR=std::pow(mdown/MR,2);
00559             double mmdI=std::pow(mdown/MI,2);
00560             double A0u=0,A1u=0, A2u=0, A3u=0, A0d=0, A1d=0;
00561
00562             if(option==0 || option==1){
00563                 A0u=A0(mmu);
00564                 A1u=A1(mmu);
00565                 A2u=A2(mmu);
00566                 A3u=A3(mmu);
00567             }
00568             if(option==0 || option==2){
00569                 A0d=(A0(mmdR)+A0(mmdI));
00570                 A1d=(A1(mmdR)-A1(mmdI));
00571             }
00572             if(option==3){
00573                 A0d=(A0(mmdR));
00574                 A1d=(A1(mmdR));
00575             }
00576             if(option==4){
00577                 A0d=(A0(mmdI));
00578                 A1d=(-A1(mmdI));
00579             }
00580
00581             CD f1(ret[j*12+4*k+0],ret[j*12+4*k+1]);
00582             C7+=f1*A2u;
00583             C8+=-2.0*f1*A0u;
00584
00585             CD f2=f1*mstrange*mbottom/mup/mup;
00586             D7+=f2*A2u;
00587             D8+=-2.0*f2*A0u;
00588
00589             C7+=-f1*A3u;
00590             C8+=-2.0*f1*A1u;
00591
00592             CD f4(ret[j*12+4*k+2],ret[j*12+4*k+3]);
00593             C7+=f4*A0d/3.0;
00594             C8+=-f4*A0d;
00595
00596             C7+=f4*A1d/3.0;
00597             C8+=-f4*A1d;
00598
00599             CD f6=f4*mstrange*mbottom/mdown/mdown;
00600             D7+=f6*A0d/3.0;
00601             D8+=-f6*A0d;
00602

```

```

00603     }
00604     uint j0=j;
00605     CC7[j]=(QCD1[j]*C7+QCD2[j]*C8)/2.0/conj(Vud[2][j])/Vud[2][i];
00606     DD7[j]=(QCD1[j]*D7+QCD2[j]*D8)/2.0/conj(Vud[2][j])/Vud[2][i];
00607     const double QCD3=(3*QCD2[j]/8+QCD1[j]);
00608     CC8[j]=QCD3*C8/2.0/conj(Vud[2][j])/Vud[2][i];
00609     DD8[j]=QCD3*D8/2.0/conj(Vud[2][j])/Vud[2][i];
00610     const CD epsilon=conj(Vud[0][j])*Vud[0][i]/conj(Vud[2][j])/
Vud[2][i];
00611     const double epsilon=norm(conj(Vud[2][j])*Vud[2][i]/Vud[1][i]);
00612     const CD R7=(C7SM_Mt+CC7[j])/C7SM_MW;
00613     const CD R8=(C8SM_Mt+CD(0)*CC8[j])/C8SM_MW;
00614     const CD R7_=(DD7[j])/C7SM_MW;
00615     const CD R8_=CD(0)*(DD8[j])/C8SM_MW;
00616
00617
00618     res[j]=a+aee*norm(epsilon)+aer*epsilon.real()+aei*epsilon.imag();
00619     res[j]+=a77*(norm(R7)+norm(R7_))+a7r*R7.real()+a7i*R7.imag();
00620     res[j]+=a88*(norm(R8)+norm(R8_))+a8r*R8.real()+a8i*R8.imag();
00621     res[j]+=a87r*(R8*conj(R7)+R8_*conj(R7_)).real()+a7er*(R7*conj(epsilon)).real()+a8er*(R8*
conj(epsilon)).real();
00622     res[j]+=a87i*(R8*conj(R7)+R8_*conj(R7_)).imag()+a7ei*(R7*conj(epsilon)).imag()+a8er*(R8*
conj(epsilon)).imag();
00623     res[j]*=calN/100*epsilon;
00624
00625     /*res[j]=a+aee*norm(epsilon)+aer*epsilon.real()+aei*epsilon.imag();
00626     res[j]+=a77*(norm(R7)+norm(R7_))+a7r*1+a7i*0;
00627     res[j]+=a88*(norm(R8)+norm(R8_))+a8r*R8.real()+a8i*R8.imag();
00628     res[j]+=a87r*1+a7er*(conj(epsilon)).real()+a8er*(R8*conj(epsilon)).real();
00629     res[j]+=a87i*0+a7ei*(conj(epsilon)).imag()+a8er*(R8*conj(epsilon)).imag();
00630     res[j]*=calN/100*epsilon;
00631     */
00632 }
00633 double r1=3.15e-4+0.00247*(norm(CC7[1])+norm(DD7[1])-0.706*CC7[1].real());
00634
00635 //ratio=res[0]/9.2e-6;
00636 //cout<<"RATIO " <<ratio<<endl;
00637 return gl.error(r1);
00638 }
00639
00640 double A0(double x) const{
00641     return x*(2+3*x-6*x*x+ x*x*x+6*x*std::log(x))/(24*std::pow(1-x,4));
00642 }
00643
00644 double A1(double x) const{
00645     return x*(-3+4*x-x*x-2*std::log(x))/(4*std::pow(1-x,3));
00646 }
00647
00648 double A2(double x) const{
00649     return x/(6*std::pow(1-x,3))*((-7+5*x+8*x*x)/6.0+x*std::log(x)/(1-x)*(-2+3*x));
00650 }
00651
00652 double A3(double x) const{
00653     return (-3+8*x-5*x*x+(6*x-4)*std::log(x))*x/(6*std::pow(1-x,3));
00654 }
00655
00656 ROOT::Math::Interpolator inter1, inter2, inter3, inter4;
00657 ROOT::Math::Interpolator Mu_[3],Md_[3];
00658
00659 const uint ii;
00660 FUNCP_CUBA fp;
00661 const gauss2obs gl,g2;
00662 mutable double ratio;
00663
00664 };
00665
00666
00667
00668
00669 /**
00670 * @brief calculus of the constraints coming from the B->mu mu decay
00671 */
00672 class calcuBmumu:public calcu{
00673     public:
00674     calcuBmumu(const Mixes& mix, const Meson & m,const
Fermion& f3, const Fermion& f4, observable *ob, const char * name):
00675         meson(m),ff3(f3),ff4(f4),
00676         o(ob),
00677         gSr("gSr"),gSi("gSi"),gPr("gPr"),gPi("gPi"),gAr("gAr"),gAi("gAi"), mixes(mix) {
00678         const ex Nq=mixes.N[tQuark][m.q2.isospin][m.
q1.flavour][m.q2.flavour];
00679         const ex Nq_=mixes.N[tQuark][m.q2.isospin][m.
q1.flavour][m.q2.flavour].conjugate();
00680         const ex Nl=mixes.N[tLepton][f3.isospin][f3.
flavour][f4.flavour];
00681         const ex Nl_=mixes.N[tLepton][f3.isospin][f4.
flavour][f3.flavour].conjugate();

```

```

00682      possymbol MR("MR"), MI("MI"), McH("McH");
00683      ex MR2=MR*MR, MI2=MI*MI, McH2=McH*McH;
00684
00685      ex cLL=Nq_*Nl_*(1/MR2-1/MI2);
00686      ex cLR=Nq_*Nl_*(1/MR2+1/MI2);
00687      ex cRL=Nq_*Nl_*(1/MR2+1/MI2);
00688      ex cRR=Nq_*Nl_*(1/MR2-1/MI2);
00689
00690      ex ggS=-(2*M_GF/sqrt(2)*(-cRL-cRR+cLL+cLR)/4).subs(mixes.replacements).evalf();
00691      ex ggP=-(2*M_GF/sqrt(2)*(+cRL-cRR-cLL+cLR)/4).subs(mixes.replacements).evalf();
00692      CD ggA=0;
00693      if(m.q2.isospin==iDown && m.q2.flavour==2 && f3.
flavour==1 && f4.flavour==1){
00694          ggA=-conj(Vud[2][m.q2.flavour])*Vud[2][m.
q1.flavour]*Y(std::pow(M_Mu[2]/M_MW,2));
00695          ggA+=-conj(Vud[1][m.q2.flavour])*Vud[1][m.
q1.flavour]*Y(std::pow(M_Mu[1]/M_MW,2));
00696          ggA*=M_GF*M_GF*M_MW*M_MW/M_PI/M_PI/2;
00697
00698      }
00699      //ex gggA=0;
00700      //ex gggA=ggA.real()+I*ggA.imag();
00701
00702      ex width=collect_common_factors(mesondwtest().subs(lst(gAr==ggA.real(),gAi==ggA.
imag(),gSr==ggS.real_part(),gSi==ggS.imag_part(),gPr==ggP.real_part(),gPi==ggP.imag_part()).subs(mixes.
replacements).evalf().real_part());
00703
00704      compile_ex(lst(width), lst(mixes.tanb,McH,MR,MI), fp);
00705
00706      }
00707      double operator()(const parameters & p) const{
00708          //double
factor=std::pow(M_GF*M_MW,4)/8/std::pow(M_PI,5)*std::sqrt(MM*MM-4*M_Ml[1]*M_Ml[1])*M_Ml[1]*M_Ml[1];
00709          int n=4,m=1;
00710          double ret=0;
00711          fp(&n,&(p.values[0]),&m,&ret);
00712
00713          return o->loglikelihood(ret);
00714      }
00715
00716      double obsvalue(const parameters & p) const{
00717          //double
factor=std::pow(M_GF*M_MW,4)/8/std::pow(M_PI,5)*std::sqrt(MM*MM-4*M_Ml[1]*M_Ml[1])*M_Ml[1]*M_Ml[1];
00718          int n=4,m=1;
00719          double ret=0;
00720          fp(&n,&(p.values[0]),&m,&ret);
00721
00722          return ret;
00723      }
00724
00725      double Y(double x) const{
00726          return 1.0113*x/8/(1-x)*(4-x+3*x*log(x)/(1-x));
00727      }
00728
00729      ex mesondwtest() const{
00730          const Fermion& f1(meson.q2), f2(meson.q1);
00731          ex mesonmass=meson.mass;
00732
00733          Fermion f3=ff3, f4=ff4;
00734          realsymbol q3("q3"), q4("q4");
00735          ex s2=pow(mesonmass,2);
00736
00737          ex v1=0, v2=0;
00738          ex mq1=mixes.mass(f1),mq2=mixes.mass(f2);
00739          ex mq3=mixes.mass(f3),mq4=mixes.mass(f4);
00740
00741          ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;
00742          scalar_products sp;
00743          sp.add(q4, q3, (s2-m2q4-m2q3)/2);
00744          sp.add(q3, q3, m2q3);
00745          sp.add(q4, q4, m2q4);
00746          ex q10=(s2+mq1*mq1-mq2*mq2)/(2*sqrt(s2)), lq1l=sqrt(q10*q10-mq1*mq1);
00747          ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);
00748
00749          ex a;
00750          a=-(gSr+I*gSi)*s2/(mq1+mq2);
00751          v1=v1+a*dirac_ONE();
00752          v2=v2+a.conjugate()*dirac_ONE();
00753          a=-(gPr+I*gPi)*s2/(mq1+mq2);
00754          v1=v1+a*dirac_gamma5();
00755          v2=v2-a.conjugate()*dirac_gamma5();
00756          ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
00757          a=(gAr+I*gAi);
00758          v1=v1+a*sl*dirac_gamma5();
00759          v2=v2+a.conjugate()*sl*dirac_gamma5();
00760
00761          ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)-mq4*dirac_ONE();

```

```

00762         ex dt=dirac_trace(vq3*v1*vq4*v2).simplify_indexed(sp);
00763         ex result=expand(dt*4*lq3l/s2/Pi/32);
00764
00765         lst ltest;
00766         //ltest.append(conjugate(gL)==pow(abs(gL),2)/gL);
00767         //ltest.append(conjugate(gR)==pow(abs(gR),2)/gR);
00768         //ltest.append(conjugate(gS)==pow(abs(gS),2)/gS);
00769         //ltest.append(conjugate(gP)==pow(abs(gP),2)/gP);
00770         //ltest.append(conjugate(gA)==pow(abs(gA),2)/gA);
00771
00772         return pow(meson.decay_factor,2)*collect_common_factors(result.subs(ltest));
00773         //return
00774     }
00775     const Meson meson;
00776     const Fermion& ff3, ff4;
00777     shared_ptr<observable> o;
00778     const realsymbol gSr, gSi, gPr, gPi, gAr, gAi;
00779     const Mixes mixes;
00780     FUNCP_CUBA fp;
00781
00782 };
00783
00784
00785
00786 /*
00787
00788 #include "defs.h"
00789
00790
00791 //constexpr double Nll(int i, int j){return UL?(i==j)*(tanb*M_Ml[i]+(-tanb-1/tanb)*M_Ml[i]*(i==GL):(i==j)*
00792 tanb*M_Ml[i]+(-tanb-1/tanb)*conj(Vnl(GL,i))*Vnl(GL,j)*M_Ml[j]);}
00793
00794 #if UL==1
00795     #define Nl(i,j) ((i==j)*(tanb*M_Ml[i]+(-tanb-1/tanb)*M_Ml[i]*(i==GL)))
00796     #define VnlNl(i,j) (Vnl[i][j])*M_Ml[j]*(tanb+(-tanb-1/tanb)*(j==GL))
00797 #else
00798     #define Nl(i,j) ((i==j)*tanb*M_Ml[i]+(-tanb-1/tanb)*conj(Vnl[GL][i])*Vnl[GL][j]*M_Ml[j])
00799     #define VnlNl(i,j) (Vnl[i][j]*M_Ml[j]*(tanb+(-tanb-1/tanb)*(i==GL)))
00800 #endif
00801
00802 #if UQ==1
00803     #define Nd(i,j) (M_Md[j]*(i==j)*(tanb+(-tanb-1/tanb)*(j==GQ))
00804     #define Nu(i,j) (M_Mu[j]*(i==j)*tanb+(-tanb-1/tanb)*Vud[i][GQ]*conj(Vud[j][GQ]))
00805     #define VudNd(i,j) (Vud[i][j]*M_Md[j]*(tanb+(-tanb-1/tanb)*(j==GQ))
00806     #define NuVud(i,j) (-M_Mu[i]*Vud[i][j]*(tanb+(-tanb-1/tanb)*(j==GQ))
00807 #else
00808     #define Nd(i,j) (M_Md[j]*((i==j)*tanb+(-tanb-1/tanb)*conj(Vud[GQ][i])*Vud[GQ][j]))
00809     #define Nu(i,j) (M_Mu[j]*(i==j)*(tanb+(-tanb-1/tanb)*M_Mu[j]*(j==GQ))
00810     #define VudNd(i,j) (Vud[i][j]*M_Md[j]*(tanb+(-tanb-1/tanb)*(i==GQ))
00811     #define NuVud(i,j) (-M_Mu[i]*Vud[i][j]*(tanb+(-tanb-1/tanb)*(i==GQ))
00812 #endif
00813
00814 */
00815 }
00816 #endif

```

## 8.13 Juca.cpp File Reference

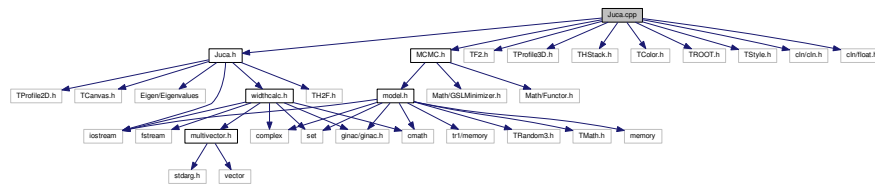
```

#include "MCMC.h"
#include "Juca.h"
#include "TF2.h"
#include "TProfile3D.h"
#include "THStack.h"
#include "TColor.h"
#include "TROOT.h"
#include "TStyle.h"
#include <cln/cln.h>
#include <cln/float.h>

```



Include dependency graph for Juca.cpp:



## Functions

- `int main()`

### 8.13.1 Function Documentation

#### 8.13.1.1 `int main()`

Definition at line 16 of file `Juca.cpp`.

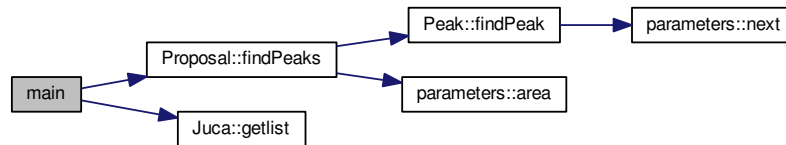
References `Proposal::findPeaks()`, `Proposal::floatPeak`, `Juca::getlist()`, and `Peak::pr`.

```

00016     {
00017         Digits=5;
00018         cln::cl_inhibit_floating_point_underflow=1;
00019
00020         //Int_t MyPalette[100];
00021         Double_t r[] = {1, 0};
00022         Double_t g[] = {1, 0};
00023         Double_t b[] = {1, 0};
00024         Double_t stop[] = {0., 1.0};
00025         TColor::CreateGradientColorTable(2, stop, r, g,b, 100);
00026
00027         //TH1F * pdf1=new TH1F("pdf1","pdf1",npoints,10,500);
00028
00029
00030         //TGraph * chi2=new TGraph(npoints);
00031
00032         uint npoints=50;
00033
00034
00035         double llmax=-20,gmax=0;
00036
00037         Juca* m=new Juca();
00038         Proposal prop4(m);
00039         prop4.findPeaks();
00040
00041         cout<<"gp "<<m->gaussprob(prop4.floatPeak.pr)<<endl;
00042         lst l=m->getlist(prop4.floatPeak.pr);
00043         for(uint i=0; i< m->size();i++){
00044             double mean=m->at(i).calculate(1);
00045             cout<<i<<" "<<mean<<" "<<sqrt(2*m->at(i).o->loglikelihood(mean))<<endl;
00046         }
00047         for(uint i=0; i< prop4.floatPeak.pr.size();i++){
00048             cout<<i<<" "<<prop4.floatPeak.pr[i].value<<endl;
00049         }
00050         delete m;
00051
00052         return 0;
00053     }

```

Here is the call graph for this function:



## 8.14 Juca.cpp

```

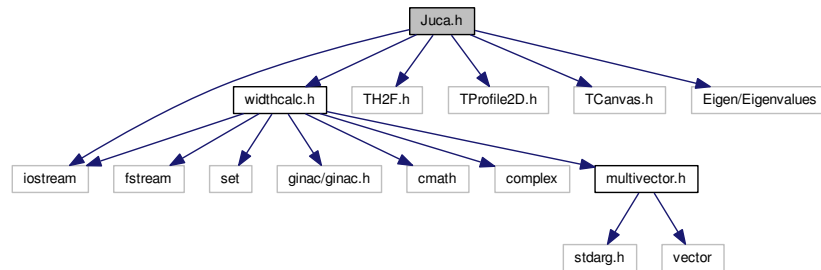
00001 #include "MCMC.h"
00002 #include "Juca.h"
00003 #include "TF2.h"
00004 #include "TProfile3D.h"
00005 #include "THStack.h"
00006 #include "TColor.h"
00007 #include "TROOT.h"
00008 #include "TStyle.h"
00009
00010
00011
00012 #include <cln/cln.h>
00013 #include <cln/float.h>
00014
00015
00016 int main(){
00017     Digits=5;
00018     cln::cl_inhibit_floating_point_underflow=1;
00019
00020     //Int_t MyPalette[100];
00021     Double_t r[] = {1, 0};
00022     Double_t g[] = {1, 0};
00023     Double_t b[] = {1, 0};
00024     Double_t stop[] = {0., 1.0};
00025     TColor::CreateGradientColorTable(2, stop, r, g,b, 100);
00026
00027     //TH1F * pdf1=new TH1F("pdf1","pdf1",npoints,10,500);
00028
00029
00030     //TGraph * chi2=new TGraph(npoints);
00031
00032     uint npoints=50;
00033
00034
00035     double llmax=-20,gmax=0;
00036
00037     Juca* m=new Juca();
00038     Proposal prop4(m);
00039     prop4.findPeaks();
00040
00041     cout<<"gp "<<m->gaussprob(prop4.floatPeak.pr)<<endl;
00042     lst l=m->getlist(prop4.floatPeak.pr);
00043     for(uint i=0; i< m->size();i++){
00044         double mean=m->at(i).calculate(1);
00045         cout<<i<<" "<<mean<<" "<<sqrt(2*m->at(i).o->loglikelihood(mean))<<endl;
00046     }
00047     for(uint i=0; i< prop4.floatPeak.pr.size();i++){
00048         cout<<i<<" "<<prop4.floatPeak.pr[i].value<<endl;
00049     }
00050     delete m;
00051
00052     return 0;
00053 }
00054

```

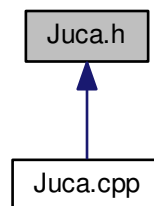
## 8.15 Juca.h File Reference

```
#include "widthcalc.h"
```

```
#include "TH2F.h"
#include "TProfile2D.h"
#include "TCanvas.h"
#include <iostream>
#include <Eigen/Eigenvalues>
Include dependency graph for Juca.h:
```



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



## Classes

- class [Juca](#)

## 8.16 Juca.h

```
00001 #ifndef JUCA_H
00002 #define JUCA_H
00003
00004 #include "widthcalc.h"
00005
00006 #include "TH2F.h"
00007 #include "TProfile2D.h"
00008 #include "TCanvas.h"
00009 #include <iostream>
00010 #include <Eigen/Eigenvalues>
00011
00012 using namespace std;
00013 using namespace Eigen;
00014
```

```

00015 class Juca: public Model{
00016 public:
00017
00018 Juca():Mu("Mu"), Md("Md"), Mc("Mc"), Ms("Ms"), Mt("Mt"), Mb("Mb"), lambda("lambda"), A("A"),rho("rho"),
    eta("eta"){
00019
00020     add("",lambda,new gauss2obs(0.22535,0.00065));
00021     add("",A,new gauss2obs(0.811, 0.017));
00022     add("",rho,new gauss2obs(0.131, 0.02));
00023     add("",eta,new gauss2obs(0.345, 0.014));
00024     add("",Mu,new gauss2obs(1.27e-3, 0.46e-3));
00025     add("",Md,new gauss2obs(2.9e-3, 1.22e-3));
00026     add("",Ms,new gauss2obs(55e-3, 16e-3));
00027     add("",Mc,new gauss2obs(0.619, 0.084));
00028     add("",Mb,new gauss2obs(2.89, 0.09));
00029     add("",Mt,new gauss2obs(171.7, 3.0));
00030 }
00031
00032 ~Juca(){}
00033
00034
00035
00036 void add(const char * s, ex pred, observable * ob){
00037     ex p=collect_common_factors(expand(pred.subs(replacements)));
00038     push_back(prediction(ob,p));
00039 }
00040
00041 parameters generateparameters() const{
00042     parameters p;
00043
00044     //buu
00045     p.push_back(freeparameter(-5,0,r));
00046     //auu
00047     p.push_back(freeparameter(-5,0,r));
00048     //add
00049     p.push_back(freeparameter(-5,0,r));
00050     //bdd
00051     p.push_back(freeparameter(-5,0,r));
00052     //eu
00053     p.push_back(freeparameter(-2,2,r));
00054     //gu
00055     p.push_back(freeparameter(-2,2,r));
00056     //ed
00057     p.push_back(freeparameter(-2,2,r));
00058     //gd
00059     p.push_back(freeparameter(-2,2,r));
00060     //nu
00061     p.push_back(freeparameter(-2,3,r));
00062     //nd
00063     p.push_back(freeparameter(-2,3,r));
00064     //cuu
00065     //p.push_back(freeparameter(-5,0,r));
00066     //cdd
00067     p.push_back(freeparameter(-5,0,r));
00068     //hu
00069     //p.push_back(freeparameter(-2,2,r));
00070     //hd
00071     p.push_back(freeparameter(-2,2,r));
00072
00073     return p;
00074 }
00075
00076
00077 1st getlist(const parameters & p) const{
00078
00079
00080     double buu = pow(10.0,p[0].value), auu = pow(10.0,p[1].value);
00081     double add = pow(10.0,p[2].value), bdd = pow(10.0,p[3].value);
00082     double eu =p[4].value, gu = p[5].value;
00083     double ed = p[6].value, gd = p[7].value;
00084     //double cuu = pow(10.0,p[10].value);
00085     double cdd = pow(10.0,p[10].value);
00086     //double hu =p[12].value;
00087     double hd= p[11].value;
00088
00089     complex<double> bu = buu*exp(complex<double>(0,gu*M_PI_2)), au = auu*exp(complex<double>(0,eu*M_PI_2));
00090     complex<double> ad = add*exp(complex<double>(0,ed*M_PI_2)), bd = bdd*exp(complex<double>(0,gd*M_PI_2));
00091     //complex<double> cu = cuu*exp(complex<double>(0,hu*M_PI_2));
00092     complex<double> cu = bu;
00093     complex<double> cd = cdd*exp(complex<double>(0,hd*M_PI_2));
00094     //complex<double> cd = bd;
00095
00096     //Matrix3cd X = Matrix3cd::Random(3,3);
00097     Matrix3cd mu,md;
00098     mu<<1,1.0,1.0+au+bu,1.0,1.0,1.0+bu,1.0+bu+au,1.0+bu,1.0+cu;
00099     md<<1,1.0,1.0+ad+bd,1.0,1.0,1.0+bd,1.0+bd+ad,1.0+bd,1.0+cd;
00100     mu=mu*pow(10.0,p[8].value);

```

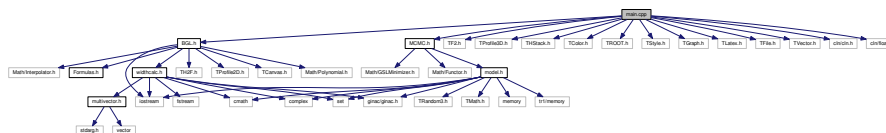
```

00101 md=md*pow(10.0,p[9].value);
00102
00103 Matrix3cd Hu = mu*mu.adjoint(), Hd = md*md.adjoint();
00104 SelfAdjointEigenSolver<Matrix3cd> usolver(Hu), dsolver(Hd);
00105 Vector3d Du=usolver.eigenvalues();//.asDiagonal();
00106 Vector3d Dd=dsolver.eigenvalues();//.asDiagonal();
00107 Matrix3cd VLd=dsolver.eigenvectors().adjoint();
00108 Matrix3cd VLu=usolver.eigenvectors().adjoint();
00109 Matrix3cd Vckm=VLu*VLd.adjoint();
00110 double lambda0=sqrt(norm(Vckm(0,1))/(norm(Vckm(0,0))+norm(Vckm(0,1))));
00111 double A0=abs(Vckm(1,2))/abs(Vckm(0,1))/lambda0;
00112 complex<double> rhoeta=-Vckm(0,0)*conj(Vckm(0,2))/Vckm(1,0)/conj(Vckm(1,2));
00113 double rho0=real(rhoeta), eta0=imag(rhoeta);
00114
00115     return lst(lambda==lambda0,A==A0,rho==rho0,eta==eta0,Mu==sqrt(abs(Du[0])),Md==sqrt(abs(Dd[0])),Mc==
sqrt(abs(Du[1])),Ms==sqrt(abs(Dd[1])),Mt==sqrt(abs(Du[2])),Mb==sqrt(abs(Dd[2])));
00116 }
00117
00118 const possymbol Mu, Md, Mc, Ms, Mt, Mb, lambda, A;
00119 const realsymbol rho, eta;
00120
00121 lst replacements;
00122 };
00123
00124
00125
00126 #endif

```

```
#include "MCMC.h"
#include "BGL.h"
#include "TF2.h"
#include "TProfile3D.h"
#include "THStack.h"
#include "TColor.h"
#include "TROOT.h"
#include "TStyle.h"
#include "TGraph.h"
#include "TLatex.h"
#include "TFile.h"
#include "TVector.h"
#include <cln/cln.h>
#include <cln/float.h>
```

Include dependency graph for main.cpp:



- `int main (int argc, char *argv[])`

*the main function takes the arguments gL gQ lup qup which specify a BGL model and runs the simulation for that model, generating a ROOT file as the output*

## 8.17.1 Function Documentation

### 8.17.1.1 `int main ( int argc, char * argv[] )`

the main function takes the arguments gL gQ lup qup which specify a BGL model and runs the simulation for that model, generating a ROOT file as the output

Definition at line 57 of file [main.cpp](#).

References [BGLmodels::BGL::BR\\_Htotaunu](#), [BGLmodels::BGL::cBmumu](#), [BGLmodels::BGL::cBsmumu](#), [Proposal::findPeaks\(\)](#), [Proposal::floatPeak](#), [BGLmodels::BGL::generateparameters\(\)](#), [BGLmodels::BGL::getlist\(\)](#), [Proposal::getNextPoint\(\)](#), [Peak::lmax](#), [Model::loglike\(\)](#), [BGLmodels::BGL::mmax](#), [BGLmodels::calcuBmumu::obsvalue\(\)](#), [parameters::p](#), [BGLmodels::BGL::planck](#), [Peak::pr](#), and [BGLmodels::BGL::stepsize](#).

```

00057                                     {
00058     // Check the number of parameters
00059     if(argc < 5){
00060         std::cerr<<"Usage: "<<argv[0]<<" gL gQ lup qup [mssm]"<<std::endl;
00061         return 1;}
00062
00063
00064     int gL=atoi(argv[1]);
00065     int gQ=atoi(argv[2]);
00066     int lup=atoi(argv[3]);
00067     int qup=atoi(argv[4]);
00068     int mssm=0;
00069     if(argc>5) mssm=atoi(argv[5]);
00070
00071
00072     Digits=5;
00073     cln::cl_inhibit_floating_point_underflow=1;
00074
00075     string ll[2][3]={{"#nu_{1}", "#nu_{2}", "#nu_{3}"}, {"e", "#mu", "#tau"}};
00076     string qq[2][3]={{"u", "c", "t"}, {"d", "s", "b"}};
00077     //Int_t MyPalette[100];
00078     Double_t r[] = {1, 0.3};
00079     Double_t g[] = {1, 0.3};
00080     Double_t b[] = {1, 0.3};
00081     Double_t stop[] = {0., 1.0};
00082     TColor::CreateGradientColorTable(2, stop, r, g,b, 100);
00083     //TH1F * pdf1=new TH1F("pdf1","pdf1",npoints,10,500);
00084     //TGraph * chi2=new TGraph(npoints);
00085
00086     uint npoints=200;
00087     double init1=-3, final1=3;
00088     double init2=10, final2=1000;
00089     double initBmumu=0, finalBmumu=2;
00090     double initBsmumu=0, finalBsmumu=5;
00091
00092     multivector<BGL *,4> ms(0,3,3,2,2);
00093     multivector<parameters,2> plots(parameters(), npoints, npoints);
00094     multivector<double,2> likely(-1000, npoints, npoints);
00095     multivector<double,2> likelyB(-1000, npoints, npoints);
00096     multivector<double,2> likely2(-1000, npoints, npoints);
00097     multivector<double,2> likely3(-1000, npoints, npoints);
00098     multivector<double,2> likely4(-1000, npoints, npoints);
00099     multivector<double,2> likely5(-1000, npoints, npoints);
00100     multivector<double,2> likely6(-1000, npoints, npoints);
00101
00102     //ofstream mass("mass.out");
00103
00104     //for(int gL=2;gL>=0;gL--)
00105     //for(int gQ=2;gQ>=0;gQ--)
00106     //for(int lup=0;lup<2;lup++)
00107     //for(int qup=0;qup<2;qup++){
00108
00109     //    for(uint qup=0;qup<2;qup++)
00110     //for(uint gL=0;gL<3;gL++)
00111     //for(uint lup=0;lup<2;lup++)
00112     //for(uint gQ=0;gQ<3;gQ++){
00113         //if(gL==0 && gQ==2 && lup==0 && qup==1) {t=1; continue;}
00114         //if(t==0) continue;
00115         double llmax=-1000, gmax=0, McHmax=1000, MRmax=1000, MImax=1000, tbmax=1;
00116         BGL* m=new BGL(gL, gQ, lup, qup);
00117         m->mmax=300;
00118         ms[gL][gQ][lup][qup]=m;
00119
00120         char name[5]="0000";

```

```

00121         name[0]+=gL;
00122         name[1]+=gQ;
00123         name[2]+=lup;
00124         name[3]+=qup;
00125
00126         /*
00127         TF1 * f1 = new TF1("f1",m,&BGL::BranchingRatio,-3,3,0,"BGL","BranchingRatio");
00128
00129         TCanvas * c1=new TCanvas("c1","",800,600);
00130         f1->Draw();
00131         c1->SaveAs("BR.png");
00132         TF2 * f2 = new TF2("f2",m,&BGL::topBranchingRatio,-3,3,80,175,0,"BGL","topBranchingRatio");
00133
00134         TCanvas * c3=new TCanvas("c3","",800,600);
00135         f2->Draw("colz");
00136         c3->SaveAs("topBR.png");
00137         */
00138
00139         Proposal prop4(m);
00140         //m->stepsize=1e-4;
00141         prop4.findPeaks(100,1);
00142         llmax=log(prop4.floatPeak.lmax);
00143         tbmax=prop4.floatPeak.pr[0].value;
00144         McHmax=prop4.floatPeak.pr[1].value;
00145         MRmax=prop4.floatPeak.pr[2].value;
00146         MImax=prop4.floatPeak.pr[3].value;
00147
00148         m->stepsize=1e-4;
00149         prop4.findPeaks(100);
00150         double llmax_=log(prop4.floatPeak.lmax);
00151         if(llmax_>llmax) {llmax=llmax_;
00152             tbmax=prop4.floatPeak.pr[0].value;
00153             McHmax=prop4.floatPeak.pr[1].value;
00154             MRmax=prop4.floatPeak.pr[2].value;
00155             MImax=prop4.floatPeak.pr[3].value;
00156         }
00157
00158         TH2F * limits4=new TH2F("limits4","Likelihood",npoints,init1,final1,npoints,init2,final2);
00159         TH2F * Bmumu_Bsmumu=new TH2F("Bmumu_Bsmumu","Likelihood",npoints,initBmumu,finalBmumu,npoints,
initBsmumu,finalBsmumu);
00160         TH2F * limits_tb_MR=new TH2F("limits_tb_MR","Likelihood",npoints,init1,final1,npoints,init2,final2)
;
00161         TH2F * limits_tb_MI=new TH2F("limits_tb_MI","Likelihood",npoints,init1,final1,npoints,init2,final2)
;
00162         TH2F * limits_MR_MI=new TH2F("limits_MR_MI","Likelihood",npoints,init2,final2,npoints,init2,final2)
;
00163         TH2F * limits_MR_McH=new TH2F("limits_MR_McH","Likelihood",npoints,init2,final2,npoints,init2,
final2);
00164         TH2F * limits_MI_McH=new TH2F("limits_MI_McH","Likelihood",npoints,init2,final2,npoints,init2,
final2);
00165
00166
00167         for(uint i=0;i<npoints;i++)
00168             for(uint j=0;j<npoints;j++) {
00169             limits4->SetBinContent(i+1,j+1,-1000);
00170             Bmumu_Bsmumu->SetBinContent(i+1,j+1,-1000);
00171             limits_MR_McH->SetBinContent(i+1,j+1,-1000);
00172             limits_MI_McH->SetBinContent(i+1,j+1,-1000);
00173             limits_MR_MI->SetBinContent(i+1,j+1,-1000);
00174             limits_tb_MR->SetBinContent(i+1,j+1,-1000);
00175             plots[i][j]=m->generateparameters();
00176         }
00177         uint steps=40e6;
00178         //uint steps=npoints*npoints;
00179         double brtaunu=1;
00180         for(uint i=steps;i--){
00181             //prop1.getNextPoint();
00182             //prop2.getNextPoint();
00183             //prop3.getNextPoint();
00184             double total=0;
00185             double gp=0;
00186             //if(i==steps) cout<<" total "<<m->loglike(prop4.floatPeak.pr)<<endl;
00187             //else{
00188             if(i>npoints*npoints){
00189                 if(i==steps/2) {
00190                     m->mmmax=1000;
00191                     m->stepsize=1e-2;
00192                     prop4.findPeaks(100);
00193                     llmax_=log(prop4.floatPeak.lmax);
00194                     if(llmax_>llmax) {llmax=llmax_;
00195                         tbmax=prop4.floatPeak.pr[0].value;
00196                         McHmax=prop4.floatPeak.pr[1].value;
00197                         MRmax=prop4.floatPeak.pr[2].value;
00198                         MImax=prop4.floatPeak.pr[3].value;
00199                     }
00200                 }
00201                 if(i<steps) prop4.getNextPoint();

```

```

00202         total=log(prop4.floatPeak.lmax);
00203         //total=m->loglike(prop4.floatPeak.pr);
00204         //gp=m->gaussprob(prop4.floatPeak.pr);
00205     }
00206     else{
00207         uint x=(i-1)/npoints, y=(i-1)%npoints;
00208         prop4.floatPeak.pr[0].value=init1+((x+0.5)*(final1-init1))/npoints;
00209         prop4.floatPeak.pr[1].value=init2+((y+0.5)*(final2-init2))/npoints;
00210         prop4.floatPeak.pr[2].value=0;
00211         prop4.floatPeak.pr[3].value=0;
00212         total=m->loglike(prop4.floatPeak.pr);
00213     }
00214     //}
00215
00216
00217     //gp=m->gaussprob_noT(prop4.floatPeak.pr);
00218     //double Btaunu=m->loglike(prop1.floatPeak.pr,m->iBtaunu);
00219     //double BDtaunu=m->loglike(prop2.floatPeak.pr,m->iBDtaunu);
00220     //double BD2taunu=m->loglike(prop3.floatPeak.pr,m->iBD2taunu);
00221     double brtaunu0=ex_to<numeric>(m->BR_Htotaunu.subs(m->
getlist(prop4.floatPeak.pr).p).evalf()).to_double();
00222     if(brtaunu0<brtaunu) brtaunu=brtaunu0;
00223     if(total>llmax) {llmax=total; gmax=gp;
00224         tbmax=prop4.floatPeak.pr[0].value;
00225         McHmax=prop4.floatPeak.pr[1].value;
00226         MRmax=prop4.floatPeak.pr[2].value;
00227         MImax=prop4.floatPeak.pr[3].value;
00228     }
00229
00230
00231
00232     uint p1=uint((prop4.floatPeak.pr[0].value-init1)/(final1-init1)*npoints);
00233     double mr=prop4.floatPeak.pr[1].value;
00234     uint p2=uint((mr-init2)/(final2-init2)*npoints);
00235     mr+=prop4.floatPeak.pr[2].value;
00236     uint p3=uint((prop4.floatPeak.pr[2].value+prop4.floatPeak.pr[1].value-init2)/(
final2-init2)*npoints);
00237     mr+=prop4.floatPeak.pr[3].value;
00238     uint p4=uint((mr-init2)/(final2-init2)*npoints);
00239     if(p1<npoints && p2<npoints && p3<npoints && p4<npoints){
00240         if(total>likely[p1][p2]){
00241             likely[p1][p2]=total;
00242             plots[p1][p2].setvalues(prop4.floatPeak.pr);
00243             limits4->SetBinContent(p1+1,p2+1,total);
00244         }
00245         if(total>likely2[p3][p4]){
00246             likely2[p3][p4]=total;
00247             //plots[gL][gQ][lup][qup][p1][p2].setvalues(prop4.floatPeak.pr);
00248             limits_MR_MI->SetBinContent(p3+1,p4+1,total);
00249         }
00250         if(total>likely3[p3][p2]){
00251             likely3[p3][p2]=total;
00252             //plots[gL][gQ][lup][qup][p1][p2].setvalues(prop4.floatPeak.pr);
00253             limits_MR_McH->SetBinContent(p3+1,p2+1,total);
00254         }
00255     }
00256     if(total>likely4[p1][p3]){
00257         likely4[p1][p3]=total;
00258         //plots[gL][gQ][lup][qup][p1][p2].setvalues(prop4.floatPeak.pr);
00259         limits_tb_MR->SetBinContent(p1+1,p3+1,total);
00260     }
00261     if(total>likely5[p4][p2]){
00262         likely5[p4][p2]=total;
00263         //plots[gL][gQ][lup][qup][p1][p2].setvalues(prop4.floatPeak.pr);
00264         limits_MI_McH->SetBinContent(p4+1,p2+1,total);
00265     }
00266     parameters ppr=m->getlist(prop4.floatPeak.pr);
00267
00268     double Bmumu=m->cBmumu->obsvalue(ppr)/(1e-10*m->
planck/1.519e-12);
00270     double Bsmumu=m->cBsmumu->obsvalue(ppr)/(1e-9*m->
planck/1.516e-12);
00271     //cout<<Bmumu<<" "<<Bsmumu<<endl;
00272     uint pB=uint((Bmumu-initBmumu)/(finalBmumu-initBmumu)*npoints);
00273     uint pBs=uint((Bsmumu-initBsmumu)/(finalBsmumu-initBsmumu)*npoints);
00274     if(pB<npoints && pBs<npoints)
00275     if(total>likelyB[pB][pBs]){
00276         likely[pB][pBs]=total;
00277         Bmumu_Bsmumu->SetBinContent(pB+1,pBs+1,total);
00278     }
00279
00280     }else cout<<"OUT!!!"<<endl;
00281
00282     if(i%100000==0){
00283         cout<<"Steps "<<i<<" logtb "<<prop4.floatPeak.pr[0].value<<" logMcH "<<prop4.
floatPeak.pr[1].value;

```



```

00284         cout<<" total "<<total<<" gmax "<<gp<<" "<<brtaunu<<endl;
00285     }
00286
00287     }
00288     //}
00289
00290     TFile f((string("h")+string(name)+string(".root")).c_str(),"recreate");
00291     TVectorD v(5);
00292     v[0] = llmax;
00293     v[1]=tbmax;
00294     v[2]=McHmax;
00295     v[3]=MRmax;
00296     v[4]=MImax;
00297     v.Write("vllmax");
00298     limits4->Write();
00299     Bmumu_Bsmumu->Write();
00300     limits_MR_MI->Write();
00301     limits_MR_McH->Write();
00302     limits_MI_McH->Write();
00303     limits_tb_MR->Write();
00304     f.Close();
00305
00306     //for(int gL=2;gL>=0;gL--)
00307     //for(int gQ=2;gQ>=0;gQ--)
00308     //for(uint lup=0;lup<2;lup++)
00309     //for(uint qup=0;qup<2;qup++)
00310     uint min1=npoints, min2=npoints, min3=npoints;
00311     uint min1l=npoints, min2l=npoints, min3l=npoints;
00312     uint min12=npoints, min22=npoints, min32=npoints;
00313
00314     for(uint i=0;i<npoints;i++)
00315     for(uint j=0;j<npoints;j++){
00316         int binmax=limits4->GetBin(i+1,j+1);
00317         double rest=limits4->GetBinContent(binmax);
00318         if(rest>=llmax) rest=1;
00319         else rest=TMath::Prob(-2*(rest-llmax),2);
00320         if(rest>=0.05 && j<min1){min1=j;}
00321         if(rest>=0.05 && j<min1l && j>uint(180*npoints/990)){min1l=j;}
00322         if(rest>=0.05 && j<min12 && j>uint(400*npoints/990)){min12=j;}
00323         limits4->SetBinContent(i+1,j+1,rest);
00324
00325         rest=Bmumu_Bsmumu->GetBinContent(binmax);
00326         if(rest>=llmax) rest=1;
00327         else rest=TMath::Prob(-2*(rest-llmax),2);
00328         Bmumu_Bsmumu->SetBinContent(i+1,j+1,rest);
00329
00330         rest=limits_MR_MI->GetBinContent(binmax);
00331         if(rest>=llmax) rest=1;
00332         else rest=TMath::Prob(-2*(rest-llmax),2);
00333         limits_MR_MI->SetBinContent(i+1,j+1,rest);
00334
00335         rest=limits_MR_McH->GetBinContent(binmax);
00336         if(rest>=llmax) rest=1;
00337         else rest=TMath::Prob(-2*(rest-llmax),2);
00338         if(rest>=0.05 && i<min2){min2=i;}
00339         if(rest>=0.05 && i<min2l && j>uint(180*npoints/990)){min2l=i;}
00340         if(rest>=0.05 && i<min22 && j>uint(400*npoints/990)){min22=i;}
00341         limits_MR_McH->SetBinContent(i+1,j+1,rest);
00342
00343         rest=limits_MI_McH->GetBinContent(binmax);
00344         if(rest>=llmax) rest=1;
00345         else rest=TMath::Prob(-2*(rest-llmax),2);
00346         if(rest>=0.05 && i<min3){min3=i;}
00347         if(rest>=0.05 && i<min3l && j>uint(180*npoints/990)){min3l=i;}
00348         if(rest>=0.05 && i<min32 && j>uint(400*npoints/990)){min32=i;}
00349         limits_MI_McH->SetBinContent(i+1,j+1,rest);
00350
00351
00352         rest=limits_tb_MR->GetBinContent(binmax);
00353         if(rest>=llmax) rest=1;
00354         else rest=TMath::Prob(-2*(rest-llmax),2);
00355         limits_tb_MR->SetBinContent(i+1,j+1,rest);
00356     }
00357     double mmin1=init2+((min1+0.5)*(final2-init2))/npoints;
00358     double mmin2=init2+((min2+0.5)*(final2-init2))/npoints;
00359     double mmin3=init2+((min3+0.5)*(final2-init2))/npoints;
00360
00361     double mmin1l=init2+((min1l+0.5)*(final2-init2))/npoints;
00362     double mmin2l=init2+((min2l+0.5)*(final2-init2))/npoints;
00363     double mmin3l=init2+((min3l+0.5)*(final2-init2))/npoints;
00364
00365     double mmin12=init2+((min12+0.5)*(final2-init2))/npoints;
00366     double mmin22=init2+((min22+0.5)*(final2-init2))/npoints;
00367     double mmin32=init2+((min32+0.5)*(final2-init2))/npoints;
00368
00369     //mass<<name<<" "<<mmin1<<" "<<mmin2<<" "<<mmin3<<endl;
00370

```

```

00371         ofstream maxs((string("maxs_")+string(name)+string(".out")).c_str());
00372         maxs<<mmin1<<" "<<mmin2<<" "<<mmin3<<endl;
00373         maxs<<mmin11<<" "<<mmin21<<" "<<mmin31<<endl;
00374         maxs<<mmin12<<" "<<mmin22<<" "<<mmin32<<endl;
00375         maxs<<llmax<<endl;
00376         //for(uint j=0;j<npoints;j++)
00377         //for(uint i=0;i<npoints;i++){
00378             //    int binmax=limits4->GetBin(i+1,j+1);
00379             //    maxs<<"("<<i<<" "<<j<<" "<<limits4->GetBinContent(binmax)<<endl;
00380             //    }
00381
00382         maxs.close();
00383
00384         limits4->SetStats(0);
00385         limits4->GetXaxis()->SetTitle("log_{10}(tan\\beta)");
00386         limits4->GetYaxis()->SetTitle("M_{H+} (GeV)");
00387
00388         Bmumu_Bsmumu->SetStats(0);
00389         Bmumu_Bsmumu->GetXaxis()->SetTitle("Br(B\\to\\mu\\mu)/10^{-10}");
00390         Bmumu_Bsmumu->GetYaxis()->SetTitle("Br(B_{s}\\to\\mu\\mu)/10^{-9}");
00391
00392
00393         limits_MR_MI->SetStats(0);
00394         limits_MR_MI->GetYaxis()->SetTitle("M_{I} (GeV)");
00395         limits_MR_MI->GetXaxis()->SetTitle("M_{R} (GeV)");
00396
00397
00398
00399         limits_MR_McH->SetStats(0);
00400         limits_MR_McH->GetYaxis()->SetTitle("M_{H+} (GeV)");
00401         limits_MR_McH->GetXaxis()->SetTitle("M_{R} (GeV)");
00402
00403         limits_MI_McH->SetStats(0);
00404         limits_MI_McH->GetYaxis()->SetTitle("M_{H+} (GeV)");
00405         limits_MI_McH->GetXaxis()->SetTitle("M_{I} (GeV)");
00406
00407         limits_tb_MR->SetStats(0);
00408         limits_tb_MR->GetXaxis()->SetTitle("log_{10}(tan\\beta)");
00409         limits_tb_MR->GetYaxis()->SetTitle("M_{R} (GeV)");
00410
00411
00412         gStyle->SetOptTitle(0);
00413
00414         Double_t contours[3];
00415         contours[0] = 0.003;
00416         contours[1] = 0.05;
00417         contours[2] = 0.32;
00418
00419
00420         double ma=0,me=.2, x0=1,y0=120;
00421
00422         limits4->SetContour(3, contours);
00423         //limits4->GetYaxis()->SetLabelOffset(0.02);
00424         limits4->GetYaxis()->SetLabelSize(0.08);
00425         limits4->GetYaxis()->SetTitleSize(0.08);
00426         limits4->GetYaxis()->SetTitleOffset(1.2);
00427         limits4->GetYaxis()->SetLimits(1,999);
00428         //limits4->GetXaxis()->SetLabelOffset(0.02);
00429         limits4->GetXaxis()->SetLabelSize(0.08);
00430         limits4->GetXaxis()->SetTitleSize(0.08);
00431         limits4->GetXaxis()->SetTitleOffset(1.2);
00432         limits4->GetXaxis()->SetLimits(-2.99,2.99);
00433
00434
00435
00436
00437         gStyle->SetPaperSize(10.,10.);
00438
00439         TLatex l;
00440         l.SetTextSize(0.08);
00441         string ss=qq[qup][gQ]+" "+l1[lup][gL];
00442
00443         TCanvas * c21=new TCanvas("c21","",int(800*(1+ma+me)),int(600*(1+ma+me)));
00444         c21->SetMargin(me,ma,me,ma);
00445         c21->SetGrid();
00446         //limits4->Draw("CONT Z LIST");
00447         limits4->Draw("CONT Z LIST");
00448         if(!mssm) l.DrawLatex(x0,y0,ss.c_str());
00449
00450         c21->SaveAs((string("pdf_")+string(name)+string(".png")).c_str());
00451
00452         delete c21;
00453
00454         // Bmumu_Bsmumu->SetContour(3, contours);
00455         //limits4->GetYaxis()->SetLabelOffset(0.02);
00456         Bmumu_Bsmumu->GetYaxis()->SetLabelSize(0.08);
00457         Bmumu_Bsmumu->GetYaxis()->SetTitleSize(0.08);

```

```

00458     Bmumu_Bsmumu->GetYaxis()->SetTitleOffset(1.2);
00459     Bmumu_Bsmumu->GetYaxis()->SetLimits(0.01,4.99);
00460     //limits4->GetXaxis()->SetLabelOffset(0.02);
00461     Bmumu_Bsmumu->GetXaxis()->SetLabelSize(0.08);
00462     Bmumu_Bsmumu->GetXaxis()->SetTitleSize(0.08);
00463     Bmumu_Bsmumu->GetXaxis()->SetTitleOffset(1.2);
00464     Bmumu_Bsmumu->GetXaxis()->SetLimits(0.01,1.99);
00465
00466     TCanvas * cB=new TCanvas("cB","",int(800*(1+ma+me)),int(600*(1+ma+me)));
00467     cB->SetMargin(me,ma,me,ma);
00468     cB->SetGrid();
00469     //limits4->Draw("CONT Z LIST");
00470     Bmumu_Bsmumu->Draw("COLZ");
00471     l.DrawLatex(x0,y0,ss.c_str());
00472
00473     cB->SaveAs((string("Bmumu_Bsmumu_")+string(name)+string(".png")).c_str());
00474
00475     delete cB;
00476
00477
00478     limits_MR_MI->SetContour(3, contours);
00479     //limits4->GetYaxis()->SetLabelOffset(0.02);
00480     limits_MR_MI->GetYaxis()->SetLabelSize(0.06);
00481     limits_MR_MI->GetYaxis()->SetTitleSize(0.06);
00482     limits_MR_MI->GetYaxis()->SetTitleOffset(1.1);
00483     limits_MR_MI->GetYaxis()->SetLimits(1,999);
00484     //limits4->GetXaxis()->SetLabelOffset(0.02);
00485     limits_MR_MI->GetXaxis()->SetLabelSize(0.06);
00486     limits_MR_MI->GetXaxis()->SetTitleSize(0.06);
00487     limits_MR_MI->GetXaxis()->SetTitleOffset(1.1);
00488     limits_MR_MI->GetXaxis()->SetLimits(1,999);
00489
00490     TCanvas * c3=new TCanvas("c3","",800,600);
00491     c3->SetMargin(me,ma,me,ma);
00492     c3->SetGrid();
00493
00494     limits_MR_MI->Draw("CONT LIST");
00495
00496     c3->SaveAs((string("pdf_")+string(name)+string("_MRMI.png")).c_str());
00497
00498     limits_MR_McH->SetContour(3, contours);
00499     //limits4->GetYaxis()->SetLabelOffset(0.02);
00500     limits_MR_McH->GetYaxis()->SetLabelSize(0.06);
00501     limits_MR_McH->GetYaxis()->SetTitleSize(0.06);
00502     limits_MR_McH->GetYaxis()->SetTitleOffset(1.1);
00503     limits_MR_McH->GetYaxis()->SetLimits(1,999);
00504     //limits4->GetXaxis()->SetLabelOffset(0.02);
00505     limits_MR_McH->GetXaxis()->SetLabelSize(0.06);
00506     limits_MR_McH->GetXaxis()->SetTitleSize(0.06);
00507     limits_MR_McH->GetXaxis()->SetTitleOffset(1.1);
00508     limits_MR_McH->GetXaxis()->SetLimits(1,999);
00509
00510     TCanvas * c4=new TCanvas("c4","",800,600);
00511     limits_MR_McH->Draw("CONT LIST");
00512     c4->SetMargin(me,ma,me,ma);
00513     c4->SetGrid();
00514     c4->SaveAs((string("pdf_")+string(name)+string("_MRMcH.png")).c_str());
00515
00516     limits_MI_McH->SetContour(3, contours);
00517     //limits4->GetYaxis()->SetLabelOffset(0.02);
00518     limits_MI_McH->GetYaxis()->SetLabelSize(0.06);
00519     limits_MI_McH->GetYaxis()->SetTitleSize(0.06);
00520     limits_MI_McH->GetYaxis()->SetTitleOffset(1.1);
00521     limits_MI_McH->GetYaxis()->SetLimits(1,999);
00522     //limits4->GetXaxis()->SetLabelOffset(0.02);
00523     limits_MI_McH->GetXaxis()->SetLabelSize(0.06);
00524     limits_MI_McH->GetXaxis()->SetTitleSize(0.06);
00525     limits_MI_McH->GetXaxis()->SetTitleOffset(1.1);
00526     limits_MI_McH->GetXaxis()->SetLimits(1,999);
00527
00528     TCanvas * c6=new TCanvas("c6","",800,600);
00529     limits_MI_McH->Draw("CONT LIST");
00530     c6->SetMargin(me,ma,me,ma);
00531     c6->SetGrid();
00532     c6->SaveAs((string("pdf_")+string(name)+string("_MIMcH.png")).c_str());
00533
00534     TCanvas * c5=new TCanvas("c5","",800,600);
00535     limits_tb_MR->Draw("colz");
00536
00537     c5->SaveAs((string("pdf_")+string(name)+string("_tbMR.png")).c_str());
00538
00539
00540     cout<<"gL "<<gL<<" gQ "<<gQ<<endl;
00541     cout<<"lup "<<lup<<" qup "<<qup<<endl;
00542     cout<<"llmax "<<llmax<<" gmax "<<gmax<<endl;
00543
00544

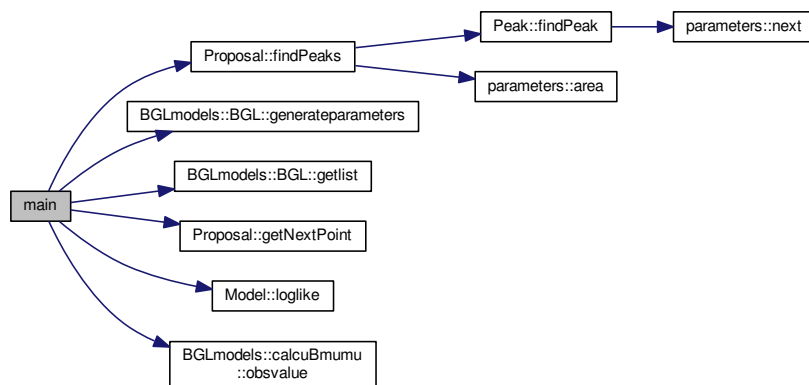
```

```

00545         delete m;
00546         delete limits4;
00547         delete Bmumu_Bsmumu;
00548         delete limits_tb_MR;
00549         delete limits_tb_MI;
00550         delete limits_MR_MI;
00551         delete limits_MI_McH;
00552         delete limits_MR_McH;
00553
00554         //mass.close();
00555         return 0;
00556
00557     }

```

Here is the call graph for this function:



## 8.18 main.cpp

```

00001 /**
00002  * @file main.c
00003  * @author Leonardo Pedro
00004  * @date May 2014
00005  * @brief Main file
00006  *
00007  * Here typically goes a more extensive explanation of what the header
00008  * defines. Doxygens tags are words preceeded by either a backslash @\
00009  * or by an at symbol @.
00010  * @see http://www.stack.nl/~dimitri/doxygen/docblocks.html
00011  * @see http://www.stack.nl/~dimitri/doxygen/commands.html
00012  */
00013
00014 #include "MCMC.h"
00015 #include "BGL.h"
00016 #include "TF2.h"
00017 #include "TProfile3D.h"
00018 #include "THStack.h"
00019 #include "TColor.h"
00020 #include "TROOT.h"
00021 #include "TStyle.h"
00022 #include "TGraph.h"
00023 #include "TLatex.h"
00024 #include "TFile.h"
00025 #include "TVector.h"
00026
00027 #include <cln/cln.h>
00028 #include <cln/float.h>
00029
00030 using namespace BGLmodels;
00031
00032 /** \mainpage Introduction
00033  * The program produces figures presenting 68%, 95% and 99% CL allowed regions in parameter space.
00034  * To wit, we represent regions where the specific BGL model is able to fit the imposed experimental
    information at least as well as the corresponding goodness levels.

```

```

00035 * Some comments are in order.
00036 * This procedure corresponds to the profile likelihood method.
00037 * In brief, for a model with parameters  $\vec{p}$ , we compute the predictions for the considered set of
    observables  $O_{\mathrm{Th}}(\vec{p})$ .
00038 * Then, using the experimental information  $O_{\mathrm{Exp}}$  available for those observables, we
    build a likelihood function  $L(\vec{O}_{\mathrm{Exp}}|\vec{O}_{\mathrm{Th}}(\vec{p}))$ 
00039 * which gives the probability of obtaining the experimental results  $O_{\mathrm{Exp}}$ 
    assuming that the model is correct.
00040 * The likelihood function  $L(\vec{O}_{\mathrm{Exp}}|\vec{O}_{\mathrm{Th}}(\vec{p}))$ 
00041 * encodes all the information on how the model is able to reproduce the observed data all over parameter
    space.
00042 * Nevertheless, the knowledge of  $L(\vec{O}_{\mathrm{Exp}}|\vec{O}_{\mathrm{Th}}(\vec{p}))$  in a
    multidimensional parameter space
00043 * can be hardly represented and one is led to the problem of reducing that information to one or
    two-dimensional subspaces.
00044 * In the profile likelihood method, for each point in the chosen subspace, the highest likelihood over the
    complementary, marginalized space, is retained. Let us clarify that likelihood
00045 * -- or chi-squared  $\chi^2 \equiv -2 \log L$  --
00046 * profiles and derived regions such as the ones we represent, are thus insensitive to the size of the space
    over which one marginalizes;
00047 * this would not be the case in a Bayesian analysis, where an integration over the marginalized space is
    involved. The profile likelihood method seems adequate to our purpose,
00048 * which is none other than exploring where in parameter space are the different BGL models able to satisfy
    experimental constraints,
00049 * without weighting in eventual fine tunings of the models or parameter space volumes.
00050 * For the numerical computations the libraries GiNaC and ROOT are used. *
00051 */
00052
00053
00054 /**
00055 * @brief the main function takes the arguments gL gQ lup qup which specify a BGL model and runs the
    simulation for that model, generating a ROOT file as the output
00056 */
00057 int main(int argc, char* argv[]){
00058     // Check the number of parameters
00059     if(argc < 5){
00060         std::cerr<<"Usage: "<<argv[0]<<" gL gQ lup qup [mssm]"<<std::endl;
00061         return 1;}
00062
00063
00064     int gL=atoi(argv[1]);
00065     int gQ=atoi(argv[2]);
00066     int lup=atoi(argv[3]);
00067     int qup=atoi(argv[4]);
00068     int mssm=0;
00069     if(argc>5) mssm=atoi(argv[5]);
00070
00071
00072     Digits=5;
00073     cln::cl_inhibit_floating_point_underflow=1;
00074
00075     string ll[2][3]={{"#nu_{1}", "#nu_{2}", "#nu_{3}"}, {"e", "#mu", "#tau"}};
00076     string qq[2][3]={{"u", "c", "t"}, {"d", "s", "b"}};
00077     //Int_t MyPalette[100];
00078     Double_t r[] = {1, 0.3};
00079     Double_t g[] = {1, 0.3};
00080     Double_t b[] = {1, 0.3};
00081     Double_t stop[] = {0., 1.0};
00082     TColor::CreateGradientColorTable(2, stop, r, g, b, 100);
00083     //TH1F * pdf1=new TH1F("pdf1", "pdf1", npoints, 10, 500);
00084     //TGraph * chi2=new TGraph(npoints);
00085
00086     uint npoints=200;
00087     double init1=-3, final1=3;
00088     double init2=10, final2=1000;
00089     double initBmumu=0, finalBmumu=2;
00090     double initBsmumu=0, finalBsmumu=5;
00091
00092     multivector<BGL *, 4> ms(0, 3, 3, 2, 2);
00093     multivector<parameters, 2> plots(parameters(), npoints, npoints);
00094     multivector<double, 2> likely(-1000, npoints, npoints);
00095     multivector<double, 2> likelyB(-1000, npoints, npoints);
00096     multivector<double, 2> likely2(-1000, npoints, npoints);
00097     multivector<double, 2> likely3(-1000, npoints, npoints);
00098     multivector<double, 2> likely4(-1000, npoints, npoints);
00099     multivector<double, 2> likely5(-1000, npoints, npoints);
00100     multivector<double, 2> likely6(-1000, npoints, npoints);
00101
00102     //ofstream mass("mass.out");
00103
00104     //for(int gL=2; gL>=0; gL--){
00105     //for(int gQ=2; gQ>=0; gQ--){
00106     //for(int lup=0; lup<2; lup++){
00107     //for(int qup=0; qup<2; qup++){
00108
00109     //    for(uint qup=0; qup<2; qup++)
00110     //for(uint gL=0; gL<3; gL++)

```

```

00111 //for(uint lup=0;lup<2;lup++)
00112 //for(uint gQ=0;gQ<3;gQ++){
00113         //if(gL==0 && gQ==2 && lup==0 && qup==1) {t=1; continue;}
00114         //if(t==0) continue;
00115         double llmax=-1000,gmax=0, McHmax=1000,MRmax=1000,MImax=1000,tbmax=1;
00116         BGL* m=new BGL(gL,gQ,lup,qup);
00117         m->mmmax=300;
00118         ms[gL][gQ][lup][qup]=m;
00119
00120         char name[5]="0000";
00121         name[0]+=gL;
00122         name[1]+=gQ;
00123         name[2]+=lup;
00124         name[3]+=qup;
00125
00126         /*
00127         TF1 * f1 = new TF1("f1",m,&BGL::BranchingRatio,-3,3,0,"BGL","BranchingRatio");
00128
00129         TCanvas * c1=new TCanvas("c1","",800,600);
00130         f1->Draw();
00131         c1->SaveAs("BR.png");
00132         TF2 * f2 = new TF2("f2",m,&BGL::topBranchingRatio,-3,3,80,175,0,"BGL","topBranchingRatio");
00133
00134         TCanvas * c3=new TCanvas("c3","",800,600);
00135         f2->Draw("colz");
00136         c3->SaveAs("topBR.png");
00137         */
00138
00139         Proposal prop4(m);
00140         //m->stepsize=1e-4;
00141         prop4.findPeaks(100,1);
00142         llmax=log(prop4.floatPeak.lmax);
00143         tbmax=prop4.floatPeak.pr[0].value;
00144         McHmax=prop4.floatPeak.pr[1].value;
00145         MRmax=prop4.floatPeak.pr[2].value;
00146         MImax=prop4.floatPeak.pr[3].value;
00147
00148         m->stepsize=1e-4;
00149         prop4.findPeaks(100);
00150         double llmax_=log(prop4.floatPeak.lmax);
00151         if(llmax_>llmax) {llmax=llmax_;
00152             tbmax=prop4.floatPeak.pr[0].value;
00153             McHmax=prop4.floatPeak.pr[1].value;
00154             MRmax=prop4.floatPeak.pr[2].value;
00155             MImax=prop4.floatPeak.pr[3].value;
00156         }
00157
00158         TH2F * limits4=new TH2F("limits4","Likelihood",npoints,init1,final1,npoints,init2,final2);
00159         TH2F * Bmumu_Bsmumu=new TH2F("Bmumu_Bsmumu","Likelihood",npoints,initBmumu,finalBmumu,npoints,
initBsmumu,finalBsmumu);
00160         TH2F * limits_tb_MR=new TH2F("limits_tb_MR","Likelihood",npoints,init1,final1,npoints,init2,final2)
;
00161         TH2F * limits_tb_MI=new TH2F("limits_tb_MI","Likelihood",npoints,init1,final1,npoints,init2,final2)
;
00162         TH2F * limits_MR_MI=new TH2F("limits_MR_MI","Likelihood",npoints,init2,final2,npoints,init2,final2)
;
00163         TH2F * limits_MR_McH=new TH2F("limits_MR_McH","Likelihood",npoints,init2,final2,npoints,init2,
final2);
00164         TH2F * limits_MI_McH=new TH2F("limits_MI_McH","Likelihood",npoints,init2,final2,npoints,init2,
final2);
00165
00166         for(uint i=0;i<npoints;i++)
00167             for(uint j=0;j<npoints;j++) {
00168                 limits4->SetBinContent(i+1,j+1,-1000);
00169                 Bmumu_Bsmumu->SetBinContent(i+1,j+1,-1000);
00170                 limits_MR_McH->SetBinContent(i+1,j+1,-1000);
00171                 limits_MI_McH->SetBinContent(i+1,j+1,-1000);
00172                 limits_MR_MI->SetBinContent(i+1,j+1,-1000);
00173                 limits_tb_MR->SetBinContent(i+1,j+1,-1000);
00174                 limits_tb_MI->SetBinContent(i+1,j+1,-1000);
00175                 plots[i][j]=m->generateparameters();
00176             }
00177         uint steps=40e6;
00178         //uint steps=npoints*npoints;
00179         double brtaunu=1;
00180         for(uint i=steps;i--){
00181             //prop1.getNextPoint();
00182             //prop2.getNextPoint();
00183             //prop3.getNextPoint();
00184             double total=0;
00185             double gp=0;
00186             //if(i==steps) cout<<" total "<<m->loglike(prop4.floatPeak.pr)<<endl;
00187             //else{
00188             if(i>npoints*npoints){
00189                 if(i==steps/2) {
00190                     m->mmmax=1000;
00191                     m->stepsize=1e-2;

```

```

00192             prop4.findPeaks(100);
00193             llmax_ = log(prop4.floatPeak.lmax);
00194             if (llmax_ > llmax) { llmax = llmax_;
00195                 tbmax = prop4.floatPeak.pr[0].value;
00196                 McHmax = prop4.floatPeak.pr[1].value;
00197                 MRmax = prop4.floatPeak.pr[2].value;
00198                 MImax = prop4.floatPeak.pr[3].value;
00199             }
00200             }
00201             if (i < steps) prop4.getNextPoint();
00202             total = log(prop4.floatPeak.lmax);
00203             //total = m->loglike(prop4.floatPeak.pr);
00204             //gp = m->gaussprob(prop4.floatPeak.pr);
00205             }
00206         else{
00207             uint x = (i-1)/npoints, y = (i-1)%npoints;
00208             prop4.floatPeak.pr[0].value = init1 + ((x+0.5)*(final1-init1))/npoints;
00209             prop4.floatPeak.pr[1].value = init2 + ((y+0.5)*(final2-init2))/npoints;
00210             prop4.floatPeak.pr[2].value = 0;
00211             prop4.floatPeak.pr[3].value = 0;
00212             total = m->loglike(prop4.floatPeak.pr);
00213             }
00214             //}
00215
00216
00217             //gp = m->gaussprob_noT(prop4.floatPeak.pr);
00218             //double Btaunu = m->loglike(prop1.floatPeak.pr, m->iBtaunu);
00219             //double BDtaunu = m->loglike(prop2.floatPeak.pr, m->iBDtaunu);
00220             //double BD2taunu = m->loglike(prop3.floatPeak.pr, m->iBD2taunu);
00221             double brtaunu0 = ex_to_numeric(m->BR_Htotaunu.subs(m->
getlist(prop4.floatPeak.pr).p).evalf()).to_double();
00222             if (brtaunu0 < brtaunu) brtaunu = brtaunu0;
00223             if (total > llmax) { llmax = total; gmax = gp;
00224                 tbmax = prop4.floatPeak.pr[0].value;
00225                 McHmax = prop4.floatPeak.pr[1].value;
00226                 MRmax = prop4.floatPeak.pr[2].value;
00227                 MImax = prop4.floatPeak.pr[3].value;
00228             }
00229
00230
00231
00232             uint p1 = uint((prop4.floatPeak.pr[0].value - init1) / (final1 - init1) * npoints)
;
00233             double mr = prop4.floatPeak.pr[1].value;
00234             uint p2 = uint((mr - init2) / (final2 - init2) * npoints);
00235             mr += prop4.floatPeak.pr[2].value;
00236             uint p3 = uint((prop4.floatPeak.pr[2].value + prop4.
floatPeak.pr[1].value - init2) / (final2 - init2) * npoints);
00237             mr += prop4.floatPeak.pr[3].value;
00238             uint p4 = uint((mr - init2) / (final2 - init2) * npoints);
00239             if (p1 < npoints && p2 < npoints && p3 < npoints && p4 < npoints){
00240                 if (total > likely[p1][p2]){
00241                     likely[p1][p2] = total;
00242                     plots[p1][p2].setvalues(prop4.floatPeak.pr);
00243                     limits4->SetBinContent(p1+1, p2+1, total);
00244                 }
00245                 if (total > likely2[p3][p4]){
00246                     likely2[p3][p4] = total;
00247                     //plots[gL][gQ][lup][qup][p1][p2].setvalues(prop4.floatPeak.pr);
00248                     limits_MR_MI->SetBinContent(p3+1, p4+1, total);
00249                 }
00250                 if (total > likely3[p3][p2]){
00251                     likely3[p3][p2] = total;
00252                     //plots[gL][gQ][lup][qup][p1][p2].setvalues(prop4.floatPeak.pr);
00253                     limits_MR_McH->SetBinContent(p3+1, p2+1, total);
00254                 }
00255             }
00256
00257             if (total > likely4[p1][p3]){
00258                 likely4[p1][p3] = total;
00259                 //plots[gL][gQ][lup][qup][p1][p2].setvalues(prop4.floatPeak.pr);
00260                 limits_tb_MR->SetBinContent(p1+1, p3+1, total);
00261             }
00262             if (total > likely5[p4][p2]){
00263                 likely5[p4][p2] = total;
00264                 //plots[gL][gQ][lup][qup][p1][p2].setvalues(prop4.floatPeak.pr);
00265                 limits_MI_McH->SetBinContent(p4+1, p2+1, total);
00266             }
00267             parameters ppr = m->getlist(prop4.floatPeak.
pr);
00268
00269             double Bmumu = m->cBmumu->obsvalue(ppr) / (1e-10 * m->
planck / 1.519e-12);
00270             double Bsmumu = m->cBsmumu->obsvalue(ppr) / (1e-9 * m->
planck / 1.516e-12);
00271             //cout << Bmumu << " " << Bsmumu << endl;
00272             uint pB = uint((Bmumu - initBmumu) / (finalBmumu - initBmumu) * npoints);

```

```

00273         uint pBs=uint((Bsmumu-initBsmumu)/(finalBsmumu-initBsmumu)*npoints);
00274         if(pB<npoints && pBs<npoints)
00275             if(total>likelyB[pB][pBs]){
00276                 likely[pB][pBs]=total;
00277                 Bmumu_Bsmumu->SetBinContent(pB+1,pBs+1,total);
00278             }
00279     }else cout<<"OUT!!!"<<endl;
00280
00281     if(i%100000==0){
00282         cout<<"Steps "<<i<<" logtb "<<prop4.floatPeak.
00283 pr[0].value<<" logMcH "<<prop4.floatPeak.pr[1].value;
00284         cout<<" total "<<total<<" gmax "<<gp<<" "<<brtaunu<<endl;
00285     }
00286
00287     }
00288     //}
00289
00290     TFile f((string("h")+string(name)+string(".root")).c_str(),"recreate");
00291     TVectorD v(5);
00292     v[0] = llmax;
00293     v[1]=tbmax;
00294     v[2]=McHmax;
00295     v[3]=MRmax;
00296     v[4]=MImax;
00297     v.Write("vllmax");
00298     limits4->Write();
00299     Bmumu_Bsmumu->Write();
00300     limits_MR_MI->Write();
00301     limits_MR_McH->Write();
00302     limits_MI_McH->Write();
00303     limits_tb_MR->Write();
00304     f.Close();
00305
00306     //for(int gL=2;gL>=0;gL--)
00307     //for(int gQ=2;gQ>=0;gQ--)
00308     //for(uint lup=0;lup<2;lup++)
00309     //for(uint qup=0;qup<2;qup++)
00310     uint min1=npoints, min2=npoints, min3=npoints;
00311     uint min11=npoints, min21=npoints, min31=npoints;
00312     uint min12=npoints, min22=npoints, min32=npoints;
00313
00314     for(uint i=0;i<npoints;i++)
00315     for(uint j=0;j<npoints;j++){
00316         int binmax=limits4->GetBin(i+1,j+1);
00317         double rest=limits4->GetBinContent(binmax);
00318         if(rest>=llmax) rest=1;
00319         else rest=TMath::Prob(-2*(rest-llmax),2);
00320         if(rest>=0.05 && j<min1){min1=j;}
00321         if(rest>=0.05 && j<min11 && j>uint(180*npoints/990)){min11=j;}
00322         if(rest>=0.05 && j<min12 && j>uint(400*npoints/990)){min12=j;}
00323         limits4->SetBinContent(i+1,j+1,rest);
00324
00325         rest=Bmumu_Bsmumu->GetBinContent(binmax);
00326         if(rest>=llmax) rest=1;
00327         else rest=TMath::Prob(-2*(rest-llmax),2);
00328         Bmumu_Bsmumu->SetBinContent(i+1,j+1,rest);
00329
00330         rest=limits_MR_MI->GetBinContent(binmax);
00331         if(rest>=llmax) rest=1;
00332         else rest=TMath::Prob(-2*(rest-llmax),2);
00333         limits_MR_MI->SetBinContent(i+1,j+1,rest);
00334
00335         rest=limits_MR_McH->GetBinContent(binmax);
00336         if(rest>=llmax) rest=1;
00337         else rest=TMath::Prob(-2*(rest-llmax),2);
00338         if(rest>=0.05 && i<min2){min2=i;}
00339         if(rest>=0.05 && i<min21 && j>uint(180*npoints/990)){min21=i;}
00340         if(rest>=0.05 && i<min22 && j>uint(400*npoints/990)){min22=i;}
00341         limits_MR_McH->SetBinContent(i+1,j+1,rest);
00342
00343         rest=limits_MI_McH->GetBinContent(binmax);
00344         if(rest>=llmax) rest=1;
00345         else rest=TMath::Prob(-2*(rest-llmax),2);
00346         if(rest>=0.05 && i<min3){min3=i;}
00347         if(rest>=0.05 && i<min31 && j>uint(180*npoints/990)){min31=i;}
00348         if(rest>=0.05 && i<min32 && j>uint(400*npoints/990)){min32=i;}
00349         limits_MI_McH->SetBinContent(i+1,j+1,rest);
00350
00351
00352         rest=limits_tb_MR->GetBinContent(binmax);
00353         if(rest>=llmax) rest=1;
00354         else rest=TMath::Prob(-2*(rest-llmax),2);
00355         limits_tb_MR->SetBinContent(i+1,j+1,rest);
00356     }
00357     double mmin1=init2+((min1+0.5)*(final2-init2))/npoints;
00358     double mmin2=init2+((min2+0.5)*(final2-init2))/npoints;

```



```

00359         double mmin3=init2+((min3+0.5)*(final2-init2))/npoints;
00360
00361         double mmin11=init2+((min11+0.5)*(final2-init2))/npoints;
00362         double mmin21=init2+((min21+0.5)*(final2-init2))/npoints;
00363         double mmin31=init2+((min31+0.5)*(final2-init2))/npoints;
00364
00365         double mmin12=init2+((min12+0.5)*(final2-init2))/npoints;
00366         double mmin22=init2+((min22+0.5)*(final2-init2))/npoints;
00367         double mmin32=init2+((min32+0.5)*(final2-init2))/npoints;
00368
00369         //mass<<name<<" "<<mmin1<<" "<<mmin2<<" "<<mmin3<<endl;
00370
00371         ofstream maxs((string("maxs_")+string(name)+string(".out")).c_str());
00372         maxs<<mmin1<<" "<<mmin2<<" "<<mmin3<<endl;
00373         maxs<<mmin11<<" "<<mmin21<<" "<<mmin31<<endl;
00374         maxs<<mmin12<<" "<<mmin22<<" "<<mmin32<<endl;
00375         maxs<<llmax<<endl;
00376         //for(uint j=0;j<npoints;j++)
00377         //for(uint i=0;i<npoints;i++){
00378             //      int binmax=limits4->GetBin(i+1,j+1);
00379             //      maxs<<"("<<i<<","<<j<<"):"<<limits4->GetBinContent(binmax)<<endl;
00380             //      }
00381
00382         maxs.close();
00383
00384         limits4->SetStats(0);
00385         limits4->GetXaxis()->SetTitle("log_{10}(tan\\beta)");
00386         limits4->GetYaxis()->SetTitle("M_{H+} (GeV)");
00387
00388         Bmumu_Bsmumu->SetStats(0);
00389         Bmumu_Bsmumu->GetXaxis()->SetTitle("Br(B\\to\\mu\\mu)/10^{-10}");
00390         Bmumu_Bsmumu->GetYaxis()->SetTitle("Br(B_{s}\\to\\mu\\mu)/10^{-9}");
00391
00392
00393         limits_MR_MI->SetStats(0);
00394         limits_MR_MI->GetYaxis()->SetTitle("M_{I} (GeV)");
00395         limits_MR_MI->GetXaxis()->SetTitle("M_{R} (GeV)");
00396
00397
00398
00399         limits_MR_McH->SetStats(0);
00400         limits_MR_McH->GetYaxis()->SetTitle("M_{H+} (GeV)");
00401         limits_MR_McH->GetXaxis()->SetTitle("M_{R} (GeV)");
00402
00403         limits_MI_McH->SetStats(0);
00404         limits_MI_McH->GetYaxis()->SetTitle("M_{H+} (GeV)");
00405         limits_MI_McH->GetXaxis()->SetTitle("M_{I} (GeV)");
00406
00407         limits_tb_MR->SetStats(0);
00408         limits_tb_MR->GetXaxis()->SetTitle("log_{10}(tan\\beta)");
00409         limits_tb_MR->GetYaxis()->SetTitle("M_{R} (GeV)");
00410
00411
00412         gStyle->SetOptTitle(0);
00413
00414         Double_t contours[3];
00415         contours[0] = 0.003;
00416         contours[1] = 0.05;
00417         contours[2] = 0.32;
00418
00419
00420         double ma=0,me=.2, x0=1,y0=120;
00421
00422         limits4->SetContour(3, contours);
00423         //limits4->GetYaxis()->SetLabelOffset(0.02);
00424         limits4->GetYaxis()->SetLabelSize(0.08);
00425         limits4->GetYaxis()->SetTitleSize(0.08);
00426         limits4->GetYaxis()->SetTitleOffset(1.2);
00427         limits4->GetYaxis()->SetLimits(1,999);
00428         //limits4->GetXaxis()->SetLabelOffset(0.02);
00429         limits4->GetXaxis()->SetLabelSize(0.08);
00430         limits4->GetXaxis()->SetTitleSize(0.08);
00431         limits4->GetXaxis()->SetTitleOffset(1.2);
00432         limits4->GetXaxis()->SetLimits(-2.99,2.99);
00433
00434
00435
00436
00437         gStyle->SetPaperSize(10.,10.);
00438
00439         TLatex l;
00440         l.SetTextSize(0.08);
00441         string ss=q[qup][gQ]+" "+l1[lup][gL];
00442
00443         TCanvas * c21=new TCanvas("c21","",int(800*(1+ma+me)),int(600*(1+ma+me)));
00444         c21->SetMargin(me,ma,me,ma);
00445         c21->SetGrid();

```

```

00446         //limits4->Draw("CONT Z LIST");
00447     limits4->Draw("CONT Z LIST");
00448     if(!mssm) l.DrawLatex(x0,y0,ss.c_str());
00449
00450     c21->SaveAs((string("pdf_")+string(name)+string(".png")).c_str());
00451
00452     delete c21;
00453
00454     // Bmumu_Bsmumu->SetContour(3, contours);
00455     //limits4->GetYaxis()->SetLabelOffset(0.02);
00456     Bmumu_Bsmumu->GetYaxis()->SetLabelSize(0.08);
00457     Bmumu_Bsmumu->GetYaxis()->SetTitleSize(0.08);
00458     Bmumu_Bsmumu->GetYaxis()->SetTitleOffset(1.2);
00459     Bmumu_Bsmumu->GetYaxis()->SetLimits(0.01,4.99);
00460     //limits4->GetXaxis()->SetLabelOffset(0.02);
00461     Bmumu_Bsmumu->GetXaxis()->SetLabelSize(0.08);
00462     Bmumu_Bsmumu->GetXaxis()->SetTitleSize(0.08);
00463     Bmumu_Bsmumu->GetXaxis()->SetTitleOffset(1.2);
00464     Bmumu_Bsmumu->GetXaxis()->SetLimits(0.01,1.99);
00465
00466     TCanvas * cB=new TCanvas("cB","",int(800*(1+ma+me)),int(600*(1+ma+me)));
00467     cB->SetMargin(me,ma,me,ma);
00468     cB->SetGrid();
00469     //limits4->Draw("CONT Z LIST");
00470     Bmumu_Bsmumu->Draw("COLZ");
00471     l.DrawLatex(x0,y0,ss.c_str());
00472
00473     cB->SaveAs((string("Bmumu_Bsmumu_")+string(name)+string(".png")).c_str());
00474
00475     delete cB;
00476
00477
00478     limits_MR_MI->SetContour(3, contours);
00479     //limits4->GetYaxis()->SetLabelOffset(0.02);
00480     limits_MR_MI->GetYaxis()->SetLabelSize(0.06);
00481     limits_MR_MI->GetYaxis()->SetTitleSize(0.06);
00482     limits_MR_MI->GetYaxis()->SetTitleOffset(1.1);
00483     limits_MR_MI->GetYaxis()->SetLimits(1,999);
00484     //limits4->GetXaxis()->SetLabelOffset(0.02);
00485     limits_MR_MI->GetXaxis()->SetLabelSize(0.06);
00486     limits_MR_MI->GetXaxis()->SetTitleSize(0.06);
00487     limits_MR_MI->GetXaxis()->SetTitleOffset(1.1);
00488     limits_MR_MI->GetXaxis()->SetLimits(1,999);
00489
00490     TCanvas * c3=new TCanvas("c3","",800,600);
00491     c3->SetMargin(me,ma,me,ma);
00492     c3->SetGrid();
00493
00494     limits_MR_MI->Draw("CONT LIST");
00495
00496     c3->SaveAs((string("pdf_")+string(name)+string("_MRMI.png")).c_str());
00497
00498     limits_MR_McH->SetContour(3, contours);
00499     //limits4->GetYaxis()->SetLabelOffset(0.02);
00500     limits_MR_McH->GetYaxis()->SetLabelSize(0.06);
00501     limits_MR_McH->GetYaxis()->SetTitleSize(0.06);
00502     limits_MR_McH->GetYaxis()->SetTitleOffset(1.1);
00503     limits_MR_McH->GetYaxis()->SetLimits(1,999);
00504     //limits4->GetXaxis()->SetLabelOffset(0.02);
00505     limits_MR_McH->GetXaxis()->SetLabelSize(0.06);
00506     limits_MR_McH->GetXaxis()->SetTitleSize(0.06);
00507     limits_MR_McH->GetXaxis()->SetTitleOffset(1.1);
00508     limits_MR_McH->GetXaxis()->SetLimits(1,999);
00509
00510     TCanvas * c4=new TCanvas("c4","",800,600);
00511     limits_MR_McH->Draw("CONT LIST");
00512     c4->SetMargin(me,ma,me,ma);
00513     c4->SetGrid();
00514     c4->SaveAs((string("pdf_")+string(name)+string("_MRMcH.png")).c_str());
00515
00516     limits_MI_McH->SetContour(3, contours);
00517     //limits4->GetYaxis()->SetLabelOffset(0.02);
00518     limits_MI_McH->GetYaxis()->SetLabelSize(0.06);
00519     limits_MI_McH->GetYaxis()->SetTitleSize(0.06);
00520     limits_MI_McH->GetYaxis()->SetTitleOffset(1.1);
00521     limits_MI_McH->GetYaxis()->SetLimits(1,999);
00522     //limits4->GetXaxis()->SetLabelOffset(0.02);
00523     limits_MI_McH->GetXaxis()->SetLabelSize(0.06);
00524     limits_MI_McH->GetXaxis()->SetTitleSize(0.06);
00525     limits_MI_McH->GetXaxis()->SetTitleOffset(1.1);
00526     limits_MI_McH->GetXaxis()->SetLimits(1,999);
00527
00528     TCanvas * c6=new TCanvas("c6","",800,600);
00529     limits_MI_McH->Draw("CONT LIST");
00530     c6->SetMargin(me,ma,me,ma);
00531     c6->SetGrid();
00532     c6->SaveAs((string("pdf_")+string(name)+string("_MIMcH.png")).c_str());

```

```

00533
00534     TCanvas * c5=new TCanvas("c5","",800,600);
00535     limits_tb_MR->Draw("colz");
00536
00537     c5->SaveAs((string("pdf_")+string(name)+string("_tbMR.png")).c_str());
00538
00539
00540     cout<<"gL "<<gL<<" gQ "<<gQ<<endl;
00541     cout<<"lup "<<lup<<" qup "<<qup<<endl;
00542     cout<<"llmax "<<llmax<<" gmax "<<gmax<<endl;
00543
00544
00545     delete m;
00546     delete limits4;
00547     delete Bmumu_Bsmumu;
00548     delete limits_tb_MR;
00549     delete limits_tb_MI;
00550     delete limits_MR_MI;
00551     delete limits_MI_McH;
00552     delete limits_MR_McH;
00553
00554     //mass.close();
00555     return 0;
00556
00557 }
00558

```

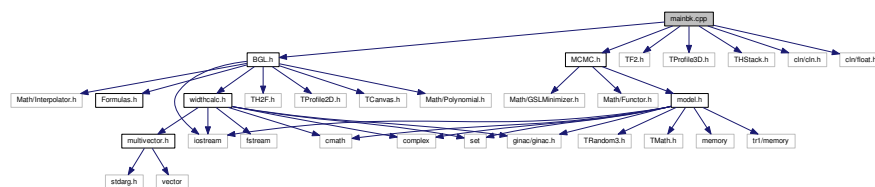
## 8.19 mainbk.cpp File Reference

```

#include "MCMC.h"
#include "BGL.h"
#include "TF2.h"
#include "TProfile3D.h"
#include "THStack.h"
#include <cln/cln.h>
#include <cln/float.h>

```

Include dependency graph for mainbk.cpp:



## Functions

- int [main](#) ()

### 8.19.1 Function Documentation

#### 8.19.1.1 int main ( )

Definition at line 8 of file [mainbk.cpp](#).

References [Proposal::findPeaks\(\)](#), [Proposal::floatPeak](#), and [Peak::pr](#).

```

00008     {
00009     Digits=5;
00010     cln::cl_inhibit_floating_point_underflow=1;
00011
00012
00013
00014
00015     //TH1F * pdf1=new TH1F("pdf1","pdf1",npoints,10,500);
00016
00017
00018     //TGraph * chi2=new TGraph(npoints);
00019     double lmax=0;
00020     double gaussmax=0;
00021     double tanbmax=0,mcHmax=0, Btaunu=0,BDtaunu=0,BD2taunu=0;
00022     uint gLm=0,gQm=0,lupm=0,qupm=0;
00023
00024     for(uint gL=0;gL<3;gL++)
00025     for(uint gQ=0;gQ<3;gQ++)
00026     for(uint lup=0;lup<2;lup++)
00027     for(uint qup=0;qup<2;qup++) {
00028         BGL* m=new BGL(gL,gQ,lup,qup);
00029         char name[5]="0000";
00030         name[0]+=gL;
00031         name[1]+=gQ;
00032         name[2]+=lup;
00033         name[3]+=qup;
00034
00035
00036
00037     /*
00038     TF1 * f1 = new TF1("f1",m,&BGL::BranchingRatio,-3,3,0,"BGL","BranchingRatio");
00039
00040     TCanvas * c1=new TCanvas("c1","",800,600);
00041     f1->Draw();
00042     c1->SaveAs("BR.png");
00043     TF2 * f2 = new TF2("f2",m,&BGL::topBranchingRatio,-3,3,80,175,0,"BGL","topBranchingRatio");
00044
00045     TCanvas * c3=new TCanvas("c3","",800,600);
00046     f2->Draw("colz");
00047     c3->SaveAs("topBR.png");
00048     */
00049     uint npoints=100;
00050     //TH2F * pdf=new TH2F("pdf","pdf",npoints,-7/log(10.0),7/log(10.0),npoints,10,500);
00051     double init=-3, final=3;
00052     double init2=1, final2=4;
00053
00054     uint steps=1e4;
00055     Proposal prop1(m,m->iBtaunu);
00056     prop1.findPeaks();
00057     Proposal prop2(m,m->iBDtaunu);
00058     prop2.findPeaks();
00059     Proposal prop3(m,m->iBD2taunu);
00060     prop3.findPeaks();
00061     Proposal prop4(m,-1);
00062     prop4.findPeaks();
00063
00064     tanbmax=pow(10.0,prop4.floatPeak.pr[0].value);
00065     mcHmax=pow(10.0,prop4.floatPeak.pr[1].value);
00066
00067     /*
00068
00069     TH2F * limits1=new TH2F("limits1","Likelihood",npoints,init,final,npoints,init2,final2);
00070     TH2F * limits2=new TH2F("limits2","Likelihood",npoints,init,final,npoints,init2,final2);
00071     TH2F * limits3=new TH2F("limits3","Likelihood",npoints,init,final,npoints,init2,final2);
00072     TH2F * limits4=new TH2F("limits4","Likelihood",npoints,init,final,npoints,init2,final2);
00073     TH2F * limits5=new TH2F("limits5","Likelihood",npoints,init,final,npoints,init2,final2);
00074     TProfile2D * like1=new TProfile2D("like1","like",npoints,init,final,npoints,init2,final2);
00075     TProfile2D * like2=new TProfile2D("like2","like",npoints,init,final,npoints,init2,final2);
00076     TProfile2D * like3=new TProfile2D("like3","like",npoints,init,final,npoints,init2,final2);
00077     TProfile2D * like4=new TProfile2D("like4","like",npoints,init,final,npoints,init2,final2);
00078     THStack hs("hs","test stacked histograms");
00079
00080
00081     for(uint i=steps;i;i--){
00082         prop1.getNextPoint();
00083         prop2.getNextPoint();
00084         prop3.getNextPoint();
00085         prop4.getNextPoint();
00086
00087         double Btaunu=m->loglike(prop1.floatPeak.pr,m->iBtaunu);
00088         double BDtaunu=m->loglike(prop2.floatPeak.pr,m->iBDtaunu);
00089         double BD2taunu=m->loglike(prop3.floatPeak.pr,m->iBD2taunu);
00090         double total=m->loglike(prop4.floatPeak.pr,-1);
00091
00092         like1->Fill(prop1.floatPeak.pr[0].value, prop1.floatPeak.pr[1].value,Btaunu);
00093         like2->Fill(prop2.floatPeak.pr[0].value, prop2.floatPeak.pr[1].value,BDtaunu);
00094         like3->Fill(prop3.floatPeak.pr[0].value, prop3.floatPeak.pr[1].value,BD2taunu);

```

```

00095         like4->Fill(prop4.floatPeak.pr[0].value, prop4.floatPeak.pr[1].value,total);
00096
00097         if(i%(steps/10)==0) cout<<"Steps " <<i<<endl;
00098         //pdf1->Fill(prop.floatPeak.pr[1].value);
00099     }
00100
00101     for(uint i=0;i<npoints;i++)
00102         for(uint j=0;j<npoints;j++)
00103         {
00104             double bmax=0;
00105             int binmax=like1->GetBin(i+1,j+1);
00106             //for(uint k=0;k<npoints;k++){
00107             //int binmax=like1->GetBin(i+1,j+1,k+1);
00108             //if(like1->GetBinEntries(bin)){
00109             //    double bc=like1->GetBinContent(bin);
00110             //    if(bc>bmax || binmax==-1) {bmax=bc; binmax=bin;}
00111             //    }
00112             //}
00113             double Btaunu=0;
00114             double BDtaunu=0;
00115             double BD2taunu=0;
00116             double rest=0;
00117             double sum=0;
00118
00119             if(like1->GetBinEntries(binmax)) {
00120                 Btaunu=like1->GetBinContent(binmax);
00121                 //cout<<i<<" "<<j<<" "<<Btaunu<<" ";
00122                 sum+=Btaunu;
00123                 Btaunu=TMath::Prob(-2*Btaunu,1);
00124                 //cout<<Btaunu<<endl;
00125                 if(Btaunu<0.05) Btaunu=0;
00126             }
00127             if(like2->GetBinEntries(binmax)) {
00128                 BDtaunu=like2->GetBinContent(binmax);
00129                 sum+=BDtaunu;
00130                 BDtaunu=TMath::Prob(-2*BDtaunu,1);
00131                 if(BDtaunu<0.05) BDtaunu=0;
00132             }
00133             if(like3->GetBinEntries(binmax)) {
00134                 BD2taunu=like3->GetBinContent(binmax);
00135                 sum+=BD2taunu;
00136                 BD2taunu=TMath::Prob(-2*BD2taunu,1);
00137                 if(BD2taunu<0.05) BD2taunu=0;
00138             }
00139             if((like3->GetBinEntries(binmax) && like2->GetBinEntries(binmax) &&
like1->GetBinEntries(binmax) && sum>-20)){
00140                 sum=TMath::Prob(-2*sum,3);
00141                 if(sum>lmax){
00142                     lmax=sum;
00143                     tanbmax=pow(10.0, (i+0.5)*1.0/npoints*6-3);
00144                     mcHmax=pow(10.0, (j+0.5)*1.0/npoints*3+1);
00145                 }
00146             }else sum=0;
00147
00148             if(like4->GetBinEntries(binmax)){
00149                 rest=like4->GetBinContent(binmax);
00150                 rest=TMath::Prob(-2*rest,m->size());
00151             }
00152             int scale=100;
00153             limits1->SetBinContent(i+1,j+1,Btaunu*scale);
00154             limits2->SetBinContent(i+1,j+1,BDtaunu*scale);
00155             limits3->SetBinContent(i+1,j+1,BD2taunu*scale);
00156             limits4->SetBinContent(i+1,j+1,rest*scale);
00157             limits5->SetBinContent(i+1,j+1,sum*scale);
00158         }
00159
00160         limits3->GetXaxis()->SetTitle("tan\\beta*MW/MH");
00161         limits3->GetYaxis()->SetTitle("cotan\\beta*MW/MH");
00162         limits1->SetStats(0);
00163         limits2->SetStats(0);
00164         limits3->SetStats(0);
00165         limits4->SetStats(0);
00166
00167         //limits4->SetMarkerStyle(7);
00168         limits1->SetMarkerColor(1);
00169         //limits1->SetMarkerStyle(7);
00170         limits2->SetMarkerColor(2);
00171         //limits2->SetMarkerStyle(7);
00172         limits3->SetMarkerColor(3);
00173         //limits3->SetMarkerStyle(7);
00174         limits4->SetMarkerColor(9);
00175         //hs.Add(limits4);
00176         hs.Add(limits3);
00177         hs.Add(limits2);
00178         hs.Add(limits1);
00179
00180         TCanvas * c2=new TCanvas("c2","",800,600);

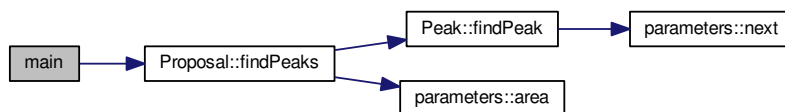
```

```

00181         limits4->Draw("colz");
00182         //hs.Draw("nostack");
00183         c2->SaveAs((string("pdf_")+string(name)+string(".png")).c_str());
00184
00185         TCanvas * c3=new TCanvas("c3","",800,600);
00186         limits1->Draw("colz");
00187         //hs.Draw("nostack");
00188         c3->SaveAs((string("pdf1_")+string(name)+string(".png")).c_str());
00189
00190         TCanvas * c4=new TCanvas("c4","",800,600);
00191         limits2->Draw("colz");
00192         //hs.Draw("nostack");
00193         c4->SaveAs((string("pdf2_")+string(name)+string(".png")).c_str());
00194
00195
00196         TCanvas * c5=new TCanvas("c5","",800,600);
00197         limits3->Draw("colz");
00198         //hs.Draw("nostack");
00199         c5->SaveAs((string("pdf3_")+string(name)+string(".png")).c_str());
00200
00201         TCanvas * c6=new TCanvas("c6","",800,600);
00202         limits5->Draw("colz");
00203         //hs.Draw("nostack");
00204         c6->SaveAs((string("pdf123_")+string(name)+string(".png")).c_str());
00205
00206         TCanvas * c7=new TCanvas("c7","",800,600);
00207         hs.Draw("nostack");
00208         c7->SaveAs((string("pdfall_")+string(name)+string(".png")).c_str());
00209         */
00210         //cout<<"Lmax " <<lmax<<" " <<TMath::Prob(-2*log(lmax),m->size())<<" GaussMax " <<gaussmax<<endl;
00211
00212         //cout<<"tanbmax " <<tanbmax<<" McHmax " <<mcHmax<<endl;
00213         cout<<"gL " <<gL<<" gQ " <<gQ<<endl;
00214         cout<<"lup " <<lup<<" qup " <<qup<<endl;
00215
00216         cout<<"Btotaunu " <<m->BtotaunuR.subs(1st(m->tanb==tanbmax,m->McH==mcHmax))<<" exp " <<1.64/0.79<<"
+/- " <<0.23*1.64/0.79<<endl;
00217         cout<<"BtoDtaunu " <<m->BtoDtaunuR.subs(1st(m->tanb==tanbmax,m->McH==mcHmax))<<" exp " <<440.0/296<<"
+/- " <<1.4*58.0/296<<endl;
00218         cout<<"BtoD2taunu " <<m->BtoD2taunuR.subs(1st(m->tanb==tanbmax,m->McH==mcHmax))<<" exp " <<332.0/252<
<" +/- " <<1.4*24.0/252<<endl;
00219
00220
00221         delete m;
00222     }
00223
00224     return 0;
00225 }
00226 }

```

Here is the call graph for this function:



## 8.20 mainbk.cpp

```

00001 #include "MCMC.h"
00002 #include "BGL.h"
00003 #include "TF2.h"
00004 #include "TProfile3D.h"
00005 #include "THStack.h"
00006 #include <cln/cln.h>
00007 #include <cln/float.h>
00008 int main(){
00009     Digits=5;
00010     cln::cl_inhibit_floating_point_underflow=1;
00011

```

```

00012
00013
00014
00015 //TH1F * pdf1=new TH1F("pdf1","pdf1",npoints,10,500);
00016
00017
00018 //TGraph * chi2=new TGraph(npoints);
00019 double lmax=0;
00020 double gaussmax=0;
00021 double tanbmax=0,mcHmax=0, Btaunu=0,BDtaunu=0,BD2taunu=0;
00022 uint gLm=0,gQm=0,lupm=0,qupm=0;
00023
00024 for(uint gL=0;gL<3;gL++)
00025 for(uint gQ=0;gQ<3;gQ++)
00026 for(uint lup=0;lup<2;lup++)
00027 for(uint qup=0;qup<2;qup++){
00028     BGL* m=new BGL(gL,gQ,lup,qup);
00029     char name[5]="0000";
00030     name[0]+=gL;
00031     name[1]+=gQ;
00032     name[2]+=lup;
00033     name[3]+=qup;
00034
00035
00036
00037 /*
00038 TF1 * f1 = new TF1("f1",m,&BGL::BranchingRatio,-3,3,0,"BGL","BranchingRatio");
00039
00040 TCanvas * c1=new TCanvas("c1","",800,600);
00041 f1->Draw();
00042 c1->SaveAs("BR.png");
00043 TF2 * f2 = new TF2("f2",m,&BGL::topBranchingRatio,-3,3,80,175,0,"BGL","topBranchingRatio");
00044
00045 TCanvas * c3=new TCanvas("c3","",800,600);
00046 f2->Draw("colz");
00047 c3->SaveAs("topBR.png");
00048 */
00049 uint npoints=100;
00050 //TH2F * pdf=new TH2F("pdf","pdf",npoints,-7/log(10.0),7/log(10.0),npoints,10,500);
00051 double init=-3, final=3;
00052 double init2=1, final2=4;
00053
00054 uint steps=1e4;
00055 Proposal prop1(m,m->iBtaunu);
00056 prop1.findPeaks();
00057 Proposal prop2(m,m->iBDtaunu);
00058 prop2.findPeaks();
00059 Proposal prop3(m,m->iBD2taunu);
00060 prop3.findPeaks();
00061 Proposal prop4(m,-1);
00062 prop4.findPeaks();
00063
00064 tanbmax=pow(10.0,prop4.floatPeak.pr[0].value);
00065 mcHmax=pow(10.0,prop4.floatPeak.pr[1].value);
00066
00067 /*
00068
00069 TH2F * limits1=new TH2F("limits1","Likelihood",npoints,init,final,npoints,init2,final2);
00070 TH2F * limits2=new TH2F("limits2","Likelihood",npoints,init,final,npoints,init2,final2);
00071 TH2F * limits3=new TH2F("limits3","Likelihood",npoints,init,final,npoints,init2,final2);
00072 TH2F * limits4=new TH2F("limits4","Likelihood",npoints,init,final,npoints,init2,final2);
00073 TH2F * limits5=new TH2F("limits5","Likelihood",npoints,init,final,npoints,init2,final2);
00074 TProfile2D * like1=new TProfile2D("like1","like",npoints,init,final,npoints,init2,final2);
00075 TProfile2D * like2=new TProfile2D("like2","like",npoints,init,final,npoints,init2,final2);
00076 TProfile2D * like3=new TProfile2D("like3","like",npoints,init,final,npoints,init2,final2);
00077 TProfile2D * like4=new TProfile2D("like4","like",npoints,init,final,npoints,init2,final2);
00078 THStack hs("hs","test stacked histograms");
00079
00080
00081 for(uint i=steps;i--){
00082     prop1.getNextPoint();
00083     prop2.getNextPoint();
00084     prop3.getNextPoint();
00085     prop4.getNextPoint();
00086
00087     double Btaunu=m->loglike(prop1.floatPeak.pr,m->iBtaunu);
00088     double BDtaunu=m->loglike(prop2.floatPeak.pr,m->iBDtaunu);
00089     double BD2taunu=m->loglike(prop3.floatPeak.pr,m->iBD2taunu);
00090     double total=m->loglike(prop4.floatPeak.pr,-1);
00091
00092     like1->Fill(prop1.floatPeak.pr[0].value, prop1.floatPeak.pr[1].value,Btaunu);
00093     like2->Fill(prop2.floatPeak.pr[0].value, prop2.floatPeak.pr[1].value,BDtaunu);
00094     like3->Fill(prop3.floatPeak.pr[0].value, prop3.floatPeak.pr[1].value,BD2taunu);
00095     like4->Fill(prop4.floatPeak.pr[0].value, prop4.floatPeak.pr[1].value,total);
00096
00097     if(i%(steps/10)==0) cout<<"Steps "<<i<<endl;
00098     //pdf1->Fill(prop.floatPeak.pr[1].value);

```

```

00099     }
00100
00101     for(uint i=0;i<npoints;i++)
00102         for(uint j=0;j<npoints;j++)
00103             {
00104                 double bcmax=0;
00105                 int binmax=like1->GetBin(i+1,j+1);
00106                 //for(uint k=0;k<npoints;k++){
00107                 //int binmax=like1->GetBin(i+1,j+1,j+1);
00108                 //if(like1->GetBinEntries(bin)){
00109                 //    double bc=like1->GetBinContent(bin);
00110                 //    if(bc>bcmax || binmax== -1) {bcmax=bc; binmax=bin;}
00111                 //    }
00112                 //}
00113                 double Btaunu=0;
00114                 double BDtaunu=0;
00115                 double BD2taunu=0;
00116                 double rest=0;
00117                 double sum=0;
00118
00119                 if(like1->GetBinEntries(binmax)) {
00120                     Btaunu=like1->GetBinContent(binmax);
00121                     //cout<<i<<" "<<j<<" "<<Btaunu<<" ";
00122                     sum+=Btaunu;
00123                     Btaunu=TMath::Prob(-2*Btaunu,1);
00124                     //cout<<Btaunu<<endl;
00125                     if(Btaunu<0.05) Btaunu=0;
00126                 }
00127                 if(like2->GetBinEntries(binmax)) {
00128                     BDtaunu=like2->GetBinContent(binmax);
00129                     sum+=BDtaunu;
00130                     BDtaunu=TMath::Prob(-2*BDtaunu,1);
00131                     if(BDtaunu<0.05) BDtaunu=0;
00132                 }
00133                 if(like3->GetBinEntries(binmax)) {
00134                     BD2taunu=like3->GetBinContent(binmax);
00135                     sum+=BD2taunu;
00136                     BD2taunu=TMath::Prob(-2*BD2taunu,1);
00137                     if(BD2taunu<0.05) BD2taunu=0;
00138                 }
00139                 if((like3->GetBinEntries(binmax) && like2->GetBinEntries(binmax) &&
like1->GetBinEntries(binmax) && sum>-20)){
00140                     sum=TMath::Prob(-2*sum,3);
00141                     if(sum>lmax){
00142                         lmax=sum;
00143                         tanbmax=pow(10.0,(i+0.5)*1.0/npoints*6-3);
00144                         mcHmax=pow(10.0,(j+0.5)*1.0/npoints*3+1);
00145                     }
00146                 }else sum=0;
00147
00148                 if(like4->GetBinEntries(binmax)){
00149                     rest=like4->GetBinContent(binmax);
00150                     rest=TMath::Prob(-2*rest,m->size());
00151                 }
00152                 int scale=100;
00153                 limits1->SetBinContent(i+1,j+1,Btaunu*scale);
00154                 limits2->SetBinContent(i+1,j+1,BDtaunu*scale);
00155                 limits3->SetBinContent(i+1,j+1,BD2taunu*scale);
00156                 limits4->SetBinContent(i+1,j+1,rest*scale);
00157                 limits5->SetBinContent(i+1,j+1,sum*scale);
00158             }
00159
00160         limits3->GetXaxis()->SetTitle("tan\\beta*MW/MH");
00161         limits3->GetYaxis()->SetTitle("cotan\\beta*MW/MH");
00162         limits1->SetStats(0);
00163         limits2->SetStats(0);
00164         limits3->SetStats(0);
00165         limits4->SetStats(0);
00166
00167         //limits4->SetMarkerStyle(7);
00168         limits1->SetMarkerColor(1);
00169         //limits1->SetMarkerStyle(7);
00170         limits2->SetMarkerColor(2);
00171         //limits2->SetMarkerStyle(7);
00172         limits3->SetMarkerColor(3);
00173         //limits3->SetMarkerStyle(7);
00174         limits4->SetMarkerColor(9);
00175         //hs.Add(limits4);
00176         hs.Add(limits3);
00177         hs.Add(limits2);
00178         hs.Add(limits1);
00179
00180         TCanvas * c2=new TCanvas("c2","",800,600);
00181         limits4->Draw("colz");
00182         //hs.Draw("nostack");
00183         c2->SaveAs((string("pdf_")+string(name)+string(".png")).c_str());
00184

```



```

00185         TCanvas * c3=new TCanvas("c3","",800,600);
00186         limits1->Draw("colz");
00187         //hs.Draw("nostack");
00188         c3->SaveAs((string("pdf1_")+string(name)+string(".png")).c_str());
00189
00190         TCanvas * c4=new TCanvas("c4","",800,600);
00191         limits2->Draw("colz");
00192         //hs.Draw("nostack");
00193         c4->SaveAs((string("pdf2_")+string(name)+string(".png")).c_str());
00194
00195
00196         TCanvas * c5=new TCanvas("c5","",800,600);
00197         limits3->Draw("colz");
00198         //hs.Draw("nostack");
00199         c5->SaveAs((string("pdf3_")+string(name)+string(".png")).c_str());
00200
00201         TCanvas * c6=new TCanvas("c6","",800,600);
00202         limits5->Draw("colz");
00203         //hs.Draw("nostack");
00204         c6->SaveAs((string("pdf123_")+string(name)+string(".png")).c_str());
00205
00206         TCanvas * c7=new TCanvas("c7","",800,600);
00207         hs.Draw("nostack");
00208         c7->SaveAs((string("pdfall_")+string(name)+string(".png")).c_str());
00209         */
00210         //cout<<"Lmax " <<lmax<<" " <<TMath::Prob(-2*log(lmax),m->size())<<" GaussMax " <<gaussmax<<endl;
00211
00212         //cout<<"tanbmax " <<tanbmax<<" McHmax " <<mcHmax<<endl;
00213         cout<<"gL " <<gL<<" gQ " <<gQ<<endl;
00214         cout<<"lup " <<lup<<" qup " <<qup<<endl;
00215
00216         cout<<"Btotaunu " <<m->BtotaunuR.subs(1st(m->tanb==tanbmax,m->McH==mcHmax))<<" exp " <<1.64/0.79<<"
+/- " <<0.23*1.64/0.79<<endl;
00217         cout<<"BtoDtaunu " <<m->BtoDtaunuR.subs(1st(m->tanb==tanbmax,m->McH==mcHmax))<<" exp " <<440.0/296<<"
+/- " <<1.4*58.0/296<<endl;
00218         cout<<"BtoD2taunu " <<m->BtoD2taunuR.subs(1st(m->tanb==tanbmax,m->McH==mcHmax))<<" exp " <<332.0/252<
<" +/- " <<1.4*24.0/252<<endl;
00219
00220
00221         delete m;
00222     }
00223
00224     return 0;
00225
00226 }
00227

```

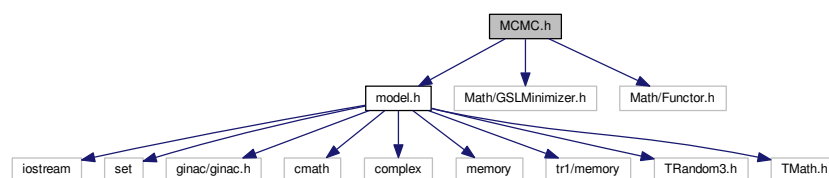
## 8.21 MCMC.h File Reference

```

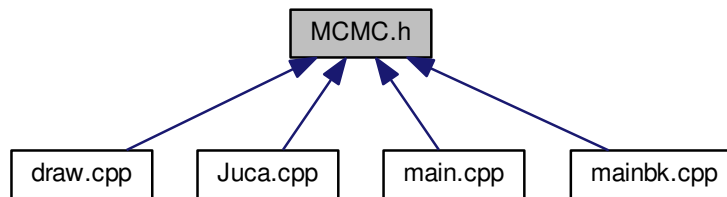
#include "model.h"
#include "Math/GSLMinimizer.h"
#include "Math/Functor.h"

```

Include dependency graph for MCMC.h:



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



## Classes

- class [Peak](#)  
A class containing the parameters of a maximum of the likelihood function.
- class [Proposal](#)  
A class containing the parameters of a proposal for the next step in the Markov Chain.

## 8.22 MCMC.h

```

00001 #ifndef MCMC_H
00002 #define MCMC_H
00003
00004 #include "model.h"
00005 #include "Math/GSLMinimizer.h"
00006 #include "Math/Functor.h"
00007
00008 using namespace std;
00009
00010 ///A class containing the parameters of a maximum of the likelihood function
00011 class Peak {
00012 public:
00013     Peak(const Model * m,int maxx=0): model(m), pr(m->generateparameters(maxx)), max(maxx){
00014         lmax=model->likelihood(pr);
00015     }
00016
00017     void findPeak(){
00018         llmax=model->loglike(pr,1,max);
00019         area=1;
00020         uint fixed=1e2;
00021         uint f=fixed;
00022         double d=1;
00023         //cout<<"f "<<f<<"llmax "<<llmax<<endl;
00024         for(uint i=1e5;i;i--){
00025             parameters p1(pr);
00026             p1.next(model->r,d);
00027
00028             double ll=llmax;
00029             if(!model->veto(p1,max)){ll=model->loglike(p1,1,max);
00030             }
00031             if(ll>llmax){pr=p1; llmax=ll; f=fixed;}
00032             else {f--; if(!f) {d/=100; f=fixed; if(d<1e-2) break;}}
00033         }
00034         cout<<"d "<<d<<"llmax "<<llmax<<endl;
00035         if(llmax<-1000) lmax=0;
00036         else{lmax=exp(llmax);
00037             area=pr.area();
00038             larea=area*lmax;
00039         }
00040     }
00041 }
00042 /*
00043 void findPeak(){

```

```

00044         llmax=model->loglike(pr);
00045         area=1;
00046         uint fixed=1e2;
00047         uint f=fixed;
00048         parameters p1(pr);
00049
00050         for(uint j=1e4;j;j--){
00051             for(uint i=0;i<pr.size(); i++){
00052                 double s=pr[i].step/1e3;
00053                 pr[i].value+=s;
00054                 if(pr[i].value>pr[i].max){ s*=-1; pr[i].value+=2*s;}
00055                 double x=(model->loglike(pr)-llmax)*pr[i].step/1e2;
00056                 pr[i].value-=s;
00057                 if(fabs(x)>pr[i].step) x*=pr[i].step/fabs(x);
00058                 p1[i].value=pr[i].value+x;
00059                 if(p1[i].value>pr[i].max) p1[i].value=pr[i].max;
00060                 else if(p1[i].value<pr[i].min) p1[i].value=pr[i].min;
00061             }
00062             if(pr.dist(p1)<1e-6) {pr.setvalues(p1); f=0; break;}
00063             cout<<"Loglike "<<llmax<<" "<<pr.dist(p1)<<endl;
00064             pr.setvalues(p1);
00065             llmax=model->loglike(pr);
00066         }
00067
00068         if(f || llmax<=-20) lmax=0;
00069         else{
00070             lmax=exp(llmax);
00071             area=pr.area();
00072             larea=area*lmax;
00073         }
00074     }
00075
00076
00077     double RosenBrock(const double *xx )
00078     {
00079         parameters p=model->generateparameters();
00080         for(uint i=0; i<p.size();i++) p[i].value=xx[i];
00081
00082         return -model->loglike(pr);
00083     }
00084
00085     void findPeak3(){
00086         llmax=model->loglike(pr);
00087         area=1;
00088         uint fixed=1e2;
00089         uint f=fixed;
00090         parameters p1(pr);
00091         ROOT::Math::GSLMinimizer min( ROOT::Math::kVectorBFGS );
00092
00093         min.SetMaxFunctionCalls(1000000);
00094         min.SetMaxIterations(100000);
00095         min.SetTolerance(0.001);
00096
00097         ROOT::Math::Functor f(&RosenBrock,pr.size());
00098         double step[2] = {0.01,0.01};
00099         double variable[2] = { -1.,1.2};
00100         char s[3]="x0";
00101
00102         min.SetFunction(f);
00103         fo
00104         // Set the free variables to be minimized!
00105         min.SetVariable(0,"x",variable[0], step[0]);
00106         min.SetVariable(1,"y",variable[1], step[1]);
00107
00108         min.Minimize();
00109
00110         for(uint j=1e4;j;j--){
00111             for(uint i=0;i<pr.size(); i++){
00112                 double s=pr[i].step/1e3;
00113                 pr[i].value+=s;
00114                 if(pr[i].value>pr[i].max){ s*=-1; pr[i].value+=2*s;}
00115                 double x=(model->loglike(pr)-llmax)*pr[i].step/1e2;
00116                 pr[i].value-=s;
00117                 if(fabs(x)>pr[i].step) x*=pr[i].step/fabs(x);
00118                 p1[i].value=pr[i].value+x;
00119                 if(p1[i].value>pr[i].max) p1[i].value=pr[i].max;
00120                 else if(p1[i].value<pr[i].min) p1[i].value=pr[i].min;
00121             }
00122             if(pr.dist(p1)<1e-6) {pr.setvalues(p1); f=0; break;}
00123             cout<<"Loglike "<<llmax<<" "<<pr.dist(p1)<<endl;
00124             pr.setvalues(p1);
00125             llmax=model->loglike(pr);
00126         }
00127
00128         if(f || llmax<=-20) lmax=0;
00129         else{
00130             lmax=exp(llmax);
00131             area=pr.area();

```

```

00131         larea=area*lmax;
00132     }
00133
00134 }
00135 */
00136 bool adjuststeps(){
00137     parameters p1(pr);
00138     for(uint i=0;i<pr.size(); i++){
00139         double s=p1[i].step;
00140         p1[i].value+=s;
00141         double x=(lmax-model->likelihood(p1))*2/lmax/s/s;
00142         double x0=std::pow(2/(pr[i].max-pr[i].min),2);
00143         if(x<x0) return 0;
00144         // cout<<"X "<<x<<endl;
00145         pr[i].step=1/sqrt(x);
00146         p1[i].value-=s;
00147     }
00148     area=pr.area();
00149     larea=area*lmax;
00150     return 1;
00151 }
00152
00153 const Model * model;
00154 parameters pr;
00155 double lmax, llmax;
00156 double area;
00157 double larea;
00158 bool max;
00159 };
00160
00161 ///A class containing the parameters of a proposal for the next step in the Markov Chain
00162 class Proposal{
00163 public:
00164
00165     Proposal(const Model * m): model(m), floatPeak(m),proposal(m){}
00166
00167     void findPeaks(uint ns=1, int max=0) {
00168         //float pmin=-100, pmax=100, s=0.1;
00169         //floatPeak.s=s;
00170         //floatPeak.lmax=0;
00171         //int imax=-1;
00172         floatPeak=Peak(model,max);
00173         floatPeak.lmax=0;
00174         floatPeak.llmax=-1000;
00175         cout<<"started"<<endl;
00176         //for(uint i=5e1;i--){
00177         for(uint i=ns;i--){
00178             Peak pp(model,max);
00179             pp.findPeak();
00180             if(pp.llmax>-15){
00181                 //for(uint j=0; j< pp.pr.size();j++){
00182                 //cout<<j<<" "<<pp.pr[j].value<<endl;
00183                 //}
00184                 //lst l=model->getlist(pp.pr);
00185                 //for(uint j=0; j< model->size();j++){
00186                 //    double mean=model->at(j).calculate(1);
00187                 //cout<<j<<" "<<mean<<" "<<sqrt(2*model->at(j).o->loglikelihood(mean))<<endl;
00188                 //}
00189             }
00190             if(pp.lmax>floatPeak.lmax){
00191                 cout<<i<<" "<<pp.lmax<<endl;
00192                 floatPeak.lmax=pp.lmax;
00193                 floatPeak.pr=pp.pr;
00194             }
00195         }
00196         floatPeak.area=floatPeak.pr.area();
00197     }
00198 }
00199
00200 void getProposal(){
00201     if(model->r->Rndm() <=0.9) {
00202         proposal.pr=floatPeak.pr;
00203         proposal.pr.next(model->r);
00204         return;
00205     }
00206
00207     proposal.pr=model->generateparameters();
00208 }
00209
00210 void getNextPoint(){
00211     getProposal();
00212     double ll=0;
00213     ll=model->likelihood(proposal.pr);
00214     if(model->r->Rndm() <=ll/floatPeak.lmax){
00215         floatPeak.lmax=ll;
00216         floatPeak.pr.setvalues(proposal.pr);
00217     }

```

```

00218 }
00219
00220 const Model * model;
00221
00222 vector<Peak> vPeak;
00223 Peak floatPeak, proposal;
00224 double total;
00225 };
00226
00227 #endif

```

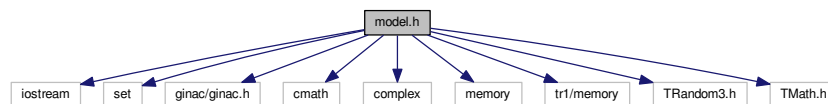
## 8.23 model.h File Reference

```

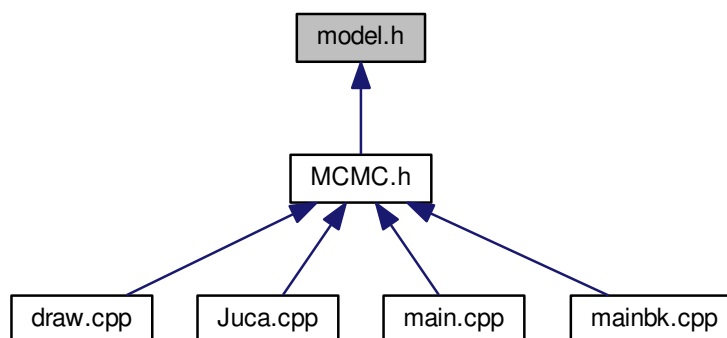
#include <iostream>
#include <set>
#include <ginac/ginac.h>
#include <cmath>
#include <complex>
#include <memory>
#include <tr1/memory>
#include "TRandom3.h"
#include "TMath.h"

```

Include dependency graph for model.h:



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



## Classes

- class [measure](#)

- A class containing the value and uncertainty of an experimental measure.*

  - class [observable](#)
- A base class representing an experimental measure.*

  - class [limitedobs](#)
- An experimental measure which is an upper limit on a parameter with a given Confidence Level.*

  - class [gaussobs](#)
- An experimental measure of a parameter which is a mean value and a standard deviation.*

  - class [gauss2obs](#)
- the same as gaussobs but with a different initializer, such that the uncertainty sigma is absolute*

  - class [freeparameter](#)
- A parameter which will be fitted in the simulation.*

  - class [discreteparameter](#)
- A parameter which will be fitted in the simulation.*

  - class [parameters](#)
- vector of parameters*

  - class [calcu](#)
- Base class to do the calculus of a constraint to the model.*

  - class [calcuba](#)
- class to do the calculus of a constraint based on a GiNaC compiled expression*

  - class [calcuex](#)
- class to do the calculus of a constraint based on a GiNaC symbolic expression*

  - class [prediction](#)
- theoretical expression for an experimental measure*

  - class [Model](#)
- Abstract class for a model.*

## Macros

- `#define _USE_MATH_DEFINES`

### 8.23.1 Macro Definition Documentation

#### 8.23.1.1 `#define _USE_MATH_DEFINES`

Definition at line 4 of file [model.h](#).

## 8.24 `model.h`

```

00001 #ifndef MODEL_H
00002 #define MODEL_H
00003
00004 #define _USE_MATH_DEFINES
00005 #include <iostream>
00006 #include <set>
00007 #include <ginac/ginac.h>
00008 #include <cmath>
00009 #include <complex>
00010 #include <memory>
00011 #include <tr1/memory>
00012 #include "TRandom3.h"
00013 #include "TMath.h"
00014
00015 //g++ teste.cpp -o teste -lcln -lginac

```

```

00016 using namespace std;
00017 //using namespace tr1;
00018 using namespace GiNaC;
00019 ///A class containing the value and uncertainty of an experimental measure
00020 class measure{
00021 public:
00022     measure(double v=0,double e=0):value(v),error(e){}
00023     measure operator*(measure m2) const{
00024         const measure & m1=*this;
00025         return measure(m1.value*m2.value,sqrt(std::pow(m1.
value*m2.error,2)+std::pow(m2.value*m1.error,2)));
00026     }
00027     measure operator/(measure m2) const{
00028         const measure & m1=*this;
00029         return measure(m1.value/m2.value,sqrt(std::pow(m1.
value/std::pow(m2.value,2)*m2.error,2)+std::pow(m1.error/m2.
value,2)));
00030     }
00031     double value;
00032     double error;
00033 };
00034 ///A base class representing an experimental measure
00035 class observable{
00036 public:
00037     observable(): copies(1) {}
00038     virtual ~observable(){}
00039
00040     /**\param hypothesis the theoretical hypothesis
00041     * \return the logarithm of the probability of measuring what was measured,
00042     * assuming that the hypothesis is true
00043     */
00044     virtual double loglikelihood(double hypothesis) const = 0;
00045     virtual double error(double hypothesis) const = 0;
00046
00047     //virtual int type() const =0;
00048     mutable uint copies;
00049 };
00050
00051 ///An experimental measure which is an upper limit on a parameter with a given Confidence Level
00052 class limitedobs: public observable{
00053 public:
00054     /**\param limit upper limit on the parameter
00055     * \param m minimum possible value of the parameter
00056     * \param p 1-Confidence Level
00057     */
00058     limitedobs(double limit, double cl=0.9, double m=0): s(fabs(limit-m)/1.282), min(m),lim(limit) {
00059         if(cl==0.95) s*=1.282/1.645;
00060         limitedobs(double limit, double cl=0.9, double m=0): s(fabs(limit-m)/(1.282+sqrt(M_PI_2))), min
(m),lim(limit) {
00061             if(cl==0.95) s*=(1.282+sqrt(M_PI_2))/(1.645+sqrt(M_PI_2));
00062         }
00063     ~limitedobs(){}
00064     double loglikelihood(double hypothesis) const {
00065         double diff=(hypothesis-min-sqrt(M_PI_2)*s)/s;
00066         if(diff<0) diff=0;
00067         return diff*diff/2;
00068     }
00069     double error(double hypothesis) const {
00070         double diff=(hypothesis-min)/s;
00071         return diff;
00072     }
00073
00074     double s,min,lim;
00075 };
00076
00077 ///An experimental measure of a parameter which is a mean value and a standard deviation
00078 class gaussobs: public observable{
00079 public:
00080
00081     /**\param mean mean value of the measure
00082     * \param sigma standard deviation of the measure
00083     */
00084     gaussobs(measure v): m(v.value), s(v.error) {}
00085     gaussobs(double mean, double sigma): m(mean), s(mean*sigma) {}
00086     ~gaussobs(){}
00087     double loglikelihood(double hypothesis) const {
00088         double diff=(m-hypothesis)/s;
00089         return diff*diff/2;
00090     }
00091     double error(double hypothesis) const {
00092         double diff=(hypothesis-m)/s;
00093         return diff;
00094     }
00095     const double m, s;
00096 };
00097
00098

```

```

00099 ///the same as gaussobs but with a different initializer, such that the uncertainty sigma is absolute
00100 class gauss2obs: public observable{
00101 public:
00102
00103 /**\param mean mean value of the measure
00104 * \param sigma standard deviation of the measure
00105 * */
00106 gauss2obs(measure v): m(v.value), s(v.error) {}
00107 gauss2obs(double mean, double sigma): m(mean), s(sigma) {}
00108 ~gauss2obs() {}
00109 double loglikelihood(double hypothesis) const {
00110     double diff=(m-hypothesis)/s;
00111     return diff*diff/2;
00112 }
00113 double error(double hypothesis) const {
00114     double diff=(hypothesis-m)/s;
00115     return diff;
00116 }
00117
00118 double expected()const {return m;}
00119 //int type() const {return 1;}
00120 const double m, s;
00121 };
00122
00123 ///A parameter which will be fitted in the simulation
00124 class freeparameter{
00125 public:
00126 /**\param mi minimum possible value for the parameter
00127 * \param ma maximum possible value for the parameter
00128 * \param r random number generator
00129 * */
00130 freeparameter(double mi, double ma, TRandom3 * r,double ss=1e-2): min(mi), max(ma), value(mi+
00131 (ma-mi)*r->Rndm()), step((ma-mi)*ss) {}
00132
00132 ///changes randomly the ::value of the parameter, the standard deviation is ::step
00133 /**\param r random number generator
00134 * */
00135 void next(TRandom3 * r,double f=1){
00136     //double x=r->Gaus()*step;
00137     value+=r->Gaus()*step*f;
00138 }
00139 ///checks if the value of the parameter is between ::min and ::max
00140 bool isvalid() const {
00141     return min<=value && value<=max;
00142 }
00143 ///probability distribution, to be used by the Markov Chain Monte Carlo simulation
00144 /**\return \f$(\frac{x-::value}{::step})^2\rf$
00145 * */
00146 double dist(double x) const {
00147     return std::pow((x-value)/step,2);
00148 }
00149 ///minimum possible value for the parameter
00150 double min;
00151 ///maximum possible value for the parameter
00152 double max;
00153 ///value of the parameter
00154 double value;
00155 ///standard deviation of the random changes of ::value in \ref next(TRandom3 *)
00156 double step;
00157
00158 };
00159
00160 ///A parameter which will be fitted in the simulation
00161 class discretparameter{
00162 public:
00163 /**\param mi minimum possible value for the parameter
00164 * \param ma maximum possible value for the parameter
00165 * \param r random number generator
00166 * */
00167 discretparameter(int mi, int ma, TRandom3 * r): min(mi), max(ma), value(mi+r->Integer(
00168 ma-mi+1)) {}
00168 ///minimum possible value for the parameter
00169 double min;
00170 ///maximum possible value for the parameter
00171 double max;
00172 ///value of the parameter
00173 double value;
00174 };
00175
00176 ///vector of parameters
00177 class parameters: public vector< freeparameter >{
00178 public:
00179
00180 //parameters() {}
00181 ///changes randomly the value of the parameters
00182 void next(TRandom3 * r, double f=1){
00183     for(iterator i=begin(); i!=end(); i++){

```



```

00184         i->next(r,f);
00185     }
00186
00187     //for(uint i=0;i<discrete.size();i++){
00188     //discrete[i].next(r);
00189     //}
00190 }
00191
00192 //checks if all the values are between their minimums and maximums
00193 bool isvalid() const{
00194     for(const_iterator i=begin();i!=end();i++){
00195         if(!i->isvalid()) return 0;
00196     }
00197     return 1;
00198 }
00199 //checks if this and another vector of parameters are within lsigma of distance
00200 double dist(const parameters& p) const{
00201     double total=0;
00202     for(uint i=0;i<size();i++){
00203         total+=at(i).dist(p[i].value);
00204     }
00205     return sqrt(total/size());
00206 }
00207
00208 void setvalues(const parameters& p){
00209
00210     for(uint i=0;i<size();i++){
00211         at(i).value=p[i].value;
00212     }
00213 }
00214
00215 double area() const{
00216     float a=1;
00217     for(const_iterator i=begin();i!=end();i++){
00218         a*=i->step;
00219     }
00220     return a;
00221 }
00222
00223 double gausslikelihood(const parameters & p2) const{
00224     double l=1;
00225     for(uint i=0;i<size();i++){
00226         l*=TMath::Gaus((p2[i].value-at(i).value)/at(i).step);
00227     }
00228     return l;
00229 }
00230
00231 lst p;
00232 vector<double> values;
00233 //vector<discreteparameters> discrete;
00234 };
00235
00236 //Base class to do the calculus of a constraint to the model
00237 class calcul{
00238     public:
00239     /**\param hipotesis the theoretical hypothesis
00240     * \return the logarithm of the probability of measuring what was measured,
00241     * assuming that the hypothesis is true
00242     */
00243     virtual double operator()(const parameters & p) const=0;
00244     //virtual int type() const =0;
00245 };
00246
00247 //class to do the calculus of a constraint based on a GiNaC compiled expression
00248 class calcula:public calcul{
00249     public:
00250     calcula(observable * ob, const FUNCP_CUBA & e0): calcul(), o(ob), e(e0){}
00251
00252     double operator()(const parameters & p) const{
00253         double ret=1000;
00254         int pass=1;
00255
00256         /* try{
00257             ret=ex_to<numeric>(e.subs(p.p,subs_options::no_pattern).evalf()).to_double();
00258         }
00259         catch(GiNaC::pole_error e){
00260             pass=0;
00261             cout<<"Pole error"<<endl;
00262         }
00263         catch(...){
00264             cout<<"Other exception"<<endl;
00265             exit(1);
00266         }
00267         */
00268         int n=p.values.size(), m=1;
00269         e(&n,&(p.values[0]),&m,&ret);
00270         if(pass) ret=o->loglikelihood(ret);

```

```

00271         else ret=1000;
00272
00273         return ret;
00274     }
00275
00276     shared_ptr<observable> o;
00277     FUNCP_CUBA e;
00278 };
00279
00280
00281 ///class to do the calculus of a constraint based on a GiNaC symbolic expression
00282 class calcuex:public calcu{
00283 public:
00284     calcuex(observable * ob, const ex & e0): calcu(), o(ob), e(e0){}
00285     ~calcuex(){}
00286
00287     double operator()(const parameters & p) const{
00288         double ret=1000;
00289         int pass=1;
00290         try{
00291             ret=ex_to<numeric>(e.subs(p.p,subs_options::no_pattern).evalf()).to_double();
00292         }
00293         catch(GiNaC::pole_error e){
00294             pass=0;
00295             cout<<"Pole error"<<endl;
00296         }
00297         catch(exception e){
00298             cout<<e.what()<<endl;
00299         }
00300         catch(...){
00301             cout<<"Other exception"<<endl;
00302             exit(1);
00303         }
00304         if(pass) ret=o->loglikelihood(ret);
00305         else ret=1000;
00306
00307         return ret;
00308     }
00309
00310     double error(const parameters & p) const{
00311         double ret=1000;
00312         int pass=1;
00313         try{
00314             cout<<e<<endl;
00315             cout<<e.subs(p.p)<<endl;
00316
00317             ret=ex_to<numeric>(e.subs(p.p,subs_options::no_pattern).evalf()).to_double();
00318         }
00319         catch(GiNaC::pole_error er){
00320             pass=0;
00321             cout<<"Pole error"<<endl;
00322         }
00323         catch(exception er){
00324             pass=0;
00325
00326             cout<<er.what()<<endl;
00327             cout<<e.subs(p.p,subs_options::no_pattern).evalf()<<endl;
00328         }
00329         catch(...){
00330             cout<<"Other exception"<<endl;
00331             exit(1);
00332         }
00333         if(pass) ret=o->error(ret);
00334         else ret=1000;
00335
00336         return ret;
00337     }
00338
00339     shared_ptr<observable> o;
00340     ex e;
00341 };
00342
00343 ///theoretical expression for an experimental measure
00344 class prediction{
00345 public:
00346     prediction(observable * ob, const FUNCP_CUBA & e0): calculate(new
00347         calcuba(ob,e0)) {}
00347     prediction(observable * ob, const ex & e0): calculate(new
00348         calcuex(ob,e0)) {}
00348     prediction(calcul * c): calculate(c) {}
00349     prediction(const prediction& p): calculate(p.calculate) {}
00350     ~prediction(){}
00351
00352     double loglikelihood(const parameters & p) const { return (*calculate)(p);}
00353     ///theoretical expression for the experimental measure
00354
00355     shared_ptr<calcu> calculate;

```

```

00356 //vector<ex> es;
00357 };
00358
00359
00360 ///Abstract class for a model
00361 class Model: public vector< prediction > {
00362 public:
00363
00364 Model(): r(new TRandom3(0)){}
00365 virtual ~Model(){delete r;};
00366 virtual parameters getlist(const parameters & p) const = 0;
00367 virtual parameters generateparameters(int max=0) const = 0;
00368 virtual int veto(const parameters & p, int max=0) const {return !p.
isvalid();}
00369
00370 ///calculates the probability of getting all the experimental measures if the model describes the reality
00371 /**\param p vector with the values of the free parameters
00372 */
00373
00374 double likelihood(const parameters & p, bool check=1, int max=0) const{
00375     if(veto(p,max) && check) return 0;
00376     double total=loglike(p,0);
00377     if(total<-1000) return 0;
00378     return exp(total);
00379 }
00380
00381 double loglike(const parameters & p, bool check=1, int max=0) const{
00382     if(veto(p,max) && check) return -1000;
00383     parameters pp(getlist(p));
00384
00385     double total=0;
00386     int n=0;
00387     for(const_iterator i=begin();i!=end();i++) {
00388         try{
00389             n++;
00390             total+=i->loglikelihood(pp);
00391         }
00392         catch(exception e){
00393             cout<<n<<e.what()<<endl;
00394             exit(1);
00395         }
00396         //catch(...){
00397             //cout<<"DD "<<n<<endl;
00398             //exit(1);
00399             //}
00400     }
00401
00402     return -total;
00403 }
00404
00405 TRandom3 * r;
00406 };
00407
00408 #endif

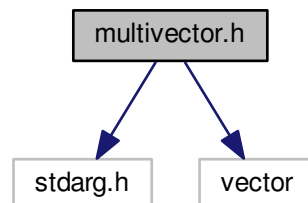
```

## 8.25 multivector.h File Reference

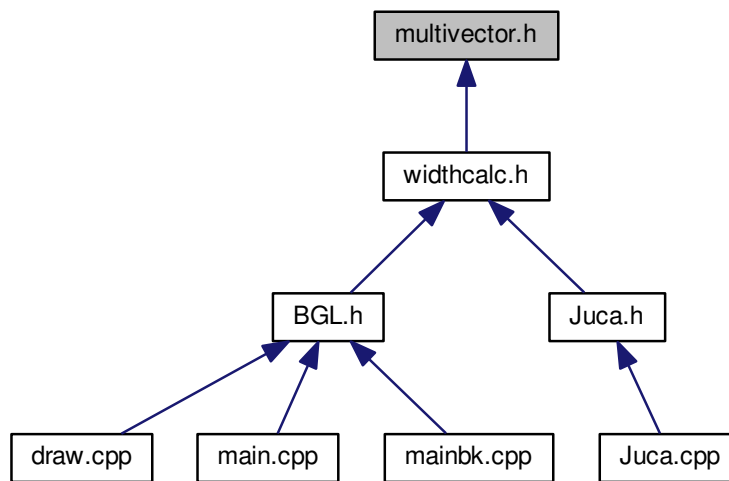
```
#include <stdarg.h>
```

```
#include <vector>
```

Include dependency graph for multivector.h:



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



## Classes

- class `std::multivector< T, N >`  
A vector of vectors of vectors of... (N times) of class T objects.
- class `std::multivector< T, 1 >`  
Specialization template class of `multivector<T,N>` for N=1.
- class `std::Matrix`

## Namespaces

- `std`

## Functions

- Matrix `std::operator*` (const Matrix &m1, const Matrix &m2)  
computes the matrix product
- Matrix `std::operator+` (const Matrix &m1, const Matrix &m2)  
computes the matrix sum

## 8.26 multivector.h

```

00001 #include <stdarg.h>
00002 #include <vector>
00003
00004 namespace std{
00005
00006     /// A vector of vectors of vectors of... (N times) of class T objects.
00007     template < class T, int N>
00008     class multivector: public vector< multivector<T, N-1> >{
00009     public:
00010         typedef vector< multivector< T,N-1> > v;
00011
00012         /// Default constructor
00013         multivector(): v() {}
00014         /// Copy constructor
00015         multivector(const multivector& m): v(m){}
00016     //Recommended constructor
00017     /** Example: multivector<double, 2> m(1.5,4,6), m is a matrix of doubles with dimensions 4x6, with all
    doubles initialized to 1.5
    * \param value the value with which every objects are initialized
    * \param ... list with the number of dimensions of each vector
    * \see multivector(const T&, va_list &)
    */
00021     multivector(const T& value, ...){
00022         va_list listPointer;
00023         va_start(listPointer,value);
00024         int n=va_arg(listPointer,int);
00025         v::insert(v::begin(),n,multivector<T,N-1>(value,listPointer));
00026         va_end(listPointer);
00027     }
00028
00029     ///Auxiliary constructor (recursive)
00030     /**\see multivector(const T&, ...)
    * \see multivector<T,1>
    */
00034     multivector(const T& value, va_list & listPointer)
00035     {
00036         int n=va_arg(listPointer,int);
00037         v::insert(v::begin(),n,multivector<T,N-1>(value,listPointer));
00038     }
00039
00040 };
00041
00042     /// Specialization template class of \ref multivector<T,N> for N=1
00043     /**\see \ref multivector<T,N>
    */
00045     template< class T >
00046     class multivector<T,1>: public vector<T>{
00047     public:
00048         typedef vector< T > v;
00049         /// Default constructor
00050         multivector(): v() {}
00051         /// Copy constructor
00052         multivector(const multivector& m): v(m){}
00053     //Recommended constructor
00054     /**\param value the value with which every objects are initialized
    * \param x number of dimensions of the vector
    */
00057     multivector(const T& value, int x): v(x,value){}
00058     ///Auxiliary constructor
00059     /**It is the last constructor to be called in the recursive constructor
    * multivector<T,N>::multivector(const T&,va_list &).
    * \see multivector<T,N>::multivector(const T&,va_list &)
    */
00063     multivector(const T& value, va_list & listPointer):
00064         v(va_arg(listPointer,int), value){}
00065 };
00066
00067     class Matrix: public multivector< ex, 2>{
00068     public:
00069
00070     Matrix(): multivector< ex,2>(0,3,3) {}
00071
00072     Matrix(const Matrix& m): multivector< ex,2>(m) {}
00073     ///constructs a symbolic matrix with the symbols names given by the argument
00074     Matrix(const char * m[3][3]): multivector< ex,2>(0,3,3){
00075         for(uint i=0;i<3;i++)
00076             for(uint j=0;j<3;j++) at(i)[j]=symbol(m[i][j]);
00077     }
00078     ///constructs a symbolic matrix with the symbols names given by the arguments
00079     Matrix(const char * name, const char ** index1, const char ** index2):
00080         multivector< ex,2>(0,3,3){
00081         for(uint i=0;i<3;i++)
00082             for(uint j=0;j<3;j++){
00083                 string res=string(name)+"_"+string(index1[i])+" "+string(index2[j])+"";

```

```

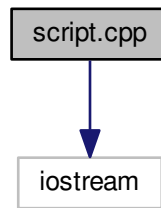
00083                                     //cout<<res<<endl;
00084                                     at(i)[j]=symbol(res.c_str());
00085                                     }
00086                                 }
00087 //constructs a diagonal matrix
00088 Matrix(ex m1, ex m2, ex m3): multivector< ex,2>(0,3,3) {
00089     at(0)[0]=m1;
00090     at(1)[1]=m2;
00091     at(2)[2]=m3;
00092 }
00093
00094 //constructs a diagonal matrix with all diagonal elements equal
00095 Matrix(ex m1): multivector< ex,2>(0,3,3){
00096     Matrix();
00097     at(0)[0]=m1;
00098     at(1)[1]=m1;
00099     at(2)[2]=m1;
00100 }
00101 //constructs a unitary matrix in the standard form
00102 Matrix(ex t12, ex t13, ex t23, ex d13): multivector< ex,2>(0,3,3) {
00103     Matrix();
00104     ex c12=cos(t12), c13=cos(t13), c23=cos(t23);
00105     ex s12=sin(t12), s13=sin(t13), s23=sin(t23);
00106     ex e13=exp(I*d13);
00107     ex e13t=ex(1)/e13;
00108
00109     ex aux[3][3]={
00110         {c12*c13,s12*c13,s13*e13t},
00111         {-s12*c23-c12*s23*s13*e13,c12*c23-s12*s23*s13*e13,s23*c13},
00112         {s12*s23-c12*c23*s13*e13,-c12*s23-s12*c23*s13*e13,c13*c23}
00113     };
00114     for(uint i=0;i<3; i++) at(i).assign(aux[i],aux[i]+3);
00115 }
00116 //used in the unitary constructor
00117 ex cs(ex t12){
00118     return (exp(I*t12)+1/exp(I*t12))/2;
00119 }
00120 //used in the unitary constructor
00121 ex sn(ex t12){
00122     return -I*(exp(I*t12)-1/exp(I*t12))/2;
00123 }
00124 //computes the hermitian conjugate of the matrix
00125 Matrix conjugate() const {
00126     Matrix res;
00127     for(uint i=0;i<3;i++)
00128         for(uint j=0;j<3;j++)
00129             res[i][j]=at(j)[i].conjugate();
00130
00131     return res;
00132 }
00133 };
00134
00135 //computes the matrix product
00136 Matrix operator*(const Matrix & m1,const Matrix & m2){
00137     Matrix res;
00138     for(uint i=0;i<3;i++)
00139         for(uint j=0;j<3;j++)
00140             for(uint k=0;k<3;k++)
00141                 res[i][j]=res[i][j]+m1[i][k]*m2[k][j];
00142     return res;
00143 }
00144
00145 //computes the matrix sum
00146 Matrix operator+(const Matrix & m1,const Matrix & m2){
00147     Matrix res;
00148     for(uint i=0;i<3;i++)
00149         for(uint j=0;j<3;j++)
00150             res[i][j]=m1[i][j]+m2[i][j];
00151     return res;
00152 }
00153
00154
00155
00156 }

```

## 8.27 script.cpp File Reference

```
#include <iostream>
```

Include dependency graph for script.cpp:



## Typedefs

- typedef unsigned int [uint](#)

## Functions

- int [main](#) ()

### 8.27.1 Typedef Documentation

#### 8.27.1.1 typedef unsigned int uint

Definition at line 4 of file [script.cpp](#).

### 8.27.2 Function Documentation

#### 8.27.2.1 int main ( )

Definition at line 5 of file [script.cpp](#).

```

00005         {
00006
00007     string g[3]={"0","1","2"};
00008     string u[3]={"0","1"};
00009
00010     string sg[3]={"1st","2nd","3rd"};
00011     string su[3]={"Down","Up"};
00012
00013     for(uint qup=0;qup<2;qup++)
00014     for(uint gQ=0;gQ<3;gQ++)
00015     {
00016         cout<<"\\pagebreak\\n";
00017         for(uint gL=0;gL<3;gL++) {
00018             cout<<"\\begin{figure}[!htb]\\n\\centering"<<endl;
00019             for(uint lup=0;lup<2;lup++) {
00020                 cout<<"\\includegraphics[width=0.49\\textwidth]{../pdfs/T_BD/pdf_";
00021                     cout<<g[gL]<<g[gQ]<<u[lup]<<u[qup];
00022                     cout<<".png}"<<endl;
00023             }
00024             cout<<"\\caption{BGL Quarks: gen "<<sg[gQ]<<" FCNC "<<su[qup];
00025             cout<<"; Leptons: gen "<<sg[gL]<<" FCNC chargedlepton/neutrino on the left/right.}"<<endl;
00026             cout<<"\\end{figure}"<<endl<<endl;
00027         }
00028     }
00029     return 0;
00030 }
  
```

## 8.28 script.cpp

```

00001 #include <iostream>
00002
00003 using namespace std;
00004 typedef unsigned int uint;
00005 int main() {
00006
00007     string g[3]={"0","1","2"};
00008     string u[3]={"0","1"};
00009
00010     string sg[3]={"1st","2nd","3rd"};
00011     string su[3]={"Down","Up"};
00012
00013     for (uint qup=0;qup<2;qup++)
00014     for (uint gQ=0;gQ<3;gQ++)
00015     {
00016         cout<<"\\pagebreak\\n";
00017         for (uint gL=0;gL<3;gL++) {
00018             cout<<"\\begin{figure}[!htb]\\n\\centering"<<endl;
00019             for (uint lup=0;lup<2;lup++) {
00020                 cout<<"\\includegraphics[width=0.49\\textwidth]{../pdfs/T_BD/pdf_"
00021                     cout<<g[GL]<<g[gQ]<<u[lup]<<u[qup];
00022                 cout<<".png}"<<endl;
00023             }
00024             cout<<"\\caption{BGL Quarks: gen "<<sg[gQ]<<" FCNC "<<su[qup];
00025             cout<<"; Leptons: gen "<<sg[GL]<<" FCNC chargedlepton/neutrino on the left/right.}"<<endl;
00026             cout<<"\\end{figure}"<<endl<<endl;
00027         }
00028     }
00029     return 0;
00030 }

```

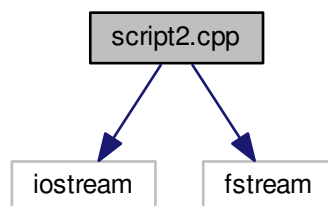
## 8.29 script2.cpp File Reference

```

#include <iostream>
#include <fstream>

```

Include dependency graph for script2.cpp:



### Typedefs

- typedef unsigned int [uint](#)

### Functions

- int [main](#) ()



## 8.29.1 Typedef Documentation

### 8.29.1.1 typedef unsigned int uint

Definition at line 6 of file [script2.cpp](#).

## 8.29.2 Function Documentation

### 8.29.2.1 int main ( )

Definition at line 7 of file [script2.cpp](#).

```

00007         {
00008
00009     string g[3]={"0","1","2"};
00010     string u[3]={"0","1"};
00011
00012     string sg[3]={"1st","2nd","3rd"};
00013     string su[3]={"Down","Up"};
00014
00015     for(uint qup=0;qup<2;qup++) {
00016     cout<<"\\pagebreak\\n";
00017     cout<<"\\begin{figure}[!htb]\\n\\centering"<<endl;
00018     for(uint gL=0;gL<3;gL++)
00019     for(uint lup=0;lup<2;lup++) {
00020     cout<<"\\begin{tikzpicture}[every node/.style={anchor=south west,inner sep=0pt},"<<endl;
00021     cout<<"x=0.307692307692\\textwidth,y=0.230769230769\\textwidth,]"<<endl;
00022
00023     for(uint gQ=0;gQ<3;gQ++)
00024     {
00025         char name[5]="0000";
00026         name[0]+=gL;
00027         name[1]+=gQ;
00028         name[2]+=lup;
00029         name[3]+=qup;
00030
00031         ifstream ff((string("pdfs/T_BD/maxs_")+string(name)+string(".out")).c_str());
00032         if(!ff.is_open()){
00033             cout<<"ERROR: maxs_*.out not found"<<endl;
00034             return 1;
00035         }
00036         int tx=0, ty=0;
00037         double xi=0,xxi=0.26,xw=0.384615384615,yi=0.25;
00038         if(gQ>0){xi=0.24+gQ; xxi+=gQ; tx=192; xw=0.307692307692;}
00039         if(gL<2 || lup<1){ty=144; yi=0.01;}
00040
00041         double McH=0,MR=0,MI=0;
00042         ff>>McH>>MR>>MI;
00043         ff.close();
00044         int mcH(McH+0.5),mR(MR+0.5),mI(MI+0.5);
00045
00046         cout<<"\\node at ("<<xi<<","0) {\\includegraphics[trim="<<tx<<" "<<ty<<" 0 0,clip,"<<endl;
00047         cout<<"width="<<xw<<"\\textwidth]{../pdfs/T_BD/pdf_"<<name<<".png}};"<<endl;
00048         cout<<"\\node at ("<<xxi<<","<<yi<<") {\\begin{minipage}{0.1\\textwidth}{<<endl;
00049         cout<<"\\scriptsize \\begin{align*} M_{H^+}&>"<<mcH<<"\\|[[-4pt]"<<endl;
00050         cout<<"M_{R^0}&>"<<mR<<"\\|[[-4pt]"<<endl;
00051         cout<<"M_{I^0}&>"<<mI<<" (GeV)\\end{align*}\\end{minipage}};"<<endl;
00052
00053
00054     }
00055     cout<<"\\end{tikzpicture}\\\\"<<endl;
00056 }
00057 cout<<"\\caption{BGL Quarks: gen ";
00058 cout<<"; Leptons: gen FCNC chargedlepton/neutrino on the left/right.}"<<endl;
00059 cout<<"\\end{figure}"<<endl<<endl;
00060 }
00061
00062
00063     for(uint qup=0;qup<2;qup++)
00064     for(uint lup=0;lup<2;lup++)
00065     for(uint gL=0;gL<3;gL++)
00066     for(uint gQ=0;gQ<3;gQ++) {
00067         cout<<"export LD_LIBRARY_PATH=.$LD_LIBRARY_PATH && ./teste "<<gL<<" "<<gQ<<" "<<lup<<" "<<qup;
00068         cout<<" "<<"teste"<<gL<<gQ<<lup<<qup<<".out"<<endl;
00069     }

```

```

00070
00071 for(uint lup=0;lup<2;lup++)
00072 for(uint qup=0;qup<2;qup++)
00073 for(uint gL=0;gL<3;gL++)
00074 for(uint gQ=0;gQ<3;gQ++){
00075     cout<<".update "<<gL<<" "<<gQ<<" "<<lup<<" "<<qup;
00076     cout<<" > "<<"teste"<<gL<<gQ<<lup<<qup<<".out"<<endl;
00077 }
00078 return 0;
00079 }

```

## 8.30 script2.cpp

```

00001 #include <iostream>
00002 #include <fstream>
00003
00004
00005 using namespace std;
00006 typedef unsigned int uint;
00007 int main(){
00008
00009 string g[3]={"0","1","2"};
00010 string u[3]={"0","1"};
00011
00012 string sg[3]={"1st","2nd","3rd"};
00013 string su[3]={"Down","Up"};
00014
00015 for(uint qup=0;qup<2;qup++){
00016     cout<<"\\pagebreak\n";
00017     cout<<"\\begin{figure}[!htb]\n\\centering"<<endl;
00018     for(uint gL=0;gL<3;gL++)
00019     for(uint lup=0;lup<2;lup++){
00020         cout<<"\\begin{tikzpicture}[every node/.style={anchor=south west,inner sep=0pt},"<<endl;
00021         cout<<"x=0.307692307692\\textwidth,y=0.230769230769\\textwidth,]"<<endl;
00022
00023         for(uint gQ=0;gQ<3;gQ++){
00024             {
00025                 char name[5]="0000";
00026                 name[0]+=gL;
00027                 name[1]+=gQ;
00028                 name[2]+=lup;
00029                 name[3]+=qup;
00030
00031                 ifstream ff((string("pdfs/T_BD/maxs_")+string(name)+string(".out")).c_str());
00032                 if(!ff.is_open()){
00033                     cout<<"ERROR: maxs_*.out not found"<<endl;
00034                     return 1;
00035                 }
00036                 int tx=0, ty=0;
00037                 double xi=0,xxi=0.26,xw=0.384615384615,yi=0.25;
00038                 if(gQ>0){xi=0.24+gQ; xxi+=gQ; tx=192; xw=0.307692307692;}
00039                 if(gL<2 || lup<1){ty=144; yi=0.01;}
00040
00041                 double McH=0,MR=0,MI=0;
00042                 ff>>McH>>MR>>MI;
00043                 ff.close();
00044                 int mCH(McH+0.5),mR(MR+0.5),mI(MI+0.5);
00045
00046                 cout<<"\\node at ("<<xi<<","0) {\\includegraphics[trim="<<tx<<" "<<ty<<" 0 0,clip,"<<endl;
00047                 cout<<"width="<<xw<<"\\textwidth]{../pdfs/T_BD/pdf_"<<name<<".png}};"<<endl;
00048                 cout<<"\\node at ("<<xxi<<","<<yi<<") {\\begin{minipage}{0.1\\textwidth}"<<endl;
00049                 cout<<"\\scriptsize \\begin{align*} M_{H^+}&>"<<mCH<<"\\[[-4pt]"<<endl;
00050                 cout<<"M_{R^0}&>"<<mR<<"\\[[-4pt]"<<endl;
00051                 cout<<"M_{I^0}&>"<<mI<<" (GeV)\\end{align*}\\end{minipage}};"<<endl;
00052
00053
00054             }
00055             cout<<"\\end{tikzpicture}\\\\"<<endl;
00056         }
00057         cout<<"\\caption{BGL Quarks: gen ";
00058         cout<<"; Leptons: gen FCNC chargedlepton/neutrino on the left/right.}"<<endl;
00059         cout<<"\\end{figure}"<<endl<<endl;
00060     }
00061
00062
00063     for(uint qup=0;qup<2;qup++)
00064     for(uint lup=0;lup<2;lup++)
00065     for(uint gL=0;gL<3;gL++)
00066     for(uint gQ=0;gQ<3;gQ++){
00067         cout<<"export LD_LIBRARY_PATH=.$LD_LIBRARY_PATH && ./teste "<<gL<<" "<<gQ<<" "<<lup<<" "<<qup;
00068         cout<<" > "<<"teste"<<gL<<gQ<<lup<<qup<<".out"<<endl;
00069     }

```

```

00070
00071 for(uint lup=0;lup<2;lup++)
00072 for(uint qup=0;qup<2;qup++)
00073 for(uint gL=0;gL<3;gL++)
00074 for(uint gQ=0;gQ<3;gQ++) {
00075     cout<<"./update "<<gL<<" "<<gQ<<" "<<lup<<" "<<qup;
00076     cout<<" > "<<"teste"<<gL<<gQ<<lup<<qup<<".out"<<endl;
00077 }
00078 return 0;
00079 }

```

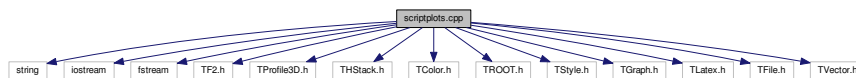
## 8.31 scriptplots.cpp File Reference

```

#include <string>
#include <iostream>
#include <fstream>
#include "TF2.h"
#include "TProfile3D.h"
#include "THStack.h"
#include "TColor.h"
#include "TROOT.h"
#include "TStyle.h"
#include "TGraph.h"
#include "TLatex.h"
#include "TFile.h"
#include "TVector.h"

```

Include dependency graph for scriptplots.cpp:



## Functions

- int [main](#) ()

### 8.31.1 Function Documentation

#### 8.31.1.1 int main ( )

Definition at line 18 of file [scriptplots.cpp](#).

```

00018     {
00019
00020     string g[3]={"0","1","2"};
00021     string u[3]={"0","1"};
00022
00023     string sg[3]={"1st","2nd","3rd"};
00024     string su[3]={"Down","Up"};
00025
00026     ofstream f[2];
00027
00028     f[0].open("draft/large_fig1.tex");
00029     f[1].open("draft/large_fig2.tex");
00030
00031     for(uint qup=0;qup<2;qup++) {

```

```

00032
00033 for (uint gL=0;gL<3;gL++)
00034 for (uint lup=0;lup<2;lup++) {
00035 f[qup]<<"\\begin{tikzpicture}[every node/.style={anchor=south west,inner sep=0pt},"<<endl;
00036 f[qup]<<"x=0.307692307692\\textwidth,y=0.230769230769\\textwidth,}"<<endl;
00037
00038 for (uint gQ=0;gQ<3;gQ++)
00039 {
00040             char name[5]="0000";
00041             name[0]+=gL;
00042             name[1]+=gQ;
00043             name[2]+=lup;
00044             name[3]+=qup;
00045             char name_[8]="0 0 0 0";
00046             name_[0]+=gL;
00047             name_[2]+=gQ;
00048             name_[4]+=lup;
00049             name_[6]+=qup;
00050
00051             //system((string("./update_pdfs/T_BD4/h")+string(name)+string("./root
pdfs/T_BD3/h")+string(name)+string("./root_pdfs/T_BD/h")+string(name)+string("./root").c_str());
00052
00053             system((string("cd pdfs/T_BD; ../../draw ./h")+string(name)+string("./root ") +string(name_)).c_str());
00054             ifstream ff((string("pdfs/T_BD/maxs_")+string(name)+string("./out").c_str()));
00055             if(!ff.is_open()){
00056                 cout<<"ERROR: maxs_*.out not found"<<endl;
00057                 return 1;
00058             }
00059             int tx=0, ty=0;
00060             double xi=0,xxi=0.26,xw=0.384615384615,yi=0.25;
00061             yi=0.25;
00062             if(gQ>0){xi=0.24+gQ; xxi+=gQ; tx=192; xw=0.307692307692;}
00063             if(gL<2 || lup<1){ty=144; yi=0.01;}
00064
00065             double McH=0,MR=0,MI=0,llmax=-1000,tbmax=0,McHmax=1000,MRmax=1000,MImax=1000;
00066             double eK_=0,sm_=0,charged_=0,neutral_=0,neutralR_=0,neutralI_=0,all_=0;
00067             ff>>McH>>MR>>MI;
00068             if(gQ>0 && qup) {
00069                 ff>>McH>>MR>>MI;
00070                 ff>>McH>>MR>>MI;
00071             }else{ ff>>llmax>>llmax>>llmax;
00072                 ff>>llmax>>llmax>>llmax;
00073             }
00074             ff>>llmax>>tbmax>>McHmax>>MRmax>>MImax;
00075             //ff>>eK_;
00076             //ff>>sm_>>charged_>>neutral_>>neutralR_>>neutralI_>>all_;
00077             ff.close();
00078             int eK_(eK_*100+0.5);
00079             int sm_(sm_*100+0.5);
00080             int charged_(charged_*100+0.5);
00081             int neutral_(neutral_*100+0.5);
00082             int neutralR_(neutralR_*100+0.5);
00083             int neutralI_(neutralI_*100+0.5);
00084             int all_(all_*100+0.5);
00085             int mCH(MCH+0.5),mR(MR+0.5),mI(MI+0.5);
00086             double tmax(int(tbmax*100+0.5)/100.0);
00087             int mCHmax(MCHmax+0.5),mRmax(MRmax+0.5),mImax(MImax+0.5);
00088             f[qup]<<"\\node at ("<<xi<<","0){\\includegraphics[trim="<<tx<< "0 0,clip,"<<endl;
00089             f[qup]<<"width="<<xw<< "\\textwidth}{../pdfs/T_BD/pdf_"<<name<<".png}};"<<endl;
00090             f[qup]<<"\\node at ("<<xxi<<","<<yi<<") {\\begin{minipage}{0.1\\textwidth}{}"<<endl;
00091             f[qup]<<"\\scriptsize \\begin{align*}"<<endl;
00092             //f[qup]<<tmax<< " & \\ " <<mCHmax<< "\\ " <<mRmax<< "\\ " <<mImax<< "\\ \\ [-4pt]"<<endl;
00093             //f[qup]<<sm_<< " & \\ " <<charged_<< "\\ " <<neutral_<< "\\ " <<neutralR_<< "\\ " <<neutralI_<< "\\
"<<all_<< "\\ \\ [-4pt]"<<endl;
00094             //f[qup]<< " & \\ " <<eK_<< "\\ \\ [-4pt]"<<endl;
00095             f[qup]<<mCH<< "<M_{H^+}/\\GeV\\ \\ \\ [-4pt]"<<endl;
00096             f[qup]<<mR<< "<M_{R^0}/\\GeV\\ \\ \\ [-4pt]"<<endl;
00097             f[qup]<<mI<< "<M_{I^0}/\\GeV\\ \\ \\ \\end{align*}}\\end{minipage}};"<<endl;
00098
00099
00100 }
00101 f[qup]<<"\\end{tikzpicture}\\ \\ \\ "<<endl;
00102 }
00103 f[qup].close();
00104 }
00105
00106 return 0;
00107 }

```

## 8.32 scriptplots.cpp

```
00001 #include <string>
```

```

00002 #include <iostream>
00003 #include <fstream>
00004
00005 #include "TF2.h"
00006 #include "TProfile3D.h"
00007 #include "THStack.h"
00008 #include "TColor.h"
00009 #include "TROOT.h"
00010 #include "TStyle.h"
00011 #include "TGraph.h"
00012 #include "TLatex.h"
00013 #include "TFile.h"
00014 #include "TVector.h"
00015
00016 using namespace std;
00017
00018 int main(){
00019
00020 string g[3]={"0","1","2"};
00021 string u[3]={"0","1"};
00022
00023 string sg[3]={"1st","2nd","3rd"};
00024 string su[3]={"Down","Up"};
00025
00026 ofstream f[2];
00027
00028 f[0].open("draft/large_fig1.tex");
00029 f[1].open("draft/large_fig2.tex");
00030
00031 for(uint qup=0;qup<2;qup++){
00032
00033 for(uint gL=0;gL<3;gL++)
00034 for(uint lup=0;lup<2;lup++){
00035 f[qup]<<"\\begin{tikzpicture}[every node/.style={anchor=south west,inner sep=0pt}],"<<endl;
00036 f[qup]<<"x=0.307692307692\\textwidth,y=0.230769230769\\textwidth,}"<<endl;
00037
00038 for(uint gQ=0;gQ<3;gQ++)
00039 {
00040 char name[5]="0000";
00041 name[0]+=gL;
00042 name[1]+=gQ;
00043 name[2]+=lup;
00044 name[3]+=qup;
00045 char name_[8]="0 0 0 0";
00046 name_[0]+=gL;
00047 name_[2]+=gQ;
00048 name_[4]+=lup;
00049 name_[6]+=qup;
00050
00051 //system((string("./update_pdfs/T_BD4/h")+string(name)+string(".root
pdfs/T_BD3/h")+string(name)+string(".root_pdfs/T_BD/h")+string(name)+string(".root")).c_str());
00052
00053 system((string("cd pdfs/T_BD; ../../draw ./h")+string(name)+string(".root ") +string(name_)).c_str());
00054 ifstream ff((string("pdfs/T_BD/maxs_")+string(name)+string(".out")).c_str());
00055 if(!ff.is_open()){
00056 cout<<"ERROR: maxs_*.out not found"<<endl;
00057 return 1;
00058 }
00059 int tx=0, ty=0;
00060 double xi=0, xxi=0.26, xw=0.384615384615, yi=0.25;
00061 yi=0.25;
00062 if(gQ>0) {xi=0.24+gQ; xxi+=gQ; tx=192; xw=0.307692307692;}
00063 if(gL<2 || lup<1) {ty=144; yi=0.01;}
00064
00065 double McH=0,MR=0,MI=0,llmax=-1000,tbmax=0,McHmax=1000,MRmax=1000,MImax=1000;
00066 double eK_=0,sm_=0,charged_=0,neutral_=0,neutralR_=0,neutralI_=0,all_=0;
00067 ff>>MCH>>MR>>MI;
00068 if(gQ>0 && qup) {
00069 ff>>MCH>>MR>>MI;
00070 ff>>MCH>>MR>>MI;
00071 }else{ ff>>llmax>>llmax>>llmax;
00072 ff>>llmax>>llmax>>llmax;
00073 }
00074 ff>>llmax>>tbmax>>MCHmax>>MRmax>>MImax;
00075 //ff>>eK_;
00076 //ff>>sm_>>charged_>>neutral_>>neutralR_>>neutralI_>>all_;
00077 ff.close();
00078 int eK_(eK_*100+0.5);
00079 int sm_(sm_*100+0.5);
00080 int charged_(charged_*100+0.5);
00081 int neutral_(neutral_*100+0.5);
00082 int neutralR_(neutralR_*100+0.5);
00083 int neutralI_(neutralI_*100+0.5);
00084 int all_(all_*100+0.5);
00085 int mCH(MCH+0.5),mR(MR+0.5),mI(MI+0.5);
00086 double tmax(int(tbmax*100+0.5)/100.0);
00087 int mCHmax(MCHmax+0.5),mRmax(MRmax+0.5),mImax(MImax+0.5);

```

```

00088 f[qup]<<"\\node at ("<<xi<<"0) {\\includegraphics[trim="<<tx<<" "<<ty<<" 0 0,clip,"<<endl;
00089 f[qup]<<"width="<<xw<<"\\textwidth]{../pdfs/T_BD/pdf_"<<name<<".png}};"<<endl;
00090 f[qup]<<"\\node at ("<<xxi<<" "<<yi<<" ) {\\begin{minipage}{0.1\\textwidth}{ "<<endl;
00091 f[qup]<<"\\scriptsize \\begin{align*}"<<endl;
00092 //f[qup]<<tmax<<" & || "<<mHmax<<"\\ "<<mRmax<<"\\ "<<mImax<<"\\\\"[-4pt]"<<endl;
00093 //f[qup]<<sm_<<" & || "<<charged_<<"\\ "<<neutral_<<"\\ "<<neutralR_<<"\\ "<<neutralI_<<"\\
"<<all_<<"\\\\"[-4pt]"<<endl;
00094 //f[qup]<<" & || "<<eK_<<"\\\\"[-4pt]"<<endl;
00095 f[qup]<<mcH<<"&<M_{H^+}/\\GeV\\\\"[-4pt]"<<endl;
00096 f[qup]<<mR<<"&<M_{R^0}/\\GeV\\\\"[-4pt]"<<endl;
00097 f[qup]<<mI<<"&<M_{I^0}/\\GeV\\end{align*}\\end{minipage}};"<<endl;
00098
00099
00100 }
00101 f[qup]<<"\\end{tikzpicture}\\\\"<<endl;
00102 }
00103 f[qup].close();
00104 }
00105
00106 return 0;
00107 }

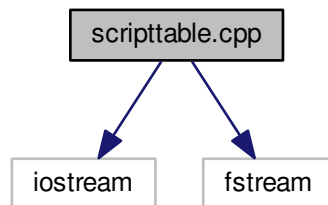
```

## 8.33 scriptable.cpp File Reference

```
#include <iostream>
```

```
#include <fstream>
```

Include dependency graph for scriptable.cpp:



### Typedefs

- typedef unsigned int [uint](#)

### Functions

- int [main](#) ()

#### 8.33.1 Typedef Documentation

##### 8.33.1.1 typedef unsigned int uint

Definition at line 5 of file [scriptable.cpp](#).

### 8.33.2 Function Documentation

#### 8.33.2.1 int main ( )

Definition at line 6 of file [scripttable.cpp](#).

```

00006         {
00007
00008     string g[3]={"0","1","2"};
00009     string u[3]={"0","1"};
00010
00011     string sg[3]={"1st","2nd","3rd"};
00012     string su[3]={"Down","Up"};
00013
00014     for (uint qup=0;qup<2;qup++)
00015     for (uint gL=0;gL<3;gL++)
00016     {
00017     for (uint lup=0;lup<2;lup++)
00018     for (uint gQ=0;gQ<3;gQ++) {
00019         char name[5]="0000";
00020         name[0]+=gL;
00021         name[1]+=gQ;
00022         name[2]+=lup;
00023         name[3]+=qup;
00024         ifstream ff((string("pdfs/T_BD/maxs_")+string(name)+string(".out")).c_str());
00025         if(!ff.is_open()){
00026             cout<<"ERROR: maxs_*.out not found"<<endl;
00027             return 1;
00028         }
00029         double McH=0,MR=0,MI=0;
00030         ff>>McH>>MR>>MI;
00031         cout<<name<<" "<<McH<<" "<<MR<<" "<<MI<<endl;
00032         ff.close();
00033     }
00034     cout<<endl;
00035 }
00036 return 0;
00037 }

```

## 8.34 scripttable.cpp

```

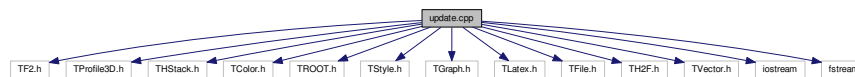
00001 #include <iostream>
00002 #include <fstream>
00003
00004 using namespace std;
00005 typedef unsigned int uint;
00006 int main() {
00007
00008     string g[3]={"0","1","2"};
00009     string u[3]={"0","1"};
00010
00011     string sg[3]={"1st","2nd","3rd"};
00012     string su[3]={"Down","Up"};
00013
00014     for (uint qup=0;qup<2;qup++)
00015     for (uint gL=0;gL<3;gL++)
00016     {
00017     for (uint lup=0;lup<2;lup++)
00018     for (uint gQ=0;gQ<3;gQ++) {
00019         char name[5]="0000";
00020         name[0]+=gL;
00021         name[1]+=gQ;
00022         name[2]+=lup;
00023         name[3]+=qup;
00024         ifstream ff((string("pdfs/T_BD/maxs_")+string(name)+string(".out")).c_str());
00025         if(!ff.is_open()){
00026             cout<<"ERROR: maxs_*.out not found"<<endl;
00027             return 1;
00028         }
00029         double McH=0,MR=0,MI=0;
00030         ff>>McH>>MR>>MI;
00031         cout<<name<<" "<<McH<<" "<<MR<<" "<<MI<<endl;
00032         ff.close();
00033     }
00034     cout<<endl;
00035 }
00036 return 0;
00037 }

```

## 8.35 update.cpp File Reference

```
#include "TF2.h"
#include "TProfile3D.h"
#include "THStack.h"
#include "TColor.h"
#include "TROOT.h"
#include "TStyle.h"
#include "TGraph.h"
#include "TLatex.h"
#include "TFile.h"
#include "TH2F.h"
#include "TVector.h"
#include <iostream>
#include <fstream>
```

Include dependency graph for update.cpp:



## Functions

- `int main (int argc, char *argv[ ])`

*the main function takes the arguments file1 file2 output, merges the results in ROOT file1 and file2 producing the output ROOT file*

### 8.35.1 Function Documentation

#### 8.35.1.1 `int main ( int argc, char * argv[ ] )`

the main function takes the arguments file1 file2 output, merges the results in ROOT file1 and file2 producing the output ROOT file

Definition at line 20 of file [update.cpp](#).

```
00020         {
00021         // Check the number of parameters
00022
00023         if(argc < 4){
00024             std::cerr<<"Usage: "<<argv[0]<<" file1 file2 output"<<std::endl;
00025             return 1;}
00026
00027         TH2F *limits4,*limits_tb_MR;
00028         TH2F *limits_MR_MI,*limits_MR_McH,*limits_MI_McH;
00029         TH2F *limits4_,*limits_tb_MR_;
00030         TH2F *limits_MR_MI_,*limits_MR_McH_,*limits_MI_McH_;
00031         TVectorD *vllmax=NULL,*vllmax_=NULL;
00032
00033         uint npoints=200;
00034
00035         TFile *f=new TFile(argv[2],"update");
00036         if(!f->IsOpen()) cout<<"NOFILE"<<endl;
00037         f->GetObject("vllmax;1",vllmax_);
00038         f->GetObject("limits4;1",limits4_);
00039         f->GetObject("limits_tb_MR;1",limits_tb_MR_);
00040         f->GetObject("limits_MR_MI;1",limits_MR_MI_);
00041         f->GetObject("limits_MR_McH;1",limits_MR_McH_);
```



```

00042         f2->GetObject("limits_MI_McH;1",limits_MI_McH_);
00043
00044         if(!vllmax_) cout<<"ERROR"<<endl;
00045
00046         TFile *f2=new TFile(argv[1],"update");
00047         if(!f2->IsOpen()) cout<<"NOFILE"<<endl;
00048
00049         f2->GetObject("vllmax;1",vllmax);
00050         f2->GetObject("limits4;1",limits4);
00051         f2->GetObject("limits_tb_MR;1",limits_tb_MR);
00052         f2->GetObject("limits_MR_MI;1",limits_MR_MI);
00053         f2->GetObject("limits_MR_McH;1",limits_MR_McH);
00054         f2->GetObject("limits_MI_McH;1",limits_MI_McH);
00055
00056         double c=(*vllmax_)[0];
00057         if(c>(*vllmax)[0]) (*vllmax)[0]=c;
00058         for(uint i=0;i<npoints;i++)
00059             for(uint j=0;j<npoints;j++) {
00060                 c=limits4->GetBinContent(i+1,j+1);
00061                 if(c>limits4->GetBinContent(i+1,j+1)) limits4->SetBinContent(i+1,j+1,c);
00062                 c=limits_tb_MR->GetBinContent(i+1,j+1);
00063                 if(c>limits_tb_MR->GetBinContent(i+1,j+1)) limits_tb_MR->SetBinContent(i+1,j+1,c);
00064                 c=limits_MR_MI->GetBinContent(i+1,j+1);
00065                 if(c>limits_MR_MI->GetBinContent(i+1,j+1)) limits_MR_MI->SetBinContent(i+1,j+1,c);
00066                 c=limits_MR_McH->GetBinContent(i+1,j+1);
00067                 if(c>limits_MR_McH->GetBinContent(i+1,j+1)) limits_MR_McH->SetBinContent(i+1,j+1,c);
00068             };
00069             c=limits_MI_McH->GetBinContent(i+1,j+1);
00070             if(c>limits_MI_McH->GetBinContent(i+1,j+1)) limits_MI_McH->SetBinContent(i+1,j+1,c);
00071         }
00072         TFile *f3=new TFile(argv[3],"recreate");
00073         if(!f3->IsOpen()) cout<<"NOFILE"<<endl;
00074
00075         vllmax->Write("vllmax");
00076         limits4->Write();
00077         limits_MR_MI->Write();
00078         limits_MR_McH->Write();
00079         limits_MI_McH->Write();
00080         limits_tb_MR->Write();
00081
00082         f3->Close();
00083         f2->Close();
00084         f->Close();
00085
00086         return 0;
00087     }
00088 }

```

## 8.36 update.cpp

```

00001 #include "TF2.h"
00002 #include "TProfile3D.h"
00003 #include "THStack.h"
00004 #include "TColor.h"
00005 #include "TROOT.h"
00006 #include "TStyle.h"
00007 #include "TGraph.h"
00008 #include "TLatex.h"
00009 #include "TFile.h"
00010 #include "TH2F.h"
00011 #include "TVector.h"
00012 #include <iostream>
00013 #include <fstream>
00014
00015 using namespace std;
00016
00017 /**
00018  * @brief the main function takes the arguments file1 file2 output, merges the results in ROOT file1 and
00019  * file2 producing the output ROOT file
00020  */
00021 int main(int argc, char* argv[]){
00022     // Check the number of parameters
00023     if(argc < 4){
00024         std::cerr<<"Usage: "<<argv[0]<<" file1 file2 output"<<std::endl;
00025         return 1;}
00026
00027     TH2F *limits4,*limits_tb_MR;
00028     TH2F *limits_MR_MI,*limits_MR_McH,*limits_MI_McH;
00029     TH2F *limits4_,*limits_tb_MR_;

```

```

00030      TH2F *limits_MR_MI_, *limits_MR_McH_, *limits_MI_McH_;
00031      TVectorD *vllmax=NULL, *vllmax_=NULL;
00032
00033      uint npoints=200;
00034
00035      TFile *f=new TFile(argv[2], "update");
00036      if(!f->IsOpen()) cout<<"NOFILE"<<endl;
00037      f->GetObject("vllmax;1", vllmax_);
00038      f->GetObject("limits4;1", limits4_);
00039      f->GetObject("limits_tb_MR;1", limits_tb_MR_);
00040      f->GetObject("limits_MR_MI;1", limits_MR_MI_);
00041      f->GetObject("limits_MR_McH;1", limits_MR_McH_);
00042      f->GetObject("limits_MI_McH;1", limits_MI_McH_);
00043
00044      if(!vllmax_) cout<<"ERROR"<<endl;
00045
00046      TFile *f2=new TFile(argv[1], "update");
00047      if(!f2->IsOpen()) cout<<"NOFILE"<<endl;
00048
00049      f2->GetObject("vllmax;1", vllmax);
00050      f2->GetObject("limits4;1", limits4);
00051      f2->GetObject("limits_tb_MR;1", limits_tb_MR);
00052      f2->GetObject("limits_MR_MI;1", limits_MR_MI);
00053      f2->GetObject("limits_MR_McH;1", limits_MR_McH);
00054      f2->GetObject("limits_MI_McH;1", limits_MI_McH);
00055
00056      double c=(*vllmax_)[0];
00057      if(c>(*vllmax)[0]) (*vllmax)[0]=c;
00058      for(uint i=0; i<npoints; i++)
00059          for(uint j=0; j<npoints; j++) {
00060              c=limits4->GetBinContent(i+1, j+1);
00061              if(c>limits4->GetBinContent(i+1, j+1)) limits4->SetBinContent(i+1, j+1, c);
00062              c=limits_tb_MR->GetBinContent(i+1, j+1);
00063              if(c>limits_tb_MR->GetBinContent(i+1, j+1)) limits_tb_MR->SetBinContent(i+1, j+1, c);
00064              c=limits_MR_MI->GetBinContent(i+1, j+1);
00065              if(c>limits_MR_MI->GetBinContent(i+1, j+1)) limits_MR_MI->SetBinContent(i+1, j+1, c);
00066              c=limits_MR_McH->GetBinContent(i+1, j+1);
00067              if(c>limits_MR_McH->GetBinContent(i+1, j+1)) limits_MR_McH->SetBinContent(i+1, j+1, c);
00068          };
00069          c=limits_MI_McH->GetBinContent(i+1, j+1);
00070          if(c>limits_MI_McH->GetBinContent(i+1, j+1)) limits_MI_McH->SetBinContent(i+1, j+1, c);
00071      };
00072      TFile *f3=new TFile(argv[3], "recreate");
00073      if(!f3->IsOpen()) cout<<"NOFILE"<<endl;
00074
00075      vllmax->Write("vllmax");
00076      limits4->Write();
00077      limits_MR_MI->Write();
00078      limits_MR_McH->Write();
00079      limits_MI_McH->Write();
00080      limits_tb_MR->Write();
00081
00082      f3->Close();
00083      f2->Close();
00084      f->Close();
00085
00086      return 0;
00087 }
00088 }
00089

```

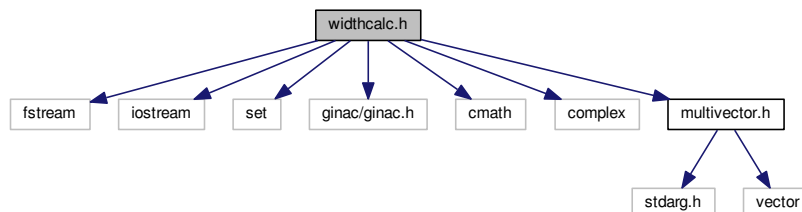
## 8.37 widthcalc.h File Reference

```

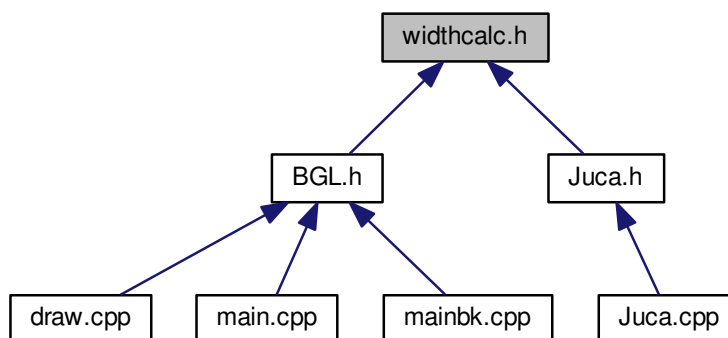
#include <fstream>
#include <iostream>
#include <set>
#include <ginac/ginac.h>
#include <cmath>
#include <complex>
#include "multivector.h"

```

Include dependency graph for widthcalc.h:



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



## Classes

- class [widthcalc](#)  
*this class calculates decay widths of one lepton to 3 leptons*

## Macros

- `#define \_USE\_MATH\_DEFINES`

### 8.37.1 Macro Definition Documentation

#### 8.37.1.1 `#define \_USE\_MATH\_DEFINES`

Definition at line 4 of file [widthcalc.h](#).

## 8.38 widthcalc.h

```

00001 #ifndef WIDTHCALC_H
00002 #define WIDTHCALC_H
00003
00004 #define _USE_MATH_DEFINES
00005 #include <fstream>
00006 #include <iostream>
00007 #include <set>
00008 #include <ginac/ginac.h>
00009 #include <cmath>
00010 #include <complex>
00011 #include "multivector.h"
00012
00013
00014 //g++ teste.cpp -o teste -lcln -lginac
00015 using namespace std;
00016 using namespace GiNaC;
00017
00018 /**
00019  * @brief this class calculates decay widths of one lepton to 3 leptons
00020  */
00021 class widthcalc{
00022
00023 public:
00024
00025 widthcalc(): M2(0,2,2,2,2,2,2,2), M22(0,2,2,2,2,2,2,2), s2("s2"), s3("s3"), mq1("mq1"), mq2("mq2"),
mq3("mq3"), mq4("mq4"){
00026
00027     integral::max_integration_level=100;
00028     integral::relative_integration_error=1e-3;
00029
00030     genM2();
00031     genM22();
00032 }
00033
00034 void genM22(){
00035     cout<<"Generating M22.dat"<<endl;
00036
00037     realsymbol mu("mu"), nu("nu"), alpha("alpha"), beta("beta"), rho("rho"), zeta("zeta");
00038     realsymbol q1("q1"), q2("q2"), q3("q3"), q4("q4");
00039     realsymbol h1("h1"), h2("h2"), h3("h3"), h4("h4");
00040
00041     varidx imu(mu,4,0), inu(nu,4,0), irho(rho,4,0);
00042     varidx ialpha(alpha,4,0), ibeta(beta,4,0), izeta(zeta,4,0);
00043
00044     varidx jmu(mu,4,1), jnu(nu,4,1), jrho(rho,4,1);
00045
00046     ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;
00047
00048     ex q2mu=indexed(q2,imu);
00049     ex q3mu=indexed(q3,imu);
00050     ex q4mu=indexed(q4,imu);
00051     ex q1mu=q2mu+q3mu+q4mu;
00052
00053     ex s4=m2q1+m2q2+m2q3+m2q4-s2-s3;
00054
00055     ex vq1=dirac_slash(q2,4)+dirac_slash(q3,4)+dirac_slash(q4,4)+mq1*dirac_ONE(), vq2=dirac_slash(q2,4)
+mq2*dirac_ONE();
00056     ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)-mq4*dirac_ONE();
00057
00058     scalar_products sp;
00059     //sp.add(q2, q3, (s4-m2q2-m2q3)/2);
00060     sp.add(q4, q3, (s2-m2q4-m2q3)/2);
00061     //sp.add(q2, q4, (s3-m2q2-m2q4)/2);
00062
00063     //sp.add(q2, q2, m2q2);
00064     sp.add(q3, q3, m2q3);
00065     sp.add(q4, q4, m2q4);
00066
00067     //sp.add(h1,h1,-1);
00068     //sp.add(h2,h2,-1);
00069     //sp.add(h3,h3,-1);
00070     //sp.add(h4,h4,-1);
00071     //sp.add(h1,h1,-1);
00072
00073     multivector<ex,3> v(0,2,2,2);
00074     v[0][0][0]=dirac_gammaL(); v[0][0][1]=dirac_gammaR();
00075     v[0][1][0]=dirac_gammaR(); v[0][1][1]=dirac_gammaL();
00076     v[1][0][0]=dirac_gammaL(); v[1][0][1]=dirac_gammaL();
00077     v[1][1][0]=dirac_gammaR(); v[1][1][1]=dirac_gammaR();
00078
00079     for(uint i=0;i<2;i++){
00080         for(uint j=0;j<2;j++){
00081             v[0][i][j]=-v[0][i][j]*s2/(mq1+mq2);
00082             v[1][i][j]=(dirac_slash(q3,4)+dirac_slash(q4,4))*v[1][i][j];

```

```

00083     }
00084
00085     //vector<ex> prop(2,0);
00086     //prop[0]=-s2/(mq1+mq2);
00087     //prop[1]=indexed(q3,imu.toggle_variance())+indexed(q4,imu.toggle_variance());
00088     /*
00089     multivector<ex,2> prop2(0,2,2);
00090     for(uint i=0;i<2;i++)
00091         for(uint j=0;j<2;j++){
00092             prop2[i][j]=prop[i]*prop[j].subs(mu==nu);
00093         }
00094     */
00095
00096     //ofstream f("M22.dat");
00097     for(uint i=0;i<2;i++)
00098         for(uint j=0;j<2;j++)
00099             for(uint k=0;k<2;k++){
00100                 for(uint l=0;l<2;l++)
00101                     for(uint m=0;m<2;m++)
00102                         for(uint n=0;n<2;n++){
00103                             //cout<<i<<" "<<j<<" "<<k<<" "<<l<<" "<<m<<" "<<n<<endl;
00104                             //cout<<dirac_trace(vq3*v[i][m][l]*vq4*v[j][n][0]).subs(mu==nu)<<endl;
00105
00106                             ex tmp=dirac_trace(vq3*v[i][m][0]*vq4*v[j][n][l])*int(pow(-1.0,double(k+1+
00107     )),);
00108
00109                             M22[i][j][k][l][m][n]=tmp.simplify_indexed(sp);
00110                             //cout<<M22[i][j][k][l][m][n]<<endl<<endl;
00111                             //f<<M22[i][j][k][l][m][n]<<endl;
00112                         }
00113     }
00114
00115 void genM2(){
00116     cout<<"Generating M2.dat"<<endl;
00117
00118     realsymbol mu("mu"), nu("nu"), alpha("alpha"), beta("beta"), rho("rho"), zeta("zeta");
00119     realsymbol q1("q1"), q2("q2"), q3("q3"), q4("q4");
00120     realsymbol h1("h1"), h2("h2"), h3("h3"), h4("h4");
00121
00122     varidx imu(mu,4,0), inu(nu,4,0), irho(rho,4,0);
00123     varidx ialpha(alpha,4,0), ibeta(beta,4,0), izeta(zeta,4,0);
00124
00125     varidx jmu(mu,4,1), jnu(nu,4,1), jrho(rho,4,1);
00126
00127     ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;
00128
00129     ex q2mu=indexed(q2,imu);
00130     ex q3mu=indexed(q3,imu);
00131     ex q4mu=indexed(q4,imu);
00132     ex q1mu=q2mu+q3mu+q4mu;
00133
00134     ex vq1=dirac_slash(q2,4)+dirac_slash(q3,4)+dirac_slash(q4,4)+mq1*dirac_ONE(), vq2=dirac_slash(q2,4)
00135     +mq2*dirac_ONE();
00136     ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)-mq4*dirac_ONE();
00137
00138     ex s4=m2q1+m2q2+m2q3+m2q4-s2-s3;
00139     scalar_products sp;
00140     sp.add(q2, q3, (s4-m2q2-m2q3)/2);
00141     sp.add(q4, q3, (s2-m2q4-m2q3)/2);
00142     sp.add(q2, q4, (s3-m2q2-m2q4)/2);
00143
00144     sp.add(q2, q2, m2q2);
00145     sp.add(q3, q3, m2q3);
00146     sp.add(q4, q4, m2q4);
00147
00148     sp.add(h1,h1,-1);
00149     sp.add(h2,h2,-1);
00150     sp.add(h3,h3,-1);
00151     sp.add(h4,h4,-1);
00152
00153     sp.add(h2,q2,0);
00154     sp.add(h3,q3,0);
00155     sp.add(h4,q4,0);
00156
00157     multivector<ex,3> v(0,2,2,2);
00158     v[0][0][0]=dirac_gammaL(); v[0][0][1]=dirac_gammaR();
00159     v[0][1][0]=dirac_gammaR(); v[0][1][1]=dirac_gammaL();
00160     v[1][0][0]=dirac_gamma(jmu)*dirac_gammaL(); v[1][0][1]=dirac_gamma(jmu)*dirac_gammaL();
00161     v[1][1][0]=dirac_gamma(jmu)*dirac_gammaR(); v[1][1][1]=dirac_gamma(jmu)*dirac_gammaR();
00162
00163     multivector<ex,7> traces(0,2,2,2,2,2,2);
00164     for(uint i=0;i<2;i++)
00165         for(uint j=0;j<2;j++)
00166             for(uint k=0;k<2;k++)
00167                 for(uint l=0;l<2;l++)
00168                     for(uint m=0;m<2;m++)

```

```

00168                                     for(uint n=0;n<2;n++){
00169                                     ex vik=v[i][k][0];
00170                                     ex vim=v[i][m][0].subs(mu==nu);
00171                                     ex vjl=v[j][l][1].subs(mu==alpha);
00172                                     ex vjn=v[j][n][1].subs(mu==beta);
00173
00174                                     traces[i][j][k][l][m][n][0]=dirac_trace(vq2*vik*vq1*vjl)*dirac_trace(vq3*
vim*vq4*vjn);
00175                                     traces[i][j][k][l][m][n][1]=-dirac_trace(vq2*vik*vq1*vjl*vq3*vim*vq4*vjn);
00176                                     }
00177
00178         vector<ex> prop(2,0);
00179         prop[0]=1;
00180         prop[1]=lorentz_g(imu,inu);
00181
00182         multivector<ex,2> prop2(0,2,2);
00183         for(uint i=0;i<2;i++)
00184             for(uint j=0;j<2;j++){
00185                 prop2[i][j]=prop[i]*prop[j].subs(1st(mu==alpha,nu==beta));
00186             }
00187
00188         //ofstream f("M2.dat");
00189         for(uint i=0;i<2;i++)
00190             for(uint j=0;j<2;j++)
00191                 for(uint k=0;k<2;k++)
00192                     for(uint l=0;l<2;l++)
00193                         for(uint m=0;m<2;m++)
00194                             for(uint n=0;n<2;n++)
00195                                 for(uint o=0;o<2;o++)
00196                                     {
00197                                         //cout<<i<<" "<<j<<" "<<k<<" "<<l<<" "<<m<<endl;
00198                                         M2[i][j][k][l][m][n][o]=(traces[i][j][k][l][m][n][o]
]*prop2[i][j]).simplify_indexed(sp);
00199                                         //cout<<M2[i][j][k][l][m][n][o]<<endl<<endl;
00200                                         //f<<M2[i][j][k][l][m][n][o]<<endl;
00201                                     }
00202     }
00203
00204
00205 ex get_integral(const multivector<ex,4>& a, const vector<ex>& mass, const
vector<int>& op, double m1, double m2, double m3, double m4) const{
00206
00207     ex q10=(s2+m1*m1-m2*m2)/(2*sqrt(s2)), lq11=sqrt(q10*q10-m1*m1);
00208     ex q30=(s2+m3*m3-m4*m4)/(2*sqrt(s2)), lq31=sqrt(q30*q30-m3*m3);
00209     ex q20=(m1*m1+m2*m2-s2)/(2*m1), lq21=sqrt(q20*q20-m2*m2);
00210
00211     ex total=0;
00212     for(uint i=0;i<a.size();i++) if(!mass[i].is_zero())
00213         for(uint j=0;j<a.size();j++){
00214             for(uint k=0;k<2;k++)
00215                 for(uint l=0;l<2;l++)
00216                     for(uint m=0;m<2;m++)
00217                         for(uint n=0;n<2;n++)
00218                             for(uint r=0;r<2;r++)
00219                                 for(uint s=0;s<2;s++){
00220                                     ex coup=a[i][r][k][m]*a[j][s][l][n].conjugate();
00221                                     if(!coup.is_zero()){
00222                                         //cout<<i<<" "<<j<<" "<<k<<" "<<l<<endl;
00223                                         ex integrand=M2[op[i]][op[j]][k][l][m][n][(r+s)%2];
00224                                         integrand=expand(integral(s3, m1*m1+m2*m2-s2)*q10*q30-2*lq11*lq31
, m1*m1+m3*m3-2*q10*q30+2*lq11*lq31, integrand)\
00225
00226                                         eval_integ()/lq11/sqrt(s2)*lq21/m1/m1);
00227                                         double mm2=m2, mm3=m3;
00228                                         if(l) {mm2=m3; mm3=m2;}
00229                                         double result=ex_to_numeric>(integral(s2,std::pow(mm3+m4,2),
std::pow(m1-mm2,2),integrand.subs(1st(m1==m1,m2==mm2,m3==mm3,m4==m4))).evalf()).to_double()/std::pow
(M_PI,3)/512;
00229                                         ex partial=result*coup/(pow(mass[i],2)*pow(mass[j],2));
00230                                         //cout<<partial<<endl;
00231                                         total=total+partial;
00232                                     }
00233                                 }
00234
00235         return total;
00236     }
00237 ex get_integral_symb(const multivector<ex,4>& a, const vector<ex>& mass,
const vector<int>& op, ex m1) const{
00238
00239     ex q10=(s2+m1*m1)/(2*sqrt(s2)), lq11=(m1*m1-s2)/(2*sqrt(s2));
00240     ex q30=sqrt(s2)/2, lq31=q30;
00241     ex q20=(m1*m1-s2)/(2*m1), lq21=q20;
00242
00243     ex total=0;
00244     for(uint i=0;i<a.size();i++) if(!mass[i].is_zero())
00245         for(uint j=0;j<a.size();j++){
00246             for(uint k=0;k<2;k++)

```

```

00247         for(uint l=0;l<2;l++)
00248         for(uint m=0;m<2;m++)
00249         for(uint n=0;n<2;n++)
00250         for(uint r=0;r<2;r++)
00251         for(uint s=0;s<2;s++){
00252             ex coup=a[i][r][k][m]*a[j][s][l][n].conjugate();
00253             if(!coup.is_zero()){
00254                 //cout<<i<<" "<<j<<" "<<k<<" "<<l<<" "<<n<<" "<<endl;
00255                 ex integrand=M2[op[i]][op[j]][k][l][m][n][(r+s)%2].subs(lst(mq1 ==
ml, mq2 == 0, mq3 == 0, mq4 == 0));
00256                 integrand=expand(integral(s3, m1*m1-2*q10*q30-2*lq1l+lq3l, m1*m1-2*
q10*q30+2*lq1l*lq3l, integrand)\
00257 eval_integ()/lq1l/sqrt(s2)*lq2l/m1/m1);
00258                 //integrand=integrand.subs(lst(mq1 == m1, mq2 == 0, mq3 == 0, mq4
== 0));
00259                 //
integrand=integrand.subs(pow(m1,2)/4-s2/2+pow(s2/m1,2)/4==pow((m1-s2/m1)/2,2));
00260
00261
00262                 double mm2=0, mm3=0, m4=0;
00263                 //if(l) {mm2=mm3; mm3=m2;}
00264                 ex result=integral(s2, std::pow(mm3+m4,2), pow(m1-mm2,2), integrand) .
eval_integ()/pow(Pi,3)/512;
00265
00266                 ex partial=result*coup/(pow(mass[i],2)*pow(mass[j],2));
00267                 total=total+partial;
00268             }
00269         }
00270         return total;
00271     }
00272     ex get_integral_meson(const multivector<ex,4>& a, const vector<ex>& mass
, const vector<int>& op, ex mm, ex m1, ex m2, ex m3, ex m4) const{
00273
00274         ex q10=(s2+mq1*mq1-mq2*mq2)/(2*sqrt(s2)), lq1l=sqrt(q10*q10-mq1*mq1);
00275         ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);
00276         ex total=0;
00277         for(uint i=0;i<a.size();i++){
00278             for(uint j=0;j<a.size();j++){
00279                 for(uint k=0;k<2;k++){
00280                     for(uint l=0;l<2;l++){
00281                         for(uint m=0;m<2;m++){
00282                             for(uint n=0;n<2;n++){
00283                                 ex coup=a[i][0][k][m]*a[j][0][l][n].conjugate();
00284                                 if(!coup.is_zero()){
00285                                     //cout<<i<<" "<<j<<" "<<k<<" "<<l<<" "<<m<<" "<<n<<" "<<endl;
00286                                     ex integrand=M22[op[i]][op[j]][k][l][m][n];
00287                                     //cout<<collect_common_factors(expand(a[i][0][k][m]))<<endl;
00288                                     //
cout<<collect_common_factors(expand(a[j][0][l][n].conjugate()))<<endl;
00289                                     integrand=expand(integral(s3, mq1*mq1+mq3*mq3-2*q10*q30-2*lq1l+lq3l
, mq1*mq1+mq3*mq3-2*q10*q30+2*lq1l*lq3l, integrand)/lq1l/s2);
00290                                     ex result=integrand.subs(lst(sqrt(s2) == mm, s2==mm*mm, mq1 == m1,
mq2 == m2, mq3 == m3, mq4 == m4))/Pi/128;
00291                                     ex mi=mass[i];
00292                                     if(mi.is_zero()) mi=mm;
00293                                     ex mj=mass[j];
00294                                     if(mj.is_zero()) mj=mm;
00295                                     ex partial=result*coup/(pow(mi,2)*pow(mj,2));
00296                                     //cout<<i<<" "<<op[i]<<" "<<j<<" "<<op[j]<<"
"<<a[i]*a[j].conjugate()/(pow(mass[i],2)*pow(mass[j],2))<<endl<<endl;
00297
00298                                     total=total+partial;
00299                                 }
00300                             }
00301                         }
00302                     }
00303                 }
00304             }
00305             return total;
00306         }
00307     }
00308     ex get_integral_meson2(const multivector<ex,4>& a, const vector<ex>&
mass, const vector<int>& op, ex mm, ex m1, ex m2, ex m3, ex m4) const{
00309
00310         ex q10=(s2+mq1*mq1-mq2*mq2)/(2*sqrt(s2)), lq1l=sqrt(q10*q10-mq1*mq1);
00311         ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);
00312         ex total=0;
00313         for(uint i=0;i<a.size();i++){
00314             for(uint j=0;j<a.size();j++){
00315                 for(uint k=0;k<2;k++){
00316                     for(uint l=0;l<2;l++){
00317                         for(uint m=0;m<2;m++){
00318                             for(uint n=0;n<2;n++){
00319                                 ex coup=a[i][0][k][m]*a[j][0][l][n].conjugate();
00320                                 if(!coup.is_zero()){
00321                                     //cout<<i<<" "<<j<<" "<<k<<" "<<l<<" "<<m<<" "<<n<<" "<<endl;
00322                                     ex integrand=M22[op[i]][op[j]][k][l][m][n];
00323                                     //cout<<collect_common_factors(expand(a[i][0][k][m]))<<endl;
00324                                     //

```

```

    cout<<collect_common_factors(expand(a[j][0][1][n].conjugate()))<<endl;
00322     integrand=expand(integral(s3, mql*mql+mq3*mq3-2*q10*q30-2*lq11*lq31
, mql*mql+mq3*mq3-2*q10*q30+2*lq11*lq31, integrand)/lq11/s2);
00323     ex result=integrand.subs(lst(sqrt(s2) == mm, s2==mm*mm, mql == m1,
mq2 == m2, mq3 == m3, mq4 == m4))/Pi/128;
00324     ex mi=mass[i];
00325     if(mi.is_zero()) mi=mm;
00326     ex mj=mass[j];
00327     if(mj.is_zero()) mj=mm;
00328     ex partial=result*coup/(pow(mi,2)*pow(mj,2));
00329     //cout<<i<<" "<<op[i]<<" "<<j<<" "<<op[j]<<"
"<<a[i]*a[j].conjugate()/(pow(mass[i],2)*pow(mass[j],2))<<endl<<endl;
00330
00331     total=total+partial;
00332 }
00333 }
00334
00335     return total;
00336 }
00337 /*
00338 ex get_integral_meson(const multivector<ex,2>& a, const vector<ex>& mass, const vector<int>& op, double
meson_mass, double m3, double m4) const{
00339
00340     ex q10=(meson_mass)/(2), lq11=sqrt(q10*q10-mql*mql);
00341     ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq31=sqrt(q30*q30-mq3*mq3);
00342     ex q20=(mql*mql+mq2*mq2-s2)/(2*mql), lq21=sqrt(q20*q20-mq2*mq2);
00343
00344
00345     ex total=0;
00346     for(uint i=0;i<a.size();i++)
00347         for(uint j=0;j<a.size();j++)
00348             for(uint k=0;k<2;k++)
00349                 for(uint l=0;l<2;l++) if(!(a[i][k]*a[j][l]).is_zero()){
00350                     //cout<<i<<" "<<j<<" "<<k<<" "<<l<<endl;
00351                     ex integrand=M2[op[i]/2][op[j]/2][op[i]%2][op[j]%2][(k+l)%2];
00352                     integrand=expand(integral(s3, mql*mql+mq3*mq3-2*q10*q30-2*lq11*
lq31, mql*mql+mq3*mq3-2*q10*q30+2*lq11*lq31, integrand)\
00353 .eval_integ()/lq11/sqrt(s2)*lq21/mql/mql);
00354                     double mm2=m2, mm3=m3;
00355                     if(l){mm2=m3; mm3=m2;}
00356                     double
result=ex_to<numeric>(integral(s2,pow(mm3+m4,2),pow(m1-mm2,2),integrand.subs(lst(mql == m1, mq2 == mm2, mq3 == mm3, mq4 == mm4),
00357 pow(mass[j],2)));
00358                     //cout<<partial<<endl;
00359                     total=total+partial;
00360 }
00361 }
00362     return total;
00363 }
00364 */
00365 multivector<ex,7> M2;
00366 multivector<ex,6> M22;
00367 realsymbol s2, s3;
00368 realsymbol mql, mq2, mq3, mq4;
00369
00370 };
00371
00372
00373
00374 #endif

```



# Index

`_USE_MATH_DEFINES`

`model.h`, [244](#)

`widthcalc.h`, [265](#)

`~BGL`

`BGLmodels::BGL`, [30](#)

`~BGL2`

`BGL2`, [56](#)

`~Juca`

`Juca`, [110](#)

`~Model`

`Model`, [132](#)

`~calcuex`

`calcuex`, [92](#)

`~gauss2obs`

`gauss2obs`, [104](#)

`~gaussobs`

`gaussobs`, [107](#)

`~limitedobs`

`limitedobs`, [115](#)

`~observable`

`observable`, [140](#)

`~prediction`

`prediction`, [150](#)

**A**

`Juca`, [112](#)

**a**

`BGLmodels::calcubtosgamma2`, [88](#)

**A0**

`BGLmodels::BGL`, [30](#)

`BGLmodels::calcubtosgamma2`, [83](#)

**A1**

`BGLmodels::BGL`, [30](#)

`BGLmodels::calcubtosgamma2`, [84](#)

**A2**

`BGLmodels::BGL`, [31](#)

`BGLmodels::calcubtosgamma2`, [84](#)

**A3**

`BGLmodels::BGL`, [31](#)

`BGLmodels::calcubtosgamma2`, [84](#)

**a77**

`BGLmodels::calcubtosgamma2`, [88](#)

**a7ei**

`BGLmodels::calcubtosgamma2`, [88](#)

**a7er**

`BGLmodels::calcubtosgamma2`, [88](#)

**a7i**

`BGLmodels::calcubtosgamma2`, [89](#)

**a7r**

`BGLmodels::calcubtosgamma2`, [89](#)

**a87i**

`BGLmodels::calcubtosgamma2`, [89](#)

**a87r**

`BGLmodels::calcubtosgamma2`, [89](#)

**a88**

`BGLmodels::calcubtosgamma2`, [89](#)

**a8ei**

`BGLmodels::calcubtosgamma2`, [89](#)

**a8er**

`BGLmodels::calcubtosgamma2`, [89](#)

**a8i**

`BGLmodels::calcubtosgamma2`, [89](#)

**a8r**

`BGLmodels::calcubtosgamma2`, [89](#)

**add**

`BGLmodels::BGL`, [31](#)

`Juca`, [110](#)

**adjuststeps**

`Peak`, [146](#)

**aee**

`BGLmodels::calcubtosgamma2`, [89](#)

**aei**

`BGLmodels::calcubtosgamma2`, [90](#)

**aer**

`BGLmodels::calcubtosgamma2`, [90](#)

**alpha**

`BGL2`, [59](#)

`BGLmodels::BGL`, [45](#)

**appendtolst**

`BGLmodels::Mixes`, [128](#)

**area**

`parameters`, [142](#)

`Peak`, [147](#)

**BGL.h**, [163](#)

**BGL2**, [51](#)

`~BGL2`, [56](#)

`alpha`, [59](#)

`BGL2`, [54](#)

`BGLtype`, [59](#)

`BR_Htotaunu`, [59](#)

`BR_toptoHq`, [59](#)

`bosons`, [59](#)

`bsgammawidth`, [57](#)

`Btaunu`, [59](#)

`BtoD2taunuR`, [59](#)

`BtoDtaunuR`, [59](#)

`BtotaunuR`, [59](#)

`conjtoabs`, [59](#)

`cos2`, [59](#)

- cp, [60](#)
- epsilonK, [60](#)
- epsK, [57](#)
- FBs, [60](#)
- FDs, [60](#)
- FB, [60](#)
- FD, [60](#)
- FK, [60](#)
- Fpi, [60](#)
- g, [60](#)
- generateparameters, [57](#)
- getlist, [57](#)
- GF, [60](#)
- iBD2taunu, [61](#)
- iBDtaunu, [61](#)
- iBtaunu, [61](#)
- inter1, [61](#)
- inter2, [61](#)
- MB0, [61](#)
- MBp, [61](#)
- MBs0, [61](#)
- MD0, [62](#)
- MDp, [62](#)
- MDs0, [62](#)
- MDsp, [62](#)
- MK0, [62](#)
- MKp, [62](#)
- McH, [61](#)
- Md, [61](#)
- Md\_, [62](#)
- mesonmixing, [58](#)
- Mh, [62](#)
- MI, [62](#)
- mixes, [62](#)
- mmmax, [63](#)
- Mpi0, [63](#)
- Mpip, [63](#)
- MR, [63](#)
- Mu, [63](#)
- mu, [63](#)
- Mu\_, [63](#)
- MW, [63](#)
- MZ, [63](#)
- planck, [63](#)
- QCD1, [64](#)
- QCD2, [64](#)
- replacements, [64](#)
- rho, [64](#)
- Sparam, [64](#)
- stepsize, [64](#)
- tanb, [64](#)
- Tparam, [64](#)
- BGLmodels, [11](#)
- BSpin, [12](#)
- C7SM\_Mb, [15](#)
- C7SM\_Mt, [15](#)
- C7SM\_MW, [15](#)
- C7SM, [14](#)
- C8SM\_Mb, [15](#)
- C8SM\_Mt, [15](#)
- C8SM\_MW, [15](#)
- C8SM, [14](#)
- cAntiParticle, [13](#)
- cParticle, [13](#)
- CD, [12](#)
- fAny, [13](#)
- FCharge, [12](#)
- fElectron, [13](#)
- FFlavour, [13](#)
- FHelicity, [13](#)
- Flsospin, [13](#)
- fMuon, [13](#)
- fTau, [13](#)
- FType, [13](#)
- hAny, [13](#)
- hLeft, [13](#)
- hRight, [13](#)
- iDown, [13](#)
- iUp, [13](#)
- M\_GF, [15](#)
- M\_Md, [15](#)
- M\_MI, [15](#)
- M\_Mu, [16](#)
- M\_MW, [16](#)
- M\_MZ, [16](#)
- M\_cos2, [15](#)
- Matrix3c, [12](#)
- mt\_mb, [16](#)
- mt\_mt, [16](#)
- mt\_mW, [16](#)
- sAny, [12](#)
- sScalar, [12](#)
- sVector, [12](#)
- tLepton, [14](#)
- tQuark, [14](#)
- Vector3c, [12](#)
- Vnl, [16](#)
- Vud, [14](#)
- BGLmodels::BGL, [19](#)
- ~BGL, [30](#)
- A0, [30](#)
- A1, [30](#)
- A2, [31](#)
- A3, [31](#)
- add, [31](#)
- alpha, [45](#)
- BGLtype, [45](#)
- BGL, [22](#)
- BR\_Htotaunu, [46](#)
- BR\_toptoHq, [46](#)
- bosons, [45](#)
- BranchingRatio, [31](#)
- bsgammawidth, [32](#)
- Btaunu, [46](#)
- BtoD2taunuR, [46](#)
- BtoDtaunuR, [46](#)

- BtotaunuR, 46
- cBmumu, 46
- cBsmumu, 46
- CHdecaycoupling, 32
- conjtoabs, 46
- cos2, 47
- cp, 47
- decaywidth, 33
- decaywidthtest2, 33
- FBs, 47
- FDs, 47
- FH1, 36
- FH2, 36
- FB, 47
- FD, 47
- fermiontomeson, 34
- fermiontomesontest, 35
- Fh1, 36
- Fh2, 37
- FK, 47
- Fpi, 47
- FW, 37
- g, 47
- GH1, 39
- GH2, 39
- gRR2, 40
- generateparameters, 37
- get\_integral\_symb, 37
- getlist, 38
- GF, 47
- GW, 40
- iBD2taunu, 48
- iBDtaunu, 48
- iBtaunu, 48
- MB0, 48
- MBp, 48
- MBs0, 48
- MD0, 48
- MDp, 48
- MDs0, 49
- MDsp, 49
- MK0, 49
- MKp, 49
- McH, 48
- Md, 48
- mesondw, 40
- mesondwtest, 42
- mesonmixing, 43
- Mh, 49
- MI, 49
- mixes, 49
- mmmax, 49
- Mpi0, 49
- Mpip, 49
- MR, 50
- Mu, 50
- mu, 50
- MW, 50
- MZ, 50
- planck, 50
- replacements, 50
- rho, 50
- stepsize, 50
- tanb, 50
- tautomu\_tautoe, 43
- topBranchingRatio, 44
- veto, 44
- wc, 51
- Y, 45
- BGLmodels::Boson, 65
  - Boson, 66
  - C, 70
  - coupling, 66
  - couplingdaggerL, 66
  - couplingdaggerR, 67
  - couplingL, 68
  - couplingR, 68
  - mass, 70
  - reset, 69
  - s, 70
- BGLmodels::Fermion, 99
  - Fermion, 99
  - flavour, 100
  - helicity, 100
  - isospin, 100
  - particle, 100
  - type, 100
- BGLmodels::Matrixx, 121
  - conjugate, 122
  - Matrixx, 122
- BGLmodels::Meson, 124
  - decay\_factor, 125
  - mass, 125
  - Meson, 125
  - q1, 125
  - q2, 125
- BGLmodels::Mixes, 126
  - appendtolst, 128
  - cp, 129
  - M, 129
  - mass, 129
  - massnum, 129
  - Mixes, 127
  - N, 129
  - N\_, 129
  - replacements, 129
  - reps, 130
  - tanb, 130
  - V, 130
  - VN\_, 130
  - VN, 130
- BGLmodels::calcuBmumu, 74
  - calcuBmumu, 76
  - ff3, 78
  - ff4, 78
  - fp, 79

- gAi, [79](#)
- gAr, [79](#)
- gPi, [79](#)
- gPr, [79](#)
- gSi, [79](#)
- gSr, [79](#)
- meson, [79](#)
- mesondwtest, [77](#)
- mixes, [79](#)
- o, [79](#)
- obsvalue, [77](#)
- operator(), [78](#)
- Y, [78](#)
- BGLmodels::calcuOblique, [94](#)
  - c1, [97](#)
  - c2, [97](#)
  - calcuOblique, [95](#)
  - F, [96](#)
  - f, [96](#)
  - G, [96](#)
  - g1, [97](#)
  - g2, [97](#)
  - lnxy\_xy, [96](#)
  - operator(), [96](#)
- BGLmodels::calcubtosgamma2, [80](#)
  - a, [88](#)
  - A0, [83](#)
  - A1, [84](#)
  - A2, [84](#)
  - A3, [84](#)
  - a77, [88](#)
  - a7ei, [88](#)
  - a7er, [88](#)
  - a7i, [89](#)
  - a7r, [89](#)
  - a87i, [89](#)
  - a87r, [89](#)
  - a88, [89](#)
  - a8ei, [89](#)
  - a8er, [89](#)
  - a8i, [89](#)
  - a8r, [89](#)
  - aee, [89](#)
  - aei, [90](#)
  - aer, [90](#)
  - calcubtosgamma2, [82](#)
  - calN, [90](#)
  - fp, [90](#)
  - g1, [90](#)
  - g2, [90](#)
  - ii, [90](#)
  - inter1, [90](#)
  - inter2, [90](#)
  - inter3, [90](#)
  - inter4, [91](#)
  - Md\_, [91](#)
  - Mu\_, [91](#)
  - operator(), [84](#)
  - ratio, [91](#)
  - width, [86](#)
- BGLtype
  - BGL2, [59](#)
  - BGLmodels::BGL, [45](#)
- BGL
  - BGLmodels::BGL, [22](#)
- BR\_Htotaunu
  - BGL2, [59](#)
  - BGLmodels::BGL, [46](#)
- BR\_toptoHq
  - BGL2, [59](#)
  - BGLmodels::BGL, [46](#)
- BSpin
  - BGLmodels, [12](#)
- Boson
  - BGLmodels::Boson, [66](#)
- bosons
  - BGL2, [59](#)
  - BGLmodels::BGL, [45](#)
- BranchingRatio
  - BGLmodels::BGL, [31](#)
- bsgammawidth
  - BGL2, [57](#)
  - BGLmodels::BGL, [32](#)
- Btaunu
  - BGL2, [59](#)
  - BGLmodels::BGL, [46](#)
- BtoD2taunuR
  - BGL2, [59](#)
  - BGLmodels::BGL, [46](#)
- BtoDtaunuR
  - BGL2, [59](#)
  - BGLmodels::BGL, [46](#)
- BtotaunuR
  - BGL2, [59](#)
  - BGLmodels::BGL, [46](#)
- C
  - BGLmodels::Boson, [70](#)
- c1
  - BGLmodels::calcuOblique, [97](#)
- c2
  - BGLmodels::calcuOblique, [97](#)
- C7SM\_Mb
  - BGLmodels, [15](#)
- C7SM\_Mt
  - BGLmodels, [15](#)
- C7SM\_MW
  - BGLmodels, [15](#)
- C7SM
  - BGLmodels, [14](#)
- C8SM\_Mb
  - BGLmodels, [15](#)
- C8SM\_Mt
  - BGLmodels, [15](#)
- C8SM\_MW
  - BGLmodels, [15](#)
- C8SM

- BGLmodels, 14
- cAntiParticle
  - BGLmodels, 13
- cBmumu
  - BGLmodels::BGL, 46
- cBsmumu
  - BGLmodels::BGL, 46
- CHdecaycoupling
  - BGLmodels::BGL, 32
- cParticle
  - BGLmodels, 13
- calcu, 71
  - operator(), 71
- calcuBmumu
  - BGLmodels::calcuBmumu, 76
- calcuOblique
  - BGLmodels::calcuOblique, 95
- calcuba, 72
  - calcuba, 73
  - e, 74
  - o, 74
  - operator(), 73
- calcubtosgamma2
  - BGLmodels::calcubtosgamma2, 82
- calcuex, 91
  - ~calcuex, 92
  - calcuex, 92
  - e, 94
  - error, 93
  - o, 94
  - operator(), 93
- calculate
  - prediction, 151
- calN
  - BGLmodels::calcubtosgamma2, 90
- CD
  - BGLmodels, 12
- chi2
  - eigen.cpp, 198
- conjtoabs
  - BGL2, 59
  - BGLmodels::BGL, 46
- conjugate
  - BGLmodels::Matrixx, 122
  - std::Matrix, 119
- copies
  - observable, 141
- cos2
  - BGL2, 59
  - BGLmodels::BGL, 47
- coupling
  - BGLmodels::Boson, 66
- couplingdaggerL
  - BGLmodels::Boson, 66
- couplingdaggerR
  - BGLmodels::Boson, 67
- couplingL
  - BGLmodels::Boson, 68
- couplingR
  - BGLmodels::Boson, 68
- cp
  - BGL2, 60
  - BGLmodels::BGL, 47
  - BGLmodels::Mixes, 129
- cs
  - std::Matrix, 120
- decay\_factor
  - BGLmodels::Meson, 125
- decaywidth
  - BGLmodels::BGL, 33
- decaywidthtest2
  - BGLmodels::BGL, 33
- defs.h, 183
  - GL, 184
  - GQ, 184
  - UL, 184
  - UQ, 184
- discreteparameter, 97
  - discreteparameter, 98
  - max, 98
  - min, 98
  - value, 98
- dist
  - freeparameter, 101
  - parameters, 142
- draw.cpp, 184
  - main, 185
- e
  - calcuba, 74
  - calcuex, 94
- eigen.cpp, 198
  - chi2, 198
  - main, 199
- eigenteste.cpp, 202
  - main, 202
- epsilonK
  - BGL2, 60
- epsK
  - BGL2, 57
- error
  - calcuex, 93
  - gauss2obs, 105
  - gaussobs, 107
  - limitedobs, 115
  - measure, 124
  - observable, 140
  - points, 149
- eta
  - Juca, 112
- expected
  - gauss2obs, 105
- F
  - BGLmodels::calcuOblique, 96
- f

- BGLmodels::calcuOblique, 96
- fAny
  - BGLmodels, 13
- FBs
  - BGL2, 60
  - BGLmodels::BGL, 47
- FCharge
  - BGLmodels, 12
- FDs
  - BGL2, 60
  - BGLmodels::BGL, 47
- fElectron
  - BGLmodels, 13
- FFlavour
  - BGLmodels, 13
- FH1
  - BGLmodels::BGL, 36
- FH2
  - BGLmodels::BGL, 36
- FHelicity
  - BGLmodels, 13
- FIsospin
  - BGLmodels, 13
- fMuon
  - BGLmodels, 13
- fTau
  - BGLmodels, 13
- FType
  - BGLmodels, 13
- FB
  - BGL2, 60
  - BGLmodels::BGL, 47
- FD
  - BGL2, 60
  - BGLmodels::BGL, 47
- Fermion
  - BGLmodels::Fermion, 99
- fermionmeson
  - BGLmodels::BGL, 34
- fermionmesontest
  - BGLmodels::BGL, 35
- ff3
  - BGLmodels::calcuBmumu, 78
- ff4
  - BGLmodels::calcuBmumu, 78
- Fh1
  - BGLmodels::BGL, 36
- Fh2
  - BGLmodels::BGL, 37
- findPeak
  - Peak, 146
- findPeaks
  - Proposal, 152
- FK
  - BGL2, 60
  - BGLmodels::BGL, 47
- flavour
  - BGLmodels::Fermion, 100
- floatPeak
  - Proposal, 154
- Formulas.h, 203
- fp
  - BGLmodels::calcuBmumu, 79
  - BGLmodels::calcubtosgamma2, 90
- Fpi
  - BGL2, 60
  - BGLmodels::BGL, 47
- freeparameter, 100
  - dist, 101
  - freeparameter, 101
  - isvalid, 102
  - max, 102
  - min, 102
  - next, 102
  - step, 102
  - value, 103
- FW
  - BGLmodels::BGL, 37
- G
  - BGLmodels::calcuOblique, 96
- g
  - BGL2, 60
  - BGLmodels::BGL, 47
- g1
  - BGLmodels::calcuOblique, 97
  - BGLmodels::calcubtosgamma2, 90
- g2
  - BGLmodels::calcuOblique, 97
  - BGLmodels::calcubtosgamma2, 90
- gAi
  - BGLmodels::calcuBmumu, 79
- gAr
  - BGLmodels::calcuBmumu, 79
- GH1
  - BGLmodels::BGL, 39
- GH2
  - BGLmodels::BGL, 39
- gPi
  - BGLmodels::calcuBmumu, 79
- gPr
  - BGLmodels::calcuBmumu, 79
- gRR2
  - BGLmodels::BGL, 40
- gSi
  - BGLmodels::calcuBmumu, 79
- gSr
  - BGLmodels::calcuBmumu, 79
- gauss2obs, 103
  - ~gauss2obs, 104
  - error, 105
  - expected, 105
  - gauss2obs, 104
  - loglikelihood, 105
  - m, 105
  - s, 105
- gausslikelihood

- parameters, 143
- gaussobs, 106
  - ~gaussobs, 107
  - error, 107
  - gaussobs, 107
  - loglikelihood, 107
  - m, 108
  - s, 108
- genM2
  - widthcalc, 156
- genM22
  - widthcalc, 157
- generateparameters
  - BGL2, 57
  - BGLmodels::BGL, 37
  - Juca, 110
  - Model, 132
- get\_integral
  - widthcalc, 158
- get\_integral\_meson
  - widthcalc, 159
- get\_integral\_meson2
  - widthcalc, 160
- get\_integral\_symb
  - BGLmodels::BGL, 37
  - widthcalc, 160
- getNextPoint
  - Proposal, 153
- getProposal
  - Proposal, 154
- getlist
  - BGL2, 57
  - BGLmodels::BGL, 38
  - Juca, 111
  - Model, 132
- GF
  - BGL2, 60
  - BGLmodels::BGL, 47
- GL
  - defs.h, 184
- GQ
  - defs.h, 184
- GW
  - BGLmodels::BGL, 40
- hAny
  - BGLmodels, 13
- hLeft
  - BGLmodels, 13
- hRight
  - BGLmodels, 13
- helicity
  - BGLmodels::Fermion, 100
- iBD2taunu
  - BGL2, 61
  - BGLmodels::BGL, 48
- iBDtaunu
  - BGL2, 61
- BGLmodels::BGL, 48
- iBtaunu
  - BGL2, 61
  - BGLmodels::BGL, 48
- iDown
  - BGLmodels, 13
- iUp
  - BGLmodels, 13
- ii
  - BGLmodels::calcubtosgamma2, 90
- inter1
  - BGL2, 61
  - BGLmodels::calcubtosgamma2, 90
- inter2
  - BGL2, 61
  - BGLmodels::calcubtosgamma2, 90
- inter3
  - BGLmodels::calcubtosgamma2, 90
- inter4
  - BGLmodels::calcubtosgamma2, 91
- isospin
  - BGLmodels::Fermion, 100
- isvalid
  - freeparameter, 102
  - parameters, 143
- Juca, 108
  - ~Juca, 110
  - A, 112
  - add, 110
  - eta, 112
  - generateparameters, 110
  - getlist, 111
  - Juca, 110
  - lambda, 112
  - Mb, 112
  - Mc, 112
  - Md, 112
  - Ms, 112
  - Mt, 112
  - Mu, 113
  - replacements, 113
  - rho, 113
- Juca.cpp, 214
  - main, 215
- Juca.h, 216
- lambda
  - Juca, 112
- larea
  - Peak, 147
- likelihood
  - Model, 132
- lim
  - limitedobs, 115
- limitedobs, 113
  - ~limitedobs, 115
  - error, 115
  - lim, 115

- limitedobs, 114
  - loglikelihood, 115
  - min, 115
  - s, 116
- llmax
  - Peak, 148
- lmax
  - Peak, 148
- lnxy\_xy
  - BGLmodels::calcuOblique, 96
- loglike
  - Model, 133
- loglikelihood
  - gauss2obs, 105
  - gaussobs, 107
  - limitedobs, 115
  - observable, 140
  - prediction, 151
- M
  - BGLmodels::Mixes, 129
- m
  - gauss2obs, 105
  - gaussobs, 108
- M2
  - widthcalc, 161
- M22
  - widthcalc, 161
- M\_GF
  - BGLmodels, 15
- M\_Md
  - BGLmodels, 15
- M\_MI
  - BGLmodels, 15
- M\_Mu
  - BGLmodels, 16
- M\_MW
  - BGLmodels, 16
- M\_MZ
  - BGLmodels, 16
- M\_cos2
  - BGLmodels, 15
- MB0
  - BGL2, 61
  - BGLmodels::BGL, 48
- MBp
  - BGL2, 61
  - BGLmodels::BGL, 48
- MBs0
  - BGL2, 61
  - BGLmodels::BGL, 48
- MCMC.h, 239
- MD0
  - BGL2, 62
  - BGLmodels::BGL, 48
- MDp
  - BGL2, 62
  - BGLmodels::BGL, 48
- MDs0
  - BGL2, 62
  - BGLmodels::BGL, 49
- MDsp
  - BGL2, 62
  - BGLmodels::BGL, 49
- MK0
  - BGL2, 62
  - BGLmodels::BGL, 49
- MKp
  - BGL2, 62
  - BGLmodels::BGL, 49
- main
  - draw.cpp, 185
  - eigen.cpp, 199
  - eigenteste.cpp, 202
  - Juca.cpp, 215
  - main.cpp, 220
  - mainbk.cpp, 233
  - script.cpp, 253
  - script2.cpp, 255
  - scriptplots.cpp, 257
  - scripttable.cpp, 261
  - update.cpp, 262
- main.cpp, 219
  - main, 220
- mainbk.cpp, 233
  - main, 233
- mass
  - BGLmodels::Boson, 70
  - BGLmodels::Meson, 125
  - BGLmodels::Mixes, 129
- massnum
  - BGLmodels::Mixes, 129
- Matrix
  - std::Matrix, 118, 119
- Matrix3c
  - BGLmodels, 12
- Matrixx
  - BGLmodels::Matrixx, 122
- max
  - discreteparameter, 98
  - freeparameter, 102
  - Peak, 148
- Mb
  - Juca, 112
- Mc
  - Juca, 112
- McH
  - BGL2, 61
  - BGLmodels::BGL, 48
- Md
  - BGL2, 61
  - BGLmodels::BGL, 48
  - Juca, 112
- Md\_
  - BGL2, 62
  - BGLmodels::calcubtosgamma2, 91
- measure, 123



- error, [124](#)
- measure, [123](#)
- operator\*, [123](#)
- operator/, [123](#)
- points, [149](#)
- value, [124](#)
- Meson
  - BGLmodels::Meson, [125](#)
- meson
  - BGLmodels::calcuBmumu, [79](#)
- mesondw
  - BGLmodels::BGL, [40](#)
- mesondwtest
  - BGLmodels::BGL, [42](#)
  - BGLmodels::calcuBmumu, [77](#)
- mesonmixing
  - BGL2, [58](#)
  - BGLmodels::BGL, [43](#)
- Mh
  - BGL2, [62](#)
  - BGLmodels::BGL, [49](#)
- MI
  - BGL2, [62](#)
  - BGLmodels::BGL, [49](#)
- min
  - discreteparameter, [98](#)
  - freeparameter, [102](#)
  - limitedobs, [115](#)
- Mixes
  - BGLmodels::Mixes, [127](#)
- mixes
  - BGL2, [62](#)
  - BGLmodels::BGL, [49](#)
  - BGLmodels::calcuBmumu, [79](#)
- mmmax
  - BGL2, [63](#)
  - BGLmodels::BGL, [49](#)
- Model, [131](#)
  - ~Model, [132](#)
  - generateparameters, [132](#)
  - getlist, [132](#)
  - likelihood, [132](#)
  - loglike, [133](#)
  - Model, [132](#)
  - r, [134](#)
  - veto, [133](#)
- model
  - Peak, [148](#)
  - Proposal, [154](#)
- model.h, [243](#)
  - \_USE\_MATH\_DEFINES, [244](#)
- Mpi0
  - BGL2, [63](#)
  - BGLmodels::BGL, [49](#)
- Mpip
  - BGL2, [63](#)
  - BGLmodels::BGL, [49](#)
- mq1
  - widthcalc, [161](#)
- mq2
  - widthcalc, [161](#)
- mq3
  - widthcalc, [161](#)
- mq4
  - widthcalc, [161](#)
- MR
  - BGL2, [63](#)
  - BGLmodels::BGL, [50](#)
- Ms
  - Juca, [112](#)
- Mt
  - Juca, [112](#)
- mt\_mb
  - BGLmodels, [16](#)
- mt\_mt
  - BGLmodels, [16](#)
- mt\_mW
  - BGLmodels, [16](#)
- Mu
  - BGL2, [63](#)
  - BGLmodels::BGL, [50](#)
  - Juca, [113](#)
- mu
  - BGL2, [63](#)
  - BGLmodels::BGL, [50](#)
- Mu\_
  - BGL2, [63](#)
  - BGLmodels::calcubtosgamma2, [91](#)
- multivector
  - std::multivector, [136](#)
  - std::multivector< T, 1 >, [138](#), [139](#)
- multivector.h, [249](#)
- MW
  - BGL2, [63](#)
  - BGLmodels::BGL, [50](#)
- MZ
  - BGL2, [63](#)
  - BGLmodels::BGL, [50](#)
- N
  - BGLmodels::Mixes, [129](#)
- N\_
  - BGLmodels::Mixes, [129](#)
- next
  - freeparameter, [102](#)
  - parameters, [143](#)
- o
  - BGLmodels::calcuBmumu, [79](#)
  - calcuba, [74](#)
  - calcuex, [94](#)
- observable, [139](#)
  - ~observable, [140](#)
  - copies, [141](#)
  - error, [140](#)
  - loglikelihood, [140](#)
  - observable, [140](#)

- obsvalue
  - BGLmodels::calcuBmumu, 77
- operator\*
  - measure, 123
  - std, 17
- operator()
  - BGLmodels::calcuBmumu, 78
  - BGLmodels::calcuOblique, 96
  - BGLmodels::calcubtosgamma2, 84
  - calcu, 71
  - calcuba, 73
  - calcuex, 93
- operator+
  - std, 17
- operator/
  - measure, 123
- p
  - parameters, 144
- parameters, 141
  - area, 142
  - dist, 142
  - gausslikelihood, 143
  - invalid, 143
  - next, 143
  - p, 144
  - setvalues, 144
  - values, 144
- particle
  - BGLmodels::Fermion, 100
- Peak, 145
  - adjuststeps, 146
  - area, 147
  - findPeak, 146
  - larea, 147
  - lmax, 148
  - lmax, 148
  - max, 148
  - model, 148
  - Peak, 146
  - pr, 148
- planck
  - BGL2, 63
  - BGLmodels::BGL, 50
- points, 148
  - error, 149
  - measure, 149
  - points, 149
  - prediction, 149
- pr
  - Peak, 148
- prediction, 149
  - ~prediction, 150
  - calculate, 151
  - loglikelihood, 151
  - points, 149
  - prediction, 150
- Proposal, 151
  - findPeaks, 152
  - floatPeak, 154
  - getNextPoint, 153
  - getProposal, 154
  - model, 154
  - Proposal, 152
  - proposal, 154
  - total, 154
  - vPeak, 155
- proposal
  - Proposal, 154
- q1
  - BGLmodels::Meson, 125
- q2
  - BGLmodels::Meson, 125
- QCD1
  - BGL2, 64
- QCD2
  - BGL2, 64
- r
  - Model, 134
- ratio
  - BGLmodels::calcubtosgamma2, 91
- replacements
  - BGL2, 64
  - BGLmodels::BGL, 50
  - BGLmodels::Mixes, 129
  - Juca, 113
- reps
  - BGLmodels::Mixes, 130
- reset
  - BGLmodels::Boson, 69
- rho
  - BGL2, 64
  - BGLmodels::BGL, 50
  - Juca, 113
- s
  - BGLmodels::Boson, 70
  - gauss2obs, 105
  - gaussobs, 108
  - limitedobs, 116
- s2
  - widthcalc, 162
- s3
  - widthcalc, 162
- sAny
  - BGLmodels, 12
- sScalar
  - BGLmodels, 12
- sVector
  - BGLmodels, 12
- script.cpp, 252
  - main, 253
  - uint, 253
- script2.cpp, 254
  - main, 255
  - uint, 255

- scriptplots.cpp, 257
  - main, 257
- scripttable.cpp, 260
  - main, 261
  - uint, 260
- setvalues
  - parameters, 144
- sn
  - std::Matrix, 120
- Sparam
  - BGL2, 64
- std, 17
  - operator\*, 17
  - operator+, 17
- std::Matrix, 116
  - conjugate, 119
  - cs, 120
  - Matrix, 118, 119
  - sn, 120
- std::multivector
  - multivector, 136
  - v, 135
- std::multivector< T, 1 >, 137
  - multivector, 138, 139
  - v, 138
- std::multivector< T, N >, 134
- step
  - freeparameter, 102
- stepsize
  - BGL2, 64
  - BGLmodels::BGL, 50
- tLepton
  - BGLmodels, 14
- tQuark
  - BGLmodels, 14
- tanb
  - BGL2, 64
  - BGLmodels::BGL, 50
  - BGLmodels::Mixes, 130
- tautomu\_tautoe
  - BGLmodels::BGL, 43
- topBranchingRatio
  - BGLmodels::BGL, 44
- total
  - Proposal, 154
- Tparam
  - BGL2, 64
- type
  - BGLmodels::Fermion, 100
- uint
  - script.cpp, 253
  - script2.cpp, 255
  - scripttable.cpp, 260
- UL
  - defs.h, 184
- update.cpp, 262
  - main, 262
- UQ
  - defs.h, 184
- V
  - BGLmodels::Mixes, 130
- v
  - std::multivector, 135
  - std::multivector< T, 1 >, 138
- VN\_
  - BGLmodels::Mixes, 130
- vPeak
  - Proposal, 155
- value
  - discreteparameter, 98
  - freeparameter, 103
  - measure, 124
- values
  - parameters, 144
- Vector3c
  - BGLmodels, 12
- veto
  - BGLmodels::BGL, 44
  - Model, 133
- VN
  - BGLmodels::Mixes, 130
- Vnl
  - BGLmodels, 16
- Vud
  - BGLmodels, 14
- wc
  - BGLmodels::BGL, 51
- width
  - BGLmodels::calcubtosgamma2, 86
- widthcalc, 155
  - genM2, 156
  - genM22, 157
  - get\_integral, 158
  - get\_integral\_meson, 159
  - get\_integral\_meson2, 160
  - get\_integral\_symb, 160
  - M2, 161
  - M22, 161
  - mq1, 161
  - mq2, 161
  - mq3, 161
  - mq4, 161
  - s2, 162
  - s3, 162
  - widthcalc, 156
- widthcalc.h, 264
  - \_USE\_MATH\_DEFINES, 265
- Y
  - BGLmodels::BGL, 45
  - BGLmodels::calcuBmumu, 78