flavour

Generated by Doxygen 1.8.11

Contents

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 $\vec{O}_{\rm Th}(\vec{p})$. Then, using the experimental information $\vec{O}_{\rm Exp}$ available for those observables, we build a likelihood function $\mathcal{L}(\vec{O}_{\rm Exp}|\vec{O}_{\rm Th}(\vec{p}))$ which gives the probability of obtaining the experimental results $\vec{O}_{\rm Exp}$ assuming that the model is correct. The likelihood function $\mathcal{L}(\vec{O}_{\rm Exp}|\vec{O}_{\rm Th}(\vec{p}))$ encodes all the information on how the model is able to reproduce the observed data all over parameter space. Nevertheless, the knowledge of $\mathcal{L}(\vec{O}_{\rm Exp}|\vec{O}_{\rm Th}(\vec{p}))$ 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:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

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->mu mu decay
- class calcubtosgamma2

  calculus of the constraints coming from the b->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 FIsospin { 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_Ml [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*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*)

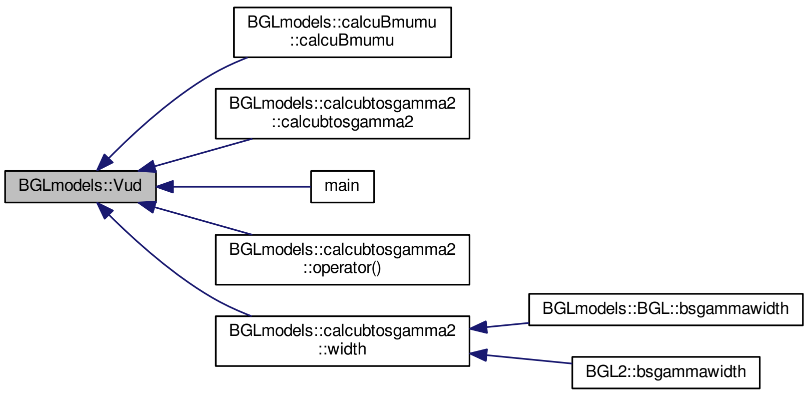
Definition at line 268 of file Formulas.h.

```
00268                        {
00269      return (-3/(x-1)*x*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:


BGLmodels::calcuBmumu ::calcuBmumu
BGLmodels::calcubtosgamma2 ::calcubtosgamma2
BGLmodels::Vud
main
BGLmodels::calcubtosgamma2 ::operator()
BGLmodels::BGL::bsgammawidth
BGLmodels::calcubtosgamma2 ::width
BGL2::bsgammawidth

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_Ml[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.

```
                                                        {
        Matrix res;
        for(uint i=0;i<3;i++)
                for(uint j=0;j<3;j++)
                        for(uint k=0;k<3;k++)
                                res[i][j]=res[i][j]+m1[i][k]*m2[k][j];
        return res;
}
```

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.

```
                                                        {
        Matrix res;
        for(uint i=0;i<3;i++)
                for(uint j=0;j<3;j++)
                                res[i][j]=m1[i][j]+m2[i][j];
        return res;
}
```

Chapter 7

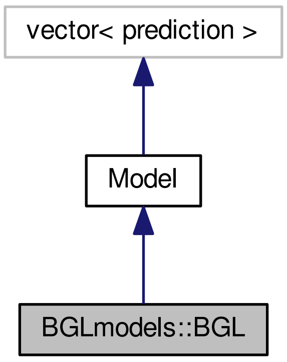
Class Documentation

7.1 BGLmodels::BGL Class Reference

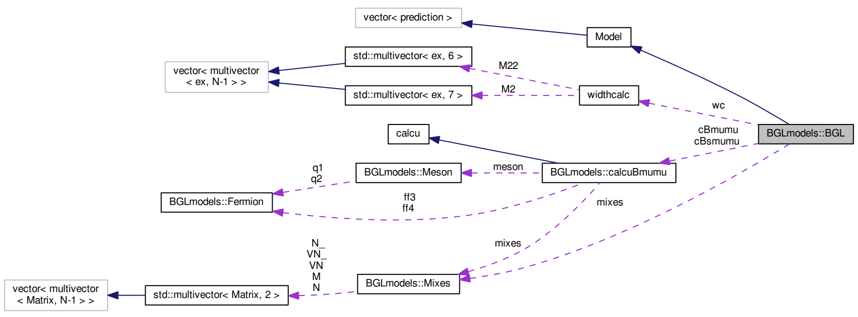
Implementation of the BGL model.

`#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=s←↩ Any) 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 mmmax
- 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.

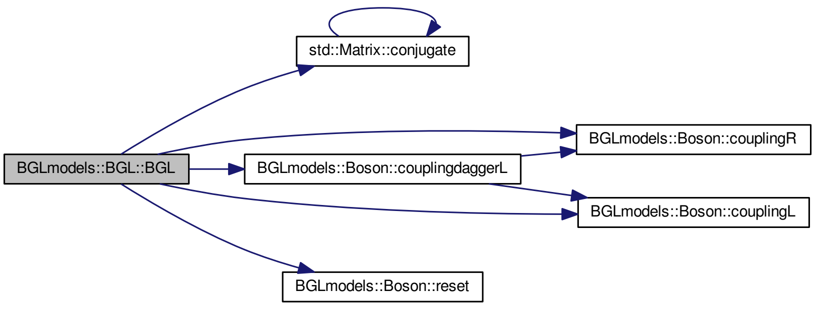
References BGLmodels::Boson::C, std::Matrix::conjugate(), BGLmodels::Boson::couplingdaggerL(), BGLmodels↩::Boson::couplingL(), BGLmodels::Boson::couplingR(), BGLmodels::cParticle, BGLmodels::fElectron, BG↩Lmodels::fMuon, BGLmodels::fTau, BGLmodels::hLeft, BGLmodels::hRight, BGLmodels::iDown, BGLmodels::iUp, 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            mmmax(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;
```

```

        boson.coupsL[0][0]=Matrix(g*sqrt(1-cos2)*0);
        boson.coupsL[1][1]=Matrix(g*sqrt(1-cos2)*(-1));
        boson.coupsL[2][2]=Matrix(g*sqrt(1-cos2)*ex(2)/3);
        boson.coupsL[3][3]=Matrix(g*sqrt(1-cos2)*ex(-1)/3);

        boson.coupsR[0][0]=Matrix(g*sqrt(1-cos2)*0);
        boson.coupsR[1][1]=Matrix(g*sqrt(1-cos2)*(-1));
        boson.coupsR[2][2]=Matrix(g*sqrt(1-cos2)*ex(2)/3);
        boson.coupsR[3][3]=Matrix(g*sqrt(1-cos2)*ex(-1)/3);

        bosons.push_back(boson);
        boson.reset();
        */
        //W+ boson
        boson.mass=MW;
        boson.s=sVector;

        for(uint t=tLepton;t<=tQuark;t++) boson.C[t][iUp][iDown][hLeft]=mixes.V[t]*Matrix(g/sqrt(ex(2)));
        Boson wboson=boson;
        bosons.push_back(boson);
        boson.reset();

        //H+ boson
        boson.mass=McH;
        boson.s=sScalar;

        for(uint t=tLepton;t<=tQuark;t++)
        for(uint i=iUp;i<=iDown;i++) boson.C[t][iUp][iDown][i]=mixes.VN[t][i]*Matrix(g/MW/sqrt(ex(2)));
        Boson chiggs=boson;
        bosons.push_back(boson);
        boson.reset();

        for(int b=bosons.size()-1;b>=0;b--){
                boson.mass=bosons[b].mass;
                boson.s=bosons[b].s;
                if(boson.s==sVector)
                        for(uint t=tLepton;t<=tQuark;t++)
                        for(uint i=iUp;i<=iDown;i++)
                        for(uint j=iUp;j<=iDown;j++)
                        for(uint h=hLeft;h<=hRight;h++){
                                boson.C[t][i][j][h]=bosons[b].C[t][j][i][h].conjugate();
                                }
                else for(uint t=tLepton;t<=tQuark;t++)
                        for(uint i=iUp;i<=iDown;i++)
                        for(uint j=iUp;j<=iDown;j++)
                        for(uint h=hLeft;h<=hRight;h++){
                                boson.C[t][i][j][hLeft]=bosons[b].C[t][j][i][hRight].conjugate();
                                boson.C[t][i][j][hRight]=bosons[b].C[t][j][i][hLeft].conjugate();
                                }
                bosons.push_back(boson);
                boson.reset();
                }

        //(R+iI)/sqrt(2) boson
        boson.mass=MR;
        boson.s=sScalar;

        for(uint t=tLepton;t<=tQuark;t++){
                        boson.C[t][iDown][iDown][hRight]=mixes.N[t][iDown]*Matrix(g/MW/ex(2));
                        boson.C[t][iUp][iUp][hLeft]=mixes.N[t][iUp].conjugate()*Matrix(g/MW/ex(2));
                        boson.C[t][iDown][iDown][hLeft]=mixes.N[t][iDown].conjugate()*Matrix(g/MW/ex(2));
                        boson.C[t][iUp][iUp][hRight]=mixes.N[t][iUp]*Matrix(g/MW/ex(2));
                }
        bosons.push_back(boson);
        boson.reset();

        //(R+iI)/sqrt(2) boson
        boson.mass=MI;
        boson.s=sScalar;

        for(uint t=tLepton;t<=tQuark;t++){
                        boson.C[t][iDown][iDown][hRight]=mixes.N[t][iDown]*Matrix(I*g/MW/ex(2));
                        boson.C[t][iUp][iUp][hLeft]=mixes.N[t][iUp].conjugate()*Matrix(I*g/MW/ex(2));
                        boson.C[t][iDown][iDown][hLeft]=mixes.N[t][iDown].conjugate()*Matrix(-I*g/MW/ex(2));
                        boson.C[t][iUp][iUp][hRight]=mixes.N[t][
iUp]*Matrix(-I*g/MW/ex(2));
                }
        bosons.push_back(boson);
        boson.reset();

        Fermion electron(tLepton,iDown,fElectron);
        Fermion electronR(tLepton,iDown,fElectron,cParticle,
hRight);

        Fermion muon(tLepton,iDown,fMuon);
        Fermion muonR(tLepton,iDown,fMuon,cParticle,
hRight);

        Fermion tau(tLepton,iDown,fTau);
        Fermion tauR(tLepton,iDown,fTau,cParticle,
hRight);
        Fermion neutrino(tLepton,iUp);
        Fermion neutrinotau(tLepton,iUp,fTau);
        Fermion neutrinomuon(tLepton,iUp,fMuon);
        Fermion neutrinoe(tLepton,iUp,fElectron);

        Fermion up(tQuark,iUp,fElectron);
        Fermion down(tQuark,iDown,fElectron);
        Fermion bottom(tQuark,iDown,fTau);
        Fermion strange(tQuark,iDown,fMuon);
        Fermion charm(tQuark,iUp,fMuon);
        Fermion top(tQuark,iUp,fTau);

        Meson Pi0d(down,down,Mpi0,Fpi);
        Meson Pi0u(down,down,Mpi0,Fpi);
        Meson Pip(up,down,Mpip,Fpi);
        Meson Pim(down,up,Mpip,Fpi);

        Meson K0(down,strange,MK0,FK);
        Meson Kp(up,strange,MKp,FK);

        Meson D0(charm,up,MD0,FD);
        Meson Dp(charm,down,MDp,FD);
        Meson Dsp(charm,strange,MDsp,FDs);

        Meson B0(down,bottom,MB0,FB);
        Meson Bp(up,bottom,MBp,FB);
        Meson Bs0(strange,bottom,MBs0,FBs);

        lst sb;
        //sb.append(mixes.M[tQuark][iUp][0][0]==0);
        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));
        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));

        //cout<<"Btaunu "<<collect_common_factors(expand(Btaunu.subs(sb).subs(conjtoabs)))<<endl;

        cout<<latex;

        ex mutoenunu=decaywidth(muon,neutrino,electron,neutrino);

        //cout<<"mutoenunu "<<mutoenunu<<endl;
        //add("mutoenunu",decaywidth(muon,neutrino,electron,neutrino),new
 gaussobs(planck/2.197034e-6,0.03));

        add("muRtoeRnunu",gRR2(muon,electron),new limitedobs(std::pow(0.035,2),0.95));

        //add("tautoenunu",decaywidth(tau,neutrino,electron,neutrino),new
 gaussobs(planck/290.6e-15*0.1782,0.03));
        add("tauRtoeRnunu",gRR2(tau,electron),new limitedobs(std::pow(0.7,2),0.95));

        //add("tautomununu",decaywidth(tau,neutrino,muon,neutrino),new
 gaussobs(planck/290.6e-15*0.1739,0.03));
        add("tauRtomuRnunu",gRR2(tau,muon),new limitedobs(std::pow(0.72,2),0.95));

        add("tautomu_tautoe",tautomu_tautoe(),new gaussobs(1.0018,0.0014/1.0018));
 //PROBLEM!!!
        cout<<"tautomu_tautoe: "<<1/1.0018<< "ERROR: "<<0.0014/1.0018<<endl;
        cout<<"ratio1 "<<tautomu_tautoe().subs(replacements)<<endl;
        cout<<"ratio2 "<<(decaywidth(tau,neutrino,muon,neutrino,
sVector)/decaywidth(tau,neutrino,electron,neutrino,sVector)).subs(
replacements)<<endl;
        //muto3e
        ex mu3e=decaywidth(muon,electron,electron,electron);
        add("muto3e", mu3e,new limitedobs(planck/2.197034e-6*1e-12));
        cout<<"mu3e "<<decaywidthtest2(muon)<<endl;

        //tauto3e
        add("tauto3e", decaywidth(tau,electron,electron,electron),new
limitedobs(planck/290.6e-15*2.7e-8));
        //tauto2e1mu+
        add("tauto2e1mup", decaywidth(tau,electron,electron,muon), new limitedobs(planck/290.6e-15*1.5e-8));
        //tauto2e1mu-
        add("tauto2e1mu", decaywidth(tau,electron,muon,electron), new limitedobs(planck/290.6e-15*1.8e-8));
        //tauto2mu1e+

        add("tauto2mu1ep", decaywidth(tau,muon,muon,electron), new limitedobs(planck/290.6e-15*1.7e-8));
        cout<<"tauto2mu1ep "<<decaywidthtest2(tau)<<endl;
        //tauto2mu1e-
        add("tauto2mu1ep", decaywidth(tau,muon,electron,muon), new limitedobs(planck/290.6e-15*2.7e-8));
        cout<<"tauto2mu1e "<<decaywidthtest2(tau)<<endl;

        //tauto3mu
        add("tauto3mu", decaywidth(tau,muon,muon,muon),new limitedobs(planck/290.6e-15*2.1e-8));
        //cout<<"tauto3mu "<<collect_common_factors(expand(decaywidth(tau,muon,muon,muon)))<<endl;


        ex piratio=1.2352e-4/(mesondw(Pip,neutrino,electron,sVector)/mesondw(Pip,neutrino,muon,sVector));
        ex picorrection=piratio.subs(replacements);
        ex pierror=picorrection*0.0001/1.2352;
        cout<<"PiRatio "<<picorrection-1<<" +/- "<<pierror<<endl;
        piratio*=mesondw(Pip,neutrino,electron)/mesondw(Pip,neutrino,muon);
        add("piontoenu_munu",piratio,new gaussobs(1.230e-4,0.003)); //PROBLEM!!!
        cout<<"piontoenu_munu: "<<1.2352e-4/1.230e-4<<" ERROR: "<<0.003<<endl;


        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));
        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;
        cout<<"tautopinu: "<<fermiontomesontest(tau,neutrino,Pip)<<endl;
        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;

        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));
    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;
        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;

    //ex pi0toemu=(mesondecaywidth(Mpi0,down,down,electron,muon)+mesondecaywidth(Mpi0,up,up,electron,muon)+mesondecaywidth(Mpi0,
    ex pi0toemu=(mesondw(Pi0d,electron,muon)+mesondw(Pi0d,muon,electron)+mesondw(Pi0u,electron,muon)+mesondw(Pi0u,muon,electron))/2;
    add("pi0toemu",pi0toemu,new limitedobs(3.6e-10*planck/8.52e-17));

        ex Kratio=2.477e-5/(mesondw(Kp,neutrino,electron,sVector)/mesondw(Kp,neutrino,muon,sVector));
        ex Kcorrection=Kratio.subs(replacements);
        ex Kerror=Kcorrection*0.001/2.477;
        cout<<"KRatio "<<Kcorrection-1<<" +/- "<<Kerror<<endl;
        Kratio*=mesondw(Kp,neutrino,electron)/mesondw(Kp,neutrino,muon);
        add("Ktoenu_munu",Kratio,new gaussobs(2.488e-5,0.005));
        cout<<"Ktoenu_munu: "<<2.477e-5/2.488e-5<<" ERROR: "<<0.005<<endl;

    ex k0Ltoemu=mesondw(K0,electron,muon)+mesondw(K0,muon,electron);
        add("K0Ltoemu",k0Ltoemu,new limitedobs((4.7e-12*planck/5.116e-8)));
    add("K0Ltoee",mesondw(K0,electron,electron),new limitedobs((9e-12*planck/5.116e-8)));

    //add("K0Ltomumu",mesondw(K0,muon,muon),new limitedobs((6.84e-9*planck/5.116e-8)));

        add("Dtoenu",mesondw(Dp,neutrino,electron),new limitedobs(8.8e-6*planck/1040e-15));
        add("Dtomunu",mesondw(Dp,neutrino,muon),new gaussobs(3.82e-4*planck/1040e-15,0.1)); //PROBLEM!!!
        cout<<"Dtomunu: "<<mesondw(Dp,neutrino,muon,sVector).subs(replacements)/(3.82e-4*planck/1040e-15)<<" ERROR: "<<0.1<<endl;

        add("Dtotaunu",mesondw(Dp,neutrino,tau),new limitedobs(1.2e-3*planck/1040e-15));  //PROBLEM!!!
        cout<<"Dtotaunu: "<<mesondw(Dp,neutrino,tau,sVector).subs(replacements)/(1.2e-3*planck/1040e-15)<<" LIMIT"<<endl;

        //D0 2.6e-7/410.1e-15

        ex D0toemu=mesondw(D0,electron,muon)+mesondw(D0,muon,electron);
        add("D0toemu",D0toemu,new limitedobs((2.6e-7*planck/410.1e-15)));
    ex D0toee=mesondw(D0,electron,electron);
        add("D0toee",D0toee,new limitedobs((7.9e-8*planck/410.1e-15)));
    ex D0tomumu=mesondw(D0,muon,muon);
        add("D0tomumu",D0tomumu,new limitedobs((1.4e-7*planck/410.1e-15)));

    //ex Dstomunu=mesondecaywidth(MDsp,strange,charm,muon,neutrino); //500e-15
        add("Dstomunu",mesondw(Dsp,neutrino,muon),new gaussobs(5.9e-3*planck/500e-15,0.33/5.9)); //PROBLEM!!!
        cout<<"Dstomunu: "<<mesondw(Dsp,neutrino,muon,sVector).subs(replacements)/(5.9e-3*planck/500e-15)<<" ERROR: "<<0.33/5.9<<endl;

        add("Dstoenu",mesondw(Dsp,neutrino,electron),new limitedobs(1.2e-4*planck/500e-15));
        add("Dstotaunu",mesondw(Dsp,neutrino,tau),new gaussobs(5.43e-2*planck/500e-15,0.31/5.43)); //PROBLEM!!!
        cout<<"Dstotaunu: "<<mesondw(Dsp,neutrino,tau,sVector).subs(replacements)/(5.43e-2*planck/500e-15)<<" ERROR: "<<0.31/5.43<<endl;

        add("Btomunu",mesondw(Bp,neutrino,muon),new limitedobs(9.8e-7*planck/1.641e-12));
        add("Btoenu",mesondw(Bp,neutrino,electron),new limitedobs(1e-6*planck/1.641e-12));

        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));

        //calcuBmumu calcutest(mixes,Bs0,muon,muon,2.9e-9*planck/1.516e-12,0.7e-9*planck/1.516e-12,"Bs_to_mumu");
        //calcuBmumu calcutest2(mixes,B0,muon,muon,3.6e-10*planck/1.519e-12,1.6e-10*planck/1.516e-19,"B_to_mumu");
        //cout<<"TESTE "<<endl;
        //double ps[4]={1,1e16,1e16,1e16};
        //double resteste=0,resteste2=0;
        //int nt=4,mt=1;
        //calcutest.fp(&nt,ps,&mt,&resteste);
        //calcutest2.fp(&nt,ps,&mt,&resteste2);
        //cout<<"TESTE "<<resteste/(2.9e-9*planck/1.516e-12)<<" "<<resteste2/(3.6e-10*planck/1.519e-12)<<endl;
        //ex B0tomumu=mesondw(B0,muon,muon);
        //cout<<"B0tomumu "<<collect_common_factors(B0tomumu)<<endl;
        //1.65e-4
        //add("B0tomumu",B0tomumu,new limitedobs((8e-10*planck/1.519e-12)));

        push_back(prediction(new calcuBmumu(mixes,B0,muon,muon,new limitedobs(6.3e-10*planck/1.519e-12),"B_to_mumu")));
        //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")));
        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")));
    push_back(prediction(new calcuBmumu(mixes,K0,muon,muon,new limitedobs(2.3e-9*planck/5.116e-8),"K0L_to_mumu")));

    cBmumu=new calcuBmumu(mixes,B0,muon,muon,new limitedobs(6.3e-10*planck/1.519e-12),"B_to_mumu");
    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");

    ex B0toetau=mesondw(B0,electron,tau)+mesondw(B0,tau,electron);
        add("B0toetau",B0toetau,new limitedobs((2.8e-5*planck/1.519e-12)));
        ex B0tomutau=mesondw(B0,muon,tau)+mesondw(B0,tau,muon);
        add("B0tomutau",B0tomutau,new limitedobs((2.2e-5*planck/1.519e-12)));
        ex B0toee=mesondw(B0,electron,electron);
        add("B0toee",B0toee,new limitedobs((8.3e-8*planck/1.519e-12)));
    ex B0totautau=mesondw(B0,tau,tau);
        add("B0totautau",B0totautau,new limitedobs((4.1e-3*planck/1.519e-12)));

        //B0s m=5.3663, life=1.472e-12 emu=2e-7, ee=2.8e-7 mumu=4.2e-8
        ex Bs0toemu=mesondw(Bs0,electron,muon)+mesondw(Bs0,muon,electron);
        add("Bs0toemu",Bs0toemu,new limitedobs((2e-7*planck/1.516e-12)));
    ex Bs0toee=mesondw(Bs0,electron,electron);
        add("Bs0toee",Bs0toee,new limitedobs((2.8e-7*planck/1.516e-12)));
//    ex Bs0tomumu=mesondw(Bs0,muon,muon);
//      add("Bs0tomumu",Bs0tomumu,new limitedobs((3.2e-9*planck/1.516e-12)));

   // add("chargedHiggs",pow(McH,-2),new limitedobs(std::pow(80.0,-2),0.9));

    cout<<"Bs0tomumu: "<<mesondwtest(Bs0,muon,muon)<<endl;
    //add("chargedHiggs",1/McH,new limitedobs(1/80.0,0));

    /*
    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];
    for(uint i=0;i<3;i++)
                for(uint j=0;j<3;j++)
                        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);
                        else llgamma2loop[i][j]=0;
        add("mutoegamma",llgamma2loop[1][0],new limitedobs(1.2e-11));
        add("tautoegamma",llgamma2loop[2][0],new limitedobs(3.3e-8));
        add("tautomugamma",llgamma2loop[2][1],new limitedobs(4.4e-8));
        //add("mutoegamma",llgamma2loop[1][0],new limitedobs(planck/2.197034e-6*1.2e-11));
        //add("tautoegamma",llgamma2loop[2][0],new limitedobs(planck/290.6e-15*3.3e-8));
        //add("tautomugamma",llgamma2loop[2][1],new limitedobs(planck/290.6e-15*4.4e-8));
    */

    Matrix llgammaCH;
        //Matrix llgammaCH=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH*12))*mixes.M[tLepton][iDown]*mixes.VN[tLepton][1].conjugate()*mixes.VN[t
        Matrix llgammaH0M,llgammaH0E;
        /*for(uint i=0;i<3;i++)
                for(uint j=0;j<3;j++)
                        for(uint k=0;k<3;k++){
                                ex z=pow(fmasses[1][i][i]/McH,2); mixes.M[tQuark][iUp][i][i]/MR,2);
                                llgammaH0M[j][k]=llgammaH0M[j][k]+(mixes.VN[0][1][j][i].conjugate()*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*z+6*z*z*log(z))/6;
                                llgammaH0M[j][k]=llgammaH0M[j][k]+(mixes.VN[0][1][i][j]*mixes.VN[0][1][k][i])/mixes.M[tLepton][iDown][i][i]/mixes.M[tLepton][iDown][j][j]*(3*z+2*z*log(z));

                                llgammaH0E[j][k]=llgammaH0E[j][k]+(mixes.VN[0][1][j][i].conjugate()*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*z+6*z*z*log(z))/6;
                                llgammaH0E[j][k]=llgammaH0E[j][k]+(mixes.VN[0][1][i][j]*mixes.VN[0][1][k][i])/mixes.M[tLepton][iDown][i][i]/mixes.M[tLepton][iDown][j][j]*(3*z+2*z*log(z));
                                }
                                */
        llgammaH0M=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH))*mixes.M[tLepton][iDown]*llgammaH0M;
        llgammaH0E=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH))*mixes.M[tLepton][iDown]*llgammaH0E;

        Matrix llgamma, llgamma2;

        for(uint i=0;i<3;i++)
                for(uint j=0;j<3;j++){
                //if(j<i) llgamma[i][j]=(llgammaCH[i][j]*llgammaCH[i][j].conjugate()+llgammaH0E[i][j]*llgammaH0E[i][j].conjugate()+llgammaH0M[i][j]*llgammaH0M[i][j].conjugate())*g*g*(1-cos2)*pow((pow(mixes.M[tLepton][iDown][i][
            ex mmuon=mixes.M[tLepton][iDown][i][i];
                        ex A,B;

            if(j<i){ for(uint k=0;k<3;k++){
                          ex mtau=mixes.M[tLepton][iDown][k][k];
                          B+=-mixes.VN[tLepton][1][k][j].conjugate()*mixes.VN[tLepton][1][k][i]/(12*pow(McH,2));
                          B+=mixes.N[tLepton][1][k][j].conjugate()*mixes.N[tLepton][1][k][i]/12*(pow(MR,-2)+pow(MI,-2));

                          A+=mixes.N[tLepton][1][j][k]*mixes.N[tLepton][1][i][k].conjugate()*(pow(MR,-2)+pow(MI,-2))/12;
                          A+=mixes.N[tLepton][1][j][k]*mixes.N[tLepton][1][k][i]/mtau/mmuon*(Fh2(pow(mtau/MR,2))-Fh2(pow(mtau/MI,2)))/4;
                        }
                         llgamma[i][j]=(A*A.conjugate()+B*B.conjugate())*alpha*pow(mmuon,5)*GF*GF/(128*pow(Pi,4));
                         }
                else if(j==i){
                        for(uint k=0;k<3;k++){
                          ex mtau=mixes.M[tLepton][iDown][k][k];
                          B+=-mixes.VN[tLepton][1][k][j].conjugate()*mixes.VN[tLepton][1][k][i]/(12*pow(McH,2));
                          B+=mixes.N[tLepton][1][k][j].conjugate()*mixes.N[tLepton][1][k][i]/12*(pow(MR,-2)+pow(MI,-2));
                          B+=mixes.N[tLepton][1][j][k].conjugate()*mixes.N[tLepton][1][i][k]/12*(pow(MR,-2)+pow(MI,-2));
                          }
                    llgamma[i][j]=-B*GF*sqrt(1/2)/(8*pow(Pi,2))*2*mmuon; //e (GeV)^-1=1/(51e6)(e cm) where e=sqrt(alpha*4*Pi)
                  }
                }
        add("mutoegamma",llgamma[1][0],new limitedobs(planck/2.197034e-6*2.4e-12));
        add("tautoegamma",llgamma[2][0],new limitedobs(planck/290.6e-15*3.3e-8));
        add("tautomugamma",llgamma[2][1],new limitedobs(planck/290.6e-15*4.4e-8));

        /*add("d_e",abs(llgamma[0][0].imag_part()),new limitedobs(10.5e-28*51e6));
        add("d_mu",abs(llgamma[1][1].imag_part()),new limitedobs(1.9e-19*51e6));
        add("d_tau",llgamma[2][2].imag_part(),new gaussobs(-0.85e-17*51e6,0.825/0.85));
        cout<<"EDM: "<<llgamma[0][0].subs(conjtoabs).subs(replacements).imag_part()<<endl;
        add("a_mu",-llgamma[1][1].real_part()*2*mixes.M[tLepton][iDown][1][1],new gaussobs(3e-9,1.0/3.0));
        */
        /*llgammaCH=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH*12))*mixes.M[tQuark][iDown]*mixes.VN[1][1].conjugate()*mixes.VN[1][1]; //4+1
        //Matrix llgammaH0M,llgammaH0E;
        for(uint i=0;i<3;i++)
                for(uint j=0;j<3;j++)
                        for(uint k=0;k<3;k++){
                                ex z=pow(mixes.M[tQuark][iUp][i][i]/MR,2);
                                llgammaH0M[j][k]=llgammaH0M[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;
                                llgammaH0M[j][k]=llgammaH0M[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));

                                llgammaH0E[j][k]=llgammaH0E[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;
                                llgammaH0E[j][k]=llgammaH0E[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));
                                }
        llgammaH0M=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH))*mixes.M[tQuark][iDown]*llgammaH0M;
        llgammaH0E=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH))*mixes.M[tQuark][iDown]*llgammaH0E;

        //Matrix llgamma;
        for(uint i=0;i<3;i++)
                for(uint j=0;j<3;j++)
                if(j<i) {llgamma[i][j]=(llgammaCH[i][j]*llgammaCH[i][j].conjugate()+llgammaH0E[i][j]*llgammaH0E[i][j].conjugate()+llgammaH0M[i][j]*llgammaH0M[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);
                                //llgamma[i][j]=llgamma[i][j].subs(lst(abs(wild()*pow(MH0,-2))==abs(wild()*pow(MH0,-2)));
                                }
            else llgamma[i][j]=0;
         */


        push_back(prediction(new calcubtosgamma2(mixes)));

        //add("btosgamma",llgamma[2][1],new gaussobs(3.55e-4,sqrt(2)*0.25/3.55),1);
            //cout<<csrc<<llgamma[2][1]<<endl;
            //cout<<latex;


        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));
        BR_Htotaunu=BR_Htotaunu.subs(replacements);

        //BR_toptoHq=decaywidth(top,bottom,chiggs);
        //ex toptoWb=decaywidth(top,bottom,wboson);
        //BR_toptoHq=BR_toptoHq/(BR_toptoHq+toptoWb);
        //BR_toptoHq=BR_toptoHq.subs(replacements);

        //cout<<"toptoWb "<<toptoWb.subs(replacements).evalf()<<endl;

        //b to c tau- nu/b to c e- nu
        //ex btocR=decaywidth(bottom,charm,tau,neutrino,sVector)/(decaywidth(bottom,charm,electron,neutrino,sVector)+decaywidth(bot
        //cout<<btocR.subs(replacements)<<endl;

        ex BtoDtaunu,BtoD2taunu, BtoDtaunuSM, KtoPi;
        for(uint i=0; i<3; i++){
                ex Wcoup=wboson.couplingL(charm,bottom)*wboson.couplingdaggerL(tau,Fermion(tLepton,iUp,FFlavour(i)));
                if(Wcoup.subs(replacements)==ex(0)) continue;
                ex chcoup_Wcoup=-pow(MW/McH,2)*(chiggs.couplingR(charm,bottom)+chiggs.couplingL(charm,bottom))*chiggs.couplingdaggerL(tau,Fermion(tLepton,iUp,FFlavour(i)))/Wcoup;
                ex chcoup2_Wcoup=-pow(MW/McH,2)*(chiggs.couplingR(charm,bottom)-chiggs.couplingL(charm,bottom))*chiggs.couplingdaggerL(tau,Fermion(tLepton,iUp,FFlavour(i)))/Wcoup;

                BtoDtaunuSM+=Wcoup*Wcoup.conjugate();
                BtoDtaunu+=Wcoup*Wcoup.conjugate()*(1+1.5*chcoup_Wcoup.real_part()+chcoup_Wcoup.conjugate()*chcoup_Wcoup);
                BtoD2taunu+=Wcoup*Wcoup.conjugate()*(1+0.12*chcoup2_Wcoup.real_part()+0.05*chcoup2_Wcoup.conjugate()*chcoup2_Wcoup);
        }
        lst 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));
        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));

        r2.append(mixes.M[1][0][1][1]==0);
        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));
        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));
        r2.append(abs(sqrt(ex(2))* GF)==sqrt(ex(2))* GF);

        BtoDtaunuSM=collect_common_factors(BtoDtaunuSM.subs(conjtoabs).subs(r2));
        BtoDtaunu=collect_common_factors(BtoDtaunu.subs(conjtoabs).subs(r2));

        BtoDtaunuR=(BtoDtaunu/BtoDtaunuSM).subs(replacements).real_part();

        BtoD2taunu=BtoD2taunu.subs(conjtoabs).subs(r2);
        BtoD2taunuR=(BtoD2taunu/BtoDtaunuSM).subs(replacements).real_part();


        //cout<<"BtoDtaunu/BtoDtaunuSM "<<expand(BtoDtaunu/BtoDtaunuSM)<<endl;
        iBDtaunu=size();
        add("BtoDtaunu_BtoDtaunuSM",BtoDtaunu/BtoDtaunuSM,new gaussobs(440.0/296, 1.4*58.0/440));

        iBD2taunu=size();
        //cout<<"BtoD2taunu/BtoD2taunuSM "<<1+collect_common_factors(expand(BtoD2taunu/BtoDtaunuSM-1))<<endl;
        add("BtoD2taunu_BtoD2taunuSM",BtoD2taunu/BtoDtaunuSM,new gaussobs(332.0/252, 1.4*24.0/332.0));



        for(uint j=0; j<2; j++){
                ex KtoPimunu, KtoPimunuSM;
        for(uint i=0; i<3; i++){
                ex Wcoup=wboson.couplingL(up,strange)*wboson.couplingdaggerL(Fermion(tLepton,iDown,FFlavour(j)),Fermion(tLepton,iUp,FFlavour(i)));
                if(Wcoup.subs(replacements)==ex(0)) continue;
                ex chcoup_Wcoup=-pow(MW/McH,2)*(chiggs.couplingR(up,strange)+chiggs.couplingL(up,strange))\
                                *chiggs.couplingdaggerL(muon,Fermion(tLepton,iUp,FFlavour(i)))/Wcoup*(pow(MKp,2)-pow(Mpip,2))\
                                /(mixes.mass(Fermion(tLepton,iDown,FFlavour(j)))*(mixes.mass(strange)-mixes.mass(up)));
                chcoup_Wcoup=collect_common_factors(expand(chcoup_Wcoup));
                KtoPimunuSM+=collect_common_factors(expand(Wcoup*Wcoup.conjugate()));
                KtoPimunu+=collect_common_factors(expand(Wcoup*Wcoup.conjugate()*pow(1+chcoup_Wcoup,2)));
        }
            KtoPimunuSM=collect_common_factors(expand(KtoPimunuSM.subs(conjtoabs).subs(r2)));
            KtoPimunu=collect_common_factors(expand(KtoPimunu.subs(conjtoabs).subs(r2)));
            KtoPimunu=expand(KtoPimunu.subs(replacements).real_part().subs(lst(abs(wild()*pow(MR,-2))==abs(wild())*pow(MR,-2))).subs(lst(log(wild()*pow(MR,-2))==log(wild())-2*log(MR))));
            KtoPimunu=expand(KtoPimunu.evalf());
            KtoPimunuSM=expand(KtoPimunuSM.subs(replacements).real_part().subs(lst(abs(wild()*pow(MR,-2))==abs(wild())*pow(MR,-2))).subs(lst(log(wild()*pow(MR,-2))==log(wild())-2*log(MR))));
            KtoPimunuSM=expand(KtoPimunuSM.evalf());
                KtoPi+=0.5*log(KtoPimunu/KtoPimunuSM);
        }

        add("KtoPi",KtoPi/(pow(MKp,2)-pow(Mpip,2)),new gaussobs(0.08, 0.11/0.08));


        //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));

    double fD=0.207;
    ex DDbar=ex(std::pow(fD,2))*mesonmixing(MD0,charm,up);
    DDbar=expand(DDbar.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))));
        DDbar=expand(DDbar.evalf());
    ex aDDbar=sqrt(DDbar.real_part()*DDbar.real_part()+DDbar.imag_part()*DDbar.imag_part());
    add("DDbar",aDDbar,new limitedobs(9.47e-15));
    cout<<DDbar<<endl;
//2|M12|<6.6e-15GeV

    double fK=0.156;
    ex KKbar=ex(std::pow(fK,2))*mesonmixing(MK0,strange,down);
    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))));
        KKbar=expand(KKbar.evalf());
        ex aKKbar=sqrt(KKbar.real_part()*KKbar.real_part()+KKbar.imag_part()*KKbar.imag_part());
    add("KKbar",aKKbar,new limitedobs(3.5e-15));
    ex eK=0.94*imag_part(KKbar)/3.5e-15/sqrt(2);
    //add("a_eK",abs(eK),new limitedobs(2.2e-3));
    add("a_eK",abs(eK),new limitedobs(20*0.0114e-3));
    cout<<abs(KKbar)<<endl;

    double fB=0.189;
    ex Vtb=mixes.V[tQuark][2][2]/mixes.V[tQuark][2][2].conjugate();
    ex Vtd=mixes.V[tQuark][2][0]/mixes.V[tQuark][2][0].conjugate();
    ex Vts=mixes.V[tQuark][2][1]/mixes.V[tQuark][2][1].conjugate();

    ex BBbar=1+ex(std::pow(fB,2))*mesonmixing(MB0,bottom,down)/(3.337e-13*Vtb*Vtd.conjugate());
    add("BBbarimag",imag_part(BBbar),new gauss2obs(-0.199,0.062));
```

```
        add("BBbarreal",real_part(BBbar),new gauss2obs(0.823,0.143));
    cout<<BBbar<<endl;
    BBbar=3.337e-13*Vtb*Vtd.conjugate();
    cout<<"Bbar "<<(abs(imag_part(BBbar))/abs(BBbar)).subs(replacements)<<endl;
        double fBs=0.225;
    ex BsBsbar=1+ex(std::pow(fBs,2))*mesonmixing(MBs0,bottom,strange)/(1.186e-11*Vtb*Vts.
conjugate());
    add("BsBsbarimag",imag_part(BsBsbar),new gauss2obs(0,0.1));
        add("BsBsbarreal",real_part(BsBsbar),new gauss2obs(0.965,0.133));
    cout<<BsBsbar<<endl;
    BsBsbar=1.186e-11*Vtb*Vts.conjugate();
    cout<<"Bbar "<<(abs(imag_part(BsBsbar))/abs(BsBsbar)).subs(replacements)<<endl;

    ex McH2=McH*McH;
    ex MR2=MR*MR;
    ex MI2=MI*MI;

    ex cu=collect_common_factors(expand(chiggs.couplingL(top,bottom)))/mixes.
mass(top)/(g/MW/sqrt(ex(2)))/mixes.V[1][2][2];
    cout<<"cu "<<cu<<endl;
    ex Zbb=(cu-0.72)/McH;
    add("Zbb",Zbb,new limitedobs(0.0024));
    cout<<"Zbb "<<Zbb<<endl;
    cout<<"SIZE "<<size()<<endl;

   push_back(prediction(new calcuOblique()));
}
```

Here is the call graph for this function:


std::Matrix::conjugate
BGLmodels::BGL::BGL
BGLmodels::Boson::couplingR
BGLmodels::Boson::couplingdaggerL
BGLmodels::Boson::couplingL
BGLmodels::Boson::reset

7.1.2.2 BGLmodels::BGL::∼BGL ( ) [inline]

Definition at line 655 of file BGL.h.

```
    {
      delete cBmumu;
      delete cBsmumu;
      }
```

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.

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

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

Definition at line 704 of file BGL.h.

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

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

Definition at line 708 of file BGL.h.

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

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

Definition at line 712 of file BGL.h.

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

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.

```
                                                                {
        //cout<<s<<endl;
        //cout<<"prediction symb"<<pred<<endl;
        //,pow(sin(wild()), 2) == 1-pow(cos(wild()), 2)
        //ex
p=expand(pred.subs(replacements).real_part().subs(lst(abs(wild()*pow(MR,-2))==abs(wild())*pow(MR,-2))).subs(lst(log(wi

        ex p=pred.subs(replacements).real_part();
        p=collect_common_factors(expand(p.evalf()));
        FUNCP_CUBA fp;

        lst l(tanb,McH,MR,MI);

        for(uint i=0;i<3;i++){
                l.append(Mu[i]);
                l.append(Md[i]);
        }
        if(sb) push_back(prediction(ob,p));
        else {
        compile_ex(lst(p), l, fp);
        //cout<<"prediction numeric"<<p<<endl;
        //cout<<"exp "<<ob->expected()<<endl<<endl;
        push_back(prediction(ob,fp));
    }
        }
```

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

Definition at line 1477 of file BGL.h.

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

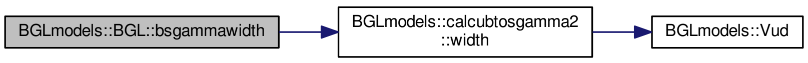
7.1.3.7 double BGLmodels::BGL::bsgammawidth (double *tanb*, double *McH*, double *MR*, double *MI*, int *option* = 0) [inline]

Definition at line 799 of file BGL.h.

References BGLmodels::calcubtosgamma2::width().

```
                                                                                            {
        parameters p=generateparameters();
        p[0].value=pow(10.0,tanb);
        p[1].value=McH;
        p[2].value=MR;
        p[3].value=MI;

        calcubtosgamma2 cal(mixes);

        return cal.width(p,option);
}
```

Here is the call graph for this function:


BGLmodels::BGL::bsgammawidth
BGLmodels::calcubtosgamma2 ::width
BGLmodels::Vud

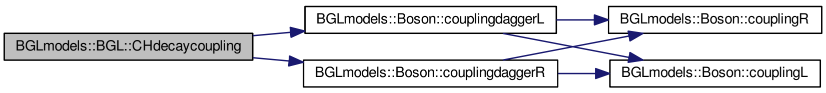
7.1.3.8 ex BGLmodels::BGL::CHdecaycoupling (Boson *higgs*, const Fermion & *ff3*, const Fermion & *ff4*) const [inline]

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.

```
                                                                                            {

        Fermion f3=ff3, f4=ff4;
        ex ret=0;
        for(uint k=fElectron;k<=fTau;k++)
        if(ff3.flavour==fAny || ff3.flavour==k){
                f3.flavour=(FFlavour)k;
        for(uint l=fElectron;l<=fTau;l++)
        if(ff4.flavour==fAny || ff4.flavour==l){
                f4.flavour=(FFlavour)l;
                ret+=higgs.couplingdaggerL(f3,f4)*higgs.couplingdaggerL(f3,f4).conjugate()+higgs.
      couplingdaggerR(f3,f4)*higgs.couplingdaggerR(f3,f4).conjugate();
        }}
        return collect_common_factors(ret.subs(conjtoabs));
}
```

Here is the call graph for this function:


BGLmodels::BGL::CHdecaycoupling
BGLmodels::Boson::couplingdaggerL
BGLmodels::Boson::couplingdaggerR
BGLmodels::Boson::couplingR
BGLmodels::Boson::couplingL

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, BG↩ Lmodels::Boson::mass, BGLmodels::Boson::s, and BGLmodels::sAny.

```
00879
              {
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           //
      ret+=wc.get_integral(a,mass,op,mixes.massnum(f1),mixes.massnum(f2),mixes.massnum(f3),mixes.massnum(f4))/pow(mixes.massnu
00915           }}}}
00916           if(ff2.flavour==ff4.flavour) ret=ret/2;
00917           return collect_common_factors(ret.subs(conjtoabs));
00918           //return
       expand(ret.subs(lst(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.massnu
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]

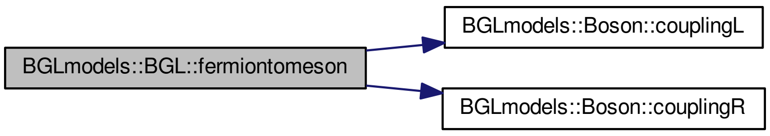
Definition at line 1314 of file BGL.h.

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), 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);
```

```
                                v1=v1+a*dirac_gammaL();
                                v2=v2+a.conjugate()*dirac_gammaR();
                                a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*
        bosons[i].couplingR(f3,f4)*s2/(mq1+mq2)/pow(bosons[i].mass,2);
                                v1=v1+a*dirac_gammaR();
                                v2=v2+a.conjugate()*dirac_gammaL();
                        }
                        else{
                                ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
                                ex a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1)
        )*bosons[i].couplingL(f3,f4)/pow(bosons[i].mass,2);
                                v1=v1+a*sl*dirac_gammaL();
                                v2=v2+a.conjugate()*sl*dirac_gammaL();
                                a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*
        bosons[i].couplingR(f3,f4)/pow(bosons[i].mass,2);
                                v1=v1+a*sl*dirac_gammaR();
                                v2=v2+a.conjugate()*sl*dirac_gammaR();
                        }
                }
                ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)+mq4*dirac_ONE();
                ex dt=dirac_trace(vq3*v1*vq4*v2).simplify_indexed(sp);
                ex result=expand(dt*2*lq31/mq4/mq4/Pi/128);

                ret+=result;
                }
                }



                return pow(meson.decay_factor,2)*collect_common_factors(ret.subs(
        conjtoabs));
                //return
         expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
        }
```

Here is the call graph for this function:


BGLmodels::BGL::fermiontomeson
BGLmodels::Boson::couplingL
BGLmodels::Boson::couplingR

7.1.3.12 ex BGLmodels::BGL::fermiontomesontest (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::fAny, BGLmodels::fElectron, BGLmodels::Fermion↩ ::flavour, BGLmodels::fTau, BGLmodels::Meson::mass, BGLmodels::Meson::q1, and BGLmodels::Meson::q2.

```
                                                                                                            {

        const Fermion& f1(meson.q1), f2(meson.q2);
        ex mesonmass=meson.mass;

        Fermion f3=ff3, f4=ff4;

        ex ret=0;

        realsymbol q3("q3"), q4("q4");

        symbol sL("sL"), sR("sR"), vL("vL"), vR("vR");
        ex s2=pow(mesonmass,2);

```

```
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), lqml=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_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_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.

```
                {
        return x/2;
        }
```

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

Definition at line 695 of file BGL.h.

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

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

Definition at line 678 of file BGL.h.

```
               {
        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));
        }
```

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().

```
                                            {
        parameters p;
        //x=log_10(tanb)
        p.push_back(freeparameter(-3,3,r,stepsize));
        //y=log_10(McH)
        if(max==1) p.push_back(freeparameter(10,10000,r,stepsize));
        else p.push_back(freeparameter(10,mmmax,r,stepsize));
        //log_10(massR)
        p.push_back(freeparameter(-200,200,r,stepsize));
        //log_10(massI)
        p.push_back(freeparameter(-50,50,r,stepsize));

        return p;
}
```

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)), 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                 //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().

```
                                                {
        //cout<<aux<<endl;
        //double
 c2=(1+sqrt(1-4*sqrt(ex_to<numeric>(mudecay.subs(lst(tanb==exp(p[0].value),McH==p[1].value))).to_double())))/2;

        double x=pow(10.0,p[0].value);
        //double y=pow(10.0,p[1].value);
        //double z=pow(10.0,p[2].value);
        //double w=pow(10.0,p[3].value);

        double y=p[1].value;
        double z=y+p[2].value;
        double w=z+p[3].value;

        parameters pp(p);
        pp[0].value=x;
        pp[2].value+=pp[1].value;
        pp[3].value+=pp[2].value;
        pp.values=vector<double>();
        for(uint i=0; i<4; i++) pp.values.push_back(pp[i].value);
        lst &l=pp.p;
        l=lst(tanb==x,McH==y,MR==z,MI==w);

        return pp;
 }
```

Here is the caller graph for this function:

BGLmodels::BGL::getlist
main

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

Definition at line 668 of file BGL.h.

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

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

Definition at line 673 of file BGL.h.

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

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, BG↩ Lmodels::Boson::mass, BGLmodels::Boson::s, BGLmodels::sScalar, BGLmodels::sVector, and BGLmodels::t↩ Lepton.

```
                                                {

        ex ret1=0,ret2=0;
        Fermion f2(tLepton,iUp);
        Fermion f4(tLepton,iUp);

        for(uint k=fElectron;k<=fTau;k++){
                f2.flavour=(FFlavour)k;
        for(uint l=fElectron;l<=fTau;l++){
                f4.flavour=(FFlavour)l;
        for(uint i=0;i<bosons.size();i++)
                if(bosons[i].s==sScalar) {
                        ex x=bosons[i].couplingdaggerR(f2,f1)*bosons[i].couplingL(f3,f4)/pow(
      bosons[i].mass,2);
                        ret1+=x*x.conjugate();
                }
                else if(bosons[i].s==sVector) {
                        ex x=bosons[i].couplingdaggerL(f2,f1)*bosons[i].couplingL(f3,f4)/pow(
      bosons[i].mass,2);
                        ret2+=x*x.conjugate();
                }
        }}
        //r2.append();
        ret2=ret2.subs(conjtoabs);
        ret1=ret1.subs(conjtoabs);
        for(uint i=0;i<3;i++){
                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))));
                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))));
        }

        cout<<ret2<<endl;
        return collect_common_factors(ret1/ret2);
}
```

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

Definition at line 664 of file BGL.h.

```
               {
        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));
        }
```

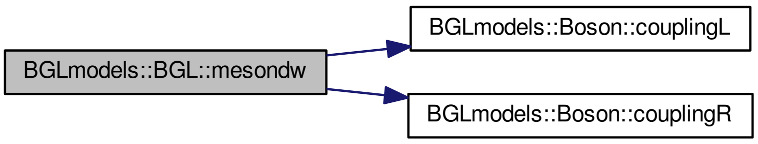
7.1.3.25 ex BGLmodels::BGL::mesondw (const Meson & *meson,* const Fermion & *ff3,* const Fermion & *ff4,* BSpin *s* = sAny) const [inline]

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 mq1=mixes.mass(f1),mq2=mixes.mass(f2),mq3=
      mixes.mass(f3),mq4=mixes.mass(f4);
01190                 ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;
01191                 scalar_products sp;
01192                 sp.add(q4, q3, (s2-m2q4-m2q3)/2);
01193                 sp.add(q3, q3, m2q3);
01194                 sp.add(q4, q4, m2q4);
01195                 ex q10=(s2+mq1*mq1-mq2*mq2)/(2*sqrt(s2)), lq1l=sqrt(q10*q10-mq1*mq1);
01196                 ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);
01197
01198         for(uint i=0;i<bosons.size();i++) if(bosons[i].s==s || s==
      sAny){
01199                 if(bosons[i].s==0){
01200                         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);
01201                         v1=v1+a*dirac_gammaL();
01202                         v2=v2+a.conjugate()*dirac_gammaR();
01203                         a=-(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*
      bosons[i].couplingR(f3,f4)*s2/(mq1+mq2)/pow(bosons[i].mass,2);
01204                         v1=v1+a*dirac_gammaR();
01205                         v2=v2+a.conjugate()*dirac_gammaL();
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*lq3l/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 }
```

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.

```
                                                                                                      {

          const Fermion& f1(meson.q1), f2(meson.q2);
          ex mesonmass=meson.mass;

          Fermion f3=ff3, f4=ff4;

          ex ret=0;

          realsymbol q3("q3"), q4("q4");
          symbol gL("gL"), gR("gR"),gVL("gVL"), gVR("gVR");
          symbol gS("gS"), gP("gP"), gA("gA");

          ex s2=pow(mesonmass,2);

          for(uint k=fElectron;k<=fTau;k++)
          if(ff3.flavour==fAny || ff3.flavour==k){
                  f3.flavour=(FFlavour)k;
          for(uint l=fElectron;l<=fTau;l++)
          if(ff4.flavour==fAny || ff4.flavour==l){
                  f4.flavour=(FFlavour)l;
                  ex v1=0, v2=0;
                  ex mq1=mixes.mass(f1),mq2=mixes.mass(f2),mq3=
      mixes.mass(f3),mq4=mixes.mass(f4);
                  ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;
                  scalar_products sp;
                  sp.add(q4, q3, (s2-m2q4-m2q3)/2);
                  sp.add(q3, q3, m2q3);
                  sp.add(q4, q4, m2q4);
                  ex q10=(s2+mq1*mq1-mq2*mq2)/(2*sqrt(s2)), lq11=sqrt(q10*q10-mq1*mq1);
                  ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq31=sqrt(q30*q30-mq3*mq3);

                  ex a;
          /*      a=-gL*s2/(mq1+mq2);
                  v1=v1+a*dirac_gammaL();
                  v2=v2+a.conjugate()*dirac_gammaR();
                  a=-gR*s2/(mq1+mq2);
                  v1=v1+a*dirac_gammaR();
                  v2=v2+a.conjugate()*dirac_gammaL();

                  ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
                  a=gA;
                  v1=v1+a*sl*dirac_gamma5();
                  v2=v2+a.conjugate()*sl*dirac_gamma5();
*/
                  a=-gS*s2/(mq1+mq2);
                  v1=v1+a*dirac_ONE();
                  v2=v2+a.conjugate()*dirac_ONE();
                  a=-gP*s2/(mq1+mq2);
                  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*lq31/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 }
```

7.1.3.27 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 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+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(
      bosons[i].mass,2);
01137                         x2+=bosons[i].couplingdaggerR(f2,f1)*bosons[i].couplingL(f32,f4)/pow(
      bosons[i].mass,2);
01138                 }
01139                 else if(bosons[i].s==sVector) {
01140                         y1+=bosons[i].couplingdaggerL(f2,f1)*bosons[i].couplingL(f31,f4)/pow(
      bosons[i].mass,2);
01141                         y2+=bosons[i].couplingdaggerL(f2,f1)*bosons[i].couplingL(f32,f4)/pow(
      bosons[i].mass,2);
01142                 }
01143                 }
01144                 ret1+=(x1*y1.conjugate()).real_part();
01145                 ret2+=(x2*y2.conjugate()).real_part();
01146                 rety1+=y1*y1.conjugate();
01147                 rety2+=y2*y2.conjugate();
01148         }}
01149         ret2=(ret2/rety2*mixes.mass(f32)/mixes.mass(f1)).subs(
      conjtoabs);
01150         ret1=(ret1/rety1*mixes.mass(f31)/mixes.mass(f1)).subs(
      conjtoabs);
01151         for(uint i=0;i<3;i++){
01152                 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))));
01153                 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))));
01154         }
01155 
01156         ex x=pow(mixes.mass(f31)/mixes.mass(f1),2);
01157         ex F1=1-8*x+8*pow(x,3)-pow(x,4)-12*pow(x,2)*log(x);
01158         ex g1=1+9*x-9*pow(x,2)-pow(x,3)+6*x*(1+x)*log(x);
01159         ex N1=1+gRR2(f1,f31)/4;
01160 
01161         x=pow(mixes.mass(f32)/mixes.mass(f1),2);
01162 
01163         ex F2=1-8*x+8*pow(x,3)-pow(x,4)-12*pow(x,2)*log(x);
01164         ex g2=1+9*x-9*pow(x,2)-pow(x,3)+6*x*(1+x)*log(x);
01165         ex N2=1+gRR2(f1,f32)/4;
01166 
01167         return collect_common_factors(N1*(F1+2/N1*ret1*g1)/N2/(F2+2/N2*ret2*g2)*F2/F1);
01168 }
```

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(lst(tanb==pow(10.0,xx[0]),
      McH==xx[1])).evalf()).to_double();
01484 }
```

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

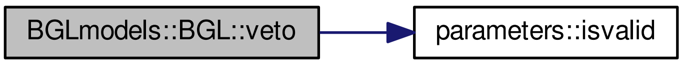
Reimplemented from Model.

Definition at line 741 of file BGL.h.

References parameters::isvalid().

```
                                                    {
        if(!p.isvalid()) return 1;
        if(max==1){
        double mr=p[1].value+p[2].value;
        if(mr<10 || mr>10000) return 1;
        mr+=p[3].value;
        if(mr<10 || mr>10000) return 1;
        return 0;
        }
        else{
        double mr=p[1].value+p[2].value;
        if(mr<10 || mr>mmmax) return 1;
        mr+=p[3].value;
        if(mr<10 || mr>mmmax) return 1;
        return 0;
        }
        }
```

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.

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

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::Ml

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::mmmax

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.

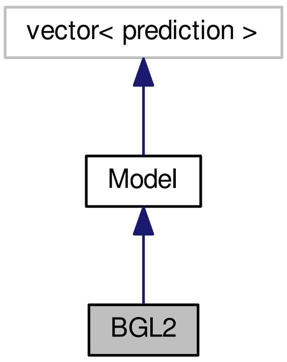
The documentation for this class was generated from the following file:

- BGL.h

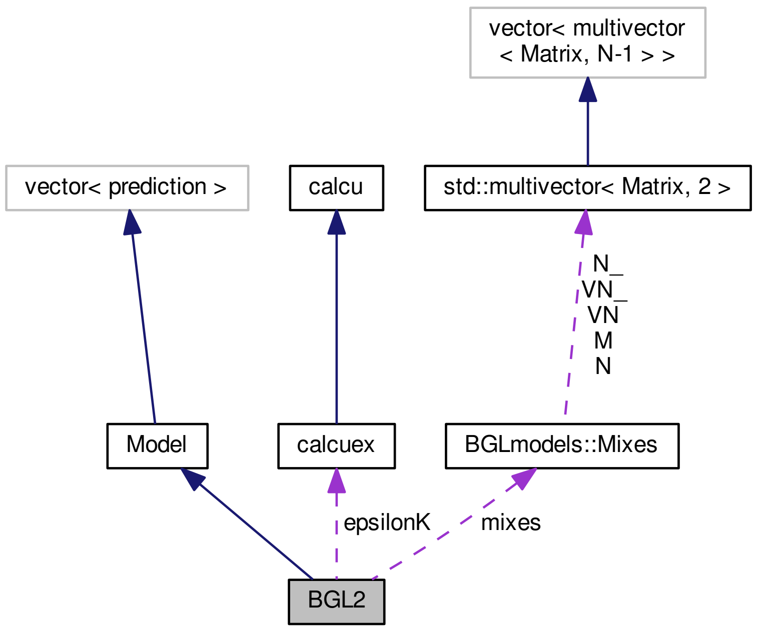
7.2 BGL2 Class Reference

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 mmmax
- 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.

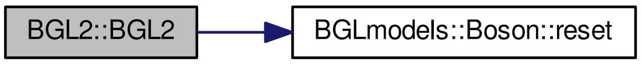
References BGLmodels::Boson::C, BGLmodels::cParticle, BGLmodels::fElectron, BGLmodels::fMuon, BG↩ Lmodels::fTau, BGLmodels::hLeft, BGLmodels::hRight, BGLmodels::iDown, BGLmodels::iUp, BGLmodels::M_MW, BGLmodels::M_MZ, BGLmodels::Boson::mass, BGLmodels::Boson::reset(), BGLmodels::Boson::s, BGLmodels↩ ::sScalar, BGLmodels::sVector, BGLmodels::tLepton, and BGLmodels::tQuark.

```
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           mmmax(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.appendtolst(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)/mixes.V[i][j][k]);
00094                                }
00095
00096        //W+ boson
00097        boson.mass=MW;
```

```
        boson.s=sVector;

        for(uint t=tLepton;t<=tQuark;t++) boson.C[t][iUp][iDown][hLeft]=mixes.V[t]*Matrix(g/sqrt(ex(2)));
        Boson wboson=boson;
        bosons.push_back(boson);
        boson.reset();

        //H+ boson
        boson.mass=McH;
        boson.s=sScalar;

        for(uint t=tLepton;t<=tQuark;t++)
        for(uint i=iUp;i<=iDown;i++) boson.C[t][iUp][iDown][i]=mixes.VN[t][i]*Matrix(g/MW/sqrt(ex(2)));
        Boson chiggs=boson;
        bosons.push_back(boson);
        boson.reset();

        for(int b=bosons.size()-1;b>=0;b--){
                boson.mass=bosons[b].mass;
                boson.s=bosons[b].s;
                if(boson.s==sVector)
                        for(uint t=tLepton;t<=tQuark;t++)
                        for(uint i=iUp;i<=iDown;i++)
                        for(uint j=iUp;j<=iDown;j++)
                        for(uint h=hLeft;h<=hRight;h++){
                                boson.C[t][i][j][h]=bosons[b].C[t][j][i][h].conjugate();
                                }
                else for(uint t=tLepton;t<=tQuark;t++)
                        for(uint i=iUp;i<=iDown;i++)
                        for(uint j=iUp;j<=iDown;j++)
                        for(uint h=hLeft;h<=hRight;h++){
                                boson.C[t][i][j][hLeft]=bosons[b].C[t][j][i][hRight].conjugate();
                                boson.C[t][i][j][hRight]=bosons[b].C[t][j][i][hLeft].conjugate();
                                }
                bosons.push_back(boson);
                boson.reset();
                }

        //(R+iI)/sqrt(2) boson
        boson.mass=MR;
        boson.s=sScalar;

        for(uint t=tLepton;t<=tQuark;t++){
                        boson.C[t][iDown][iDown][hRight]=mixes.N[t][iDown]*Matrix(g/MW/ex(2));
                        boson.C[t][iUp][iUp][hLeft]=mixes.N[t][iUp].conjugate()*Matrix(g/MW/ex(2));
                        boson.C[t][iDown][iDown][hLeft]=mixes.N[t][iDown].conjugate()*Matrix(g/MW/ex(2));
                        boson.C[t][iUp][iUp][hRight]=mixes.N[t][iUp]*Matrix(g/MW/ex(2));
                }
        bosons.push_back(boson);
        boson.reset();

        //(R+iI)/sqrt(2) boson
        boson.mass=MI;
        boson.s=sScalar;

        for(uint t=tLepton;t<=tQuark;t++){
                        boson.C[t][iDown][iDown][hRight]=mixes.N[t][iDown]*Matrix(I*g/MW/ex(2));
                        boson.C[t][iUp][iUp][hLeft]=mixes.N[t][iUp].conjugate()*Matrix(I*g/MW/ex(2));
                        boson.C[t][iDown][iDown][hLeft]=mixes.N[t][iDown].conjugate()*Matrix(-I*g/MW/ex(2));
                        boson.C[t][iUp][iUp][hRight]=mixes.N[t][iUp]*Matrix(-I*g/MW/ex(2));
                }
        bosons.push_back(boson);
        boson.reset();

        Fermion electron(tLepton,iDown,fElectron);
        Fermion electronR(tLepton,iDown,fElectron,cParticle,hRight);

        Fermion muon(tLepton,iDown,fMuon);
        Fermion muonR(tLepton,iDown,fMuon,cParticle,hRight);

        Fermion tau(tLepton,iDown,fTau);
        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

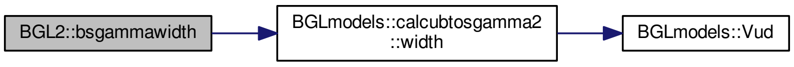
7.2.3.1 double BGL2::bsgammawidth (double *tanb_*, double *McH_*, double *MR_*, double *MI_*, int *option* = 0) [inline]

Definition at line 298 of file draw.cpp.

References BGLmodels::calcubtosgamma2::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         calcubtosgamma2 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=lst(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.

```
                                                    {
        //cout<<aux<<endl;
        //double
 c2=(1+sqrt(1-4*sqrt(ex_to<numeric>(mudecay.subs(lst(tanb==exp(p[0].value),McH==p[1].value))).to_double())))/2;

        double x=pow(10.0,p[0].value);
        //double y=pow(10.0,p[1].value);
        //double z=pow(10.0,p[2].value);
        //double w=pow(10.0,p[3].value);

        double y=p[1].value;
        double z=y+p[2].value;
        double w=z+p[3].value;

        parameters pp(p);
        pp[0].value=x;
        pp[2].value+=pp[1].value;
        pp[3].value+=pp[2].value;
        pp.values=vector<double>();
        for(uint i=0; i<4; i++) pp.values.push_back(pp[i].value);
        lst &l=pp.p;
        l=lst(tanb==x,McH==y,MR==z,MI==w);
    l.append(QCD1==inter1.Eval(y));
        l.append(QCD2==inter2.Eval(y));

        for(uint i=0;i<3;i++){
                l.append(Mu[i]==Mu_[i].Eval(log(y)));
                l.append(Md[i]==Md_[i].Eval(log(y)));
        }
        return pp;
}
```

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.

```
                                                                                {

        ex ret=0;

                ex v1=0, v2=0;
                ex mq1=mixes.mass(f1),mq2=mixes.mass(f2);
                ex m2q1=mq1*mq1, m2q2=mq2*mq2;

        for(uint i=0;i<bosons.size();i++)
                if(bosons[i].s==0){
                        ex a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1)
);
                        v1=v1+pow(a/bosons[i].mass,2);

                        ex b=(bosons[i].couplingdaggerR(f2,f1)+bosons[i].couplingdaggerL(f2,f1)
);
                        v2=v2+pow(b/bosons[i].mass,2);
                }

        ex fc=mesonmass/(mixes.massnum(f1)+mixes.massnum(f2));
        fc=pow(fc,2);

        ret=2*(-v1*(1+11*fc)+v2*(1+fc))*mesonmass/96;

        return collect_common_factors(ret.subs(conjtoabs));
        //return
 expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
}
```

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 lst 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::Ml

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::mmmax

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:

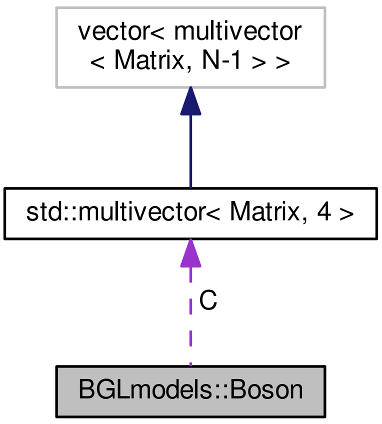
- draw.cpp

7.3 BGLmodels::Boson Class Reference

Gauge boson.

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

Collaboration diagram for BGLmodels::Boson:


vector< multivector < Matrix, N-1 > >
std::multivector< Matrix, 4 >
C
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

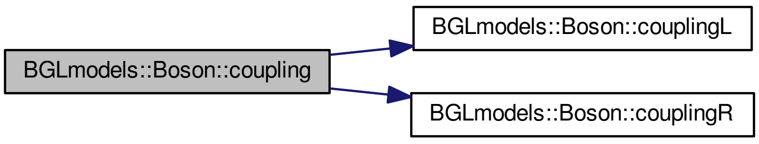
7.3.3.1 ex BGLmodels::Boson::coupling (const Fermion & *f2*, const Fermion & *f1*, ex *mu*) [inline]

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:


BGLmodels::Boson::coupling
BGLmodels::Boson::couplingL
BGLmodels::Boson::couplingR

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

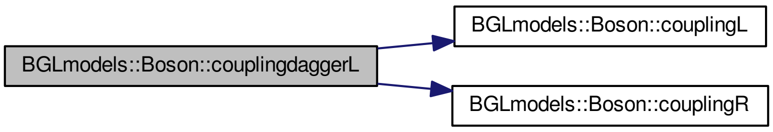
Definition at line 51 of file BGL.h.

References couplingL(), couplingR(), s, and BGLmodels::sScalar.

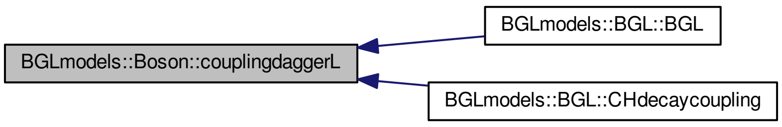
Referenced by BGLmodels::BGL::BGL(), and BGLmodels::BGL::CHdecaycoupling().

```
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:


BGLmodels::Boson::couplingdaggerL
BGLmodels::Boson::couplingL
BGLmodels::Boson::couplingR

Here is the caller graph for this function:


BGLmodels::Boson::couplingdaggerL
BGLmodels::BGL::BGL
BGLmodels::BGL::CHdecaycoupling

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

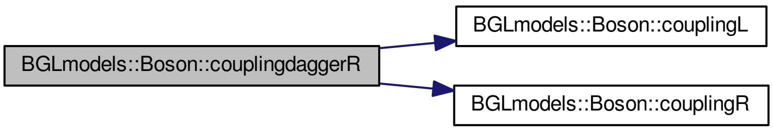
Definition at line 55 of file BGL.h.

References couplingL(), couplingR(), s, and BGLmodels::sScalar.

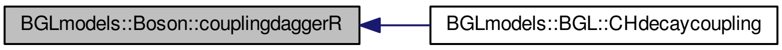
Referenced by BGLmodels::BGL::CHdecaycoupling().

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

Here is the call graph for this function:


BGLmodels::Boson::couplingdaggerR
BGLmodels::Boson::couplingL
BGLmodels::Boson::couplingR

Here is the caller graph for this function:


BGLmodels::Boson::couplingdaggerR
BGLmodels::BGL::CHdecaycoupling

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

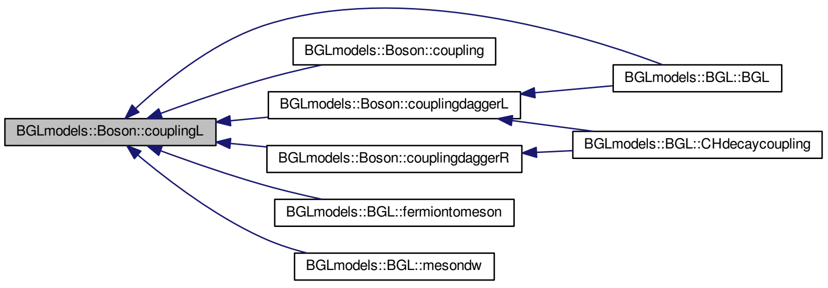
Definition at line 26 of file BGL.h.

References C, BGLmodels::Fermion::flavour, BGLmodels::Fermion::helicity, BGLmodels::hLeft, BGLmodels::h↩ Right, BGLmodels::Fermion::isospin, s, BGLmodels::sScalar, BGLmodels::sVector, and BGLmodels::Fermion↩ ::type.

Referenced by BGLmodels::BGL::BGL(), coupling(), couplingdaggerL(), couplingdaggerR(), BGLmodels::BGL↩ ::fermiontomeson(), and BGLmodels::BGL::mesondw().

```
                                                                         {
                bool quiralfilter=0;
                if(f1.type!=f2.type) return 0;
                if(s==sScalar){
                        if(f1.helicity!=hRight && f2.helicity!=hLeft) quiralfilter=1;
                        }
                else if(s==sVector){
                        if(f1.helicity!=hRight && f2.helicity!=hRight) quiralfilter=1;
                        }

                if(quiralfilter) return C[f2.type][f2.isospin][f1.isospin][
      hLeft][f2.flavour][f1.flavour];
                return 0;
                }
```

Here is the caller graph for this function:


BGLmodels::Boson::couplingL
BGLmodels::Boson::coupling
BGLmodels::Boson::couplingdaggerL
BGLmodels::Boson::couplingdaggerR
BGLmodels::BGL::fermiontomeson
BGLmodels::BGL::mesondw
BGLmodels::BGL::BGL
BGLmodels::BGL::CHdecaycoupling

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

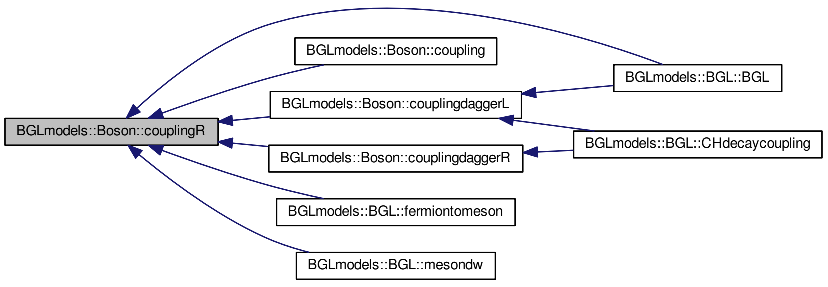
Definition at line 39 of file BGL.h.

References C, BGLmodels::Fermion::flavour, BGLmodels::Fermion::helicity, BGLmodels::hLeft, BGLmodels::h↩ Right, BGLmodels::Fermion::isospin, s, BGLmodels::sScalar, BGLmodels::sVector, and BGLmodels::Fermion↩ ::type.

Referenced by BGLmodels::BGL::BGL(), coupling(), couplingdaggerL(), couplingdaggerR(), BGLmodels::BGL↩ ::fermiontomeson(), and BGLmodels::BGL::mesondw().

```
                                                        {
                bool quiralfilter=0;
                if(f1.type!=f2.type) return 0;
                if(s==sScalar){
                        if(f2.helicity!=hRight && f1.helicity!=hLeft) quiralfilter=1;
                        }
                else if(s==sVector){
                        if(f1.helicity!=hLeft && f2.helicity!=hLeft) quiralfilter=1;
                        }
                if(quiralfilter) return C[f2.type][f2.isospin][f1.isospin][
     hRight][f2.flavour][f1.flavour];
                return 0;
                }
```

Here is the caller graph for this function:


BGLmodels::Boson::couplingR
BGLmodels::Boson::coupling
BGLmodels::Boson::couplingdaggerL
BGLmodels::Boson::couplingdaggerR
BGLmodels::BGL::fermiontomeson
BGLmodels::BGL::mesondw
BGLmodels::BGL::BGL
BGLmodels::BGL::CHdecaycoupling

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

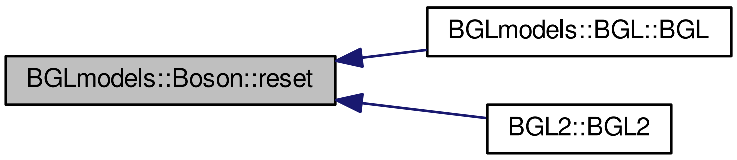
Definition at line 63 of file BGL.h.

References C.

Referenced by BGLmodels::BGL::BGL(), and BGL2::BGL2().

```
                    {
                C=multivector<Matrix,4>(Matrix(),2,2,2,2);
                }
```

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↩ ::fermiontomeson(), 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::fermiontomeson(), 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:

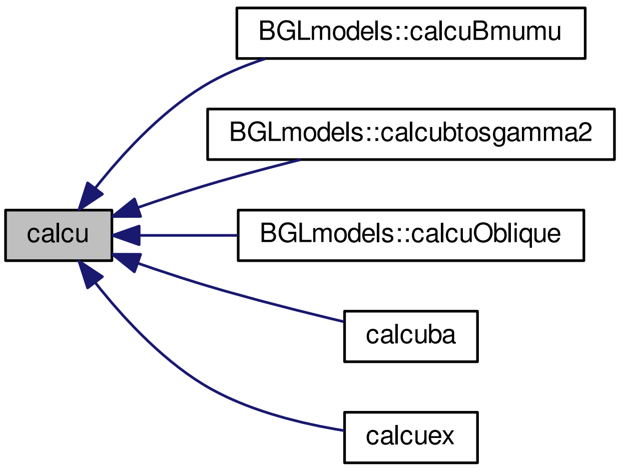
- BGL.h

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

hipothesis	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::calcubtosgamma2, calcuex, calcuba, and BGLmodels↩
::calcuOblique.

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

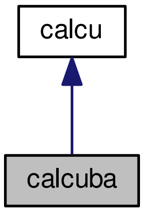
- model.h

7.5 calcuba Class Reference

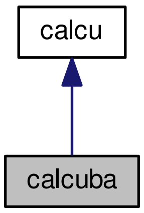
class to do the calculus of a constraint based on a GiNaC compiled expression

```
#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

hipothesis	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:

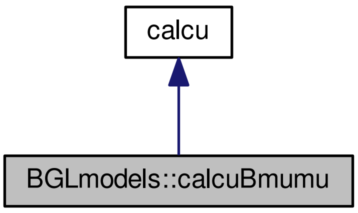
- model.h

7.6 BGLmodels::calcuBmumu Class Reference

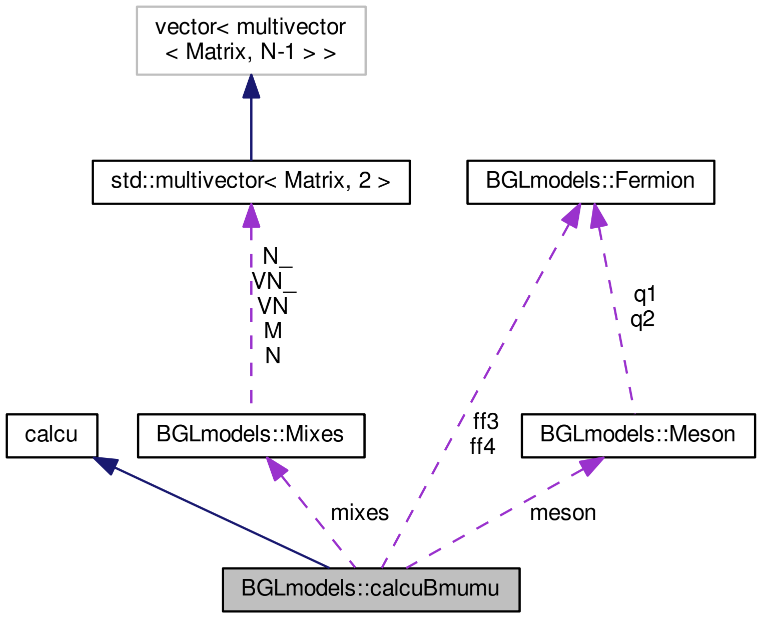
calculus of the constraints coming from the B->mu mu decay

```
#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->mu mu decay

Definition at line 672 of file Formulas.h.

7.6.2 Constructor & Destructor Documentation

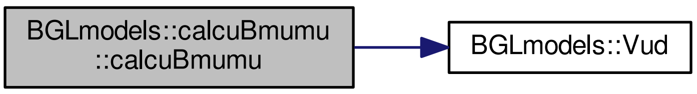
7.6.2.1 BGLmodels::calcuBmumu::calcuBmumu (const Mixes & *mix,* const Meson & *m,* const Fermion & *f3,* const Fermion & *f4,* observable ∗ *ob,* const char ∗ *name*) [inline]

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 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                   }
```

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
      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]

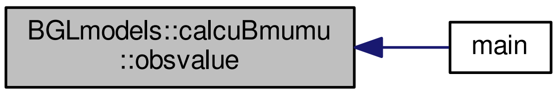
Definition at line 716 of file Formulas.h.

References parameters::values.

Referenced by main().

```
00716                                         {
00717            //double
      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         }
```

Here is the caller graph for this function:



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

Parameters

hipothesis	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.

```
                                                    {
          //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];
          int n=4,m=1;
          double ret=0;
          fp(&n,&(p.values[0]),&m,&ret);

          return o->loglikelihood(ret);
       }
```

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

Definition at line 725 of file Formulas.h.

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

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:

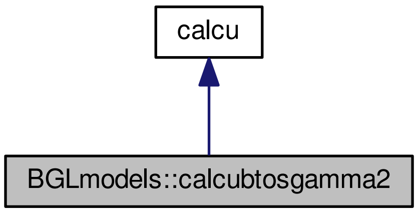
- Formulas.h

7.7 BGLmodels::calcubtosgamma2 Class Reference

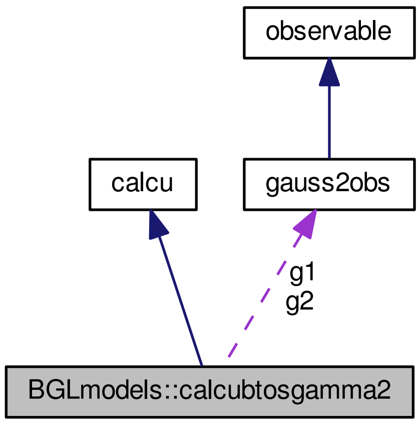
calculus of the constraints coming from the b->s gamma decay

`#include <Formulas.h>`

Inheritance diagram for BGLmodels::calcubtosgamma2:



Collaboration diagram for BGLmodels::calcubtosgamma2:



Public Member Functions

- calcubtosgamma2 (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->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]

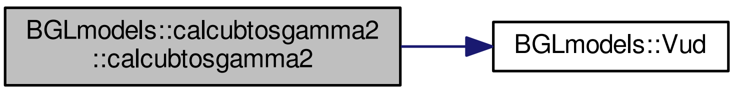
Definition at line 286 of file Formulas.h.

References BGLmodels::C7SM_Mt, BGLmodels::C8SM_Mt, BGLmodels::iDown, BGLmodels::iUp, BGLmodels::↩
Mixes::N_, BGLmodels::Mixes::replacements, BGLmodels::Mixes::tanb, BGLmodels::tQuark, 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+aee*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()+
      a8er*(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                         //      cout<<a<<" ";
00352                         }
```

```
                } //cout<<endl;
                }
        for(uint i=0; i<3;i++) {
                Md_[i].SetData(m_[0],m_[2*i+1]);
                Mu_[i].SetData(m_[0],m_[2*i+2]);
        }
//      cout<<"Eval "<<Mu_[2].Eval(log(100.0))<<endl;
//      cout<<"Eval "<<Md_[2].Eval(log(100.0))<<endl;

        finter2.close();

    ifstream finter3("interpolation2.dat");

    if(!finter3.is_open()){
                cout<<"ERROR: interpolation2.dat not found"<<endl;
                exit(1);
                }
    vector<double> vinter20, vinter21, vinter22;
    while(!finter3.eof()){
                double a=0,b=0,c=0;
                finter3>>a>>b>>c;
                if(a!=0){
                // cout<<a<<" "<<b<<" "<<c<<endl;
                 vinter20.push_back(a);
                 vinter21.push_back(b);
                 vinter22.push_back(c);
            }
                }

        inter3.SetData(vinter20,vinter21);
        inter4.SetData(vinter20,vinter22);

        finter3.close();

        vector<ex> vex(24);

           const uint i=ii;
        for(uint j=0;j<2;j++)
        for(uint k=0;k<3;k++){
            vex[j*6+k*2+0]=mixes.VN_[tQuark][iUp][k][j].conjugate()*mixes.VN_[
tQuark][iUp][k][i];
            vex[j*6+k*2+1]=mixes.N_[tQuark][iDown][j][k]*mixes.N_[
tQuark][iDown][i][k].conjugate();
            vex[j+k*2+12]=-mixes.VN_[tQuark][iUp][k][j].conjugate()*mixes.VN_[
tQuark][iDown][k][i];
            vex[j+k*2+18]=mixes.VN_[tQuark][iDown][k][j].conjugate()*mixes.VN_[
tQuark][iDown][k][i];
                }
        lst l;
    for(uint k=0;k<vex.size();k++){
                vex[k]=vex[k].subs(mixes.replacements).evalf();
                l.append(vex[k].real_part());
                l.append(vex[k].imag_part());
                }
        compile_ex(l, lst(mixes.tanb), fp);
        }
```

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.

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

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

Definition at line 644 of file Formulas.h.

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

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

Definition at line 648 of file Formulas.h.

```
                            {
            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));
            }
```

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

Definition at line 652 of file Formulas.h.

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

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

Parameters

hipothesis	the theoretical hypothesis

Returns

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

Implements calcu.

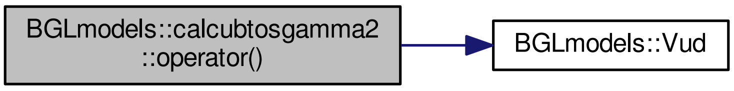
Definition at line 406 of file Formulas.h.

References BGLmodels::C7SM_Mt, BGLmodels::C8SM_Mt, BGLmodels::mt_mt, and BGLmodels::Vud().

```
                                                  {
             double tanb=p[0].value;
             double y=p[1].value;
```

```
                double z=p[2].value;
                double w=p[3].value;
            double McH=y, MR=z, MI=w;

            double y0=y;
            if(y<mt_mt) y0=mt_mt;
             double QCD1[2]={inter3.Eval(y0),inter1.Eval(y)};
             double QCD2[2]={inter4.Eval(y0),inter2.Eval(y)};

             double Mu[3],Md[3];

            for(uint i=0;i<3;i++){
                    Mu[i]=Mu_[i].Eval(log(y));
                    Md[i]=Md_[i].Eval(log(z));
            }
                const uint i=ii;
                CD CC7[2],DD7[2],CC8[2],DD8[2];
                double res[2];
               // constexpr double C7SM_[2]={C7SM_Mt,C7SM_Mb};
               // constexpr double C8SM_[2]={C8SM_Mt,C8SM_Mb};

                std::array<double,48> ret;
                const int n=1,m=48;
                fp(&n,&(tanb),&m,&(ret[0]));
                for(uint j=0;j<2;j++){
                     const double mbottom=Md[i];
                     const double mstrange=Md[j];
                             //ex mbottom=mixes.M[tQuark][iDown][i][i];
                             //ex mstrange=mixes.M[tQuark][iDown][j][j];

                             CD C7,D7,C8,D8;
                             for(uint k=0;k<3;k++){
                             double mup=Mu[k];
                             double mdown=Md[k];
                             //ex mup=mixes.M[tQuark][iUp][k][k];
                             //ex mdown=mixes.M[tQuark][iDown][k][k];
                             //f1+=
                             double mmu=std::pow(mup/McH,2);
                             double mmdR=std::pow(mdown/MR,2);
                             double mmdI=std::pow(mdown/MI,2);

                             double A0u=A0(mmu);
                             double A1u=A1(mmu);
                             double A2u=A2(mmu);
                             double A3u=A3(mmu);
                 double A0d=(A0(mmdR)+A0(mmdI));
                             double A1d=(A1(mmdR)-A1(mmdI));

                             CD f1(ret[j*12+4*k+0],ret[j*12+4*k+1]);
                             C7+=f1*A2u;
                             C8+=-2.0*f1*A0u;

                             CD f2=CD(ret[36+j*2+4*k+0],ret[36+j*2+4*k+1])*mstrange*mbottom/mup/mup;
                             //CD f2=f1*mstrange*mbottom/mup/mup;
                             D7+=f2*A2u;
                             D8+=-2.0*f2*A0u;

                             CD f12(ret[24+j*2+4*k+0],ret[24+j*2+4*k+1]);
                             C7+=f12*A3u;
                             C8+=2.0*f12*A1u;

                             CD f4(ret[j*12+4*k+2],ret[j*12+4*k+3]);
                             C7+=f4*A0d/3.0;
                             C8+=-f4*A0d;

                             C7+=f4*A1d/3.0;
                             C8+=-f4*A1d;

                             CD f6=f4*mstrange*mbottom/mdown/mdown;
                             D7+=f6*A0d/3.0;
                             D8+=-f6*A0d;
                             }
                     uint j0=j;
                     CC7[j]=(QCD1[j]*C7+QCD2[j]*C8)/2.0/conj(Vud[2][j])/Vud[2][i];
                     DD7[j]=(QCD1[j]*D7+QCD2[j]*D8)/2.0/conj(Vud[2][j])/Vud[2][i];
                     const double QCD3=(3*QCD2[j]/8+QCD1[j]);
                     CC8[j]=QCD3*C8/2.0/conj(Vud[2][j])/Vud[2][i];
                     DD8[j]=QCD3*D8/2.0/conj(Vud[2][j])/Vud[2][i];
                     const CD epsilon=conj(Vud[0][j])*Vud[0][i]/conj(Vud[2][j])/
      Vud[2][i];
                     const double upsilon=norm(conj(Vud[2][j])*Vud[2][i]/Vud[1][i]);
                     const CD R7=(C7SM_Mt+CC7[j])/C7SM_MW;
                     const CD R8=(C8SM_Mt+CD(0)*CC8[j])/C8SM_MW;
                     const CD R7_=(DD7[j])/C7SM_MW;
                     const CD R8_=CD(0)*(DD8[j])/C8SM_MW;


                     res[j]=a+aee*norm(epsilon)+aer*epsilon.real()+aei*epsilon.imag();
                     res[j]+=a77*(norm(R7)+norm(R7_))+a7r*R7.real()+a7i*R7.imag();
                     res[j]+=a88*(norm(R8)+norm(R8_))+a8r*R8.real()+a8i*R8.imag();
                     res[j]+=a87r*(R8*conj(R7)+R8_*conj(R7_)).real()+a7er*(R7*conj(epsilon)).real()+
     a8er*(R8*conj(epsilon)).real();
                     res[j]+=a87i*(R8*conj(R7)+R8_*conj(R7_)).imag()+a7ei*(R7*conj(epsilon)).imag()+
     a8er*(R8*conj(epsilon)).imag();
                     res[j]*=calN/100*upsilon;

                     /*res[j]=a+aee*norm(epsilon)+aer*epsilon.real()+aei*epsilon.imag();
                     res[j]+=a77*(norm(R7)+norm(R7_))+a7r*1+a7i*0;
                     res[j]+=a88*(norm(R8)+norm(R8_))+a8r*R8.real()+a8i*R8.imag();
                     res[j]+=a87r*1+a7er*(conj(epsilon)).real()+a8er*(R8*conj(epsilon)).real();
                     res[j]+=a87i*0+a7ei*(conj(epsilon)).imag()+a8er*(R8*conj(epsilon)).imag();
                     res[j]*=calN/100*upsilon;
                     */
             }
             double r1=3.15e-4+0.00247*(norm(CC7[1])+norm(DD7[1])-0.706*CC7[1].real());

             //ratio=res[0]/9.2e-6;
             //cout<<"RATIO "<<ratio<<endl;
             return g1.loglikelihood(r1)+0*g2.loglikelihood(res[0]);
                }
```

Here is the call graph for this function:



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

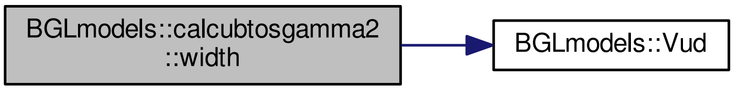
Definition at line 517 of file Formulas.h.

References BGLmodels::C7SM_Mt, BGLmodels::C8SM_Mt, BGLmodels::mt_mt, and BGLmodels::Vud().

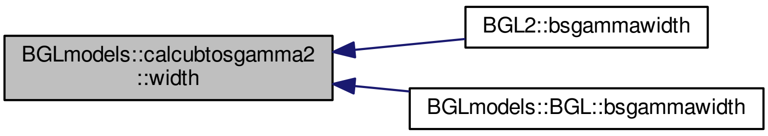
Referenced by BGL2::bsgammawidth(), and BGLmodels::BGL::bsgammawidth().

```
                                                                   {
         double tanb=p[0].value;
         double y=p[1].value;
           double z=p[2].value;
           double w=p[3].value;
        double McH=y, MR=z, MI=w;

        double y0=y;
        if(y<mt_mt) y0=mt_mt;
         double QCD1[2]={inter3.Eval(y0),inter1.Eval(y)};
         double QCD2[2]={inter4.Eval(y0),inter2.Eval(y)};

         double Mu[3],Md[3];

        for(uint i=0;i<3;i++){
                Mu[i]=Mu_[i].Eval(log(y));
                Md[i]=Md_[i].Eval(log(z));
        }
           const uint i=ii;
           CD CC7[2],DD7[2],CC8[2],DD8[2];
           double res[2];
          // constexpr double C7SM_[2]={C7SM_Mt,C7SM_Mb};
          // constexpr double C8SM_[2]={C8SM_Mt,C8SM_Mb};

           std::array<double,24> ret;
           const int n=1,m=24;
           fp(&n,&(tanb),&m,&(ret[0]));
           for(uint j=0;j<2;j++){
                const double mbottom=Md[i];
                const double mstrange=Md[j];
                        //ex mbottom=mixes.M[tQuark][iDown][i][i];
                        //ex mstrange=mixes.M[tQuark][iDown][j][j];

                        CD C7,D7,C8,D8;
                        for(uint k=0;k<3;k++){
                        double mup=Mu[k];
                        double mdown=Md[k];
                        //ex mup=mixes.M[tQuark][iUp][k][k];
                        //ex mdown=mixes.M[tQuark][iDown][k][k];
                        //f1+=
                        double mmu=std::pow(mup/McH,2);
                        double mmdR=std::pow(mdown/MR,2);
                        double mmdI=std::pow(mdown/MI,2);
                        double A0u=0,A1u=0, A2u=0, A3u=0, A0d=0, A1d=0;

                        if(option==0 || option==1){
                        A0u=A0(mmu);
                        A1u=A1(mmu);
                        A2u=A2(mmu);
                        A3u=A3(mmu);
                        }
            if(option==0 || option==2){
            A0d=(A0(mmdR)+A0(mmdI));
                        A1d=(A1(mmdR)-A1(mmdI));
                        }
                        if(option==3){
            A0d=(A0(mmdR));
                        A1d=(A1(mmdR));
                        }
                        if(option==4){
            A0d=(A0(mmdI));
                        A1d=(-A1(mmdI));
                        }

                        CD f1(ret[j*12+4*k+0],ret[j*12+4*k+1]);
                        C7+=f1*A2u;
                        C8+=-2.0*f1*A0u;

                        CD f2=f1*mstrange*mbottom/mup/mup;
                        D7+=f2*A2u;
                        D8+=-2.0*f2*A0u;

                        C7+=-f1*A3u;
                        C8+=-2.0*f1*A1u;

                        CD f4(ret[j*12+4*k+2],ret[j*12+4*k+3]);
                        C7+=f4*A0d/3.0;
                        C8+=-f4*A0d;

                        C7+=f4*A1d/3.0;
                        C8+=-f4*A1d;

                        CD f6=f4*mstrange*mbottom/mdown/mdown;
                        D7+=f6*A0d/3.0;
                        D8+=-f6*A0d;

                        }
                uint j0=j;
                CC7[j]=(QCD1[j]*C7+QCD2[j]*C8)/2.0/conj(Vud[2][j])/Vud[2][i];
                DD7[j]=(QCD1[j]*D7+QCD2[j]*D8)/2.0/conj(Vud[2][j])/Vud[2][i];
                const double QCD3=(3*QCD2[j]/8+QCD1[j]);
                CC8[j]=QCD3*C8/2.0/conj(Vud[2][j])/Vud[2][i];
                DD8[j]=QCD3*D8/2.0/conj(Vud[2][j])/Vud[2][i];
                const CD epsilon=conj(Vud[0][j])*Vud[0][i]/conj(Vud[2][j])/Vud[2][i];
                const double upsilon=norm(conj(Vud[2][j])*Vud[2][i]/Vud[1][i]);
                const CD R7=(C7SM_Mt+CC7[j])/C7SM_MW;
                const CD R8=(C8SM_Mt+CD(0)*CC8[j])/C8SM_MW;
                const CD R7_=(DD7[j])/C7SM_MW;
                const CD R8_=CD(0)*(DD8[j])/C8SM_MW;


                res[j]=a+aee*norm(epsilon)+aer*epsilon.real()+aei*epsilon.imag();
                res[j]+=a77*(norm(R7)+norm(R7_))+a7r*R7.real()+a7i*R7.imag();
                res[j]+=a88*(norm(R8)+norm(R8_))+a8r*R8.real()+a8i*R8.imag();
                res[j]+=a87r*(R8*conj(R7)+R8_*conj(R7_)).real()+a7er*(R7*conj(epsilon)).real()+a8er*(R8*conj(epsilon)).real();
                res[j]+=a87i*(R8*conj(R7)+R8_*conj(R7_)).imag()+a7ei*(R7*conj(epsilon)).imag()+a8er*(R8*conj(epsilon)).imag();
                res[j]*=calN/100*upsilon;

                /*res[j]=a+aee*norm(epsilon)+aer*epsilon.real()+aei*epsilon.imag();
                res[j]+=a77*(norm(R7)+norm(R7_))+a7r*1+a7i*0;
                res[j]+=a88*(norm(R8)+norm(R8_))+a8r*R8.real()+a8i*R8.imag();
                res[j]+=a87r*1+a7er*(conj(epsilon)).real()+a8er*(R8*conj(epsilon)).real();
                res[j]+=a87i*0+a7ei*(conj(epsilon)).imag()+a8er*(R8*conj(epsilon)).imag();
                res[j]*=calN/100*upsilon;
                */
        }
        double r1=3.15e-4+0.00247*(norm(CC7[1])+norm(DD7[1])-0.706*CC7[1].real());

        //ratio=res[0]/9.2e-6;
        //cout<<"RATIO "<<ratio<<endl;
        return g1.error(r1);
           }
```

Here is the call graph for this function:



Here is the caller graph for this function:


BGLmodels::calcubtosgamma2 ::width
BGL2::bsgammawidth
BGLmodels::BGL::bsgammawidth

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::a7er =-0.7827 [static]

Definition at line 283 of file Formulas.h.

7.7.4.5 constexpr double BGLmodels::calcubtosgamma2::a7i =0.3546 [static]

Definition at line 284 of file Formulas.h.

7.7.4.6 constexpr double BGLmodels::calcubtosgamma2::a7r =4.8802 [static]

Definition at line 283 of file Formulas.h.

7.7.4.7 constexpr double BGLmodels::calcubtosgamma2::a87i =-0.0487 [static]

Definition at line 284 of file Formulas.h.

7.7.4.8 constexpr double BGLmodels::calcubtosgamma2::a87r =0.1923 [static]

Definition at line 284 of file Formulas.h.

7.7.4.9 constexpr double BGLmodels::calcubtosgamma2::a88 =0.0197 [static]

Definition at line 283 of file Formulas.h.

7.7.4.10 constexpr double BGLmodels::calcubtosgamma2::a8ei =-0.0661 [static]

Definition at line 284 of file Formulas.h.

7.7.4.11 constexpr double BGLmodels::calcubtosgamma2::a8er =-0.0601 [static]

Definition at line 284 of file Formulas.h.

7.7.4.12 constexpr double BGLmodels::calcubtosgamma2::a8i =-0.0987 [static]

Definition at line 284 of file Formulas.h.

7.7.4.13 constexpr double BGLmodels::calcubtosgamma2::a8r =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:

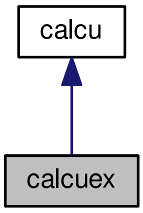
- Formulas.h

7.8 calcuex Class Reference

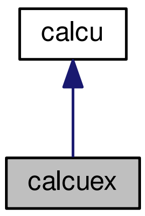
class to do the calculus of a constraint based on a GiNaC symbolic expression

`#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 : calcu(), 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

hipothesis	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;
```

```
            }
           catch(...){
            cout<<"Other exception"<<endl;
            exit(1);
           }
           if(pass) ret=o->loglikelihood(ret);
           else ret=1000;

           return ret;
       }
```

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:

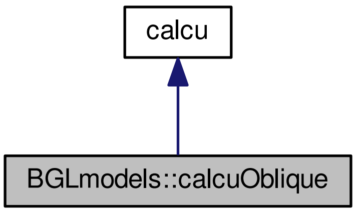
- model.h

7.9 BGLmodels::calcuOblique Class Reference

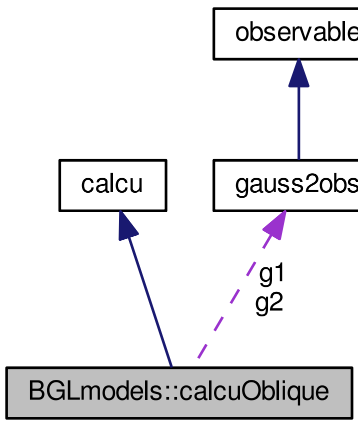
calculus of the constraints coming from the oblique parameters

`#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.

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

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

Definition at line 242 of file Formulas.h.

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

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

Definition at line 252 of file Formulas.h.

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

7.9.3.4 double BGLmodels::calcuOblique::lnxy_xy (double *x*, double *y*) const [inline]

Definition at line 248 of file Formulas.h.

```
                                          {
        if(x==y) return 1/y;
        return log(x/y)/(x-y);
    }
```

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

Parameters

hipothesis	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.

```
                                                    {
            double y=p[1].value;
            double z=p[2].value;
            double w=p[3].value;

            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));
            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;

            double T1=c1*TT-c2*Sparam;
        double T2=c2*TT+c1*Sparam;

         return g1.loglikelihood(T1)+g2.loglikelihood(T2);
    }
```

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

mi	minimum possible value for the parameter
ma	maximum possible value for the parameter
r	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↩ ::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

mi	minimum possible value for the parameter
ma	maximum possible value for the parameter
r	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

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

Definition at line 146 of file model.h.

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

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.

```
                         {
        return min<=value && value<=max;
    }
```

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

r	random number generator

Definition at line 135 of file model.h.

```
                                      {
         //double x=r->Gaus()*step;
         value+=r->Gaus()*step*f;
         }
```

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:

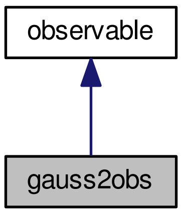
- model.h

7.13 gauss2obs Class Reference

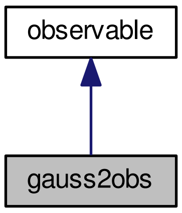
the same as gaussobs but with a different initializer, such that the uncertainty sigma is absolute

```
#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

mean	mean value of the measure
sigma	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

hipothesis	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:

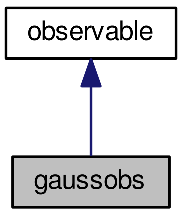
- model.h

7.14 gaussobs Class Reference

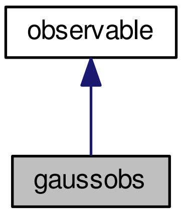
An experimental measure of a parameter which is a mean value and a standard deviation.

```
#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 hipothesis) const
- double error (double hipothesis) 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

mean	mean value of the measure
sigma	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 *hipothesis*) const [inline], [virtual]

Implements observable.

Definition at line 91 of file model.h.

```
00091                                           {
00092         double diff=(hipothesis-m)/s;
00093         return diff;
00094         }
```

7.14.3.2 double gaussobs::loglikelihood (double *hipothesis*) const [inline], [virtual]

Parameters

hipothesis	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.

```
                                                  {
          double diff=(m-hipothesis)/s;
          return diff*diff/2;
          }
```

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.

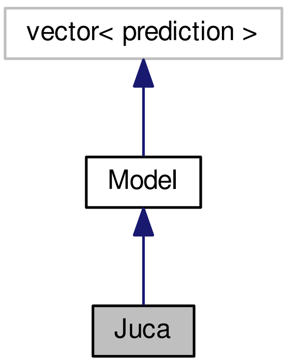
The documentation for this class was generated from the following file:

- model.h

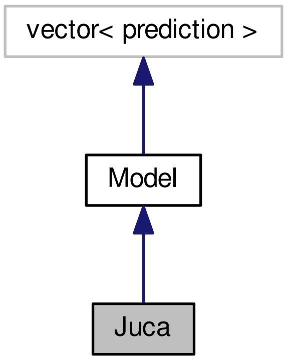
7.15 Juca Class Reference

```
#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
- lst 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
- lst 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 lst Juca::getlist (const parameters & *p*) const [inline], [virtual]

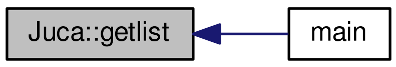
Implements Model.

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])),
      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 }
```

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 lst 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:

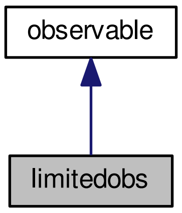
- Juca.h

7.16 limitedobs Class Reference

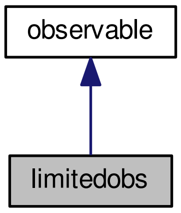
An experimental measure which is an upper limit on a parameter with a given Confidence Level.

```
#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 hipothesis) const
- double error (double hipothesis) 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

limit	upper limit on the parameter
m	minimum possible value of the parameter
p	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) {
             if(cl==0.95) s*=(1.282+sqrt(M_PI_2))/(1.645+sqrt(M_PI_2));
             }
```

7.16.2.2 limitedobs::∼limitedobs ( ) [inline]

Definition at line 63 of file model.h.

```
{}
```

7.16.3 Member Function Documentation

7.16.3.1 double limitedobs::error (double *hipothesis*) const [inline], [virtual]

Implements observable.

Definition at line 69 of file model.h.

```
                                               {
            double diff=(hipothesis-min )/s;
            return diff;
            }
```

7.16.3.2 double limitedobs::loglikelihood (double *hipothesis*) const [inline], [virtual]

Parameters

hipothesis	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.

```
                                                      {
             double diff=(hipothesis-min-sqrt(M_PI_2)*s)/s;
             if(diff<0) diff=0;
             return diff*diff/2;
            }
```

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.

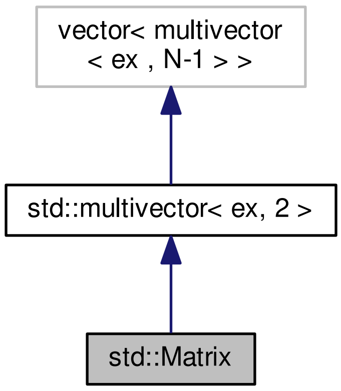
The documentation for this class was generated from the following file:

- model.h

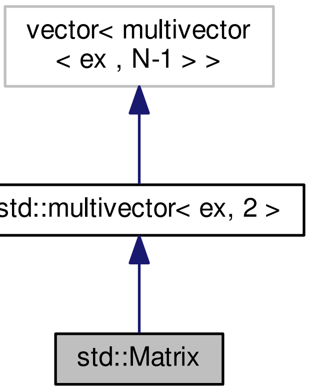
7.17 std::Matrix Class Reference

```
#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.

```
                : multivector< ex,2>(0,3,3){
        Matrix();
        at(0)[0]=m1;
        at(1)[1]=m1;
        at(2)[2]=m1;
        }
```

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.

```
                                        : multivector< ex,2>(0,3,3) {
        Matrix();
        ex c12=cos(t12), c13=cos(t13), c23=cos(t23);
        ex s12=sin(t12), s13=sin(t13), s23=sin(t23);
        ex e13=exp(I*d13);
        ex e13t=ex(1)/e13;

        ex aux[3][3]={
                        {c12*c13,s12*c13,s13*e13t},
                        {-s12*c23-c12*s23*s13*e13,c12*c23-s12*s23*s13*e13,s23*c13},
                        {s12*s23-c12*c23*s13*e13,-c12*s23-s12*c23*s13*e13,c13*c23}
                        };
        for(uint i=0;i<3; i++) at(i).assign(aux[i],aux[i]+3);
        }
```

7.17.3 Member Function Documentation

7.17.3.1 Matrix std::Matrix::conjugate ( ) const [inline]

computes the hermitian conjugate of the matrix

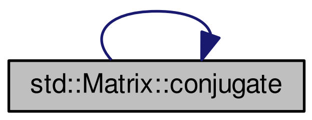
Definition at line 125 of file multivector.h.

References conjugate().

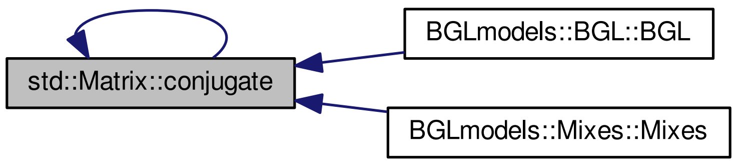
Referenced by BGLmodels::BGL::BGL(), conjugate(), and BGLmodels::Mixes::Mixes().

```
                         {
        Matrix res;
        for(uint i=0;i<3;i++)
                for(uint j=0;j<3;j++)
                        res[i][j]=at(j)[i].conjugate();

        return res;
}
```

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.

```
                {
        return (exp(I*t12)+1/exp(I*t12))/2;
        }
```

7.17.3.3 ex std::Matrix::sn (ex *t12*) [inline]

used in the unitary constructor

Definition at line 121 of file multivector.h.

```
                {
        return -I*(exp(I*t12)-1/exp(I*t12))/2;
        }
```

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

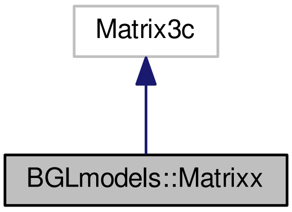
- multivector.h

7.18 BGLmodels::Matrixx Class Reference

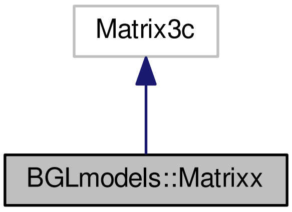
a class to represent the mixing matrices VCKM and VPMNS

```
#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.

```
: Matrix3c(){}
```

7.18.2.2 BGLmodels::Matrixx::Matrixx (const Matrix3c & *a*) [inline]

Definition at line 75 of file Formulas.h.

```
: 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.

```
                                                                                           :
        Matrix3c({
                Vector3c({c12*c13,s12*c13,s13*e13t}),
                Vector3c({-s12*c23-c12*s23*s13*e13,c12*c23-s12*s23*s13*e13,s23*c13}),
                Vector3c({s12*s23-c12*c23*s13*e13,-c12*s23-s12*c23*s13*e13,c13*c23})
                }){}
```

7.18.2.4 BGLmodels::Matrixx::Matrixx (double *t12*, double *t13*, double *t23*, double *d13*) [inline]

Definition at line 86 of file Formulas.h.

```
                                                          :
        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.

```
                            {
        Matrixx res;
        for(uint i=0;i<3;i++)
                for(uint j=0;j<3;j++)
                        res[i][j]=std::conj((*this)[j][i]);
        return res;
    }
```

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.

```
                                {
                const measure & m1=*this;
                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)));
                }
```

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:

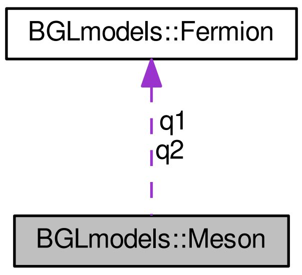
- model.h

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::BG↩ L::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::BG←L::fermiontomesontest(), BGLmodels::BGL::mesondw(), and BGLmodels::BGL::mesondwtest().

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

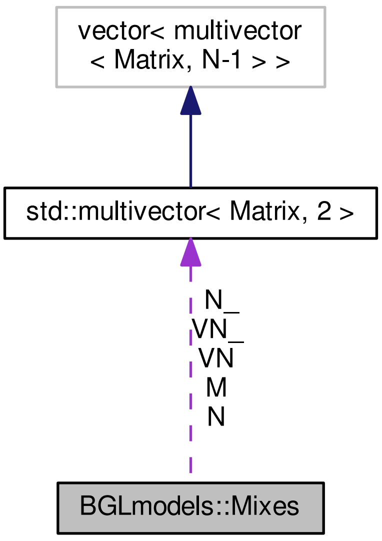
- Formulas.h

7.21 BGLmodels::Mixes Class Reference

definition of the couplings for the different BGL models

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

Collaboration diagram for BGLmodels::Mixes:


vector< multivector < Matrix, N-1 > >
std::multivector< Matrix, 2 >
N_
VN_
VN
M
N
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 & Destructor Documentation

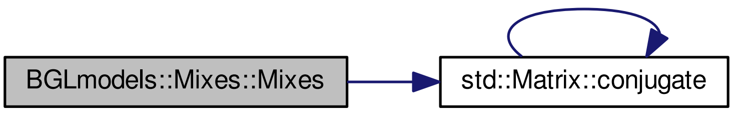
7.21.2.1 BGLmodels::Mixes::Mixes (ex *tanb0*, ex *cp0*, int *genL* = 2, int *genQ* = 2, int *lup* = 0, int *qup* = 0, int *mssm* = 0) [inline]

Definition at line 108 of file Formulas.h.

References std::Matrix::conjugate(), BGLmodels::iDown, BGLmodels::iUp, BGLmodels::tLepton, and BGLmodels::tQuark.

```
00108                                                                               :M(
      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   }
```

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);
```

```

        reps.append(M[1][1][0][0]==5.3e-3);
        reps.append(M[1][1][1][1]==95e-3);
        reps.append(M[1][1][2][2]==4.2);

        vector< Matrix > Vn;
      Vn.push_back(Matrix(33.6*M_PI/180,9.11*M_PI/180,40.4*M_PI/180,M_PI/4).conjugate());
      Vn.push_back(Matrix(13.04*M_PI/180,0.201*M_PI/180,2.38*M_PI/180,1.2));
      for(uint i=0; i<2;i++)
              for(uint j=0; j<3;j++)
                      for(uint k=0; k<3;k++)
                              reps.append(V[i][j][k]==Vn[i][j][k]);
        }
```

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:

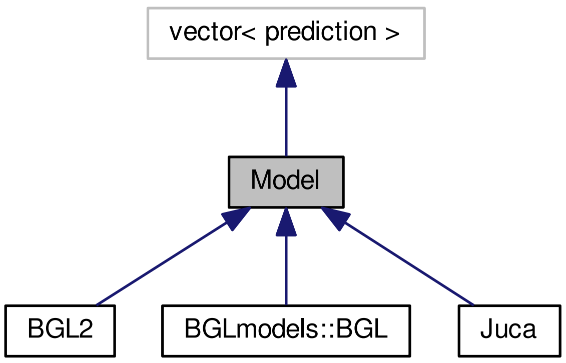
- Formulas.h

7.22 Model Class Reference

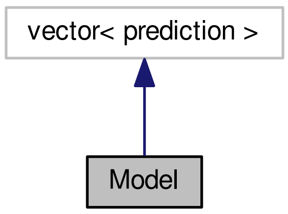
Abstract class for a model.

`#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

p	vector with the values of the free parameters

Definition at line 374 of file model.h.

```
                                                                        {
           if(veto(p,max) && check) return 0;
           double total=loglike(p,0);
           if(total<-1000) return 0;
           return exp(total);
           }
```

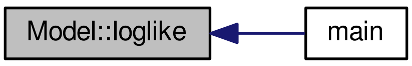
7.22.3.4 double Model::loglike (const parameters & *p*, bool *check* = 1, int *max* = 0) const [inline]

Definition at line 381 of file model.h.

Referenced by main().

```
                                                                      {
           if(veto(p,max) && check) return -1000;
           parameters pp(getlist(p));

           double total=0;
           int n=0;
           for(const_iterator i=begin();i!=end();i++) {
           try{
                   n++;
                   total+=i->loglikelihood(pp);
                                   }
       catch(exception e){
             cout<<n<<e.what()<<endl;
             exit(1);
            }
           //catch(...){
                   //cout<<"DD "<<n<<endl;
                   //exit(1);
                   //}
 }

           return -total;
           }
```

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]

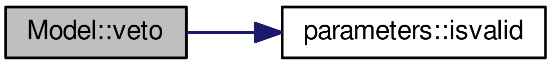
Reimplemented in BGLmodels::BGL.

Definition at line 368 of file model.h.

References parameters::isvalid().

```
{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:

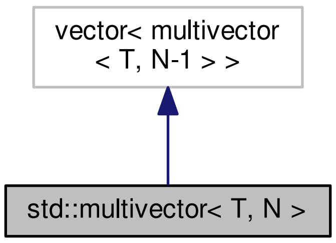
- model.h

7.23 std::multivector$<$ T, N $>$ Class Template Reference

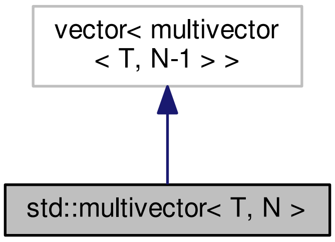
A vector of vectors of vectors of... (N times) of class T objects.

```
#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.

```
: 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.

```
: 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

value	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.

```
                        {
        va_list listPointer;
        va_start(listPointer,value);
        int n=va_arg(listPointer,int);
        v::insert(v::begin(),n,multivector<T,N-1>(value,listPointer));
        va_end(listPointer);
        }
```

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.

```
                    {
                            int n=va_arg(listPointer,int);
                            v::insert(v::begin(),n,multivector<T,N-1>(value,listPointer));
                            }
```

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

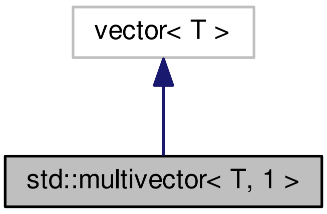
- multivector.h

7.24 std::multivector< T, 1 > Class Template Reference

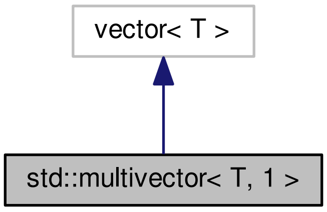
Specialization template class of multivector<T,N> for N=1.

`#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

value	the value with which every objects are initialized
x	number of dimensions of the vector

Definition at line 57 of file multivector.h.

```
: 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.

```
                                                :
                v(va_arg(listPointer,int), value){}
```

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

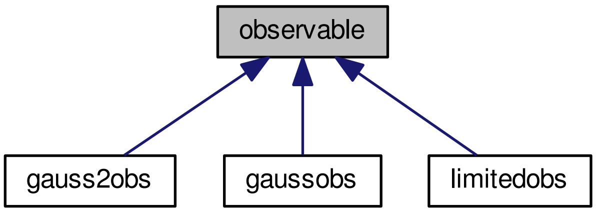
- multivector.h

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

hipothesis	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:

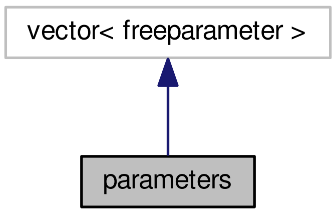
- model.h

7.26 parameters Class Reference

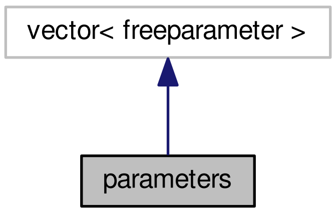
vector of parameters

`#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

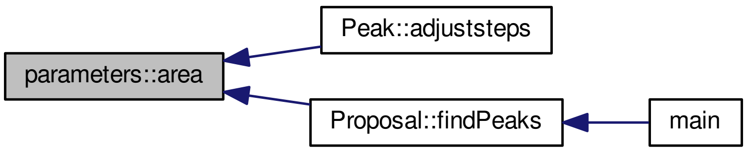
7.26.2.1 double parameters::area ( ) const [inline]

Definition at line 215 of file model.h.

Referenced by Peak::adjuststeps(), and Proposal::findPeaks().

```
                        {
        float a=1;
        for(const_iterator i=begin();i!=end();i++){
                a*=i->step;
                }
        return a;
        }
```

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.

```
                                   {
        double total=0;
        for(uint i=0;i<size();i++){
                total+=at(i).dist(p[i].value);
                }
        return sqrt(total/size());
}
```

7.26.2.3 double parameters::gausslikelihood (const parameters & *p2*) const [inline]

Definition at line 223 of file model.h.

```
                                                {
        double l=1;
        for(uint i=0;i<size();i++){
                l*=TMath::Gaus((p2[i].value-at(i).value)/at(i).step);
        }
        return l;
        }
```

7.26.2.4 bool parameters::isvalid ( ) const [inline]

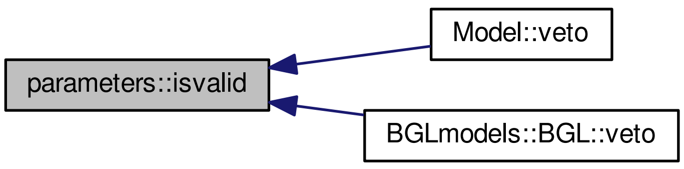
checks if all the values are between their minimums and maximums

Definition at line 193 of file model.h.

Referenced by Model::veto(), and BGLmodels::BGL::veto().

```
                    {
        for(const_iterator i=begin();i!=end();i++){
                if(!i->isvalid()) return 0;
                }
        return 1;
        }
```

Here is the caller graph for this function:



7.26.2.5 void parameters::next (TRandom3 ∗ *r*, double *f* = `1`) `[inline]`

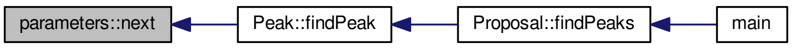
changes randomly the value of the parameters

Definition at line 182 of file model.h.

Referenced by Peak::findPeak().

```
                                        {
            for(iterator i=begin();i!=end();i++){
                    i->next(r,f);
                    }

            //for(uint i=0;i<discrete.size();i++){
                    //discrete[i].next(r);
                    //}
            }
```

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.

```
                                        {

            for(uint i=0;i<size();i++){
                    at(i).value=p[i].value;
                    }
            }
```

7.26.3 Member Data Documentation

7.26.3.1 lst 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(), calcuba↩ ::operator()(), and BGLmodels::calcuBmumu::operator()().

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

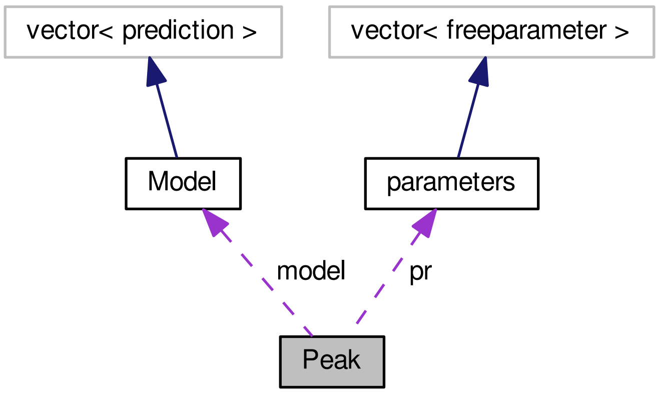
- model.h

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.

```
                                        : model(m), pr(m->
generateparameters(maxx)), max(maxx){
                lmax=model->likelihood(pr);
                }
```

7.27.3 Member Function Documentation

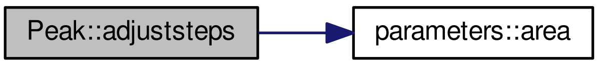
7.27.3.1 bool Peak::adjuststeps ( ) [inline]

Definition at line 136 of file MCMC.h.

References parameters::area().

```
                          {
                 parameters p1(pr);
                 for(uint i=0;i<pr.size(); i++){
                           double s=p1[i].step;
                           p1[i].value+=s;
                           double x=(lmax-model->likelihood(p1))*2/
lmax/s/s;
                           double x0=std::pow(2/(pr[i].max-pr[i].min),2);
                           if(x<x0) return 0;
                          // cout<<"X "<<x<<endl;
                           pr[i].step=1/sqrt(x);
                           p1[i].value-=s;
                           }
           area=pr.area();
           larea=area*lmax;
           return 1;
        }
```

Here is the call graph for this function:



7.27.3.2 void Peak::findPeak ( ) [inline]

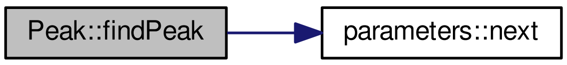
Definition at line 17 of file MCMC.h.

References parameters::next().

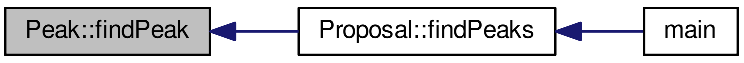
Referenced by Proposal::findPeaks().

```
                        {
            llmax=model->loglike(pr,1,max);
            area=1;
           uint fixed=1e2;
           uint f=fixed;
           double d=1;
           //cout<<"f "<<f<<"llmax "<<llmax<<endl;
                 for(uint i=1e5;i;i--){
                         parameters p1(pr);
                         p1.next(model->r,d);

                         double l1=llmax;
                         if(!model->veto(p1,max)){l1=model->
      loglike(p1,1,max);
                                 }
                         if(l1>llmax){pr=p1;  llmax=l1; f=fixed;}
                         else {f--; if(!f) {d/=100; f=fixed; if(d<1e-2) break;}}
                     }
            cout<<"d "<<d<<"llmax "<<llmax<<endl;
            if(llmax<-1000) lmax=0;
            else{lmax=exp(llmax);
                 area=pr.area();
             larea=area*lmax;
             }

        }
```

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::llmax

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:

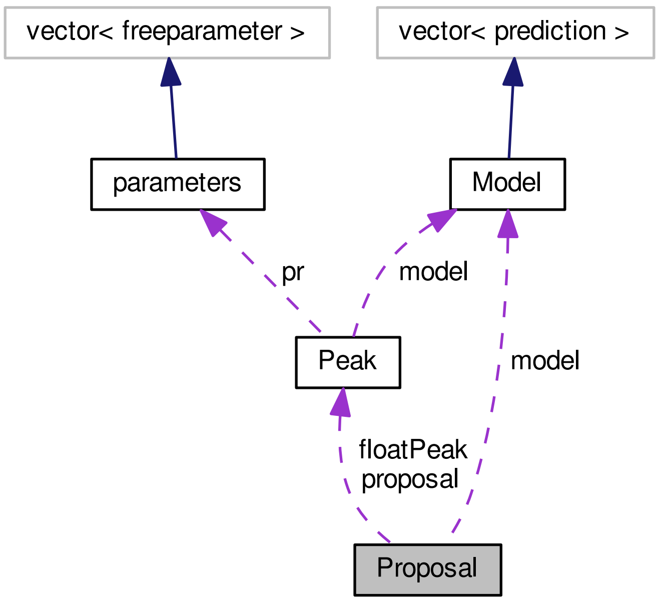
- model.h

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]

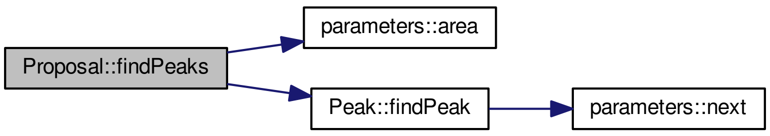
Definition at line 167 of file MCMC.h.

References parameters::area(), Peak::findPeak(), Peak::llmax, Peak::lmax, and Peak::pr.

Referenced by main().

```
                                        {
            //float pmin=-100, pmax=100, s=0.1;
        //floatPeak.s=s;
        //floatPeak.lmax=0;
        //int imax=-1;
        floatPeak=Peak(model,max);
        floatPeak.lmax=0;
        floatPeak.llmax=-1000;
       cout<<"started"<<endl;
            //for(uint i=5e1;i;i--){
            for(uint i=ns;i;i--){
                    Peak pp(model,max);
                    pp.findPeak();
                    if(pp.llmax>-15){
                    //for(uint j=0; j< pp.pr.size();j++){
                    //cout<<j<<" "<<pp.pr[j].value<<endl;
                    //}
                    //lst l=model->getlist(pp.pr);
                    //for(uint j=0; j< model->size();j++){
                    //      double mean=model->at(j).calculate(l);
                    //cout<<j<<" "<<mean<<" "<<sqrt(2*model->at(j).o->loglikelihood(mean))<<endl;
                    //}
                    }
                    if(pp.lmax>floatPeak.lmax){
                            cout<<i<<" "<<pp.lmax<<endl;
                            floatPeak.lmax=pp.lmax;
                            floatPeak.pr=pp.pr;
                            }
            }
            floatPeak.area=floatPeak.pr.area();

 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

Proposal::findPeaks
main

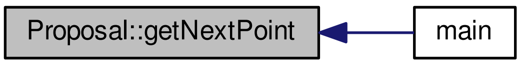
7.30.3.2 void Proposal::getNextPoint ( ) [inline]

Definition at line 210 of file MCMC.h.

Referenced by main().

```
                    {
        getProposal();
        double ll=0;
        ll=model->likelihood(proposal.pr);
        if(model->r->Rndm()<=ll/floatPeak.lmax){
                floatPeak.lmax=ll;
                floatPeak.pr.setvalues(proposal.pr);
        }
}
```

Here is the caller graph for this function:



7.30.3.3 void Proposal::getProposal ( ) [inline]

Definition at line 200 of file MCMC.h.

```
                   {
        if(model->r->Rndm()<=0.9) {
                proposal.pr=floatPeak.pr;
                proposal.pr.next(model->r);
                return;
                }

        proposal.pr=model->generateparameters();
}
```

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< Peak > Proposal::vPeak

Definition at line 222 of file MCMC.h.

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

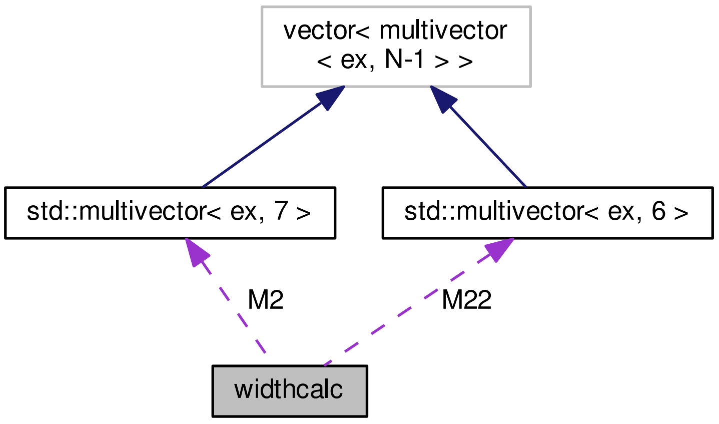
- MCMC.h

7.31 widthcalc Class Reference

this class calculates decay widths of one lepton to 3 leptons

```
#include <widthcalc.h>
```

Collaboration diagram for widthcalc:


vector< multivector < ex, N-1 > >
std::multivector< ex, 7 >
std::multivector< ex, 6 >
M2
M22
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), 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
```

```

        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();
        ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)-mq4*dirac_ONE();

        ex s4=m2q1+m2q2+m2q3+m2q4-s2-s3;
    scalar_products sp;
    sp.add(q2, q3, (s4-m2q2-m2q3)/2);
    sp.add(q4, q3, (s2-m2q4-m2q3)/2);
    sp.add(q2, q4, (s3-m2q2-m2q4)/2);

    sp.add(q2, q2, m2q2);
    sp.add(q3, q3, m2q3);
    sp.add(q4, q4, m2q4);

    sp.add(h1,h1,-1);
    sp.add(h2,h2,-1);
    sp.add(h3,h3,-1);
    sp.add(h4,h4,-1);

    sp.add(h2,q2,0);
    sp.add(h3,q3,0);
    sp.add(h4,q4,0);

    multivector<ex,3> v(0,2,2,2);
        v[0][0][0]=dirac_gammaL(); v[0][0][1]=dirac_gammaR();
        v[0][1][0]=dirac_gammaR(); v[0][1][1]=dirac_gammaL();
        v[1][0][0]=dirac_gamma(jmu)*dirac_gammaL(); v[1][0][1]=dirac_gamma(jmu)*dirac_gammaL();
        v[1][1][0]=dirac_gamma(jmu)*dirac_gammaR(); v[1][1][1]=dirac_gamma(jmu)*dirac_gammaR();

        multivector<ex,7> traces(0,2,2,2,2,2,2,2);
        for(uint i=0;i<2;i++)
          for(uint j=0;j<2;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 vik=v[i][k][0];
                                ex vim=v[i][m][0].subs(mu==nu);
                                ex vjl=v[j][l][1].subs(mu==alpha);
                                ex vjn=v[j][n][1].subs(mu==beta);

                                traces[i][j][k][l][m][n][0]=dirac_trace(vq2*vik*vq1*vjl)*dirac_trace(vq3*vim*vq4*vjn);
                                traces[i][j][k][l][m][n][1]=-dirac_trace(vq2*vik*vq1*vjl*vq3*vim*vq4*vjn);
                        }

        vector<ex> prop(2,0);
        prop[0]=1;
        prop[1]=lorentz_g(imu,inu);

        multivector<ex,2> prop2(0,2,2);
        for(uint i=0;i<2;i++)
          for(uint j=0;j<2;j++){
                                prop2[i][j]=prop[i]*prop[j].subs(lst(mu==alpha,nu==beta));
                        }

        //ofstream f("M2.dat");
        for(uint i=0;i<2;i++)
          for(uint j=0;j<2;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 o=0;o<2;o++)
                                        {
                                                        //cout<<i<<" "<<j<<" "<<k<<" "<<l<<" "<<m<<endl;
                                                        M2[i][j][k][l][m][n][o]=(traces[i][j][k][l][m][n][o]*prop2[i][j]).simplify_indexed(sp);
                                                        //cout<<M2[i][j][k][l][m][n][o]<<endl<<endl;
                                                        //f<<M2[i][j][k][l][m][n][o]<<endl;
                                                }
}
```

7.31.3.2 void widthcalc::genM22 ( ) [inline]

Definition at line 34 of file widthcalc.h.

```
00034             {
00035         cout<<"Generating M22.dat"<<endl;
```

```

    realsymbol mu("mu"), nu("nu"), alpha("alpha"), beta("beta"), rho("rho"), zeta("zeta");
    realsymbol q1("q1"), q2("q2"), q3("q3"), q4("q4");
    realsymbol h1("h1"), h2("h2"), h3("h3"), h4("h4");

    varidx imu(mu,4,0), inu(nu,4,0), irho(rho,4,0);
    varidx ialpha(alpha,4,0), ibeta(beta,4,0), izeta(zeta,4,0);

    varidx jmu(mu,4,1), jnu(nu,4,1), jrho(rho,4,1);

    ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;

    ex q2mu=indexed(q2,imu);
    ex q3mu=indexed(q3,imu);
    ex q4mu=indexed(q4,imu);
    ex q1mu=q2mu+q3mu+q4mu;

    ex s4=m2q1+m2q2+m2q3+m2q4-s2-s3;

    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();
    ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)-mq4*dirac_ONE();

  scalar_products sp;
  //sp.add(q2, q3, (s4-m2q2-m2q3)/2);
  sp.add(q4, q3, (s2-m2q4-m2q3)/2);
  //sp.add(q2, q4, (s3-m2q2-m2q4)/2);

  //sp.add(q2, q2, m2q2);
  sp.add(q3, q3, m2q3);
  sp.add(q4, q4, m2q4);

  //sp.add(h1,h1,-1);
  //sp.add(h2,h2,-1);
  //sp.add(h3,h3,-1);
  //sp.add(h4,h4,-1);
  //sp.add(h1,h1,-1);

  multivector<ex,3> v(0,2,2,2);
    v[0][0][0]=dirac_gammaL(); v[0][0][1]=dirac_gammaR();
    v[0][1][0]=dirac_gammaR(); v[0][1][1]=dirac_gammaL();
    v[1][0][0]=dirac_gammaL(); v[1][0][1]=dirac_gammaL();
    v[1][1][0]=dirac_gammaR(); v[1][1][1]=dirac_gammaR();

    for(uint i=0;i<2;i++)
     for(uint j=0;j<2;j++){
        v[0][i][j]=-v[0][i][j]*s2/(mq1+mq2);
        v[1][i][j]=(dirac_slash(q3,4)+dirac_slash(q4,4))*v[1][i][j];
    }

    //vector<ex> prop(2,0);
    //prop[0]=-s2/(mq1+mq2);
    //prop[1]=indexed(q3,imu.toggle_variance())+indexed(q4,imu.toggle_variance());
    /*
    multivector<ex,2> prop2(0,2,2);
    for(uint i=0;i<2;i++)
     for(uint j=0;j<2;j++){
                prop2[i][j]=prop[i]*prop[j].subs(mu==nu);
            }
    */

    //ofstream f("M22.dat");
    for(uint i=0;i<2;i++)
     for(uint j=0;j<2;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++){
                //cout<<i<<" "<<j<<" "<<k<<" "<<l<<" "<<m<<" "<<n<<endl;
                //cout<<dirac_trace(vq3*v[i][m][1]*vq4*v[j][n][0].subs(mu==nu))<<endl;

                ex tmp=dirac_trace(vq3*v[i][m][0]*vq4*v[j][n][1])*int(pow(-1.0,double(k+1+l)));
                M22[i][j][k][l][m][n]=tmp.simplify_indexed(sp);
                //cout<<M22[i][j][k][l][m][n]<<endl<<endl;
                //f<<M22[i][j][k][l][m][n]<<endl;
            }
}
```

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         ex q10=(s2+mq1*mq1-mq2*mq2)/(2*sqrt(s2)), lq1l=sqrt(q10*q10-
      mq1*mq1);
00208     ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-
      mq3*mq3);
00209     ex q20=(mq1*mq1+mq2*mq2-s2)/(2*mq1), lq2l=sqrt(q20*q20-mq2*mq2);
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, mq1*mq1+mq3*mq3-2*q10*q30-2*lq1l*
      lq3l, mq1*mq1+mq3*mq3-2*q10*q30+2*lq1l*lq3l, integrand)\
00225                                                                                                          .
      eval_integ()/lq1l/sqrt(s2)*lq2l/mq1/mq1);
00226                                         double mm2=m2, mm3=m3;
00227                                         if(l) {mm2=m3; mm3=m2;}
00228                                         double result=ex_to<numeric>(integral(s2,std::pow(mm3+m4,2),
      std::pow(m1-mm2,2),integrand.subs(lst(mq1 == m1, mq2 == mm2, mq3 == mm3, mq4 == 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 }
```

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         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         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+mq1*mq1-mq2*mq2)/(2*sqrt(s2)), lq1l=sqrt(q10*q10-
      mq1*mq1);
00308     ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-
      mq3*mq3);
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                                         //
      cout<<collect_common_factors(expand(a[j][0][l][n].conjugate()))<<endl;
00322                                         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);
00323                                         ex result=integrand.subs(lst(sqrt(s2) == mm,
      s2==mm*mm, mq1 == 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 }
```

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)), lq1l=(m1*m1-s2)/(2*sqrt(s2));
00240     ex q30=sqrt(s2)/2, lq3l=q30;
00241     ex q20=(m1*m1-s2)/(2*m1), lq2l=q20;
```

```

    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<<endl;
                                        ex integrand=M2[op[i]][op[j]][k][l][m][n][(r+s)%2].subs(lst(mq1 == m1, mq2 == 0, mq3 == 0, mq4 == 0));
                                        integrand=expand(integral(s3, m1*m1-2*q10*q30-2*lq1l*lq3l, m1*m1-2*q10*q30+2*lq1l*lq3l, integrand)\
                                        .eval_integ()/lq1l/sqrt(s2)*lq2l/m1/m1);
                                        //integrand=integrand.subs(lst(mq1 == m1, mq2 == 0, mq3 == 0, mq4 == 0));
                                        //integrand=integrand.subs(pow(m1,2)/4-s2/2+pow(s2/m1,2)/4==pow((m1-s2/m1)/2,2));


                                        double mm2=0, mm3=0, m4=0;
                                        //if(l) {mm2=m3; mm3=m2;}
                                        ex result=integral(s2,std::pow(mm3+m4,2),pow(m1-mm2,2),integrand).eval_integ()/pow(Pi,3)/512;

                                    ex partial=result*coup/(pow(mass[i],2)*pow(mass[j],2));
                                    total=total+partial;
                        }
                        }
        return total;
}
```

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

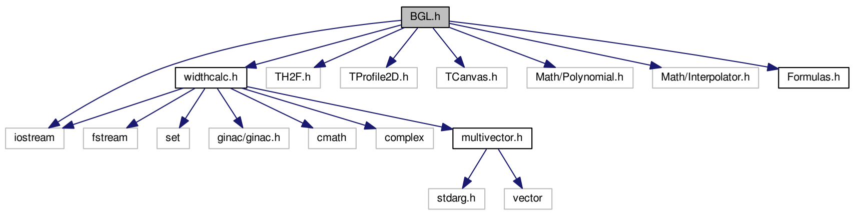
Chapter 8

File Documentation

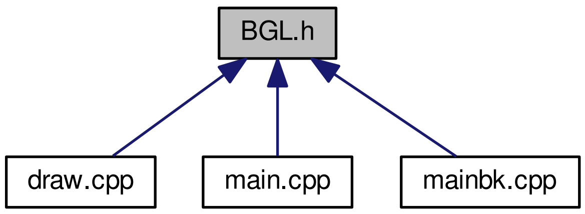
8.1 BGL.h File Reference

```
#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

```
#ifndef BGL_H
#define BGL_H

#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"


namespace BGLmodels{

/**
* @brief Gauge boson
*/
class Boson {
public:

        Boson(): C(Matrix(),2,2,2,2){}

        ex couplingL(const Fermion& f2,const Fermion& f1) const {
                bool quiralfilter=0;
                if(f1.type!=f2.type) return 0;
                if(s==sScalar){
                        if(f1.helicity!=hRight && f2.helicity!=
hLeft) quiralfilter=1;
                        }
                else if(s==sVector){
                        if(f1.helicity!=hRight && f2.helicity!=
hRight) quiralfilter=1;
                        }

                if(quiralfilter) return C[f2.type][f2.isospin][f1.
isospin][hLeft][f2.flavour][f1.flavour];
                return 0;
                }
        ex couplingR(const Fermion& f2,const Fermion& f1) const {
                bool quiralfilter=0;
                if(f1.type!=f2.type) return 0;
                if(s==sScalar){
                        if(f2.helicity!=hRight && f1.helicity!=
hLeft) quiralfilter=1;
                        }
                else if(s==sVector){
                        if(f1.helicity!=hLeft && f2.helicity!=
hLeft) quiralfilter=1;
                        }
                if(quiralfilter) return C[f2.type][f2.isospin][f1.
isospin][hRight][f2.flavour][f1.flavour];
                return 0;
                }
        ex couplingdaggerL(const Fermion& f2,const Fermion& f1) const {
                if(s==sScalar) return couplingR(f1,f2).conjugate();
                return couplingL(f1,f2).conjugate();
                }
        ex couplingdaggerR(const Fermion& f2,const Fermion& f1) const {
                if(s==sScalar) return couplingL(f1,f2).conjugate();
                return couplingR(f1,f2).conjugate();
                }
        ex coupling(const Fermion& f2,const Fermion& f1, ex mu){
                if(s==sScalar) return couplingL(f2,f1)*dirac_gammaL()+couplingR(f2,f1)*dirac_gammaR();
                else return couplingL(f2,f1)*dirac_gammaL()+couplingR(f2,f1)*dirac_gammaR();
                }
        void reset(){
                C=multivector<Matrix,4>(Matrix(),2,2,2,2);
                }
    BSpin s;
    ex mass;
    multivector<Matrix,4> C;
};



//Boson::Type Boson::dagger[2][2]={Boson::scalarright,Boson::scalarleft,Boson::vectorleft,Boson::vectorright};

/**
* @brief Implementation of the BGL model
*/
class BGL: public Model{
public:

BGL(int genL=2,int genQ=2, int lup=0, int qup=0, int mssm=0):
           planck(6.58211928e-25),
           GF("G_F"),
           MZ("M_Z"),
           MW("M_W"),
           Mpip("Mpip",0.1396,"M_{\\pi^+}",domain::real),
           Mpi0("Mpi0",0.1349766,"M_{\\pi^0}",domain::real),
           MBp("MBp",5.279,"M_{B^+}",domain::real),
           MB0("MB0",5.2795,"M_{B^0}",domain::real),
           MBs0("MBs0",5.3663,"M_{B_s^0}",domain::real),
           MKp("MKp",0.493677,"MKp",domain::real),
           MK0("MK0",0.497614,"MK0",domain::real),
           MDp("MDp",1.86957,"MDp",domain::real),
           MD0("MD0",1.86480,"MD0",domain::real),
           MDsp("MDsp",1.96845,"MDsp",domain::real),
           MDs0("MDs0",0),
           Fpi("Fpi",0.132,"Fpi",domain::real),
           FB("FB",0.189,"FB",domain::real),
           FBs("FBs",0.225,"FBs",domain::real),
           FK("FK",0.159,"FK",domain::real),
           FD("FD",0.208,"FD",domain::real),
           FDs("FDs",0.248,"FDs",domain::real),
           //alpha(7.297352e-3*4*M_PI),
           cos2(pow(MW/MZ,2)),
           g(sqrt(GF*8/sqrt(ex(2)))*MW),
           //g(sqrt(4*Pi*alpha/(1-cos2))),
           tanb("tg\\beta"),
           cp("cp"),
           McH("M_{H^+}"),
       MR("M_{R}"),
       MI("M_{I}"),
           mixes(tanb,cp, genL,genQ, lup, qup, mssm),
           mu("\\mu"),
           BGLtype(4,0),
           mmmax(1000),
           stepsize(1e-2)
           //muwidth(planck/2.197034e-6)
           {
        alpha=pow(g,2)*(1-cos2)/(4*Pi);
        replacements.append(GF==1.166371e-5);
        replacements.append(MZ==M_MZ);
        replacements.append(MW==M_MW);

    mixes.appendtolst(replacements);

    replacements.append(Pi==M_PI);
    replacements.append(sqrt(ex(2))==sqrt(2));

        //cout<<pow(sqrt(2)/8*pow(g/MW,2),2)<<endl;
        //cout<<pow(1.166,2)<<endl;

        Boson boson;

        realsymbol q3("q3");
        ex vq3=dirac_slash(q3,4);
        varidx jmu(mu,4,1);

        for(uint i=0;i<2;i++)
                for(uint j=0;j<3;j++)
```

```
                        for(uint k=0;k<3;k++){
                                conjtoabs.append(conjugate(mixes.V[i][j][k])==pow(abs(mixes.V[i][j][k]),2)/mixes.V[i][j][k]);
                                }
        /*
        //Gamma boson
        boson.mass=0;
        boson.s=Boson::vector;

        boson.coupsL[0][0]=Matrix(g*sqrt(1-cos2)*0);
        boson.coupsL[1][1]=Matrix(g*sqrt(1-cos2)*(-1));
        boson.coupsL[2][2]=Matrix(g*sqrt(1-cos2)*ex(2)/3);
        boson.coupsL[3][3]=Matrix(g*sqrt(1-cos2)*ex(-1)/3);

        boson.coupsR[0][0]=Matrix(g*sqrt(1-cos2)*0);
        boson.coupsR[1][1]=Matrix(g*sqrt(1-cos2)*(-1));
        boson.coupsR[2][2]=Matrix(g*sqrt(1-cos2)*ex(2)/3);
        boson.coupsR[3][3]=Matrix(g*sqrt(1-cos2)*ex(-1)/3);

        bosons.push_back(boson);
        boson.reset();
        */
        //W+ boson
        boson.mass=MW;
        boson.s=sVector;

        for(uint t=tLepton;t<=tQuark;t++) boson.C[t][iUp][iDown][hLeft]=mixes.V[t]*Matrix(g/sqrt(ex(2)));
        Boson wboson=boson;
        bosons.push_back(boson);
        boson.reset();

        //H+ boson
        boson.mass=McH;
        boson.s=sScalar;

        for(uint t=tLepton;t<=tQuark;t++)
        for(uint i=iUp;i<=iDown;i++) boson.C[t][iUp][iDown][i]=mixes.VN[t][i]*Matrix(g/MW/sqrt(ex(2)));
        Boson chiggs=boson;
        bosons.push_back(boson);
        boson.reset();

        for(int b=bosons.size()-1;b>=0;b--){
                boson.mass=bosons[b].mass;
                boson.s=bosons[b].s;
                if(boson.s==sVector)
                        for(uint t=tLepton;t<=tQuark;t++)
                        for(uint i=iUp;i<=iDown;i++)
                        for(uint j=iUp;j<=iDown;j++)
                        for(uint h=hLeft;h<=hRight;h++){
                                boson.C[t][i][j][h]=bosons[b].C[t][j][i][h].conjugate();
                                }
                else for(uint t=tLepton;t<=tQuark;t++)
                        for(uint i=iUp;i<=iDown;i++)
                        for(uint j=iUp;j<=iDown;j++)
                        for(uint h=hLeft;h<=hRight;h++){
                                boson.C[t][i][j][hLeft]=bosons[b].C[t][j][i][hRight].conjugate();
                                boson.C[t][i][j][hRight]=bosons[b].C[t][j][i][hLeft].conjugate();
                                }
                bosons.push_back(boson);
                boson.reset();
                }

        //(R+iI)/sqrt(2) boson
        boson.mass=MR;
        boson.s=sScalar;

        for(uint t=tLepton;t<=tQuark;t++){
                        boson.C[t][iDown][iDown][hRight]=mixes.N[t][iDown]*Matrix(g/MW/ex(2));
                        boson.C[t][iUp][iUp][hLeft]=mixes.N[t][iUp].conjugate()*Matrix(g/MW/ex(2));
                        boson.C[t][iDown][iDown][hLeft]=mixes.N[t][iDown].conjugate()*Matrix(g/MW/ex(2));
                        boson.C[t][iUp][iUp][hRight]=mixes.N[t][iUp]*Matrix(g/MW/ex(2));
                }
        bosons.push_back(boson);
        boson.reset();

        //(R+iI)/sqrt(2) boson
        boson.mass=MI;
        boson.s=sScalar;

        for(uint t=tLepton;t<=tQuark;t++){
                        boson.C[t][iDown][iDown][hRight]=mixes.N[t][iDown]*Matrix(I*g/MW/ex(2));
                        boson.C[t][iUp][iUp][hLeft]=mixes.N[t][iUp].conjugate()*Matrix(I*g/MW/ex(2));
                        boson.C[t][iDown][iDown][hLeft]=mixes.N[t][iDown].conjugate()*Matrix(-I*g/MW/ex(2));
                        boson.C[t][iUp][iUp][hRight]=mixes.N[t][iUp]*Matrix(-I*g/MW/ex(2));
                }
        bosons.push_back(boson);
        boson.reset();

        Fermion electron(tLepton,iDown,fElectron);
        Fermion electronR(tLepton,iDown,fElectron,cParticle,hRight);

        Fermion muon(tLepton,iDown,fMuon);
        Fermion muonR(tLepton,iDown,fMuon,cParticle,hRight);

        Fermion tau(tLepton,iDown,fTau);
        Fermion tauR(tLepton,iDown,fTau,cParticle,hRight);
        Fermion neutrino(tLepton,iUp);
        Fermion neutrinotau(tLepton,iUp,fTau);
        Fermion neutrinomuon(tLepton,iUp,fMuon);
        Fermion neutrinoe(tLepton,iUp,fElectron);

        Fermion up(tQuark,iUp,fElectron);
        Fermion down(tQuark,iDown,fElectron);
        Fermion bottom(tQuark,iDown,fTau);
        Fermion strange(tQuark,iDown,fMuon);
        Fermion charm(tQuark,iUp,fMuon);
        Fermion top(tQuark,iUp,fTau);

        Meson Pi0d(down,down,Mpi0,Fpi);
        Meson Pi0u(down,down,Mpi0,Fpi);
        Meson Pip(up,down,Mpip,Fpi);
        Meson Pim(down,up,Mpip,Fpi);

        Meson K0(down,strange,MK0,FK);
        Meson Kp(up,strange,MKp,FK);

        Meson D0(charm,up,MD0,FD);
        Meson Dp(charm,down,MDp,FD);
        Meson Dsp(charm,strange,MDsp,FDs);

        Meson B0(down,bottom,MB0,FB);
        Meson Bp(up,bottom,MBp,FB);
        Meson Bs0(strange,bottom,MBs0,FBs);

        lst sb;
        //sb.append(mixes.M[tQuark][iUp][0][0]==0);
        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));
        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));

        //cout<<"Btaunu "<<collect_common_factors(expand(Btaunu.subs(sb).subs(conjtoabs)))<<endl;

        cout<<latex;

        ex mutoenunu=decaywidth(muon,neutrino,electron,neutrino);

        //cout<<"mutoenunu "<<mutoenunu<<endl;
        //add("mutoenunu",decaywidth(muon,neutrino,electron,neutrino),new gaussobs(planck/2.197034e-6,0.03));

        add("muRtoeRnunu",gRR2(muon,electron),new limitedobs(std::pow(0.035,2),0.95));

        //add("tautoenunu",decaywidth(tau,neutrino,electron,neutrino),new gaussobs(planck/290.6e-15*0.1782,0.03));
        add("tauRtoeRnunu",gRR2(tau,electron),new limitedobs(std::pow(0.7,2),0.95));

        //add("tautomununu",decaywidth(tau,neutrino,muon,neutrino),new gaussobs(planck/290.6e-15*0.1739,0.03));
        add("tauRtomuRnunu",gRR2(tau,muon),new limitedobs(std::pow(0.72,2),0.95));

        add("tautomu_tautoe",tautomu_tautoe(),new gaussobs(1.0018,0.0014/1.0018)); //PROBLEM!!!
        cout<<"tautomu_tautoe: "<<1/1.0018<< "ERROR: "<<0.0014/1.0018<<endl;
        cout<<"ratio1 "<<tautomu_tautoe().subs(replacements)<<endl;
        cout<<"ratio2 "<<(decaywidth(tau,neutrino,muon,neutrino,sVector)/decaywidth(tau,neutrino,electron,neutrino,sVector)).subs(replacements)<<endl;
        //muto3e
        ex mu3e=decaywidth(muon,electron,electron,electron);
        add("muto3e", mu3e,new limitedobs(planck/2.197034e-6*1e-12));
        cout<<"mu3e "<<decaywidthtest2(muon)<<endl;

        //tauto3e
        add("tauto3e", decaywidth(tau,electron,electron,electron),new limitedobs(planck/290.6e-15*2.7e-8));
        //tauto2e1mu+
        add("tauto2e1mup", decaywidth(tau,electron,electron,muon), new limitedobs(planck/290.6e-15*1.5e-8));
        //tauto2e1mu-
        add("tauto2e1mu", decaywidth(tau,electron,muon,electron), new limitedobs(planck/290.6e-15*1.8e-8));
        //tauto2mu1e+

        add("tauto2mu1ep", decaywidth(tau,muon,muon,electron), new limitedobs(planck/290.6e-15*1.7e-8));
        cout<<"tauto2mu1ep "<<decaywidthtest2(tau)<<endl;
        //tauto2mu1e-
        add("tauto2mu1ep", decaywidth(tau,muon,electron,muon), new limitedobs(planck/290.6e-15*2.7e-8));
        cout<<"tauto2mu1e "<<decaywidthtest2(tau)<<endl;

        //tauto3mu
        add("tauto3mu", decaywidth(tau,muon,muon,muon),new limitedobs(planck/290.6e-15*2.1e-8));
        //cout<<"tauto3mu "<<collect_common_factors(expand(decaywidth(tau,muon,muon,muon)))<<endl;


        ex piratio=1.2352e-4/(mesondw(Pip,neutrino,electron,sVector)/mesondw(Pip,neutrino,muon,sVector));
        ex picorrection=piratio.subs(replacements);
        ex pierror=picorrection*0.0001/1.2352;
        cout<<"PiRatio "<<picorrection-1<<" +/- "<<pierror<<endl;
        piratio*=mesondw(Pip,neutrino,electron)/mesondw(Pip,neutrino,muon);
        add("piontoenu_munu",piratio,new gaussobs(1.230e-4,0.003)); //PROBLEM!!!
        cout<<"piontoenu_munu: "<<1.2352e-4/1.230e-4<<" ERROR: "<<0.003<<endl;


        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));
        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;
        cout<<"tautopinu: "<<fermiontomesontest(tau,neutrino,Pip)<<endl;
        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;

        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));
        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;
        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;

    //ex pi0toemu=(mesondecaywidth(Mpi0,down,down,electron,muon)+mesondecaywidth(Mpi0,up,up,electron,muon)+mesondecaywidth(Mpi0,
    ex pi0toemu=(mesondw(Pi0d,electron,muon)+mesondw(Pi0d,muon,electron)+mesondw(Pi0u,electron,muon)+mesondw(Pi0u,muon,electron))/2;
    add("pi0toemu",pi0toemu,new limitedobs(3.6e-10*planck/8.52e-17));

        ex Kratio=2.477e-5/(mesondw(Kp,neutrino,electron,sVector)/mesondw(Kp,neutrino,muon,sVector));
        ex Kcorrection=Kratio.subs(replacements);
        ex Kerror=Kcorrection*0.001/2.477;
        cout<<"KRatio "<<Kcorrection-1<<" +/- "<<Kerror<<endl;
        Kratio*=mesondw(Kp,neutrino,electron)/mesondw(Kp,neutrino,muon);
        add("Ktoenu_munu",Kratio,new gaussobs(2.488e-5,0.005));
        cout<<"Ktoenu_munu: "<<2.477e-5/2.488e-5<<" ERROR: "<<0.005<<endl;

    ex k0Ltoemu=mesondw(K0,electron,muon)+mesondw(K0,muon,electron);
        add("K0Ltoemu",k0Ltoemu,new limitedobs((4.7e-12*planck/5.116e-8)));
    add("K0Ltoee",mesondw(K0,electron,electron),new limitedobs((9e-12*planck/5.116e-8)));

    //add("K0Ltomumu",mesondw(K0,muon,muon),new limitedobs((6.84e-9*planck/5.116e-8)));

        add("Dtoenu",mesondw(Dp,neutrino,electron),new limitedobs(8.8e-6*planck/1040e-15));
        add("Dtomunu",mesondw(Dp,neutrino,muon),new gaussobs(3.82e-4*planck/1040e-15,0.1)); //PROBLEM!!!
        cout<<"Dtomunu: "<<mesondw(Dp,neutrino,muon,sVector).subs(replacements)/(3.82e-4*planck/1040e-15)<<" ERROR: "<<0.1<<endl;

        add("Dtotaunu",mesondw(Dp,neutrino,tau),new limitedobs(1.2e-3*planck/1040e-15));  //PROBLEM!!!
        cout<<"Dtotaunu: "<<mesondw(Dp,neutrino,tau,sVector).subs(replacements)/(1.2e-3*planck/1040e-15)<<" LIMIT"<<endl;

        //D0 2.6e-7/410.1e-15

        ex D0toemu=mesondw(D0,electron,muon)+mesondw(D0,muon,electron);
        add("D0toemu",D0toemu,new limitedobs((2.6e-7*planck/410.1e-15)));
    ex D0toee=mesondw(D0,electron,electron);
        add("D0toee",D0toee,new limitedobs((7.9e-8*planck/410.1e-15)));
    ex D0tomumu=mesondw(D0,muon,muon);
        add("D0tomumu",D0tomumu,new limitedobs((1.4e-7*planck/410.1e-15)));

    //ex Dstomunu=mesondecaywidth(MDsp,strange,charm,muon,neutrino); //500e-15
        add("Dstomunu",mesondw(Dsp,neutrino,muon),new gaussobs(5.9e-3*planck/500e-15,0.33/5.9)); // PROBLEM!!!
        cout<<"Dstomunu: "<<mesondw(Dsp,neutrino,muon,sVector).subs(replacements)/(5.9e-3*planck/500e-15)<<" ERROR: "<<0.33/5.9<<endl;

        add("Dstoenu",mesondw(Dsp,neutrino,electron),new limitedobs(1.2e-4*planck/500e-15));
        add("Dstotaunu",mesondw(Dsp,neutrino,tau),new gaussobs(5.43e-2*planck/500e-15,0.31/5.43)); //PROBLEM!!!
        cout<<"Dstotaunu: "<<mesondw(Dsp,neutrino,tau,sVector).subs(replacements)/(5.43e-2*planck/500e-15)<<" ERROR: "<<0.31/5.43<<endl;

        add("Btomunu",mesondw(Bp,neutrino,muon),new limitedobs(9.8e-7*planck/1.641e-12));
        add("Btoenu",mesondw(Bp,neutrino,electron),new limitedobs(1e-6*planck/1.641e-12));

        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));

        //calcuBmumu calcutest(mixes,Bs0,muon,muon,2.9e-9*planck/1.516e-12,0.7e-9*planck/1.516e-12,"Bs_to_mumu");
        //calcuBmumu calcutest2(mixes,B0,muon,muon,3.6e-10*planck/1.519e-12,1.6e-10*planck/1.516e-19,"B_to_mumu");
        //cout<<"TESTE "<<endl;
        //double ps[4]={1,1e16,1e16,1e16};
        //double resteste=0,resteste2=0;
        //int nt=4,mt=1;
        //calcutest.fp(&nt,ps,&mt,&resteste);
        //calcutest2.fp(&nt,ps,&mt,&resteste2);
        //cout<<"TESTE "<<resteste/(2.9e-9*planck/1.516e-12)<<" "<<resteste2/(3.6e-10*planck/1.519e-12)<<endl;
        //ex B0tomumu=mesondw(B0,muon,muon);
        //cout<<"B0tomumu "<<collect_common_factors(B0tomumu)<<endl;
        //1.65e-4
        //add("B0tomumu",B0tomumu,new limitedobs((8e-10*planck/1.519e-12)));

        push_back(prediction(new calcuBmumu(mixes,B0,muon,muon,new limitedobs(6.3e-10*planck/1.519e-12),"B_to_mumu")));
        //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")));
        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")));
    push_back(prediction(new calcuBmumu(mixes,K0,muon,muon,new limitedobs(2.3e-9*planck/5.116e-8),"K0L_to_mumu")));

    cBmumu=new calcuBmumu(mixes,B0,muon,muon,new limitedobs(6.3e-10*planck/1.519e-12),"B_to_mumu");
    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");

    ex B0toetau=mesondw(B0,electron,tau)+mesondw(B0,tau,electron);
        add("B0toetau",B0toetau,new limitedobs((2.8e-5*planck/1.519e-12)));
        ex B0tomutau=mesondw(B0,muon,tau)+mesondw(B0,tau,muon);
        add("B0tomutau",B0tomutau,new limitedobs((2.2e-5*planck/1.519e-12)));
        ex B0toee=mesondw(B0,electron,electron);
        add("B0toee",B0toee,new limitedobs((8.3e-8*planck/1.519e-12)));
    ex B0totautau=mesondw(B0,tau,tau);
        add("B0totautau",B0totautau,new limitedobs((4.1e-3*planck/1.519e-12)));

        //B0s m=5.3663, life=1.472e-12 emu=2e-7, ee=2.8e-7 mumu=4.2e-8
        ex Bs0toemu=mesondw(Bs0,electron,muon)+mesondw(Bs0,muon,electron);
        add("Bs0toemu",Bs0toemu,new limitedobs((2e-7*planck/1.516e-12)));
    ex Bs0toee=mesondw(Bs0,electron,electron);
        add("Bs0toee",Bs0toee,new limitedobs((2.8e-7*planck/1.516e-12)));
//    ex Bs0tomumu=mesondw(Bs0,muon,muon);
//      add("Bs0tomumu",Bs0tomumu,new limitedobs((3.2e-9*planck/1.516e-12)));

   // add("chargedHiggs",pow(McH,-2),new limitedobs(std::pow(80.0,-2),0.9));

    cout<<"Bs0tomumu: "<<mesondwtest(Bs0,muon,muon)<<endl;
    //add("chargedHiggs",1/McH,new limitedobs(1/80.0,0));

    /*
    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];
    for(uint i=0;i<3;i++)
                for(uint j=0;j<3;j++)
                        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);
                        else llgamma2loop[i][j]=0;
        add("mutoegamma",llgamma2loop[1][0],new limitedobs(1.2e-11));
        add("tautoegamma",llgamma2loop[2][0],new limitedobs(3.3e-8));
        add("tautomugamma",llgamma2loop[2][1],new limitedobs(4.4e-8));
        //add("mutoegamma",llgamma2loop[1][0],new limitedobs(planck/2.197034e-6*1.2e-11));
        //add("tautoegamma",llgamma2loop[2][0],new limitedobs(planck/290.6e-15*3.3e-8));
        //add("tautomugamma",llgamma2loop[2][1],new limitedobs(planck/290.6e-15*4.4e-8));
    */

    Matrix llgammaCH;
        //Matrix
 llgammaCH=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH*12))*mixes.M[tLepton][iDown]*mixes.VN[tLepton][1].conjugate()*mixes.VN[t
        Matrix llgammaH0M,llgammaH0E;
        /*for(uint i=0;i<3;i++)
                for(uint j=0;j<3;j++)
                        for(uint k=0;k<3;k++){
                                ex z=pow(fmasses[1][i][i]/McH,2); mixes.M[tQuark][iUp][i][i]/MR,2);
                                llgammaH0M[j][k]=llgammaH0M[j][k]+(mixes.VN[0][1][j][i].conjugate()*
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
*z+6*z*z*log(z))/6;
                                llgammaH0M[j][k]=llgammaH0M[j][k]+(mixes.VN[0][1][i][j]*
mixes.VN[0][1][k][i])/mixes.M[tLepton][iDown][i][i]/mixes.M[tLepton][iDown][j][j]*(3*z+2*z*log(z));

                                llgammaH0E[j][k]=llgammaH0E[j][k]+(mixes.VN[0][1][j][i].conjugate()*
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
*z+6*z*z*log(z))/6;
                                llgammaH0E[j][k]=llgammaH0E[j][k]+(mixes.VN[0][1][i][j]*
mixes.VN[0][1][k][i])/mixes.M[tLepton][iDown][i][i]/mixes.M[tLepton][iDown][j][j]*(3*z+2*z*log(z));
                                }
                                */
        llgammaH0M=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH))*mixes.M[tLepton][
iDown]*llgammaH0M;
        llgammaH0E=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH))*mixes.M[tLepton][
iDown]*llgammaH0E;

        Matrix llgamma, llgamma2;

        for(uint i=0;i<3;i++)
                for(uint j=0;j<3;j++){
                //if(j<i)
 llgamma[i][j]=(llgammaCH[i][j]*llgammaCH[i][j].conjugate()+llgammaH0E[i][j]*llgamm
aH0E[i][j].conjugate()+llgammaH0M[i][j]*llgammaH0M[i][j].conjugate())*g*g*(1-cos2)*pow((pow(mixes.M[tLepton][iDown][i][i
            ex mmuon=mixes.M[tLepton][iDown][i][i];
                        ex A,B;

            if(j<i){ for(uint k=0;k<3;k++){
                          ex mtau=mixes.M[tLepton][iDown][k][k];
                          B+=-mixes.VN[tLepton][1][k][j].conjugate()*mixes.VN[
tLepton][1][k][i]/(12*pow(McH,2));
                          B+=mixes.N[tLepton][1][k][j].conjugate()*mixes.N[
tLepton][1][k][i]/12*(pow(MR,-2)+pow(MI,-2));

                          A+=mixes.N[tLepton][1][j][k]*mixes.N[tLepton][1][i][k].
conjugate()*(pow(MR,-2)+pow(MI,-2))/12;
                          A+=mixes.N[tLepton][1][j][k]*mixes.N[tLepton][1][k][i]/mtau/mmuon*(
Fh2(pow(mtau/MR,2))-Fh2(pow(mtau/MI,2)))/4;
                        }
                         llgamma[i][j]=(A*A.conjugate()+B*B.conjugate())*alpha*pow(mmuon,5)*GF*GF/(128*pow(
Pi,4));
                         }
                else if(j==i){
                        for(uint k=0;k<3;k++){
                          ex mtau=mixes.M[tLepton][iDown][k][k];
                          B+=-mixes.VN[tLepton][1][k][j].conjugate()*mixes.VN[
tLepton][1][k][i]/(12*pow(McH,2));
                          B+=mixes.N[tLepton][1][k][j].conjugate()*mixes.N[
tLepton][1][k][i]/12*(pow(MR,-2)+pow(MI,-2));
                          B+=mixes.N[tLepton][1][j][k].conjugate()*mixes.N[
tLepton][1][i][k]/12*(pow(MR,-2)+pow(MI,-2));
                          }
                     llgamma[i][j]=-B*GF*sqrt(1/2)/(8*pow(Pi,2))*2*mmuon; //e (GeV)^-1=1/(51e6)(e cm) where
 e=sqrt(alpha*4*Pi)
                   }
                }
        add("mutoegamma",llgamma[1][0],new limitedobs(planck/2.197034e-6*2.4e-12));
        add("tautoegamma",llgamma[2][0],new limitedobs(planck/290.6e-15*3.3e-8));
        add("tautomugamma",llgamma[2][1],new limitedobs(planck/290.6e-15*4.4e-8));

        /*add("d_e",abs(llgamma[0][0].imag_part()),new limitedobs(10.5e-28*51e6));
        add("d_mu",abs(llgamma[1][1].imag_part()),new limitedobs(1.9e-19*51e6));
        add("d_tau",llgamma[2][2].imag_part(),new gaussobs(-0.85e-17*51e6,0.825/0.85));
        cout<<"EDM: "<<llgamma[0][0].subs(conjtoabs).subs(replacements).imag_part()<<endl;
        add("a_mu",-llgamma[1][1].real_part()*2*mixes.M[tLepton][iDown][1][1],new gaussobs(3e-9,1.0/3.0));
        */
        /*llgammaCH=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH*12))*mixes.M[tQuark][iDown]*
mixes.VN[1][1].conjugate()*mixes.VN[1][1]; //4+1
        //Matrix llgammaH0M,llgammaH0E;
        for(uint i=0;i<3;i++)
                for(uint j=0;j<3;j++)
```

```
                        for(uint k=0;k<3;k++){
                                ex z=pow(mixes.M[tQuark][iUp][i][i]/MR,2);
                                llgammaH0M[j][k]=llgammaH0M[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;
                                llgammaH0M[j][k]=llgammaH0M[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));

                                llgammaH0E[j][k]=llgammaH0E[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;
                                llgammaH0E[j][k]=llgammaH0E[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));
                                }
        llgammaH0M=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH))*mixes.M[tQuark][iDown]*llgammaH0M;
        llgammaH0E=Matrix(g*g/(16*Pi*4*Pi*MW*MW*McH*McH))*mixes.M[tQuark][iDown]*llgammaH0E;

        //Matrix llgamma;
        for(uint i=0;i<3;i++)
                for(uint j=0;j<3;j++)
                if(j<i) {llgamma[i][j]=(llgammaCH[i][j]*llgammaCH[i][j].conjugate()+llgammaH0E[i][j]*llgammaH0E[i][j].conjugate()+llgammaH0M[i][j]*llgammaH0M[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);
                                //llgamma[i][j]=llgamma[i][j].subs(lst(abs(wild()*pow(MH0,-2))==abs(wild())*pow(MH0,-2)));
                                }
            else llgamma[i][j]=0;
         */


        push_back(prediction(new calcubtosgamma2(mixes)));

        //add("btosgamma",llgamma[2][1],new gaussobs(3.55e-4,sqrt(2)*0.25/3.55),1);
            //cout<<csrc<<llgamma[2][1]<<endl;
            //cout<<latex;


        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));
        BR_Htotaunu=BR_Htotaunu.subs(replacements);

        //BR_toptoHq=decaywidth(top,bottom,chiggs);
        //ex toptoWb=decaywidth(top,bottom,wboson);
        //BR_toptoHq=BR_toptoHq/(BR_toptoHq+toptoWb);
        //BR_toptoHq=BR_toptoHq.subs(replacements);

        //cout<<"toptoWb "<<toptoWb.subs(replacements).evalf()<<endl;

        //b to c tau- nu/b to c e- nu
        //ex
 btocR=decaywidth(bottom,charm,tau,neutrino,sVector)/(decaywidth(bottom,charm,electron,neutrino,sVector)+decaywidth(bott
        //cout<<btocR.subs(replacements)<<endl;

        ex BtoDtaunu,BtoD2taunu, BtoDtaunuSM, KtoPi;
        for(uint i=0; i<3; i++){
                ex Wcoup=wboson.couplingL(charm,bottom)*wboson.couplingdaggerL(tau,Fermion(tLepton,iUp,FFlavour(i)));
                if(Wcoup.subs(replacements)==ex(0)) continue;
                ex chcoup_Wcoup=-pow(MW/McH,2)*(chiggs.couplingR(charm,bottom)+chiggs.couplingL(charm,bottom))*chiggs.couplingdaggerL(tau,Fermion(tLepton,iUp,FFlavour(i)))/Wcoup;
                ex chcoup2_Wcoup=-pow(MW/McH,2)*(chiggs.couplingR(charm,bottom)-chiggs.couplingL(charm,bottom))*chiggs.couplingdaggerL(tau,Fermion(tLepton,iUp,FFlavour(i)))/Wcoup;

                BtoDtaunuSM+=Wcoup*Wcoup.conjugate();
                BtoDtaunu+=Wcoup*Wcoup.conjugate()*(1+1.5*chcoup_Wcoup.real_part()+chcoup_Wcoup.conjugate()*chcoup_Wcoup);
                BtoD2taunu+=Wcoup*Wcoup.conjugate()*(1+0.12*chcoup2_Wcoup.real_part()+0.05*chcoup2_Wcoup.conjugate()*chcoup2_Wcoup);
        }
        lst 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));
        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));

        r2.append(mixes.M[1][0][1][1]==0);
        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));
        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));
        r2.append(abs(sqrt(ex(2))* GF)==sqrt(ex(2))* GF);

        BtoDtaunuSM=collect_common_factors(BtoDtaunuSM.subs(conjtoabs).subs(r2));
        BtoDtaunu=collect_common_factors(BtoDtaunu.subs(conjtoabs).subs(r2));

        BtoDtaunuR=(BtoDtaunu/BtoDtaunuSM).subs(replacements).real_part();

        BtoD2taunu=BtoD2taunu.subs(conjtoabs).subs(r2);
        BtoD2taunuR=(BtoD2taunu/BtoDtaunuSM).subs(replacements).real_part();


        //cout<<"BtoDtaunu/BtoDtaunuSM "<<expand(BtoDtaunu/BtoDtaunuSM)<<endl;
        iBDtaunu=size();
        add("BtoDtaunu_BtoDtaunuSM",BtoDtaunu/BtoDtaunuSM,new gaussobs(440.0/296, 1.4*58.0/440));

        iBD2taunu=size();
        //cout<<"BtoD2taunu/BtoD2taunuSM "<<1+collect_common_factors(expand(BtoD2taunu/BtoDtaunuSM-1))<<endl;
        add("BtoD2taunu_BtoD2taunuSM",BtoD2taunu/BtoDtaunuSM,new gaussobs(332.0/252, 1.4*24.0/332.0));



        for(uint j=0; j<2; j++){
                ex KtoPimunu, KtoPimunuSM;
        for(uint i=0; i<3; i++){
                ex Wcoup=wboson.couplingL(up,strange)*wboson.couplingdaggerL(Fermion(tLepton,iDown,FFlavour(j)),Fermion(tLepton,iUp,FFlavour(i)));
                if(Wcoup.subs(replacements)==ex(0)) continue;
                ex chcoup_Wcoup=-pow(MW/McH,2)*(chiggs.couplingR(up,strange)+chiggs.couplingL(up,strange))\
                                *chiggs.couplingdaggerL(muon,Fermion(tLepton,iUp,FFlavour(i)))/Wcoup*(pow(MKp,2)-pow(Mpip,2))\
                                /(mixes.mass(Fermion(tLepton,iDown,FFlavour(j)))*(mixes.mass(strange)-mixes.mass(up)));
                chcoup_Wcoup=collect_common_factors(expand(chcoup_Wcoup));
                KtoPimunuSM+=collect_common_factors(expand(Wcoup*Wcoup.conjugate()));
                KtoPimunu+=collect_common_factors(expand(Wcoup*Wcoup.conjugate()*pow(1+chcoup_Wcoup,2)));
        }
            KtoPimunuSM=collect_common_factors(expand(KtoPimunuSM.subs(conjtoabs).subs(r2)));
            KtoPimunu=collect_common_factors(expand(KtoPimunu.subs(conjtoabs).subs(r2)));
            KtoPimunu=expand(KtoPimunu.subs(replacements).real_part().subs(lst(abs(wild()*pow(MR,-2))==abs(wild())*pow(MR,-2))).subs(lst(log(wild()*pow(MR,-2))==log(wild())-2*log(MR))));
            KtoPimunu=expand(KtoPimunu.evalf());
            KtoPimunuSM=expand(KtoPimunuSM.subs(replacements).real_part().subs(lst(abs(wild()*pow(MR,-2))==abs(wild())*pow(MR,-2))).subs(lst(log(wild()*pow(MR,-2))==log(wild())-2*log(MR))));
            KtoPimunuSM=expand(KtoPimunuSM.evalf());
                KtoPi+=0.5*log(KtoPimunu/KtoPimunuSM);
        }

        add("KtoPi",KtoPi/(pow(MKp,2)-pow(Mpip,2)),new gaussobs(0.08, 0.11/0.08));


        //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));

    double fD=0.207;
    ex DDbar=ex(std::pow(fD,2))*mesonmixing(MD0,charm,up);
    DDbar=expand(DDbar.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))));
        DDbar=expand(DDbar.evalf());
    ex aDDbar=sqrt(DDbar.real_part()*DDbar.real_part()+DDbar.imag_part()*DDbar.imag_part());
    add("DDbar",aDDbar,new limitedobs(9.47e-15));
    cout<<DDbar<<endl;
//2|M12|<6.6e-15GeV

    double fK=0.156;
    ex KKbar=ex(std::pow(fK,2))*mesonmixing(MK0,strange,down);
    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))));
        KKbar=expand(KKbar.evalf());
        ex aKKbar=sqrt(KKbar.real_part()*KKbar.real_part()+KKbar.imag_part()*KKbar.imag_part());
    add("KKbar",aKKbar,new limitedobs(3.5e-15));
    ex eK=0.94*imag_part(KKbar)/3.5e-15/sqrt(2);
    //add("a_eK",abs(eK),new limitedobs(2.2e-3));
    add("a_eK",abs(eK),new limitedobs(20*0.0114e-3));
    cout<<abs(KKbar)<<endl;

    double fB=0.189;
    ex Vtb=mixes.V[tQuark][2][2]/mixes.V[tQuark][2][2].conjugate();
    ex Vtd=mixes.V[tQuark][2][0]/mixes.V[tQuark][2][0].conjugate();
    ex Vts=mixes.V[tQuark][2][1]/mixes.V[tQuark][2][1].conjugate();

    ex BBbar=1+ex(std::pow(fB,2))*mesonmixing(MB0,bottom,down)/(3.337e-13*Vtb*Vtd.conjugate());
    add("BBbarimag",imag_part(BBbar),new gauss2obs(-0.199,0.062));
        add("BBbarreal",real_part(BBbar),new gauss2obs(0.823,0.143));
    cout<<BBbar<<endl;
    BBbar=3.337e-13*Vtb*Vtd.conjugate();
    cout<<"Bbar "<<(abs(imag_part(BBbar))/abs(BBbar)).subs(replacements)<<endl;
        double fBs=0.225;
    ex BsBsbar=1+ex(std::pow(fBs,2))*mesonmixing(MBs0,bottom,strange)/(1.186e-11*Vtb*Vts.conjugate());
    add("BsBsbarimag",imag_part(BsBsbar),new gauss2obs(0,0.1));
        add("BsBsbarreal",real_part(BsBsbar),new gauss2obs(0.965,0.133));
    cout<<BsBsbar<<endl;
    BsBsbar=1.186e-11*Vtb*Vts.conjugate();
    cout<<"Bbar "<<(abs(imag_part(BsBsbar))/abs(BsBsbar)).subs(replacements)<<endl;

    ex McH2=McH*McH;
    ex MR2=MR*MR;
    ex MI2=MI*MI;

    ex cu=collect_common_factors(expand(chiggs.couplingL(top,bottom)))/mixes.mass(top)/(g/MW/sqrt(
ex(2)))/mixes.V[1][2][2];
    cout<<"cu "<<cu<<endl;
    ex Zbb=(cu-0.72)/McH;
    add("Zbb",Zbb,new limitedobs(0.0024));
    cout<<"Zbb "<<Zbb<<endl;
    cout<<"SIZE "<<size()<<endl;

   push_back(prediction(new calcuOblique()));
}

~BGL(){
        delete cBmumu;
        delete cBsmumu;
        }

ex Y(ex x) const{
                return 1.0113*x/8/(1-x)*(4-x+3*x*log(x)/(1-x));
                }

ex GW(ex x) const{
        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));
        }

ex GH1(ex x) const{
        return x*(x*((39-14*x)*x-6)+6*x*(3*x-8)*log(x)-19)/(36*pow(x-1,4));
        //return -x/12;
        }

ex GH2(ex x) const{
        return x*((x-1)*(11*x-21)+(16-6*x)*log(x))/(6*pow(x-1,3));
        //return x/2;
        }

ex FW(ex x) const{
        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));
        }

ex FH1(ex x) const{
        return -x/12;
        }

ex FH2(ex x) const{
        return x/2;
        }

ex Fh1(ex x) const{
        //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));
        return x/3;
        }

ex Fh2(ex x) const{
        //return (-3*x+4*pow(x,2)-pow(x,3)-2*x*log(x))/pow(1-x,3);
        return -2*(3/2+log(x))*x;
        }

ex A0(ex x) const{
        return x*(2+3*x-6*x*x+ x*x*x+6*x*log(x))/(24*pow(1-x,4));
        }

ex A1(ex x) const{
        return x*(-3+4*x-x*x-2*log(x))/(4*pow(1-x,3));
        }

ex A2(ex x) const{
        return x/(6*pow(1-x,3))*((-7+5*x-8*x*x)/6.0+x*log(x)/(1-x)*(-2+3*x));
        }

ex A3(ex x) const{
        return (-3+8*x-5*x*x+(6*x-4)*log(x))*x/(6*pow(1-x,3));
        }

void add(const char * s, ex pred, observable * ob, bool sb=0){
        //cout<<s<<endl;
        //cout<<"prediction symb"<<pred<<endl;
        //,pow(sin(wild()), 2) == 1-pow(cos(wild()), 2)
        //ex
    p=expand(pred.subs(replacements).real_part().subs(lst(abs(wild()*pow(MR,-2))==abs(wild())*pow(MR,-2))).subs(lst(log(wi

        ex p=pred.subs(replacements).real_part();
        p=collect_common_factors(expand(p.evalf()));
        FUNCP_CUBA fp;

        lst l(tanb,McH,MR,MI);

        for(uint i=0;i<3;i++){
                l.append(Mu[i]);
                l.append(Md[i]);
        }
        if(sb) push_back(prediction(ob,p));
        else {
        compile_ex(lst(p), l, fp);
        //cout<<"prediction numeric"<<p<<endl;
        //cout<<"exp "<<ob->expected()<<endl<<endl;
        push_back(prediction(ob,fp));
   }
        }

int veto(const parameters & p, int max=0) const{
        if(!p.isvalid()) return 1;
        if(max==1){
        double mr=p[1].value+p[2].value;
        if(mr<10 || mr>10000) return 1;
        mr+=p[3].value;
        if(mr<10 || mr>10000) return 1;
        return 0;
        }
        else{
        double mr=p[1].value+p[2].value;
        if(mr<10 || mr>mmmax) return 1;
        mr+=p[3].value;
        if(mr<10 || mr>mmmax) return 1;
        return 0;
        }
        }

parameters generateparameters(int max=0) const{
        parameters p;
        //x=log_10(tanb)
        p.push_back(freeparameter(-3,3,r,stepsize));
        //y=log_10(McH)
        if(max==1) p.push_back(freeparameter(10,10000,r,stepsize));
        else p.push_back(freeparameter(10,mmmax,r,stepsize));
        //log_10(massR)
        p.push_back(freeparameter(-200,200,r,stepsize));
        //log_10(massI)
        p.push_back(freeparameter(-50,50,r,stepsize));

        return p;
}

parameters getlist(const parameters & p) const{
        //cout<<aux<<endl;
        //double
    c2=(1+sqrt(1-4*sqrt(ex_to<numeric>(mudecay.subs(lst(tanb==exp(p[0].value),McH==p[1].value))).to_double())))/2;

        double x=pow(10.0,p[0].value);
        //double y=pow(10.0,p[1].value);
        //double z=pow(10.0,p[2].value);
        //double w=pow(10.0,p[3].value);

        double y=p[1].value;
        double z=y+p[2].value;
        double w=z+p[3].value;

        parameters pp(p);
        pp[0].value=x;
        pp[2].value+=pp[1].value;
        pp[3].value+=pp[2].value;
        pp.values=vector<double>();
        for(uint i=0; i<4; i++) pp.values.push_back(pp[i].value);
        lst &l=pp.p;
        l=lst(tanb==x,McH==y,MR==z,MI==w);

        return pp;
}

double bsgammawidth(double tanb,double McH,double MR,double MI, int option=0){
        parameters p=generateparameters();
        p[0].value=pow(10.0,tanb);
        p[1].value=McH;
        p[2].value=MR;
        p[3].value=MI;

        calcubtosgamma2 cal(mixes);

        return cal.width(p,option);
}
/*
ex decaywidth2(const Fermion& f1, const Fermion& ff2, const Fermion& ff3, const Fermion& ff4, BSpin s=sAny)
 const{

        Fermion f2=ff2,f3=ff3, f4=ff4;

        ex ret=0;

        realsymbol q1("q1"), q2("q2"), q3("q3"), q4("q2"), s3("s3");
        ex s2=pow(mixes.mass(f1),2);

        for(uint j=fElectron;j<=fTau;j++)
        if(ff2.flavour==fAny || ff2.flavour==j){
                f2.flavour=(FFlavour)j;
        for(uint k=fElectron;k<=fTau;k++)
        if(ff3.flavour==fAny || ff3.flavour==k){
                f3.flavour=(FFlavour)k;
        for(uint l=fElectron;l<=fTau;l++)
        if(ff4.flavour==fAny || ff4.flavour==l){
                f4.flavour=(FFlavour)l;
                ex v1=0, v2=0;
                ex mq1=mixes.mass(f1),mq2=mixes.mass(f2),mq3=mixes.mass(f3),mq4=mixes.mass(f4);
                ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;
                ex s4=m2q1+m2q2+m2q3+m2q4-s2-s3;

                scalar_products sp;
                sp.add(q2, q3, (s4-m2q2-m2q3)/2);
                sp.add(q4, q3, (s2-m2q4-m2q3)/2);
                sp.add(q2, q4, (s3-m2q2-m2q4)/2);

                sp.add(q2, q2, m2q2);
                sp.add(q3, q3, m2q3);
                sp.add(q4, q4, m2q4);

                ex q10=(s2+mq1*mq1-mq2*mq2)/(2*sqrt(s2)), lq1l=sqrt(q10*q10-mq1*mq1);
                ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);

        for(uint i=0;i<bosons.size();i++)if(bosons[i].s==s || s==sAny){
                if(bosons[i].s==0){
                        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);
                        v1=v1+a*dirac_gammaL();
                        v2=v2+a.conjugate()*dirac_gammaR();
                        a=-(bosons[i].couplingR(f2,f1)-bosons[i].couplingL(f2,f1))*
bosons[i].couplingdaggerR(f3,f4)*s2/(mq1+mq2)/pow(bosons[i].mass,2);
                        v1=v1+a*dirac_gammaR();
                        v2=v2+a.conjugate()*dirac_gammaL();
                }
                else{
                        ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
                        ex a=(bosons[i].couplingR(f2,f1)-bosons[i].couplingL(f2,f1))*
bosons[i].couplingdaggerL(f3,f4)/pow(bosons[i].mass,2);
                        v1=v1+a*sl*dirac_gammaL();
                        v2=v2+a.conjugate()*sl*dirac_gammaL();
                        a=(bosons[i].couplingR(f2,f1)-bosons[i].couplingL(f2,f1))*
bosons[i].couplingdaggerR(f3,f4)/pow(bosons[i].mass,2);
                        v1=v1+a*sl*dirac_gammaR();
                        v2=v2+a.conjugate()*sl*dirac_gammaR();
                }
        }
        ex vq3=dirac_slash(q3,4)-mq3*dirac_ONE(), vq4=dirac_slash(q4,4)+mq4*dirac_ONE();
        ex dt=dirac_trace(vq3*v1*vq4*v2).simplify_indexed(sp);
        cout<<"dt: "<<dt<<endl;
        ex result=expand(dt*4*lq3l/s2/Pi/128);

        ret+=result;
        }
        }

        return collect_common_factors(ret.subs(conjtoabs));
        //return expand(ret.subs(lst(exp(-I*wild())==1/exp(I*
wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
}
*/

ex decaywidth(const Fermion& ff1, const Fermion& ff2, const
Fermion& ff3, const Fermion& ff4, BSpin s=sAny) const{
        multivector<ex,4> a(0,bosons.size(),2,2,2);
        vector<ex> mass(bosons.size(),0);
        vector<int> op(bosons.size(),0);
        ex ret=0;
        Fermion f1=ff1, f2=ff2,f3=ff3, f4=ff4;


        for(uint i=fElectron;i<=fTau;i=i+1)
        if(ff1.flavour==fAny || ff1.flavour==i){
                f1.flavour=(FFlavour)i;
        for(uint j=fElectron;j<=fTau;j++)
        if(ff2.flavour==fAny || ff2.flavour==j){
                f2.flavour=(FFlavour)j;
        for(uint k=fElectron;k<=fTau;k++)
        if(ff3.flavour==fAny || ff3.flavour==k){
                f3.flavour=(FFlavour)k;
        for(uint l=fElectron;l<=fTau;l++)
        if(ff4.flavour==fAny || ff4.flavour==l){
                f4.flavour=(FFlavour)l;
        for(uint i=0;i<bosons.size();i++) if(bosons[i].s==s || s==sAny){
                        op[i]=bosons[i].s;
                        mass[i]=bosons[i].mass;
                        a[i][0][0][0]=bosons[i].couplingdaggerL(f2,f1)*bosons[i].couplingL(f3,f4);
                        a[i][0][0][1]=bosons[i].couplingdaggerL(f2,f1)*bosons[i].couplingR(f3,f4);
                        a[i][0][1][0]=bosons[i].couplingdaggerR(f2,f1)*bosons[i].couplingL(f3,f4);
                        a[i][0][1][1]=bosons[i].couplingdaggerR(f2,f1)*bosons[i].couplingR(f3,f4);

                        a[i][1][0][0]=bosons[i].couplingdaggerL(f3,f1)*bosons[i].couplingL(f2,f4);
                        a[i][1][0][1]=bosons[i].couplingdaggerL(f3,f1)*bosons[i].couplingR(f2,f4);
                        a[i][1][1][0]=bosons[i].couplingdaggerR(f3,f1)*bosons[i].couplingL(f2,f4);
                        a[i][1][1][1]=bosons[i].couplingdaggerR(f3,f1)*bosons[i].couplingR(f2,f4);
        }

        ret+=wc.get_integral_symb(a,mass,op,mixes.mass(f1));
        //
ret+=wc.get_integral(a,mass,op,mixes.massnum(f1),mixes.massnum(f2),mixes.massnum(f3),mixes.massnum(f4))/pow(mixes.massnu
        }}}}
        if(ff2.flavour==ff4.flavour) ret=ret/2;
        return collect_common_factors(ret.subs(conjtoabs));
        //return
 expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
}

ex get_integral_symb(const multivector<ex,3>& a, ex m1) const{
        realsymbol s2("s2"), s3("s3");
        realsymbol q1("q1"), q2("q2"), q3("q3"), q4("q4");

        ex m2q1=m1*m1;

        ex vq1=dirac_slash(q2,4)+dirac_slash(q3,4)+dirac_slash(q4,4)+m1*dirac_ONE(), vq2=dirac_slash(q2,4);
        ex vq3=dirac_slash(q3,4), vq4=dirac_slash(q4,4);

        ex s4=m2q1-s2-s3;
    scalar_products sp;
    sp.add(q2, q3, (s4)/2);
    sp.add(q4, q3, (s2)/2);
    sp.add(q2, q4, (s3)/2);

    sp.add(q2, q2, 0);
    sp.add(q3, q3, 0);
    sp.add(q4, q4, 0);

    multivector<ex,2> v(0,2,2);
        v[0][0]=dirac_gammaL(); v[0][1]=dirac_gammaR();
        v[1][0]=dirac_gammaR(); v[1][1]=dirac_gammaL();

        multivector<ex,5> traces(0,2,2,2,2,2);
            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 vk=v[k][0];
                                ex vm=v[m][0];
                                ex vl=v[l][1];
                                ex vn=v[n][1];

                                traces[k][l][m][n][0]=dirac_trace(vq2*vk*vq1*vl)*dirac_trace(vq3*vm*vq4*vn)
;
                                traces[k][l][m][n][1]=-dirac_trace(vq2*vk*vq1*vl*vq3*vm*vq4*vn);
                        }

            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 o=0;o<2;o++)
                                        {
                                                        traces[k][l][m][n][o]=(traces[k][l][m][n][o]).
simplify_indexed(sp);
                                                }

        ex q10=(s2+m1*m1)/(2*sqrt(s2)), lq1l=(m1*m1-s2)/(2*sqrt(s2));
    ex q30=sqrt(s2)/2, lq3l=q30;
    ex q20=(m1*m1-s2)/(2*m1), lq2l=q20;

    ex total=0;
    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[r][k][m]*a[s][l][n].conjugate();
                ex integrand=traces[k][l][m][n][(r+s)%2];
                integrand=expand(integral(s3, 0, m1*m1-s2, integrand).eval_integ()/lq1l/sqrt(s2)*lq2l/m1/m1);
                //double mm2=0, mm3=0, m4=0;
                ex result=integral(s2,0,m1*m1,integrand).eval_integ()/pow(Pi,3)/512;
            ex partial=result*coup;
                total=total+partial;
                }
        return total;
}

ex decaywidthtest2(const Fermion& ff1) const{
        multivector<ex,3> a(0,2,2,2);
        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}");

                        a[0][0][0]=gLL;
                        a[0][0][1]=gLR;
                        a[0][1][0]=gRL;
                        a[0][1][1]=gRR;

                        a[1][0][0]=cLL;
                        a[1][0][1]=cLR;
                        a[1][1][0]=cRL;
                        a[1][1][1]=cRR;

        ex ret=get_integral_symb(a,mixes.mass(ff1));
        //
ret+=wc.get_integral(a,mass,op,mixes.massnum(f1),mixes.massnum(f2),mixes.massnum(f3),mixes.massnum(f4))/pow(mixes.massnu

        return collect_common_factors(ret.subs(conjtoabs));
        //return expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
}

/*
ex decaywidthtest(const Fermion& f1, const Fermion& f2, const Fermion& ff3, const Fermion& ff4, BSpin s=sAny) const{

        Fermion f1=ff1, f2=ff2,f3=ff3, f4=ff4;

        ex ret=0;

        realsymbol q1("q1"), q2("q2"), q3("q3"), q4("q4");
        symbol gL("gL"), gR("gR");
        ex s2("s2"), s3("s3");

        for(uint i=fElectron;i<=fTau;i=i+1)
        if(ff1.flavour==fAny || ff1.flavour==i){
                f1.flavour=(FFlavour)i;
        for(uint j=fElectron;j<=fTau;j++)
        if(ff2.flavour==fAny || ff2.flavour==j){
                f2.flavour=(FFlavour)j;
        for(uint k=fElectron;k<=fTau;k++)
        if(ff3.flavour==fAny || ff3.flavour==k){
                f3.flavour=(FFlavour)k;
        for(uint l=fElectron;l<=fTau;l++)
        if(ff4.flavour==fAny || ff4.flavour==l){
                f4.flavour=(FFlavour)l;
                ex v1=0, v2=0;
                ex mq1=mixes.mass(f1),mq2=mixes.mass(f2),mq3=mixes.mass(f3),mq4=mixes.mass(f4);
                ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;

                ex s4=m2q1+m2q2+m2q3+m2q4-s2-s3;

                scalar_products sp;

                sp.add(q4, q3, (s2-m2q4-m2q3)/2);
                sp.add(q3, q3, m2q3);
                sp.add(q4, q4, m2q4);
                ex q10=(s2+mq1*mq1-mq2*mq2)/(2*sqrt(s2)), lq1l=sqrt(q10*q10-mq1*mq1);
                ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);

                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();
            ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)-mq4*dirac_ONE();
```

```

                ex a;

                a=gL;
                v1=v1+a*dirac_gammaL();
                v2=v2+a.conjugate()*dirac_gammaR();
                a=gR;
                v1=v1+a*dirac_gammaR();
                v2=v2+a.conjugate()*dirac_gammaL();

                / *}
                else{
                        ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
                        ex a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*bosons[i].couplingL(f3,f4)/pow(bosons[i].mass,2);
                        v1=v1+a*sl*dirac_gammaL();
                        v2=v2+a.conjugate()*sl*dirac_gammaL();
                        a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*bosons[i].couplingR(f3,f4)/pow(bosons[i].mass,2);
                        v1=v1+a*sl*dirac_gammaR();
                        v2=v2+a.conjugate()*sl*dirac_gammaR();
                }* /

        ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)-mq4*dirac_ONE();
        ex dt=dirac_trace(vq3*v1*vq4*v2).simplify_indexed(sp);
        ex result=expand(dt*4*lq3l/s2/Pi/128);

        ret+=result;
        }
        }
        lst ltest;
        ltest.append(conjugate(gL)==pow(abs(gL),2)/gL);
        ltest.append(conjugate(gR)==pow(abs(gR),2)/gR);
        return pow(meson.decay_factor,2)*collect_common_factors(ret.subs(conjtoabs).subs(ltest));
        //return expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
}
*/

ex gRR2(const Fermion& f1, const Fermion& f3) const{

        ex ret1=0,ret2=0;
        Fermion f2(tLepton,iUp);
        Fermion f4(tLepton,iUp);

        for(uint k=fElectron;k<=fTau;k++){
                f2.flavour=(FFlavour)k;
        for(uint l=fElectron;l<=fTau;l++){
                f4.flavour=(FFlavour)l;
        for(uint i=0;i<bosons.size();i++)
                if(bosons[i].s==sScalar) {
                        ex x=bosons[i].couplingdaggerR(f2,f1)*bosons[i].couplingL(f3,f4)/pow(bosons[i].mass,2);
                        ret1+=x*x.conjugate();
                }
                else if(bosons[i].s==sVector) {
                        ex x=bosons[i].couplingdaggerL(f2,f1)*bosons[i].couplingL(f3,f4)/pow(bosons[i].mass,2);
                        ret2+=x*x.conjugate();
                }
        }}
        //r2.append();
        ret2=ret2.subs(conjtoabs);
        ret1=ret1.subs(conjtoabs);
        for(uint i=0;i<3;i++){
                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))));
                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))));
        }

        cout<<ret2<<endl;
        return collect_common_factors(ret1/ret2);
}

ex tautomu_tautoe() const{

        ex ret1=0,ret2=0, rety1=0, rety2=0;

        Fermion f1(tLepton,iDown,fTau);
        Fermion f31(tLepton,iDown,fMuon);
        Fermion f32(tLepton,iDown,fElectron);

        Fermion f2(tLepton,iUp);
        Fermion f4(tLepton,iUp);


        for(uint k=fElectron;k<=fTau;k++){
                f2.flavour=(FFlavour)k;
        for(uint l=fElectron;l<=fTau;l++){
                f4.flavour=(FFlavour)l;
                ex x1=0, x2=0, y1=0, y2=0;
            for(uint i=0;i<bosons.size();i++){
                if(bosons[i].s==sScalar) {
                        x1+=bosons[i].couplingdaggerR(f2,f1)*bosons[i].couplingL(f31,f4)/pow(bosons[i].
mass,2);
                        x2+=bosons[i].couplingdaggerR(f2,f1)*bosons[i].couplingL(f32,f4)/pow(bosons[i].mass
,2);
                }
                else if(bosons[i].s==sVector) {
                        y1+=bosons[i].couplingdaggerL(f2,f1)*bosons[i].couplingL(f31,f4)/pow(bosons[i].
mass,2);
                        y2+=bosons[i].couplingdaggerL(f2,f1)*bosons[i].couplingL(f32,f4)/pow(bosons[i].mass
,2);
                }
                }
                ret1+=(x1*y1.conjugate()).real_part();
                ret2+=(x2*y2.conjugate()).real_part();
                rety1+=y1*y1.conjugate();
                rety2+=y2*y2.conjugate();
        }}
        ret2=(ret2/rety2*mixes.mass(f32)/mixes.mass(f1)).subs(conjtoabs);
        ret1=(ret1/rety1*mixes.mass(f31)/mixes.mass(f1)).subs(conjtoabs);
        for(uint i=0;i<3;i++){
                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))));
                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))));
        }

        ex x=pow(mixes.mass(f31)/mixes.mass(f1),2);
        ex F1=1-8*x+8*pow(x,3)-pow(x,4)-12*pow(x,2)*log(x);
        ex g1=1+9*x-9*pow(x,2)-pow(x,3)+6*x*(1+x)*log(x);
        ex N1=1+gRR2(f1,f31)/4;

        x=pow(mixes.mass(f32)/mixes.mass(f1),2);

        ex F2=1-8*x+8*pow(x,3)-pow(x,4)-12*pow(x,2)*log(x);
        ex g2=1+9*x-9*pow(x,2)-pow(x,3)+6*x*(1+x)*log(x);
        ex N2=1+gRR2(f1,f32)/4;

        return collect_common_factors(N1*(F1+2/N1*ret1*g1)/N2/(F2+2/N2*ret2*g2)*F2/F1);
}

ex mesondw(const Meson & meson, const Fermion& ff3, const
Fermion& ff4, BSpin s=sAny) const{

        const Fermion& f1(meson.q1), f2(meson.q2);
        ex mesonmass=meson.mass;

        Fermion f3=ff3, f4=ff4;

        ex ret=0;

        realsymbol q3("q3"), q4("q4");
        ex s2=pow(mesonmass,2);

        for(uint k=fElectron;k<=fTau;k++)
        if(ff3.flavour==fAny || ff3.flavour==k){
                f3.flavour=(FFlavour)k;
        for(uint l=fElectron;l<=fTau;l++)
        if(ff4.flavour==fAny || ff4.flavour==l){
                f4.flavour=(FFlavour)l;
                ex v1=0, v2=0;
                ex mq1=mixes.mass(f1),mq2=mixes.mass(f2),mq3=mixes.mass(f3),mq4=mixes.mass(f4);
                ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;
                scalar_products sp;
                sp.add(q4, q3, (s2-m2q4-m2q3)/2);
                sp.add(q3, q3, m2q3);
                sp.add(q4, q4, m2q4);
                ex q10=(s2+mq1*mq1-mq2*mq2)/(2*sqrt(s2)), lq1l=sqrt(q10*q10-mq1*mq1);
                ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);

        for(uint i=0;i<bosons.size();i++) if(bosons[i].s==s || s==sAny){
                if(bosons[i].s==0){
                        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);
                        v1=v1+a*dirac_gammaL();
                        v2=v2+a.conjugate()*dirac_gammaR();
                        a=-(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*bosons[i].
couplingR(f3,f4)*s2/(mq1+mq2)/pow(bosons[i].mass,2);
                        v1=v1+a*dirac_gammaR();
                        v2=v2+a.conjugate()*dirac_gammaL();
                }
                else{
                        ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
                        ex a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*bosons[i].
couplingL(f3,f4)/pow(bosons[i].mass,2);
                        v1=v1+a*sl*dirac_gammaL();
                        v2=v2+a.conjugate()*sl*dirac_gammaL();
                        a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*bosons[i].
couplingR(f3,f4)/pow(bosons[i].mass,2);
                        v1=v1+a*sl*dirac_gammaR();
                        v2=v2+a.conjugate()*sl*dirac_gammaR();
                }
        }
        ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)-mq4*dirac_ONE();
        ex dt=dirac_trace(vq3*v1*vq4*v2).simplify_indexed(sp);
        ex result=expand(dt*4*lq3l/s2/Pi/128);

        ret+=result;
        }
        }

        return pow(meson.decay_factor,2)*collect_common_factors(ret.subs(conjtoabs));
        //return
 expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
}


ex mesondwtest(const Meson & meson, const Fermion& ff3, const
Fermion& ff4, BSpin s=sAny) const{

        const Fermion& f1(meson.q1), f2(meson.q2);
        ex mesonmass=meson.mass;

        Fermion f3=ff3, f4=ff4;

        ex ret=0;

        realsymbol q3("q3"), q4("q4");
        symbol gL("gL"), gR("gR"),gVL("gVL"), gVR("gVR");
        symbol gS("gS"), gP("gP"), gA("gA");

        ex s2=pow(mesonmass,2);

        for(uint k=fElectron;k<=fTau;k++)
        if(ff3.flavour==fAny || ff3.flavour==k){
                f3.flavour=(FFlavour)k;
        for(uint l=fElectron;l<=fTau;l++)
        if(ff4.flavour==fAny || ff4.flavour==l){
                f4.flavour=(FFlavour)l;
                ex v1=0, v2=0;
                ex mq1=mixes.mass(f1),mq2=mixes.mass(f2),mq3=mixes.mass(f3),mq4=mixes.mass(f4);
                ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;
                scalar_products sp;
                sp.add(q4, q3, (s2-m2q4-m2q3)/2);
                sp.add(q3, q3, m2q3);
                sp.add(q4, q4, m2q4);
                ex q10=(s2+mq1*mq1-mq2*mq2)/(2*sqrt(s2)), lq1l=sqrt(q10*q10-mq1*mq1);
                ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);

                ex a;
        /*      a=-gL*s2/(mq1+mq2);
                v1=v1+a*dirac_gammaL();
                v2=v2+a.conjugate()*dirac_gammaR();
                a=-gR*s2/(mq1+mq2);
                v1=v1+a*dirac_gammaR();
                v2=v2+a.conjugate()*dirac_gammaL();

                ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
                a=gA;
                v1=v1+a*sl*dirac_gamma5();
                v2=v2+a.conjugate()*sl*dirac_gamma5();
*/
                a=-gS*s2/(mq1+mq2);
                v1=v1+a*dirac_ONE();
                v2=v2+a.conjugate()*dirac_ONE();
                a=-gP*s2/(mq1+mq2);
                v1=v1+a*dirac_gamma5();
                v2=v2-a.conjugate()*dirac_gamma5();
                ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
        //      a=gA;
        //      v1=v1+a*sl*dirac_gamma5();
        //      v2=v2+a.conjugate()*sl*dirac_gamma5();

                /*}
                else{
                        ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
                        ex a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*
bosons[i].couplingL(f3,f4)/pow(bosons[i].mass,2);
```

```
                                v1=v1+a*sl*dirac_gammaL();
                                v2=v2+a.conjugate()*sl*dirac_gammaL();
                                a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*
bosons[i].couplingR(f3,f4)/pow(bosons[i].mass,2);
                                v1=v1+a*sl*dirac_gammaR();
                                v2=v2+a.conjugate()*sl*dirac_gammaR();
                        }*/

        ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)-mq4*dirac_ONE();
        ex dt=dirac_trace(vq3*v1*vq4*v2).simplify_indexed(sp);
        ex result=expand(dt*4*lq3l/s2/Pi/128);

        ret+=result;
        }
        }
        lst ltest;
        ltest.append(conjugate(gL)==pow(abs(gL),2)/gL);
        ltest.append(conjugate(gR)==pow(abs(gR),2)/gR);
        ltest.append(conjugate(gS)==pow(abs(gS),2)/gS);
        ltest.append(conjugate(gP)==pow(abs(gP),2)/gP);
        ltest.append(conjugate(gA)==pow(abs(gA),2)/gA);

        return pow(meson.decay_factor,2)*collect_common_factors(expand(ret.subs(conjtoabs).subs
(ltest)));
        //return
 expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
}

ex fermiontomeson(const Fermion& ff4, const Fermion& ff3, const
Meson & meson, BSpin s=sAny) const{

        const Fermion& f1(meson.q1), f2(meson.q2);
        ex mesonmass=meson.mass;

        Fermion f3=ff3, f4=ff4;

        ex ret=0;

        realsymbol q3("q3"), q4("q4");
        ex s2=pow(mesonmass,2);

        for(uint k=fElectron;k<=fTau;k++)
        if(ff3.flavour==fAny || ff3.flavour==k){
                f3.flavour=(FFlavour)k;
        for(uint l=fElectron;l<=fTau;l++)
        if(ff4.flavour==fAny || ff4.flavour==l){
                f4.flavour=(FFlavour)l;
                ex v1=0, v2=0;
                ex mq1=mixes.mass(f1),mq2=mixes.mass(f2),mq3=mixes.mass(f3),mq4=mixes.mass(f4);
                ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;
                scalar_products sp;
                sp.add(q4, q3, -(s2-m2q4-m2q3)/2);
                sp.add(q3, q3, m2q3);
                sp.add(q4, q4, m2q4);
                //ex qm0=(s2-mq3*mq3+mq4*mq4)/(2*mq4), lqml=sqrt(qm0*qm0-s2);
                ex q30=(-s2+mq3*mq3+mq4*mq4)/(2*mq4), lq3l=sqrt(q30*q30-mq3*mq3);
        //ex q30=-(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);

        for(uint i=0;i<bosons.size();i++)if(bosons[i].s==s || s==sAny){
                if(bosons[i].s==0){
                        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);
                        v1=v1+a*dirac_gammaL();
                        v2=v2+a.conjugate()*dirac_gammaR();
                        a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*bosons[i].
couplingR(f3,f4)*s2/(mq1+mq2)/pow(bosons[i].mass,2);
                        v1=v1+a*dirac_gammaR();
                        v2=v2+a.conjugate()*dirac_gammaL();
                }
                else{
                        ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
                        ex a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*bosons[i].
couplingL(f3,f4)/pow(bosons[i].mass,2);
                        v1=v1+a*sl*dirac_gammaL();
                        v2=v2+a.conjugate()*sl*dirac_gammaL();
                        a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1))*bosons[i].
couplingR(f3,f4)/pow(bosons[i].mass,2);
                        v1=v1+a*sl*dirac_gammaR();
                        v2=v2+a.conjugate()*sl*dirac_gammaR();
                }
        }
        ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)+mq4*dirac_ONE();
        ex dt=dirac_trace(vq3*v1*vq4*v2).simplify_indexed(sp);
        ex result=expand(dt*2*lq3l/mq4/mq4/Pi/128);

        ret+=result;
        }
        }



        return pow(meson.decay_factor,2)*collect_common_factors(ret.subs(conjtoabs));
        //return
 expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
}

ex fermiontomesontest(const Fermion& ff4, const Fermion& ff3, const
Meson & meson, BSpin s=sAny) const{

        const Fermion& f1(meson.q1), f2(meson.q2);
        ex mesonmass=meson.mass;

        Fermion f3=ff3, f4=ff4;

        ex ret=0;

        realsymbol q3("q3"), q4("q4");

        symbol sL("sL"), sR("sR"), vL("vL"), vR("vR");
        ex s2=pow(mesonmass,2);

        for(uint k=fElectron;k<=fTau;k++)
        if(ff3.flavour==fAny || ff3.flavour==k){
                f3.flavour=(FFlavour)k;
        for(uint l=fElectron;l<=fTau;l++)
        if(ff4.flavour==fAny || ff4.flavour==l){
                f4.flavour=(FFlavour)l;
                ex v1=0, v2=0;
                ex mq1=mixes.mass(f1),mq2=mixes.mass(f2),mq3=mixes.mass(f3),mq4=mixes.mass(f4);
                ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;
                scalar_products sp;
                sp.add(q4, q3, -(s2-m2q4-m2q3)/2);
                sp.add(q3, q3, m2q3);
                sp.add(q4, q4, m2q4);
                //ex qm0=(s2-mq3*mq3+mq4*mq4)/(2*mq4), lqml=sqrt(qm0*qm0-s2);
                ex q30=(-s2+mq3*mq3+mq4*mq4)/(2*mq4), lq3l=sqrt(q30*q30-mq3*mq3);
        //ex q30=-(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);


                        ex a=sL;
                        v1=v1+a*dirac_gammaL();
                        v2=v2+a.conjugate()*dirac_gammaR();
                        a=sR;
                        v1=v1+a*dirac_gammaR();
                        v2=v2+a.conjugate()*dirac_gammaL();

                        ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
                        a=vL;
                        v1=v1+a*sl*dirac_gammaL();
                        v2=v2+a.conjugate()*sl*dirac_gammaL();
                        a=vR;
                        v1=v1+a*sl*dirac_gammaR();
                        v2=v2+a.conjugate()*sl*dirac_gammaR();

        ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)+mq4*dirac_ONE();
        ex dt=dirac_trace(vq3*v1*vq4*v2).simplify_indexed(sp);
        ex result=expand(dt*2*lq3l/mq4/mq4/Pi/128);

        ret+=result;
        }
        }

        return pow(meson.decay_factor,2)*collect_common_factors(ret.subs(conjtoabs));
        //return
 expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
}

ex mesonmixing(ex mesonmass, const Fermion& f1, const Fermion& f2) const{

        ex ret=0;

                ex v1=0, v2=0;
                ex mq1=mixes.mass(f1),mq2=mixes.mass(f2);
                ex m2q1=mq1*mq1, m2q2=mq2*mq2;

        for(uint i=0;i<bosons.size();i++)
                if(bosons[i].s==0){
                        ex a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1));
                        v1=v1+pow(a/bosons[i].mass,2);

                        ex b=(bosons[i].couplingdaggerR(f2,f1)+bosons[i].couplingdaggerL(f2,f1));
                        v2=v2+pow(b/bosons[i].mass,2);
                }

        ex fc=mesonmass/(mixes.massnum(f1)+mixes.massnum(f2));
        fc=pow(fc,2);

        ret=2*(-v1*(1+11*fc)+v2*(1+fc))*mesonmass/96;

        return collect_common_factors(ret.subs(conjtoabs));
        //return
 expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
}

ex CHdecaycoupling(Boson higgs, const Fermion& ff3, const
Fermion& ff4) const{

        Fermion f3=ff3, f4=ff4;
        ex ret=0;
        for(uint k=fElectron;k<=fTau;k++)
        if(ff3.flavour==fAny || ff3.flavour==k){
                f3.flavour=(FFlavour)k;
        for(uint l=fElectron;l<=fTau;l++)
        if(ff4.flavour==fAny || ff4.flavour==l){
                f4.flavour=(FFlavour)l;
                ret+=higgs.couplingdaggerL(f3,f4)*higgs.
couplingdaggerL(f3,f4).conjugate()+higgs.couplingdaggerR(f3,f4)*higgs.
couplingdaggerR(f3,f4).conjugate();
        }}
        return collect_common_factors(ret.subs(conjtoabs));
}


double BranchingRatio(double * xx,double * p){
        return ex_to<numeric>(BR_Htotaunu.subs(tanb==pow(10.0,xx[0])).evalf()).to_double();
}


double topBranchingRatio(double * xx,double * p){
        return ex_to<numeric>(BR_toptoHq.subs(lst(tanb==pow(10.0,xx[0]),McH==xx[1])).evalf()).to_double();
}

widthcalc wc;

const double planck;
const possymbol GF, MZ, MW, Mh;
const constant Mpip, Mpi0, MBp,MB0,MBs0, MKp,MK0,MDp,MD0,MDsp,MDs0;
const constant Fpi, FB,FBs, FK,FD,FDs;
ex cos2, g, alpha;
const possymbol tanb, cp, McH, MR, MI, rho;
possymbol Mu[3],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, iBDtaunu, iBD2taunu;
vector<int> BGLtype;

double mmmax, stepsize;

calcuBmumu * cBmumu;
calcuBmumu * cBsmumu;


};


}
#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.

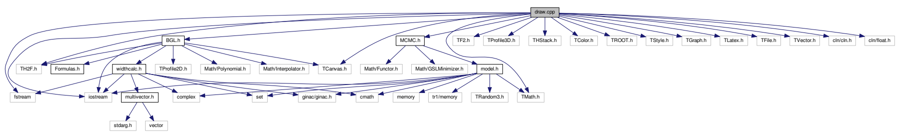
8.4 defs.h

```
#define GL 2
#define GQ 2
#define UL 0
#define UQ 0



```

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>
```

Include dependency graph for draw.cpp:



Classes

- class BGL2

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

Functions

- 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 Function Documentation

8.5.1.1 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

Definition at line 357 of file draw.cpp.

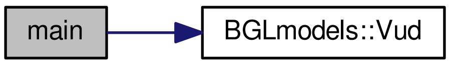
References BGLmodels::Vud().

```
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);
```

```
  //TGraph * chi2=new TGraph(npoints);

   uint npoints=200;
   double init1=-3, final1=3;
   double init2=10, final2=1000;
   double initBmumu=0, finalBmumu=3;
   double initBsmumu=0, finalBsmumu=6;

   double llmax=-1000,McHmax=1000,MRmax=1000,MImax=1000,tbmax=1;

       TFile *f=new TFile(argv[1],"read");
       if(!f->IsOpen()) cout<<"NOFILE"<<endl;
       //f->ShowStreamerInfo();

       TH2F *limits4,*Bmumu_Bsmumu,*limits_tb_MR,*limits_tb_MI;
       TH2F *limits_MR_MI,*limits_MR_McH,*limits_MI_McH;

       f->GetObject("limits4;1",limits4);
       f->GetObject("Bmumu_Bsmumu;1",Bmumu_Bsmumu);
       f->GetObject("limits_tb_MR;1",limits_tb_MR);
       f->GetObject("limits_tb_MI;1",limits_tb_MI);
       f->GetObject("limits_MR_MI;1",limits_MR_MI);
       f->GetObject("limits_MR_McH;1",limits_MR_McH);
       f->GetObject("limits_MI_McH;1",limits_MI_McH);

       TVectorD* vllmax=NULL;

       f->GetObject("vllmax;1",vllmax);
       if(!vllmax) cout<<"ERROR"<<endl;
       llmax=(*vllmax)[0];
       //tbmax=(*vllmax)[1];
       //McHmax=(*vllmax)[2];
       //MRmax=McHmax+(*vllmax)[3];
       //MImax=MRmax+(*vllmax)[4];
       cout<<llmax<<" "<<tbmax<<" "<<McHmax<<" "<<MRmax<<" "<<MImax<<" "<<endl;

 /*BGL2* m=new BGL2(gL,gQ,lup,qup);
 double sm_(m->bsgammawidth(tbmax,McHmax,MRmax,MImax,5));
 double charged_(m->bsgammawidth(tbmax,McHmax,MRmax,MImax,1));
 double neutral_(m->bsgammawidth(tbmax,McHmax,MRmax,MImax,2));
 double neutralR_(m->bsgammawidth(tbmax,McHmax,MRmax,MImax,3));
 double neutralI_(m->bsgammawidth(tbmax,McHmax,MRmax,MImax,4));
 double eK_(m->epsK(tbmax,McHmax,MRmax,MImax));

 double all_(m->bsgammawidth(tbmax,McHmax,MRmax,MImax,0));
 */
       //for(int gL=2;gL>=0;gL--)
       //for(int gQ=2;gQ>=0;gQ--)
       //for(uint lup=0;lup<2;lup++)
       //for(uint qup=0;qup<2;qup++)
       uint min1=npoints, min2=npoints, min3=npoints;
       uint min11=npoints, min21=npoints, min31=npoints;
       uint min12=npoints, min22=npoints, min32=npoints;

       for(uint i=0;i<npoints;i++)
       for(uint j=0;j<npoints;j++){
                       int binmax=limits4->GetBin(i+1,j+1);
                       double rest=limits4->GetBinContent(binmax);
                       if(rest>=llmax) rest=1;
                       else rest=TMath::Prob(-2*(rest-llmax),2);
                       if(rest>=0.05 && j<min1){min1=j;}
                       if(rest>=0.05 && j<min11 && j>uint(180*npoints/990)){min11=j;}
                       if(rest>=0.05 && j<min12 && j>uint(400*npoints/990)){min12=j;}
                       limits4->SetBinContent(i+1,j+1,rest);

                       rest=Bmumu_Bsmumu->GetBinContent(binmax);
                       if(rest>=llmax) rest=1;
                       else rest=TMath::Prob(-2*(rest-llmax),2);
                       //int nn=4;
                       //int ii=(i/nn)*nn, jj=(j/nn)*nn;
                       //for(int iii=ii;iii<ii+n;++iii)
                       //for(int iii=ii;iii<ii+n;++iii)
                       Bmumu_Bsmumu->SetBinContent(i+1,j+1,rest);

                       rest=limits_MR_MI->GetBinContent(binmax);
                       if(rest>=llmax) rest=1;
                       else rest=TMath::Prob(-2*(rest-llmax),2);
                       limits_MR_MI->SetBinContent(i+1,j+1,rest);

                       rest=limits_MR_McH->GetBinContent(binmax);
                       if(rest>=llmax) rest=1;
                       else rest=TMath::Prob(-2*(rest-llmax),2);
                       if(rest>=0.05 && i<min2){min2=i;}
                       if(rest>=0.05 && i<min21 && j>uint(180*npoints/990)){min21=i;}
                       if(rest>=0.05 && i<min22 && j>uint(400*npoints/990)){min22=i;}
                       limits_MR_McH->SetBinContent(i+1,j+1,rest);

```

```
                        rest=limits_MI_McH->GetBinContent(binmax);
                        if(rest>=llmax) rest=1;
                        else rest=TMath::Prob(-2*(rest-llmax),2);
                        if(rest>=0.05 && i<min3){min3=i;}
                        if(rest>=0.05 && i<min31 && j>uint(180*npoints/990)){min31=i;}
                        if(rest>=0.05 && i<min32 && j>uint(400*npoints/990)){min32=i;}
                        limits_MI_McH->SetBinContent(i+1,j+1,rest);


                        rest=limits_tb_MR->GetBinContent(binmax);
                        if(rest>=llmax) rest=1;
                        else rest=TMath::Prob(-2*(rest-llmax),2);
                        limits_tb_MR->SetBinContent(i+1,j+1,rest);
                }
                double mmin1=init2+((min1+0.5)*(final2-init2))/npoints;
                double mmin2=init2+((min2+0.5)*(final2-init2))/npoints;
                double mmin3=init2+((min3+0.5)*(final2-init2))/npoints;

                double mmin11=init2+((min11+0.5)*(final2-init2))/npoints;
                double mmin21=init2+((min21+0.5)*(final2-init2))/npoints;
                double mmin31=init2+((min31+0.5)*(final2-init2))/npoints;

                double mmin12=init2+((min12+0.5)*(final2-init2))/npoints;
                double mmin22=init2+((min22+0.5)*(final2-init2))/npoints;
                double mmin32=init2+((min32+0.5)*(final2-init2))/npoints;

                //mass<<name<<" "<<mmin1<<" "<<mmin2<<" "<<mmin3<<endl;

                ofstream maxs((string("maxs_")+string(name)+string(".out")).c_str());
                maxs<<mmin1<<" "<<mmin2<<" "<<mmin3<<endl;
                maxs<<mmin11<<" "<<mmin21<<" "<<mmin31<<endl;
                maxs<<mmin12<<" "<<mmin22<<" "<<mmin32<<endl;
                maxs<<llmax<<" "<<tbmax<<" "<<McHmax<<" "<<MRmax<<" "<<MImax<<" "<<endl;
                //maxs<<eK_<<endl;
                //maxs<<sm_<<" "<<charged_<<" "<<neutral_<<" "<<neutralR_<<" "<<neutralI_<<" "<<all_<<endl;

                //for(uint j=0;j<npoints;j++)
        //for(uint i=0;i<npoints;i++){
                //      int binmax=limits4->GetBin(i+1,j+1);
                //      maxs<<"("<<i<<","<<j<<"):"<<limits4->GetBinContent(binmax)<<endl;
                //      }

        maxs.close();

           double ma=0,me=.2, x0=1,y0=120;
   gStyle->SetOptTitle(0);
   gStyle->SetPaperSize(10.,10.);
   TCanvas * c21=new TCanvas("c21","",int(800*(1+ma+me)),int(600*(1+ma+me)));
        c21->SetMargin(me,ma,me,ma);
        c21->SetGrid();

    limits4->SetStats(0);
    limits4->GetXaxis()->SetTitle("log_{10}(tan\\beta)");
    limits4->GetYaxis()->SetTitle("M_{H+} (GeV)");

     Bmumu_Bsmumu->SetStats(0);
    Bmumu_Bsmumu->GetXaxis()->SetTitle("Br(B\\to\\mu\\mu)/10^{-10}");
    Bmumu_Bsmumu->GetYaxis()->SetTitle("Br(B_{s}\\to\\mu\\mu)/10^{-9}");

    limits_MR_MI->SetStats(0);
    limits_MR_MI->GetYaxis()->SetTitle("M_{I} (GeV)");
    limits_MR_MI->GetXaxis()->SetTitle("M_{R} (GeV)");


    limits_MR_McH->SetStats(0);
    limits_MR_McH->GetYaxis()->SetTitle("M_{H+} (GeV)");
    limits_MR_McH->GetXaxis()->SetTitle("M_{R} (GeV)");

    limits_MI_McH->SetStats(0);
    limits_MI_McH->GetYaxis()->SetTitle("M_{H+} (GeV)");
    limits_MI_McH->GetXaxis()->SetTitle("M_{I} (GeV)");

    limits_tb_MR->SetStats(0);
    limits_tb_MR->GetXaxis()->SetTitle("log_{10}(tan\\beta)");
    limits_tb_MR->GetYaxis()->SetTitle("M_{R} (GeV)");



   Double_t contours[3];
   contours[0] = 0.003;
   contours[1] = 0.05;
   contours[2] = 0.32;




    limits4->SetContour(3, contours);
    //limits4->GetYaxis()->SetLabelOffset(0.02);
    limits4->GetYaxis()->SetLabelSize(0.08);
    limits4->GetYaxis()->SetTitleSize(0.08);
    limits4->GetYaxis()->SetTitleOffset(1.2);
    limits4->GetYaxis()->SetLimits(1,999);



    //limits4->GetXaxis()->SetLabelOffset(0.02);
    limits4->GetXaxis()->SetLabelSize(0.08);
    limits4->GetXaxis()->SetTitleSize(0.08);
    limits4->GetXaxis()->SetTitleOffset(1.2);
    limits4->GetXaxis()->SetLimits(-2.99,2.99);



    TLatex l;
    l.SetTextSize(0.08);
    string ss=qq[qup][gQ]+","+ll[lup][gL];


        limits4->Draw("CONT Z LIST");
    //limits4->Draw("CONT LIST");
    //limits4->Draw("colz");

    l.DrawLatex(x0,y0,ss.c_str());

    c21->SaveAs((string("pdf_")+string(name)+string(".png")).c_str());

    delete c21;
    //Bmumu_Bsmumu->SetBit(TH1::kCanRebin);
    Bmumu_Bsmumu->Rebin2D(2,2);

    //Bmumu_Bsmumu->SetContour(3, contours);
    //limits4->GetYaxis()->SetLabelOffset(0.02);
    Bmumu_Bsmumu->GetYaxis()->SetLabelSize(0.08);
   Bmumu_Bsmumu->GetYaxis()->SetTitleSize(0.08);
    Bmumu_Bsmumu->GetYaxis()->SetTitleOffset(0.8);
    Bmumu_Bsmumu->GetYaxis()->SetLimits(0.01,4.99);
    //Bmumu_Bsmumu->GetYaxis()->SetRangeUser(0.01, 3.49);
   // Bmumu_Bsmumu->GetYaxis()->SetLimits(0.01,3.49);
    //limits4->GetXaxis()->SetLabelOffset(0.02);
    Bmumu_Bsmumu->GetYaxis()->SetNdivisions(5, kTRUE);
    Bmumu_Bsmumu->GetXaxis()->SetNdivisions(5, kTRUE);

    Bmumu_Bsmumu->GetXaxis()->SetLabelSize(0.08);
    Bmumu_Bsmumu->GetXaxis()->SetTitleSize(0.08);
   Bmumu_Bsmumu->GetXaxis()->SetTitleOffset(1.2);
    Bmumu_Bsmumu->GetXaxis()->SetLimits(0.01,1.99);
    //Bmumu_Bsmumu->GetXaxis()->SetRangeUser(0., 2);

     TCanvas * cB=new TCanvas("cB","",int(800*(1+ma+me)),int(600*(1+ma+me)));
        cB->SetMargin(.14,ma,me,ma);
        cB->SetGrid();
        //limits4->Draw("CONT Z LIST");
    Bmumu_Bsmumu->Draw("COLZ");

    l.DrawLatex(1.5,1,ss.c_str());

    cB->SaveAs((string("Bmumu_Bsmumu_")+string(name)+string(".png")).c_str());

    delete cB;

    limits_MR_MI->SetContour(3, contours);
    //limits4->GetYaxis()->SetLabelOffset(0.02);
    limits_MR_MI->GetYaxis()->SetLabelSize(0.06);
    limits_MR_MI->GetYaxis()->SetTitleSize(0.06);
    limits_MR_MI->GetYaxis()->SetTitleOffset(1.1);
    limits_MR_MI->GetYaxis()->SetLimits(1,999);
    //limits4->GetXaxis()->SetLabelOffset(0.02);
    limits_MR_MI->GetXaxis()->SetLabelSize(0.06);
    limits_MR_MI->GetXaxis()->SetTitleSize(0.06);
    limits_MR_MI->GetXaxis()->SetTitleOffset(1.1);
    limits_MR_MI->GetXaxis()->SetLimits(1,999);

        TCanvas * c3=new TCanvas("c3","",800,600);
        c3->SetMargin(me,ma,me,ma);
        c3->SetGrid();

        limits_MR_MI->Draw("CONT LIST");

    c3->SaveAs((string("pdf_")+string(name)+string("_MRMI.png")).c_str());

        limits_MR_McH->SetContour(3, contours);
        //limits4->GetYaxis()->SetLabelOffset(0.02);
    limits_MR_McH->GetYaxis()->SetLabelSize(0.06);
    limits_MR_McH->GetYaxis()->SetTitleSize(0.06);
    limits_MR_McH->GetYaxis()->SetTitleOffset(1.1);
    limits_MR_McH->GetYaxis()->SetLimits(1,999);
    //limits4->GetXaxis()->SetLabelOffset(0.02);
    limits_MR_McH->GetXaxis()->SetLabelSize(0.06);
    limits_MR_McH->GetXaxis()->SetTitleSize(0.06);
    limits_MR_McH->GetXaxis()->SetTitleOffset(1.1);
    limits_MR_McH->GetXaxis()->SetLimits(1,999);

        TCanvas * c4=new TCanvas("c4","",800,600);
        limits_MR_McH->Draw("CONT LIST");
    c4->SetMargin(me,ma,me,ma);
        c4->SetGrid();
    c4->SaveAs((string("pdf_")+string(name)+string("_MRMcH.png")).c_str());

        limits_MI_McH->SetContour(3, contours);
        //limits4->GetYaxis()->SetLabelOffset(0.02);
    limits_MI_McH->GetYaxis()->SetLabelSize(0.06);
    limits_MI_McH->GetYaxis()->SetTitleSize(0.06);
    limits_MI_McH->GetYaxis()->SetTitleOffset(1.1);
    limits_MI_McH->GetYaxis()->SetLimits(1,999);
    //limits4->GetXaxis()->SetLabelOffset(0.02);
    limits_MI_McH->GetXaxis()->SetLabelSize(0.06);
    limits_MI_McH->GetXaxis()->SetTitleSize(0.06);
    limits_MI_McH->GetXaxis()->SetTitleOffset(1.1);
    limits_MI_McH->GetXaxis()->SetLimits(1,999);

        TCanvas * c6=new TCanvas("c6","",800,600);
        limits_MI_McH->Draw("CONT LIST");
    c6->SetMargin(me,ma,me,ma);
        c6->SetGrid();
    c6->SaveAs((string("pdf_")+string(name)+string("_MIMcH.png")).c_str());

        TCanvas * c5=new TCanvas("c5","",800,600);
        limits_tb_MR->Draw("colz");

    c5->SaveAs((string("pdf_")+string(name)+string("_tbMR.png")).c_str());

    //delete m;
    //mass.close();
    f->Close();
        delete f;
        return 0;

}
```

Here is the call graph for this function:



8.6 draw.cpp

```
#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>

using namespace BGLmodels;

/**
* @brief A second implementation of the BGL model, for testing purposes
*/
class BGL2: public Model{
public:

BGL2(int genL=2,int genQ=2, int lup=0, int qup=0, int mssm=0):
           planck(6.58211928e-25),
           GF("G_F"),
           MZ("M_Z"),
           MW("M_W"),
           Mpip("Mpip",0.1396,"M_{\\pi^+}",domain::real),
           Mpi0("Mpi0",0.1349766,"M_{\\pi^0}",domain::real),
           MBp("MBp",5.279,"M_{B^+}",domain::real),
           MB0("MB0",5.2795,"M_{B^0}",domain::real),
           MBs0("MBs0",5.3663,"M_{B_s^0}",domain::real),
           MKp("MKp",0.493677,"MKp",domain::real),
           MK0("MK0",0.497614,"MK0",domain::real),
           MDp("MDp",1.86957,"MDp",domain::real),
           MD0("MD0",1.86480,"MD0",domain::real),
           MDsp("MDsp",1.96845,"MDsp",domain::real),
           MDs0("MDs0",0),
           Fpi("Fpi",0.132,"Fpi",domain::real),
           FB("FB",0.189,"FB",domain::real),
           FBs("FBs",0.225,"FBs",domain::real),
           FK("FK",0.159,"FK",domain::real),
           FD("FD",0.208,"FD",domain::real),
           FDs("FDs",0.248,"FDs",domain::real),
           //alpha(7.297352e-3*4*M_PI),
           cos2(pow(MW/MZ,2)),
           g(sqrt(GF*8/sqrt(ex(2)))*MW),
           //g(sqrt(4*Pi*alpha/(1-cos2))),
           tanb("tg\\beta"),
           cp("cp"),
           McH("M_{H^+}"),
       MR("M_{R}"),
       MI("M_{I}"),
       Tparam("T_param"),
       Sparam("S_param"),
       QCD1("QCD_1"),
       QCD2("QCD_2"),
           mixes(tanb,cp, genL,genQ, lup, qup, mssm),
           mu("\\mu"),
           BGLtype(4,0),
           mmmax(1000),
           stepsize(1e-2)
           {
        alpha=pow(g,2)*(1-cos2)/(4*Pi);
        replacements.append(GF==1.166371e-5);
        replacements.append(MZ==M_MZ);
        replacements.append(MW==M_MW);

    mixes.appendtolst(replacements);

    replacements.append(Pi==M_PI);
    replacements.append(sqrt(ex(2))==sqrt(2));
        replacements.append(Pi==M_PI);
    replacements.append(sqrt(ex(2))==sqrt(2));

        Boson boson;

        realsymbol q3("q3");
        ex vq3=dirac_slash(q3,4);
        varidx jmu(mu,4,1);

        for(uint i=0;i<2;i++)
                for(uint j=0;j<3;j++)
                        for(uint k=0;k<3;k++){
                                conjtoabs.append(conjugate(mixes.V[i][j][k])==pow(abs(mixes.V[i][j][k]),2)/mixes.V[i][j][k]);
                                }

        //W+ boson
        boson.mass=MW;
        boson.s=sVector;

        for(uint t=tLepton;t<=tQuark;t++) boson.C[t][iUp][iDown][hLeft]=mixes.V[t]*Matrix(g/sqrt(ex(2)));
        Boson wboson=boson;
        bosons.push_back(boson);
        boson.reset();

        //H+ boson
        boson.mass=McH;
        boson.s=sScalar;

        for(uint t=tLepton;t<=tQuark;t++)
        for(uint i=iUp;i<=iDown;i++) boson.C[t][iUp][iDown][i]=mixes.VN[t][i]*Matrix(g/MW/sqrt(ex(2)));
        Boson chiggs=boson;
        bosons.push_back(boson);
        boson.reset();

        for(int b=bosons.size()-1;b>=0;b--){
                boson.mass=bosons[b].mass;
                boson.s=bosons[b].s;
                if(boson.s==sVector)
                        for(uint t=tLepton;t<=tQuark;t++)
                        for(uint i=iUp;i<=iDown;i++)
                        for(uint j=iUp;j<=iDown;j++)
                        for(uint h=hLeft;h<=hRight;h++){
                                boson.C[t][i][j][h]=bosons[b].C[t][j][i][h].conjugate();
                                }
                else for(uint t=tLepton;t<=tQuark;t++)
                        for(uint i=iUp;i<=iDown;i++)
                        for(uint j=iUp;j<=iDown;j++)
                        for(uint h=hLeft;h<=hRight;h++){
                                boson.C[t][i][j][hLeft]=bosons[b].C[t][j][i][hRight].conjugate();
                                boson.C[t][i][j][hRight]=bosons[b].C[t][j][i][hLeft].conjugate();
                                }
                bosons.push_back(boson);
                boson.reset();
                }

        //(R+iI)/sqrt(2) boson
        boson.mass=MR;
        boson.s=sScalar;

        for(uint t=tLepton;t<=tQuark;t++){
                        boson.C[t][iDown][iDown][hRight]=mixes.N[t][iDown]*Matrix(g/MW/ex(2));
                        boson.C[t][iUp][iUp][hLeft]=mixes.N[t][iUp].conjugate()*Matrix(g/MW/ex(2));
                        boson.C[t][iDown][iDown][hLeft]=mixes.N[t][iDown].conjugate()*Matrix(g/MW/ex(2));
                        boson.C[t][iUp][iUp][hRight]=mixes.N[t][iUp]*Matrix(g/MW/ex(2));
                }
        bosons.push_back(boson);
        boson.reset();

        //(R+iI)/sqrt(2) boson
        boson.mass=MI;
        boson.s=sScalar;

        for(uint t=tLepton;t<=tQuark;t++){
                        boson.C[t][iDown][iDown][hRight]=mixes.N[t][iDown]*Matrix(I*g/MW/ex(2));
                        boson.C[t][iUp][iUp][hLeft]=mixes.N[t][iUp].conjugate()*Matrix(I*g/MW/ex(2));
                        boson.C[t][iDown][iDown][hLeft]=mixes.N[t][iDown].conjugate()*Matrix(-I*g/MW/ex(2));
                        boson.C[t][iUp][iUp][hRight]=mixes.N[t][iUp]*Matrix(-I*g/MW/ex(2));
                }
        bosons.push_back(boson);
        boson.reset();

        Fermion electron(tLepton,iDown,fElectron);
        Fermion electronR(tLepton,iDown,fElectron,cParticle,hRight);

        Fermion muon(tLepton,iDown,fMuon);
        Fermion muonR(tLepton,iDown,fMuon,cParticle,hRight);

        Fermion tau(tLepton,iDown,fTau);
        Fermion tauR(tLepton,iDown,fTau,cParticle,hRight);
        Fermion neutrino(tLepton,iUp);
        Fermion neutrinotau(tLepton,iUp,fTau);
        Fermion neutrinomuon(tLepton,iUp,fMuon);
        Fermion neutrinoe(tLepton,iUp,fElectron);

        Fermion up(tQuark,iUp,fElectron);
        Fermion down(tQuark,iDown,fElectron);
        Fermion bottom(tQuark,iDown,fTau);
        Fermion strange(tQuark,iDown,fMuon);
        Fermion charm(tQuark,iUp,fMuon);
        Fermion top(tQuark,iUp,fTau);

        Meson Pi0d(down,down,Mpi0,Fpi);
        Meson Pi0u(down,down,Mpi0,Fpi);
        Meson Pip(up,down,Mpip,Fpi);
        Meson Pim(down,up,Mpip,Fpi);

        Meson K0(down,strange,MK0,FK);
        Meson Kp(up,strange,MKp,FK);

        Meson D0(charm,up,MD0,FD);
        Meson Dp(charm,down,MDp,FD);
        Meson Dsp(charm,strange,MDsp,FDs);

        Meson B0(down,bottom,MB0,FB);
        Meson Bp(up,bottom,MBp,FB);
        Meson Bs0(strange,bottom,MBs0,FBs);

        lst sb;
        //sb.append(mixes.M[tQuark][iUp][0][0]==0);
        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));
        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));

        //cout<<pow(sqrt(2)/8*pow(g/MW,2),2)<<endl;
        //cout<<pow(1.166,2)<<endl;
         double fK=0.156;
    ex KKbar=ex(std::pow(fK,2))*mesonmixing(MK0,strange,down);
    ex eK=0.94*imag_part(KKbar)/3.5e-15/sqrt(2);
    cout<<"KKbar "<<KKbar<<endl;
    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))));
        KKbar=expand(KKbar.evalf());
        ex aKKbar=sqrt(KKbar.real_part()*KKbar.real_part()+KKbar.imag_part()*KKbar.imag_part());

    eK=0.94*imag_part(KKbar)/3.5e-15/sqrt(2);
    eK=eK.subs(replacements).real_part();
        eK=collect_common_factors(expand(eK.evalf()));
        cout<<"eK"<<eK<<endl;
    //add("a_eK",abs(eK),new limitedobs(2.2e-3));
    epsilonK=new calcuex(new limitedobs(2*0.011e-3),abs(eK));


}

~BGL2(){epsilonK->~calcuex();}

parameters generateparameters(int max=0) const{
        parameters p;
        //x=log_10(tanb)
        p.push_back(freeparameter(-3,3,r,stepsize));
        //y=log_10(McH)
        if(max==1) p.push_back(freeparameter(10,10000,r,stepsize));
        else p.push_back(freeparameter(10,mmmax,r,stepsize));
        //log_10(massR)
        p.push_back(freeparameter(-200,200,r,stepsize));
        //log_10(massI)
        p.push_back(freeparameter(-50,50,r,stepsize));

        return p;
}


parameters getlist(const parameters & p) const{
        //cout<<aux<<endl;
        //double c2=(1+sqrt(1-4*sqrt(ex_to<numeric>(mudecay.subs(lst(tanb==exp(p[0].value),McH==p[1].value))).to_double())))/2;

        double x=pow(10.0,p[0].value);
        //double y=pow(10.0,p[1].value);
        //double z=pow(10.0,p[2].value);
        //double w=pow(10.0,p[3].value);

        double y=p[1].value;
        double z=y+p[2].value;
        double w=z+p[3].value;

        parameters pp(p);
        pp[0].value=x;
        pp[2].value+=pp[1].value;
        pp[3].value+=pp[2].value;
        pp.values=vector<double>();
        for(uint i=0; i<4; i++) pp.values.push_back(pp[i].value);
        lst &l=pp.p;
        l=lst(tanb==x,McH==y,MR==z,MI==w);
    l.append(QCD1==inter1.Eval(y));
        l.append(QCD2==inter2.Eval(y));

        for(uint i=0;i<3;i++){
                l.append(Mu[i]==Mu_[i].Eval(log(y)));
                l.append(Md[i]==Md_[i].Eval(log(y)));
        }
        return pp;
}

ex mesonmixing(ex mesonmass, const Fermion& f1, const Fermion& f2) const{

        ex ret=0;

                ex v1=0, v2=0;
                ex mq1=mixes.mass(f1),mq2=mixes.mass(f2);
                ex m2q1=mq1*mq1, m2q2=mq2*mq2;

        for(uint i=0;i<bosons.size();i++)
                if(bosons[i].s==0){
                        ex a=(bosons[i].couplingdaggerR(f2,f1)-bosons[i].couplingdaggerL(f2,f1));
                        v1=v1+pow(a/bosons[i].mass,2);

                        ex b=(bosons[i].couplingdaggerR(f2,f1)+bosons[i].couplingdaggerL(f2,f1));
                        v2=v2+pow(b/bosons[i].mass,2);
                }

        ex fc=mesonmass/(mixes.massnum(f1)+mixes.massnum(f2));
        fc=pow(fc,2);

        ret=2*(-v1*(1+11*fc)+v2*(1+fc))*mesonmass/96;

        return collect_common_factors(ret.subs(conjtoabs));
        //return
 expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
}

double bsgammawidth(double tanb_,double McH_,double MR_,double MI_, int option=0){
        parameters p=generateparameters();
        p[0].value=pow(10.0,tanb_);
        p[1].value=McH_;
        p[2].value=MR_;
        p[3].value=MI_;
        calcubtosgamma2 cal(mixes);

        return cal.width(p,option);
}

double epsK(double tanb_,double McH_,double MR_,double MI_, int option=0){
        parameters p=generateparameters();
        p[0].value=pow(10.0,tanb_);
        p[1].value=McH_;
        p[2].value=MR_;
        p[3].value=MI_;
    p.p=lst(tanb==p[0].value,McH==p[1].value,MR==p[2].value,MI==p[3].value);


         return epsilonK->error(p);
}

const double planck;
const possymbol GF, MZ, MW, Mh;
const constant Mpip, Mpi0, MBp,MB0,MBs0, MKp,MK0,MDp,MD0,MDsp,MDs0;
const constant Fpi, FB,FBs, FK,FD,FDs;
ex cos2, g, alpha;
const possymbol tanb, cp, McH, MR, MI, rho;
const realsymbol Tparam, Sparam, QCD1, QCD2;
possymbol Mu[3],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, iBDtaunu, iBD2taunu;
vector<int> BGLtype;
ROOT::Math::Interpolator inter1, inter2;
ROOT::Math::Interpolator Mu_[3],Md_[3];
double mmmax, stepsize;

calcuex * epsilonK;

};


/**
* @brief 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
*/
int main(int argc, char* argv[]){
    // Check the number of parameters

    if(argc<6){
        std::cerr<<"Usage: "<<argv[0]<<" inputfile gL gQ lup qup"<<std::endl;
        return 1;}
    CD cmu=conj(Vud[1][0])*Vud[1][1];
    CD umu=conj(Vud[0][0])*Vud[0][1];


   cout<<"RATIO "<<cmu*cmu<<endl;
   cout<<umu*umu<<endl;

   int gL=atoi(argv[2]);
   int gQ=atoi(argv[3]);
   int lup=atoi(argv[4]);
   int qup=atoi(argv[5]);
   char name[5]="0000";

   name[0]+=gL;
   name[1]+=gQ;
   name[2]+=lup;
   name[3]+=qup;
   string ll[2][3]={{"#nu_{1}","#nu_{2}","#nu_{3}"},{"e","#mu","#tau"}};
   string qq[2][3]={{"u","c","t"},{"d","s","b"}};
   //Int_t MyPalette[100];
   Double_t r[]    = {1, 0.3};
   Double_t g[]    = {1, 0.3};
   Double_t b[]    = {1, 0.3};
   Double_t stop[] = {0., 1.0};
   TColor::CreateGradientColorTable(2, stop, r, g,b, 100);
   //TH1F * pdf1=new TH1F("pdf1","pdf1",npoints,10,500);
   //TGraph * chi2=new TGraph(npoints);

    uint npoints=200;
    double init1=-3, final1=3;
    double init2=10, final2=1000;
    double initBmumu=0, finalBmumu=3;
    double initBsmumu=0, finalBsmumu=6;

    double llmax=-1000,McHmax=1000,MRmax=1000,MImax=1000,tbmax=1;

        TFile *f=new TFile(argv[1],"read");
        if(!f->IsOpen()) cout<<"NOFILE"<<endl;
        //f->ShowStreamerInfo();

        TH2F *limits4,*Bmumu_Bsmumu,*limits_tb_MR,*limits_tb_MI;
        TH2F *limits_MR_MI,*limits_MR_McH,*limits_MI_McH;

        f->GetObject("limits4;1",limits4);
        f->GetObject("Bmumu_Bsmumu;1",Bmumu_Bsmumu);
        f->GetObject("limits_tb_MR;1",limits_tb_MR);
        f->GetObject("limits_tb_MI;1",limits_tb_MI);
        f->GetObject("limits_MR_MI;1",limits_MR_MI);
        f->GetObject("limits_MR_McH;1",limits_MR_McH);
        f->GetObject("limits_MI_McH;1",limits_MI_McH);

        TVectorD* vllmax=NULL;

        f->GetObject("vllmax;1",vllmax);
        if(!vllmax) cout<<"ERROR"<<endl;
        llmax=(*vllmax)[0];
        //tbmax=(*vllmax)[1];
        //McHmax=(*vllmax)[2];
        //MRmax=McHmax+(*vllmax)[3];
        //MImax=MRmax+(*vllmax)[4];
        cout<<llmax<<" "<<tbmax<<" "<<McHmax<<" "<<MRmax<<" "<<MImax<<" "<<endl;

  /*BGL2* m=new BGL2(gL,gQ,lup,qup);
  double sm_(m->bsgammawidth(tbmax,McHmax,MRmax,MImax,5));
  double charged_(m->bsgammawidth(tbmax,McHmax,MRmax,MImax,1));
  double neutral_(m->bsgammawidth(tbmax,McHmax,MRmax,MImax,2));
  double neutralR_(m->bsgammawidth(tbmax,McHmax,MRmax,MImax,3));
  double neutralI_(m->bsgammawidth(tbmax,McHmax,MRmax,MImax,4));
  double eK_(m->epsK(tbmax,McHmax,MRmax,MImax));

  double all_(m->bsgammawidth(tbmax,McHmax,MRmax,MImax,0));
  */
        //for(int gL=2;gL>=0;gL--)
        //for(int gQ=2;gQ>=0;gQ--)
        //for(uint lup=0;lup<2;lup++)
        //for(uint qup=0;qup<2;qup++)
        uint min1=npoints, min2=npoints, min3=npoints;
        uint min11=npoints, min21=npoints, min31=npoints;
        uint min12=npoints, min22=npoints, min32=npoints;

        for(uint i=0;i<npoints;i++)
        for(uint j=0;j<npoints;j++){
                        int binmax=limits4->GetBin(i+1,j+1);
                        double rest=limits4->GetBinContent(binmax);
                        if(rest>=llmax) rest=1;
                        else rest=TMath::Prob(-2*(rest-llmax),2);
                        if(rest>=0.05 && j<min1){min1=j;}
                        if(rest>=0.05 && j<min11 && j>uint(180*npoints/990)){min11=j;}
                        if(rest>=0.05 && j<min12 && j>uint(400*npoints/990)){min12=j;}
                        limits4->SetBinContent(i+1,j+1,rest);

                        rest=Bmumu_Bsmumu->GetBinContent(binmax);
                        if(rest>=llmax) rest=1;
                        else rest=TMath::Prob(-2*(rest-llmax),2);
                        //int nn=4;
                        //int ii=(i/nn)*nn, jj=(j/nn)*nn;
                        //for(int iii=ii;iii<ii+n;++iii)
                        //for(int iii=ii;iii<ii+n;++iii)
                        Bmumu_Bsmumu->SetBinContent(i+1,j+1,rest);

                        rest=limits_MR_MI->GetBinContent(binmax);
                        if(rest>=llmax) rest=1;
                        else rest=TMath::Prob(-2*(rest-llmax),2);
                        limits_MR_MI->SetBinContent(i+1,j+1,rest);

                        rest=limits_MR_McH->GetBinContent(binmax);
                        if(rest>=llmax) rest=1;
                        else rest=TMath::Prob(-2*(rest-llmax),2);
                        if(rest>=0.05 && i<min2){min2=i;}
                        if(rest>=0.05 && i<min21 && j>uint(180*npoints/990)){min21=i;}
                        if(rest>=0.05 && i<min22 && j>uint(400*npoints/990)){min22=i;}
                        limits_MR_McH->SetBinContent(i+1,j+1,rest);

                        rest=limits_MI_McH->GetBinContent(binmax);
                        if(rest>=llmax) rest=1;
                        else rest=TMath::Prob(-2*(rest-llmax),2);
                        if(rest>=0.05 && i<min3){min3=i;}
                        if(rest>=0.05 && i<min31 && j>uint(180*npoints/990)){min31=i;}
                        if(rest>=0.05 && i<min32 && j>uint(400*npoints/990)){min32=i;}
                        limits_MI_McH->SetBinContent(i+1,j+1,rest);


                        rest=limits_tb_MR->GetBinContent(binmax);
                        if(rest>=llmax) rest=1;
                        else rest=TMath::Prob(-2*(rest-llmax),2);
                        limits_tb_MR->SetBinContent(i+1,j+1,rest);
                }
                double mmin1=init2+((min1+0.5)*(final2-init2))/npoints;
                double mmin2=init2+((min2+0.5)*(final2-init2))/npoints;
                double mmin3=init2+((min3+0.5)*(final2-init2))/npoints;

                double mmin11=init2+((min11+0.5)*(final2-init2))/npoints;
                double mmin21=init2+((min21+0.5)*(final2-init2))/npoints;
                double mmin31=init2+((min31+0.5)*(final2-init2))/npoints;

                double mmin12=init2+((min12+0.5)*(final2-init2))/npoints;
                double mmin22=init2+((min22+0.5)*(final2-init2))/npoints;
                double mmin32=init2+((min32+0.5)*(final2-init2))/npoints;

                //mass<<name<<" "<<mmin1<<" "<<mmin2<<" "<<mmin3<<endl;

                ofstream maxs((string("maxs_")+string(name)+string(".out")).c_str());
                maxs<<mmin1<<" "<<mmin2<<" "<<mmin3<<endl;
                maxs<<mmin11<<" "<<mmin21<<" "<<mmin31<<endl;
                maxs<<mmin12<<" "<<mmin22<<" "<<mmin32<<endl;
                maxs<<llmax<<" "<<tbmax<<" "<<McHmax<<" "<<MRmax<<" "<<MImax<<" "<<endl;
                //maxs<<eK_<<endl;
                //maxs<<sm_<<" "<<charged_<<" "<<neutral_<<" "<<neutralR_<<" "<<neutralI_<<" "<<all_<<endl;

                //for(uint j=0;j<npoints;j++)
        //for(uint i=0;i<npoints;i++){
                //      int binmax=limits4->GetBin(i+1,j+1);
                //      maxs<<"("<<i<<","<<j<<"):"<<limits4->GetBinContent(binmax)<<endl;
```

```
                //      }

        maxs.close();

            double ma=0,me=.2, x0=1,y0=120;
   gStyle->SetOptTitle(0);
   gStyle->SetPaperSize(10.,10.);
   TCanvas * c21=new TCanvas("c21","",int(800*(1+ma+me)),int(600*(1+ma+me)));
        c21->SetMargin(me,ma,me,ma);
        c21->SetGrid();

    limits4->SetStats(0);
    limits4->GetXaxis()->SetTitle("log_{10}(tan\\beta)");
    limits4->GetYaxis()->SetTitle("M_{H+} (GeV)");

     Bmumu_Bsmumu->SetStats(0);
    Bmumu_Bsmumu->GetXaxis()->SetTitle("Br(B\\to\\mu\\mu)/10^{-10}");
    Bmumu_Bsmumu->GetYaxis()->SetTitle("Br(B_{s}\\to\\mu\\mu)/10^{-9}");

    limits_MR_MI->SetStats(0);
    limits_MR_MI->GetYaxis()->SetTitle("M_{I} (GeV)");
    limits_MR_MI->GetXaxis()->SetTitle("M_{R} (GeV)");


    limits_MR_McH->SetStats(0);
    limits_MR_McH->GetYaxis()->SetTitle("M_{H+} (GeV)");
    limits_MR_McH->GetXaxis()->SetTitle("M_{R} (GeV)");

    limits_MI_McH->SetStats(0);
    limits_MI_McH->GetYaxis()->SetTitle("M_{H+} (GeV)");
    limits_MI_McH->GetXaxis()->SetTitle("M_{I} (GeV)");

    limits_tb_MR->SetStats(0);
    limits_tb_MR->GetXaxis()->SetTitle("log_{10}(tan\\beta)");
    limits_tb_MR->GetYaxis()->SetTitle("M_{R} (GeV)");



   Double_t contours[3];
   contours[0] = 0.003;
   contours[1] = 0.05;
   contours[2] = 0.32;




    limits4->SetContour(3, contours);
    //limits4->GetYaxis()->SetLabelOffset(0.02);
    limits4->GetYaxis()->SetLabelSize(0.08);
    limits4->GetYaxis()->SetTitleSize(0.08);
    limits4->GetYaxis()->SetTitleOffset(1.2);
    limits4->GetYaxis()->SetLimits(1,999);



    //limits4->GetXaxis()->SetLabelOffset(0.02);
    limits4->GetXaxis()->SetLabelSize(0.08);
    limits4->GetXaxis()->SetTitleSize(0.08);
    limits4->GetXaxis()->SetTitleOffset(1.2);
    limits4->GetXaxis()->SetLimits(-2.99,2.99);



    TLatex l;
    l.SetTextSize(0.08);
    string ss=qq[qup][gQ]+","+ll[lup][gL];


        limits4->Draw("CONT Z LIST");
    //limits4->Draw("CONT LIST");
    //limits4->Draw("colz");

    l.DrawLatex(x0,y0,ss.c_str());

    c21->SaveAs((string("pdf_")+string(name)+string(".png")).c_str());

    delete c21;
    //Bmumu_Bsmumu->SetBit(TH1::kCanRebin);
    Bmumu_Bsmumu->Rebin2D(2,2);

    //Bmumu_Bsmumu->SetContour(3, contours);
    //limits4->GetYaxis()->SetLabelOffset(0.02);
    Bmumu_Bsmumu->GetYaxis()->SetLabelSize(0.08);
   Bmumu_Bsmumu->GetYaxis()->SetTitleSize(0.08);
    Bmumu_Bsmumu->GetYaxis()->SetTitleOffset(0.8);
    Bmumu_Bsmumu->GetYaxis()->SetLimits(0.01,4.99);
    //Bmumu_Bsmumu->GetYaxis()->SetRangeUser(0.01, 3.49);
```

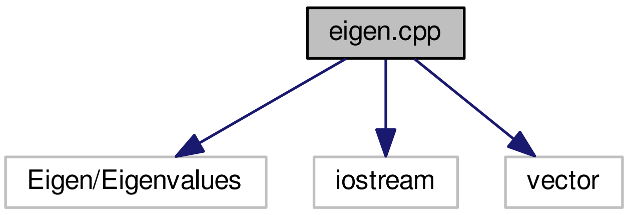
```
  // Bmumu_Bsmumu->GetYaxis()->SetLimits(0.01,3.49);
   //limits4->GetXaxis()->SetLabelOffset(0.02);
   Bmumu_Bsmumu->GetYaxis()->SetNdivisions(5, kTRUE);
   Bmumu_Bsmumu->GetXaxis()->SetNdivisions(5, kTRUE);

   Bmumu_Bsmumu->GetXaxis()->SetLabelSize(0.08);
   Bmumu_Bsmumu->GetXaxis()->SetTitleSize(0.08);
  Bmumu_Bsmumu->GetXaxis()->SetTitleOffset(1.2);
   Bmumu_Bsmumu->GetXaxis()->SetLimits(0.01,1.99);
   //Bmumu_Bsmumu->GetXaxis()->SetRangeUser(0., 2);

    TCanvas * cB=new TCanvas("cB","",int(800*(1+ma+me)),int(600*(1+ma+me)));
       cB->SetMargin(.14,ma,me,ma);
       cB->SetGrid();
       //limits4->Draw("CONT Z LIST");
   Bmumu_Bsmumu->Draw("COLZ");

   l.DrawLatex(1.5,1,ss.c_str());

   cB->SaveAs((string("Bmumu_Bsmumu_")+string(name)+string(".png")).c_str());

   delete cB;

   limits_MR_MI->SetContour(3, contours);
   //limits4->GetYaxis()->SetLabelOffset(0.02);
   limits_MR_MI->GetYaxis()->SetLabelSize(0.06);
   limits_MR_MI->GetYaxis()->SetTitleSize(0.06);
   limits_MR_MI->GetYaxis()->SetTitleOffset(1.1);
   limits_MR_MI->GetYaxis()->SetLimits(1,999);
   //limits4->GetXaxis()->SetLabelOffset(0.02);
   limits_MR_MI->GetXaxis()->SetLabelSize(0.06);
   limits_MR_MI->GetXaxis()->SetTitleSize(0.06);
   limits_MR_MI->GetXaxis()->SetTitleOffset(1.1);
   limits_MR_MI->GetXaxis()->SetLimits(1,999);

       TCanvas * c3=new TCanvas("c3","",800,600);
       c3->SetMargin(me,ma,me,ma);
       c3->SetGrid();

       limits_MR_MI->Draw("CONT LIST");

   c3->SaveAs((string("pdf_")+string(name)+string("_MRMI.png")).c_str());

       limits_MR_McH->SetContour(3, contours);
       //limits4->GetYaxis()->SetLabelOffset(0.02);
   limits_MR_McH->GetYaxis()->SetLabelSize(0.06);
   limits_MR_McH->GetYaxis()->SetTitleSize(0.06);
   limits_MR_McH->GetYaxis()->SetTitleOffset(1.1);
   limits_MR_McH->GetYaxis()->SetLimits(1,999);
   //limits4->GetXaxis()->SetLabelOffset(0.02);
   limits_MR_McH->GetXaxis()->SetLabelSize(0.06);
   limits_MR_McH->GetXaxis()->SetTitleSize(0.06);
   limits_MR_McH->GetXaxis()->SetTitleOffset(1.1);
   limits_MR_McH->GetXaxis()->SetLimits(1,999);

       TCanvas * c4=new TCanvas("c4","",800,600);
       limits_MR_McH->Draw("CONT LIST");
   c4->SetMargin(me,ma,me,ma);
       c4->SetGrid();
   c4->SaveAs((string("pdf_")+string(name)+string("_MRMcH.png")).c_str());

       limits_MI_McH->SetContour(3, contours);
       //limits4->GetYaxis()->SetLabelOffset(0.02);
   limits_MI_McH->GetYaxis()->SetLabelSize(0.06);
   limits_MI_McH->GetYaxis()->SetTitleSize(0.06);
   limits_MI_McH->GetYaxis()->SetTitleOffset(1.1);
   limits_MI_McH->GetYaxis()->SetLimits(1,999);
   //limits4->GetXaxis()->SetLabelOffset(0.02);
   limits_MI_McH->GetXaxis()->SetLabelSize(0.06);
   limits_MI_McH->GetXaxis()->SetTitleSize(0.06);
   limits_MI_McH->GetXaxis()->SetTitleOffset(1.1);
   limits_MI_McH->GetXaxis()->SetLimits(1,999);

       TCanvas * c6=new TCanvas("c6","",800,600);
       limits_MI_McH->Draw("CONT LIST");
   c6->SetMargin(me,ma,me,ma);
       c6->SetGrid();
   c6->SaveAs((string("pdf_")+string(name)+string("_MIMcH.png")).c_str());

       TCanvas * c5=new TCanvas("c5","",800,600);
       limits_tb_MR->Draw("colz");

   c5->SaveAs((string("pdf_")+string(name)+string("_tbMR.png")).c_str());

   //delete m;
   //mass.close();
   f->Close();
```

```
        delete f;
        return 0;

}

```

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.

```
                        {
        double buu = v[0], auu = v[1];
        double add = v[2], bdd = v[3];
        double eu =v[4], gu = v[5];
        double ed = v[6], gd = v[7];
complex<double> bu = buu*exp(complex<double>(0,gu*M_PI_2)), au = auu*exp(complex<double>(0,eu*M_PI_2));
complex<double> ad = add*exp(complex<double>(0,ed*M_PI_2)), bd = bdd*exp(complex<double>(0,gd*M_PI_2));
complex<double> cd = bd - ad;
complex<double> cu =bu + au;

//Matrix3cd X = Matrix3cd::Random(3,3);
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));
comp.push_back(points(sqrt(Dd[1]), 55e-3, 16e-3));
comp.push_back(points(sqrt(Du[1]), 0.619, 0.084));
comp.push_back(points(sqrt(Dd[2]), 2.89, 0.09));
comp.push_back(points(sqrt(Du[2]), 171.7, 3.0));

double chi=0;

for(uint i=0;i<comp.size();i++){
        double aa=(comp[i].prediction-comp[i].measure)/comp[i].error;
        cout<<i<<" "<<aa<<endl;
        chi+=pow((comp[i].prediction-comp[i].measure)/comp[i].error,2);
        }
chi/=comp.size();

cout<<lambda<<" "<<A<<" "<<rho<<" "<<eta<<endl;
cout<<sqrt(Du[2])<<" "<<sqrt(Du[1])<<" "<<sqrt(Du[0])<<endl;
cout<<sqrt(Dd[2])<<" "<<sqrt(Dd[1])<<" "<<sqrt(Dd[0])<<endl;
cout<<chi<<endl;
/*
cout << "The eigenvalues of mu are:" << endl << es.eigenvalues() << endl;
cout << "The matrix of eigenvectors, V, is:" << endl << es.eigenvectors() << endl << endl;
double lambda = es.eigenvalues()[0];
cout << "Consider the first eigenvalue, lambda = " << lambda << endl;
Vector3cd v = es.eigenvectors().col(0);
cout << "If v is the corresponding eigenvector, then lambda * v = " << endl << lambda * v << endl;
cout << "... and A * v = " << endl << A * v << endl << endl;
Matrix3d D = es.eigenvalues().asDiagonal();
Matrix3cd V = es.eigenvectors();
cout << "Finally, V * D * V^(-1) = " << endl << V * D * V.inverse() << endl;
*/
return 0;
}
```

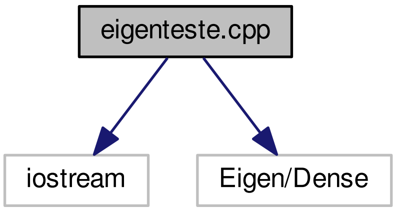
8.8 eigen.cpp

```
#include <Eigen/Eigenvalues>
#include <iostream>
#include <vector>
using namespace std;
using namespace Eigen;

class points{
public:
        points(double p, double m, double e):prediction(p), measure(m),error(e){}
double prediction, measure, error;
};

double chi2(double * v){
        double buu = v[0], auu = v[1];
        double add = v[2], bdd = v[3];
        double eu =v[4], gu = v[5];
        double ed = v[6], gd = v[7];
complex<double> bu = buu*exp(complex<double>(0,gu*M_PI_2)), au = auu*exp(complex<double>(0,eu*M_PI_2));
complex<double> ad = add*exp(complex<double>(0,ed*M_PI_2)), bd = bdd*exp(complex<double>(0,gd*M_PI_2));
complex<double> cd = bd - ad;
complex<double> cu =bu + au;

//Matrix3cd X = Matrix3cd::Random(3,3);
Matrix3cd mu,md;
mu<<1,1,1.0+au+bu,1,1,1.0+bu,1.0+bu+au,1.0+bu,1.0+cu;
md<<1,1,1.0+ad+bd,1,1,1.0+bd,1.0+bd+ad,1.0+bd,1.0+cd;
        mu=mu*v[8];
        md=md*v[9];

Matrix3cd Hu = mu*mu.adjoint(), Hd = md*md.adjoint();
SelfAdjointEigenSolver<Matrix3cd> usolver(Hu), dsolver(Hd);
Vector3d Du=usolver.eigenvalues();//.asDiagonal();
Vector3d Dd=dsolver.eigenvalues();//.asDiagonal();
Matrix3cd VLd=dsolver.eigenvectors().adjoint();
Matrix3cd VLu=usolver.eigenvectors().adjoint();
Matrix3cd Vckm=VLu*VLd.adjoint();
double lambda=sqrt(norm(Vckm(0,1))/(norm(Vckm(0,0))+norm(Vckm(0,1))));
double A=abs(Vckm(1,2))/abs(Vckm(0,1))/lambda;
complex<double> rhoeta=-Vckm(0,0)*conj(Vckm(0,2))/Vckm(1,0)/conj(Vckm(1,2));
double rho=real(rhoeta), eta=imag(rhoeta);

vector<points> comp;
comp.push_back(points(lambda, 0.22535, 0.00065));
comp.push_back(points(A, 0.811, 0.017));
comp.push_back(points(rho, 0.131, 0.02));
comp.push_back(points(eta, 0.345, 0.014));
comp.push_back(points(eta, 0.345, 0.014));
comp.push_back(points(Du[0], 1.27e-3, 0.46e-3));
comp.push_back(points(Dd[0], 2.9e-3, 1.22e-3));
comp.push_back(points(Dd[1], 55e-3, 16e-3));
comp.push_back(points(Du[1], 0.619, 0.084));
comp.push_back(points(Dd[2], 2.89, 0.09));
comp.push_back(points(Du[2], 171.7, 3.0));

double chi=0;

for(uint i=0;i<comp.size();i++){
        chi+=pow((comp[i].prediction-comp[i].measure)/comp[i].error,2);
        }
return chi/comp.size();
}

int main(){
double buu = 0.0162, auu = 0.0009;
double add = 0.018, bdd = 0.09;
//double x = 1.0/3;
double eu = -1.8, gu = -1.0/2;
double ed = -1.0/2, gd = -2;
complex<double> bu = buu*exp(complex<double>(0,gu*M_PI_2)), au = auu*exp(complex<double>(0,eu*M_PI_2));
complex<double> ad = add*exp(complex<double>(0,ed*M_PI_2)), bd = bdd*exp(complex<double>(0,gd*M_PI_2));
complex<double> cd = exp(complex<double>(0,0.09/1.8));

//Matrix3cd X = Matrix3cd::Random(3,3);
Matrix3cd mu,md;
mu<<1,1,1.0+au+bu,1,1,1.0+bu,1.0+bu+au,1.0+bu,1.0+bu;
//md<<1,1,1.0+ad+bd,1,1,1.0+bd,1.0+bd+ad,1.0+bd,1.0+bd;
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);
mu=mu*57.4822;
md=md*1.0147;

Matrix3cd Hu = mu*mu.adjoint(), Hd = md*md.adjoint();
SelfAdjointEigenSolver<Matrix3cd> usolver(Hu), dsolver(Hd);
Vector3d Du=usolver.eigenvalues();//.asDiagonal();
Vector3d Dd=dsolver.eigenvalues();//.asDiagonal();
Matrix3cd VLd=dsolver.eigenvectors().adjoint();
Matrix3cd VLu=usolver.eigenvectors().adjoint();
Matrix3cd Vckm=VLu*VLd.adjoint();
double lambda=sqrt(norm(Vckm(0,1))/(norm(Vckm(0,0))+norm(Vckm(0,1))));
double A=abs(Vckm(1,2))/abs(Vckm(0,1))/lambda;
complex<double> rhoeta=-Vckm(0,0)*conj(Vckm(0,2))/Vckm(1,0)/conj(Vckm(1,2));
double rho=real(rhoeta), eta=imag(rhoeta);
double fu=173.5/sqrt(Du[2]), fd=2.89/sqrt(Dd[2]);

vector<points> comp;
comp.push_back(points(lambda, 0.22535, 0.00065));
comp.push_back(points(A, 0.811, 0.017));
comp.push_back(points(rho, 0.131, 0.02));
comp.push_back(points(eta, 0.345, 0.014));
comp.push_back(points(sqrt(Du[0]), 1.27e-3, 0.46e-3));
comp.push_back(points(sqrt(Dd[0]), 2.9e-3, 1.22e-3));
comp.push_back(points(sqrt(Dd[1]), 55e-3, 16e-3));
comp.push_back(points(sqrt(Du[1]), 0.619, 0.084));
comp.push_back(points(sqrt(Dd[2]), 2.89, 0.09));
comp.push_back(points(sqrt(Du[2]), 171.7, 3.0));

double chi=0;

for(uint i=0;i<comp.size();i++){
        double aa=(comp[i].prediction-comp[i].measure)/comp[i].error;
        cout<<i<<" "<<aa<<endl;
        chi+=pow((comp[i].prediction-comp[i].measure)/comp[i].error,2);
        }
chi/=comp.size();

cout<<lambda<<" "<<A<<" "<<rho<<" "<<eta<<endl;
cout<<sqrt(Du[2])<<" "<<sqrt(Du[1])<<" "<<sqrt(Du[0])<<endl;
cout<<sqrt(Dd[2])<<" "<<sqrt(Dd[1])<<" "<<sqrt(Dd[0])<<endl;
cout<<chi<<endl;
/*
cout << "The eigenvalues of mu are:" << endl << es.eigenvalues() << endl;
cout << "The matrix of eigenvectors, V, is:" << endl << es.eigenvectors() << endl << endl;
double lambda = es.eigenvalues()[0];
cout << "Consider the first eigenvalue, lambda = " << lambda << endl;
Vector3cd v = es.eigenvectors().col(0);
cout << "If v is the corresponding eigenvector, then lambda * v = " << endl << lambda * v << endl;
cout << "... and A * v = " << endl << A * v << endl << endl;
Matrix3d D = es.eigenvalues().asDiagonal();
Matrix3cd V = es.eigenvectors();
cout << "Finally, V * D * V^(-1) = " << endl << V * D * V.inverse() << endl;
*/
return 0;
}
```

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.

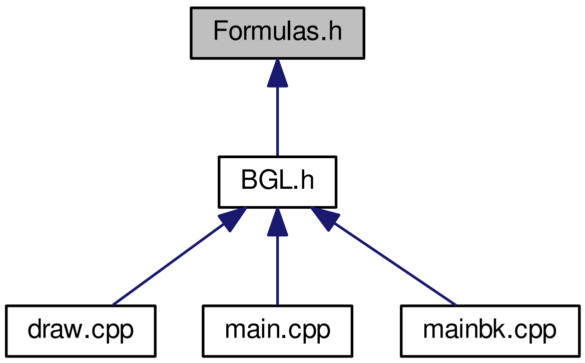
```
00005 {
00006 MatrixXd m(2,2);
00007 m(0,0) = 3;
00008 m(1,0) = 2.5;
00009 m(0,1) = -1;
00010 m(1,1) = m(1,0) + m(0,1);
00011 std::cout << m << std::endl;
00012 }
```

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::calcubtosgamma2

    calculus of the constraints coming from the b->s gamma decay
- class BGLmodels::calcuBmumu

    calculus of the constraints coming from the B->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::f←↩ Any }
- 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

```
#ifndef Formulas_H
#define Formulas_H

/*
#include "widthcalc.h"

#include "TH2F.h"
#include "TProfile2D.h"
#include "TCanvas.h"
#include <iostream>

#include "Math/Polynomial.h"
#include "Math/Interpolator.h"
#include <complex>
#include <cmath>
*/
using namespace std;

namespace BGLmodels{

```

```

enum FType{tLepton,tQuark};
enum FIsospin{iUp,iDown};
enum FFlavour{fElectron,fMuon,fTau,fAny};
enum FCharge{cParticle,cAntiParticle};
enum FHelicity{hLeft,hRight,hAny};
enum BSpin{sScalar, sVector,sAny};

/**
* @brief a fermion properties
*/
class Fermion{
public:

    Fermion(FType t, FIsospin i, FFlavour f=fAny,
FCharge p=cParticle, FHelicity h=hAny): type(t), isospin(i), flavour(f), particle(
p), helicity(h){}

    FType type;
    FIsospin isospin;
    FFlavour flavour;
    FCharge particle;
    FHelicity helicity;
};

/**
* @brief a meson properties
*/
class Meson{
public:

    Meson(const Fermion& qq1, const Fermion& qq2, ex m, ex d): q1(qq1), q2(qq2), mass(m)
, decay_factor(d){}

    Fermion q1, q2;
    ex mass;
    ex decay_factor;
};

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_Ml[3]={0.510998910e-3,105.6583715e-3,1776.82e-3};

typedef std::complex<double> CD;
typedef std::array<CD,3> Vector3c;
typedef std::array<std::array<CD,3>,3> Matrix3c;

/**
* @brief a class to represent the mixing matrices VCKM and VPMNS
*/
class Matrixx: public Matrix3c{
public:
Matrixx(): Matrix3c(){}
Matrixx(const Matrix3c& a): Matrix3c(a){}
///constructs a unitary Matrixx in the standard form
//Matrixx(std::initializer_list<Vector3c> l):

Matrixx(double c12, double c13, double c23,double s12, double s13, double s23, CD e13,CD e13t):
        Matrix3c({
                Vector3c({c12*c13,s12*c13,s13*e13t}),
                Vector3c({-s12*c23-c12*s23*s13*e13,c12*c23-s12*s23*s13*e13,s23*c13}),
                Vector3c({s12*s23-c12*c23*s13*e13,-c12*s23-s12*c23*s13*e13,c13*c23})
                }){}

Matrixx(double t12, double t13, double t23, double d13):
        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))){}
///computes the hermitian conjugate of the Matrixx
const Matrixx conjugate() const{
        Matrixx res;
        for(uint i=0;i<3;i++)
                for(uint j=0;j<3;j++)
                        res[i][j]=std::conj((*this)[j][i]);
        return res;
    }
};

const Matrixx Vnl=Matrixx(33.6*M_PI/180,9.11*M_PI/180,40.4*M_PI/180,M_PI/4).
conjugate();
const Matrixx Vud(13.04*M_PI/180,0.201*M_PI/180,2.38*M_PI/180,1.2);


/**
```

```
* @brief definition of the couplings for the different BGL models
*/
class Mixes{
public:

  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)
  {
          tanb=tanb0;
          M[tLepton][iDown]=Matrix(possymbol("m_e"),possymbol("m_\\mu"),possymbol("m_\\tau"));
          M[tQuark][iUp]=Matrix(possymbol("m_u"),possymbol("m_c"),possymbol("m_t"));
          M[tQuark][iDown]=Matrix(possymbol("m_d"),possymbol("m_s"),possymbol("m_b"));
          const char * ln[3]={"1","2","3"};
          const char * ll[3]={"e","\\mu","\\tau"};
          const char * lu[3]={"u","c","t"};
          const char * ld[3]={"d","s","b"};

        V.push_back(Matrix("U",ln,ll));
        V.push_back(Matrix("V",lu,ld));


        int up[2];
        up[0]=lup;
        up[1]=qup;

        vector< Matrix > delta;

          vector<int> gL(3,0);
          gL[genL]=1;
          //Leptons
          delta.push_back(Matrix(gL[0],gL[1],gL[2]));
          //Quarks
          vector<int> gQ(3,0);
          gQ[genQ]=1;
          delta.push_back(Matrix(gQ[0],gQ[1],gQ[2]));

      for(uint i=0;i<2;i++){
                if(mssm){
                //Nu
            N_[i][0]=0;
        //Nd
        N_[i][1]=0;
        //VNd
        VN_[i][1]=Matrix(tanb)*V[i];
        //Nu*V
        VN_[i][0]=Matrix(1/tanb)*V[i];
        }
        else if(up[i]){
                //Nu
                N_[i][0]=Matrix(tanb)+Matrix(-tanb-1/tanb)*V[i]*delta[i]*V[i].conjugate();
        //Nd
        N_[i][1]=Matrix(tanb)+Matrix(-tanb-1/tanb)*delta[i];
        //VNd
        VN_[i][1]=Matrix(tanb)*V[i]+Matrix(-tanb-1/tanb)*V[i]*delta[i];
        //Nu*V
        VN_[i][0]=Matrix(-1)*(Matrix(tanb)*V[i]+Matrix(-tanb-1/tanb)*V[i]*delta[i]);
                }else{
                //Nu
                N_[i][0]=Matrix(tanb)+Matrix(-tanb-1/tanb)*delta[i];
        //Nd
        N_[i][1]=Matrix(tanb)+Matrix(-tanb-1/tanb)*V[i].conjugate()*delta[i]*V[i];
        //VNd
        VN_[i][1]=Matrix(tanb)*V[i]+Matrix(-tanb-1/tanb)*delta[i]*V[i];
        //Nu*V
        VN_[i][0]=Matrix(-1)*(Matrix(tanb)*V[i]+Matrix(-tanb-1/tanb)*delta[i]*V[i]);
                }

                N[i][0]=N_[i][0]*M[i][0];
                N[i][1]=N_[i][1]*M[i][1];
                VN[i][1]=VN_[i][1]*M[i][1];
                VN[i][0]=M[i][0]*VN_[i][0];
      }
      appendtolst(replacements);


  }
  ex mass(const Fermion& f) const{return M[f.type][f.isospin][f.flavour][f.flavour];}
  double massnum(const Fermion& f) const{return ex_to<numeric>(M[f.type][f.isospin][f.flavour][f.flavour].subs(replacements)).to_double();}

  void appendtolst(lst & reps) const{//,vector<ex>& var_errors
          reps.append(M[0][1][0][0]==0.510998910e-3);
          reps.append(M[0][1][1][1]==105.6583715e-3);
          reps.append(M[0][1][2][2]==1776.82e-3);

          reps.append(M[1][0][0][0]==2.4e-3);
          reps.append(M[1][0][1][1]==1.29);
          reps.append(M[1][0][2][2]==172.9);

          reps.append(M[1][1][0][0]==5.3e-3);
          reps.append(M[1][1][1][1]==95e-3);
          reps.append(M[1][1][2][2]==4.2);

          vector< Matrix > Vn;
        Vn.push_back(Matrix(33.6*M_PI/180,9.11*M_PI/180,40.4*M_PI/180,M_PI/4).conjugate());
        Vn.push_back(Matrix(13.04*M_PI/180,0.201*M_PI/180,2.38*M_PI/180,1.2));
        for(uint i=0; i<2;i++)
                for(uint j=0; j<3;j++)
                        for(uint k=0; k<3;k++)
                                reps.append(V[i][j][k]==Vn[i][j][k]);
          }

  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;
};


/**
* @brief calculus of the constraints coming from the oblique parameters
*/
class calcuOblique:public calcu{
        public:

        calcuOblique(): c1(0.741),c2(0.671), g1(c1*0.02,0.0397), g2(c2*0.02,0.1579) {}
        double operator()(const parameters & p) const{
           double y=p[1].value;
           double z=p[2].value;
           double w=p[3].value;

           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));
           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;

           double T1=c1*TT-c2*Sparam;
       double T2=c2*TT+c1*Sparam;

        return g1.loglikelihood(T1)+g2.loglikelihood(T2);
    }
        double F(double x,double y) const{
         if(x==y) return 0;
         return (x+y)/2-x*y*log(x/y)/(x-y);
        }
    double f(double t,double r) const{
         if(r==0) return 0;
         if(r<0) return 2*sqrt(-r)*atan(sqrt(-r)/t);
         return sqrt(r)*log(fabs((t-sqrt(r))/(t+sqrt(r))));
        }

    double lnxy_xy(double x, double y) const{
         if(x==y) return 1/y;
         return log(x/y)/(x-y);
    }
    double G(double x,double y,double z) const{
         double t=x+y-z;
         double r=std::pow(z,2)-2*z*(x+y)+std::pow(x-y,2);
         return -16.0/3+5*(x+y)/z-2*std::pow((x-y)/z,2)+r/std::pow(z,3)*f(t,r)+\
                        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));
        }
   const double  c1,c2;
   const gauss2obs g1,g2;
};

constexpr double mt_mt=163.3,mt_mW=174.2,mt_mb=261.8;

constexpr double C7SM(double x){
        return ((1/(x-1)+3)*x*std::log(x)+(-8*x*x-5*x+7)/6)*x/4/std::pow(x-1,3);
        }

constexpr double C8SM(double x){
        return (-3/(x-1)*x*std::log(x)+(-x*x+5*x+2)/2)*x/4/std::pow(x-1,3);
        }

constexpr double C7SM_MW=C7SM(std::pow(mt_mW/M_MW,2)),C7SM_Mt=C7SM(std::pow(mt_mt/M_MW,2)),C7SM_Mb=-0.353;
constexpr double C8SM_MW=C8SM(std::pow(mt_mW/M_MW,2)),C8SM_Mt=C8SM(std::pow(mt_mt/M_MW,2)),C8SM_Mb=C8SM(std::pow(mt_mb/M_MW,2));


/**
* @brief calculus of the constraints coming from the b->s gamma decay
*/
class calcubtosgamma2:public calcu{
        public:

constexpr static double calN=2.567e-3;
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=0.5680;
constexpr static double a8er=-0.0601,a87r=0.1923,a7i=0.3546,a8i=-0.0987,aei=2.4997,a87i=-0.0487,a7ei=-0.9067,a8ei=-0.0661;

        calcubtosgamma2(const Mixes& mixes):
          ii(2),
          g1(3.43e-4,sqrt(2)*0.23e-4),
          g2(9.2e-6,4e-6),
          ratio(0){
                //cout<<"C7 "<<C7SM_Mt<<" "<<C7SM_MW<<" "<<C7SM(std::pow(261.8/M_MW,2))<<endl;
                double res[2];
           constexpr double C7SM_[2]={C7SM_Mt,C7SM_Mt};
           constexpr double C8SM_[2]={C8SM_Mt,C8SM_Mt};
           for(uint j=0; j<2; j++){
                const uint i=2;
                const CD epsilon=conj(Vud[0][j])*Vud[0][i]/conj(Vud[2][j])/Vud[2][i];
                const double upsilon=norm(conj(Vud[2][j])*Vud[2][i]/Vud[1][i]);
                const CD R7=(C7SM_Mt)/C7SM_MW;
                const CD R8=(C8SM_Mt)/C8SM_MW;
                const CD R7_=0;
                const CD R8_=0;

                res[j]=a+aee*norm(epsilon)+aer*epsilon.real()+aei*epsilon.imag();
                res[j]+=a77*(norm(R7)+norm(R7_))+a7r*R7.real()+a7i*R7.imag();
                res[j]+=a88*(norm(R8)+norm(R8_))+a8r*R8.real()+a8i*R8.imag();
                res[j]+=a87r*(R8*conj(R7)+R8_*conj(R7_)).real()+a7er*(R7*conj(epsilon)).real()+a8er*(R8*conj(epsilon)).real();
                res[j]+=a87i*(R8*conj(R7)+R8_*conj(R7_)).imag()+a7ei*(R7*conj(epsilon)).imag()+a8er*(R8*conj(epsilon)).imag();
                res[j]*=calN/100*upsilon;
        }
        //cout<<"Btosgamma "<<res[0]/9.2e-6<<" "<<res[1]/3.15e-4<<endl;
                ifstream finter("interpolation.dat");

    if(!finter.is_open()){
                cout<<"ERROR: interpolation.dat not found"<<endl;
                exit(1);
                }
    vector<double> vinter0, vinter1, vinter2;
    while(!finter.eof()){
                double a=0,b=0,c=0;
                finter>>a>>b>>c;
                if(a!=0){
                // cout<<a<<" "<<b<<" "<<c<<endl;
                 vinter0.push_back(a);
                 vinter1.push_back(b);
                 vinter2.push_back(c);
            }
                }

        inter1.SetData(vinter0,vinter1);
        inter2.SetData(vinter0,vinter2);

        finter.close();

            ifstream finter2("masses.dat");

    if(!finter2.is_open()){
                cout<<"ERROR: masses.dat not found"<<endl;
                exit(1);
                }
    vector<vector<double> > m_(7);
    while(!finter2.eof()){
                for(uint i=0; i<7;i++) {
                        double a=0;
                        finter2>>a;

                        if(a!=0) {
                                if(i==0) a=log(a);
                                else if(i<4) a*=1e-3;
                                m_[i].push_back(a);
                        //      cout<<a<<" ";
                        }
                } //cout<<endl;
                }
        for(uint i=0; i<3;i++) {
                Md_[i].SetData(m_[0],m_[2*i+1]);
                Mu_[i].SetData(m_[0],m_[2*i+2]);
        }
//      cout<<"Eval "<<Mu_[2].Eval(log(100.0))<<endl;
//      cout<<"Eval "<<Md_[2].Eval(log(100.0))<<endl;

        finter2.close();

    ifstream finter3("interpolation2.dat");

    if(!finter3.is_open()){
                cout<<"ERROR: interpolation2.dat not found"<<endl;
                exit(1);
                }
    vector<double> vinter20, vinter21, vinter22;
    while(!finter3.eof()){
                double a=0,b=0,c=0;
                finter3>>a>>b>>c;
                if(a!=0){
                // cout<<a<<" "<<b<<" "<<c<<endl;
                 vinter20.push_back(a);
                 vinter21.push_back(b);
                 vinter22.push_back(c);
            }
                }

        inter3.SetData(vinter20,vinter21);
        inter4.SetData(vinter20,vinter22);

        finter3.close();

        vector<ex> vex(24);

           const uint i=ii;
        for(uint j=0;j<2;j++)
        for(uint k=0;k<3;k++){
              vex[j*6+k*2+0]=mixes.VN_[tQuark][iUp][k][j].conjugate()*mixes.
      VN_[tQuark][iUp][k][i];
              vex[j*6+k*2+1]=mixes.N_[tQuark][iDown][j][k]*mixes.N_[
      tQuark][iDown][i][k].conjugate();
              vex[j+k*2+12]=-mixes.VN_[tQuark][iUp][k][j].conjugate()*mixes.
      VN_[tQuark][iDown][k][i];
              vex[j+k*2+18]=mixes.VN_[tQuark][iDown][k][j].conjugate()*mixes.
      VN_[tQuark][iDown][k][i];
                }
        lst l;
    for(uint k=0;k<vex.size();k++){
                vex[k]=vex[k].subs(mixes.replacements).evalf();
                l.append(vex[k].real_part());
                l.append(vex[k].imag_part());
                }
        compile_ex(l, lst(mixes.tanb), fp);
        }

        double operator()(const parameters & p) const{
         double tanb=p[0].value;
         double y=p[1].value;
           double z=p[2].value;
           double w=p[3].value;
        double McH=y, MR=z, MI=w;

        double y0=y;
        if(y<mt_mt) y0=mt_mt;
         double QCD1[2]={inter3.Eval(y0),inter1.Eval(y)};
         double QCD2[2]={inter4.Eval(y0),inter2.Eval(y)};

         double Mu[3],Md[3];

        for(uint i=0;i<3;i++){
                Mu[i]=Mu_[i].Eval(log(y));
                Md[i]=Md_[i].Eval(log(z));
        }
           const uint i=ii;
           CD CC7[2],DD7[2],CC8[2],DD8[2];
           double res[2];
          // constexpr double C7SM_[2]={C7SM_Mt,C7SM_Mb};
          // constexpr double C8SM_[2]={C8SM_Mt,C8SM_Mb};

           std::array<double,48> ret;
           const int n=1,m=48;
```

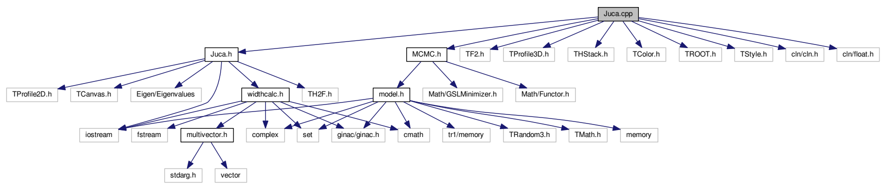
```
   fp(&n,&(tanb),&m,&(ret[0]));
   for(uint j=0;j<2;j++){
        const double mbottom=Md[i];
        const double mstrange=Md[j];
                //ex mbottom=mixes.M[tQuark][iDown][i][i];
                //ex mstrange=mixes.M[tQuark][iDown][j][j];

                CD C7,D7,C8,D8;
                for(uint k=0;k<3;k++){
                double mup=Mu[k];
                double mdown=Md[k];
                //ex mup=mixes.M[tQuark][iUp][k][k];
                //ex mdown=mixes.M[tQuark][iDown][k][k];
                //f1+=
                double mmu=std::pow(mup/McH,2);
                double mmdR=std::pow(mdown/MR,2);
                double mmdI=std::pow(mdown/MI,2);

                double A0u=A0(mmu);
                double A1u=A1(mmu);
                double A2u=A2(mmu);
                double A3u=A3(mmu);
    double A0d=(A0(mmdR)+A0(mmdI));
                double A1d=(A1(mmdR)-A1(mmdI));

                CD f1(ret[j*12+4*k+0],ret[j*12+4*k+1]);
                C7+=f1*A2u;
                C8+=-2.0*f1*A0u;

                CD f2=CD(ret[36+j*2+4*k+0],ret[36+j*2+4*k+1])*mstrange*mbottom/mup/mup;
                //CD f2=f1*mstrange*mbottom/mup/mup;
                D7+=f2*A2u;
                D8+=-2.0*f2*A0u;

                CD f12(ret[24+j*2+4*k+0],ret[24+j*2+4*k+1]);
                C7+=f12*A3u;
                C8+=2.0*f12*A1u;

                CD f4(ret[j*12+4*k+2],ret[j*12+4*k+3]);
                C7+=f4*A0d/3.0;
                C8+=-f4*A0d;

                C7+=f4*A1d/3.0;
                C8+=-f4*A1d;

                CD f6=f4*mstrange*mbottom/mdown/mdown;
                D7+=f6*A0d/3.0;
                D8+=-f6*A0d;
                }
        uint j0=j;
        CC7[j]=(QCD1[j]*C7+QCD2[j]*C8)/2.0/conj(Vud[2][j])/Vud[2][i];
        DD7[j]=(QCD1[j]*D7+QCD2[j]*D8)/2.0/conj(Vud[2][j])/Vud[2][i];
        const double QCD3=(3*QCD2[j]/8+QCD1[j]);
        CC8[j]=QCD3*C8/2.0/conj(Vud[2][j])/Vud[2][i];
        DD8[j]=QCD3*D8/2.0/conj(Vud[2][j])/Vud[2][i];
        const CD epsilon=conj(Vud[0][j])*Vud[0][i]/conj(Vud[2][j])/Vud[2][i];
        const double upsilon=norm(conj(Vud[2][j])*Vud[2][i]/Vud[1][i]);
        const CD R7=(C7SM_Mt+CC7[j])/C7SM_MW;
        const CD R8=(C8SM_Mt+CD(0)*CC8[j])/C8SM_MW;
        const CD R7_=(DD7[j])/C7SM_MW;
        const CD R8_=CD(0)*(DD8[j])/C8SM_MW;


        res[j]=a+aee*norm(epsilon)+aer*epsilon.real()+aei*epsilon.imag();
        res[j]+=a77*(norm(R7)+norm(R7_))+a7r*R7.real()+a7i*R7.imag();
        res[j]+=a88*(norm(R8)+norm(R8_))+a8r*R8.real()+a8i*R8.imag();
        res[j]+=a87r*(R8*conj(R7)+R8_*conj(R7_)).real()+a7er*(R7*conj(epsilon)).real()+a8er*(R8*conj(epsilon)).real();
        res[j]+=a87i*(R8*conj(R7)+R8_*conj(R7_)).imag()+a7ei*(R7*conj(epsilon)).imag()+a8er*(R8*conj(epsilon)).imag();
        res[j]*=calN/100*upsilon;

        /*res[j]=a+aee*norm(epsilon)+aer*epsilon.real()+aei*epsilon.imag();
        res[j]+=a77*(norm(R7)+norm(R7_))+a7r*1+a7i*0;
        res[j]+=a88*(norm(R8)+norm(R8_))+a8r*R8.real()+a8i*R8.imag();
        res[j]+=a87r*1+a7er*(conj(epsilon)).real()+a8er*(R8*conj(epsilon)).real();
        res[j]+=a87i*0+a7ei*(conj(epsilon)).imag()+a8er*(R8*conj(epsilon)).imag();
        res[j]*=calN/100*upsilon;
        */
}
double r1=3.15e-4+0.00247*(norm(CC7[1])+norm(DD7[1])-0.706*CC7[1].real());

//ratio=res[0]/9.2e-6;
//cout<<"RATIO "<<ratio<<endl;
return g1.loglikelihood(r1)+0*g2.loglikelihood(res[0]);
   }
```

```

        double width(const parameters & p, int option=0) const{
         double tanb=p[0].value;
         double y=p[1].value;
           double z=p[2].value;
           double w=p[3].value;
        double McH=y, MR=z, MI=w;

        double y0=y;
        if(y<mt_mt) y0=mt_mt;
         double QCD1[2]={inter3.Eval(y0),inter1.Eval(y)};
         double QCD2[2]={inter4.Eval(y0),inter2.Eval(y)};

         double Mu[3],Md[3];

        for(uint i=0;i<3;i++){
                Mu[i]=Mu_[i].Eval(log(y));
                Md[i]=Md_[i].Eval(log(z));
        }
           const uint i=ii;
           CD CC7[2],DD7[2],CC8[2],DD8[2];
           double res[2];
          // constexpr double C7SM_[2]={C7SM_Mt,C7SM_Mb};
          // constexpr double C8SM_[2]={C8SM_Mt,C8SM_Mb};

           std::array<double,24> ret;
           const int n=1,m=24;
           fp(&n,&(tanb),&m,&(ret[0]));
           for(uint j=0;j<2;j++){
                const double mbottom=Md[i];
                const double mstrange=Md[j];
                        //ex mbottom=mixes.M[tQuark][iDown][i][i];
                        //ex mstrange=mixes.M[tQuark][iDown][j][j];

                        CD C7,D7,C8,D8;
                        for(uint k=0;k<3;k++){
                        double mup=Mu[k];
                        double mdown=Md[k];
                        //ex mup=mixes.M[tQuark][iUp][k][k];
                        //ex mdown=mixes.M[tQuark][iDown][k][k];
                        //f1+=
                        double mmu=std::pow(mup/McH,2);
                        double mmdR=std::pow(mdown/MR,2);
                        double mmdI=std::pow(mdown/MI,2);
                        double A0u=0,A1u=0, A2u=0, A3u=0, A0d=0, A1d=0;

                        if(option==0 || option==1){
                        A0u=A0(mmu);
                        A1u=A1(mmu);
                        A2u=A2(mmu);
                        A3u=A3(mmu);
                        }
            if(option==0 || option==2){
            A0d=(A0(mmdR)+A0(mmdI));
                        A1d=(A1(mmdR)-A1(mmdI));
                        }
                        if(option==3){
            A0d=(A0(mmdR));
                        A1d=(A1(mmdR));
                        }
                        if(option==4){
            A0d=(A0(mmdI));
                        A1d=(-A1(mmdI));
                        }

                        CD f1(ret[j*12+4*k+0],ret[j*12+4*k+1]);
                        C7+=f1*A2u;
                        C8+=-2.0*f1*A0u;

                        CD f2=f1*mstrange*mbottom/mup/mup;
                        D7+=f2*A2u;
                        D8+=-2.0*f2*A0u;

                        C7+=-f1*A3u;
                        C8+=-2.0*f1*A1u;

                        CD f4(ret[j*12+4*k+2],ret[j*12+4*k+3]);
                        C7+=f4*A0d/3.0;
                        C8+=-f4*A0d;

                        C7+=f4*A1d/3.0;
                        C8+=-f4*A1d;

                        CD f6=f4*mstrange*mbottom/mdown/mdown;
                        D7+=f6*A0d/3.0;
                        D8+=-f6*A0d;

                        }
                uint j0=j;
                CC7[j]=(QCD1[j]*C7+QCD2[j]*C8)/2.0/conj(Vud[2][j])/Vud[2][i];
                DD7[j]=(QCD1[j]*D7+QCD2[j]*D8)/2.0/conj(Vud[2][j])/Vud[2][i];
                const double QCD3=(3*QCD2[j]/8+QCD1[j]);
                CC8[j]=QCD3*C8/2.0/conj(Vud[2][j])/Vud[2][i];
                DD8[j]=QCD3*D8/2.0/conj(Vud[2][j])/Vud[2][i];
                const CD epsilon=conj(Vud[0][j])*Vud[0][i]/conj(Vud[2][j])/Vud[2][i];
                const double upsilon=norm(conj(Vud[2][j])*Vud[2][i]/Vud[1][i]);
                const CD R7=(C7SM_Mt+CC7[j])/C7SM_MW;
                const CD R8=(C8SM_Mt+CD(0)*CC8[j])/C8SM_MW;
                const CD R7_=(DD7[j])/C7SM_MW;
                const CD R8_=CD(0)*(DD8[j])/C8SM_MW;


                res[j]=a+aee*norm(epsilon)+aer*epsilon.real()+aei*epsilon.imag();
                res[j]+=a77*(norm(R7)+norm(R7_))+a7r*R7.real()+a7i*R7.imag();
                res[j]+=a88*(norm(R8)+norm(R8_))+a8r*R8.real()+a8i*R8.imag();
                res[j]+=a87r*(R8*conj(R7)+R8_*conj(R7_)).real()+a7er*(R7*conj(epsilon)).real()+a8er*(R8*conj(epsilon)).real();
                res[j]+=a87i*(R8*conj(R7)+R8_*conj(R7_)).imag()+a7ei*(R7*conj(epsilon)).imag()+a8er*(R8*conj(epsilon)).imag();
                res[j]*=calN/100*upsilon;

                /*res[j]=a+aee*norm(epsilon)+aer*epsilon.real()+aei*epsilon.imag();
                res[j]+=a77*(norm(R7)+norm(R7_))+a7r*1+a7i*0;
                res[j]+=a88*(norm(R8)+norm(R8_))+a8r*R8.real()+a8i*R8.imag();
                res[j]+=a87r*1+a7er*(conj(epsilon)).real()+a8er*(R8*conj(epsilon)).real();
                res[j]+=a87i*0+a7ei*(conj(epsilon)).imag()+a8er*(R8*conj(epsilon)).imag();
                res[j]*=calN/100*upsilon;
                */
        }
        double r1=3.15e-4+0.00247*(norm(CC7[1])+norm(DD7[1])-0.706*CC7[1].real());

        //ratio=res[0]/9.2e-6;
        //cout<<"RATIO "<<ratio<<endl;
        return g1.error(r1);
           }

double A0(double x) const{
        return x*(2+3*x-6*x*x+ x*x*x+6*x*std::log(x))/(24*std::pow(1-x,4));
        }

double A1(double x) const{
        return x*(-3+4*x-x*x-2*std::log(x))/(4*std::pow(1-x,3));
        }

double A2(double x) const{
        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));
        }

double A3(double x) const{
        return (-3+8*x-5*x*x+(6*x-4)*std::log(x))*x/(6*std::pow(1-x,3));
        }

ROOT::Math::Interpolator inter1, inter2, inter3, inter4;
ROOT::Math::Interpolator Mu_[3],Md_[3];

const uint ii;
FUNCP_CUBA fp;
 const gauss2obs g1,g2;
mutable double ratio;

};




/**
* @brief calculus of the constraints coming from the B->mu mu decay
*/
class calcuBmumu:public calcu{
        public:
        calcuBmumu(const Mixes& mix, const Meson & m,const Fermion& f3, const Fermion& f4, observable *ob, const char * name):
                meson(m),ff3(f3),ff4(f4),
                o(ob),
                gSr("gSr"),gSi("gSi"),gPr("gPr"),gPi("gPi"),gAr("gAr"),gAi("gAi"), mixes(mix) {
                        const ex Nq=mixes.N[tQuark][m.q2.isospin][m.q1.flavour][m.q2.flavour];
                        const ex Nq_=mixes.N[tQuark][m.q2.isospin][m.q1.flavour][m.q2.flavour].conjugate();
                        const ex Nl=mixes.N[tLepton][f3.isospin][f3.flavour][f4.flavour];
                        const ex Nl_=mixes.N[tLepton][f3.isospin][f4.flavour][f3.flavour].conjugate();
                        possymbol MR("MR"),MI("MI"),McH("McH");
                        ex MR2=MR*MR,MI2=MI*MI,McH2=McH*McH;

                        ex cLL=Nq_*Nl_*(1/MR2-1/MI2);
                        ex cLR=Nq_*Nl*(1/MR2+1/MI2);
                        ex cRL=Nq*Nl_*(1/MR2+1/MI2);
                        ex cRR=Nq_*Nl*(1/MR2-1/MI2);

                        ex ggS=-(2*M_GF/sqrt(2)*(-cRL-cRR+cLL+cLR)/4).subs(mixes.replacements).evalf();
                        ex ggP=-(2*M_GF/sqrt(2)*(+cRL-cRR-cLL+cLR)/4).subs(mixes.replacements).evalf();
                        CD ggA=0;
                        if(m.q2.isospin==iDown && m.q2.flavour==2 && f3.flavour==1 && f4.flavour==1){
                                ggA=-conj(Vud[2][m.q2.flavour])*Vud[2][m.q1.flavour]*Y(std::pow(M_Mu[2]/M_MW,2));
                                ggA+=-conj(Vud[1][m.q2.flavour])*Vud[1][m.q1.flavour]*Y(std::pow(M_Mu[1]/M_MW,2));
                                ggA*=M_GF*M_GF*M_MW*M_MW/M_PI/M_PI/2;

                        }
                        //ex gggA=0;
                        //ex gggA=ggA.real()+I*ggA.imag();

                        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());

                        compile_ex(lst(width), lst(mixes.tanb,McH,MR,MI), fp);

                }
        double operator()(const parameters & p) const{
           //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];
           int n=4,m=1;
           double ret=0;
           fp(&n,&(p.values[0]),&m,&ret);

           return o->loglikelihood(ret);
        }

double obsvalue(const parameters & p) const{
           //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];
           int n=4,m=1;
           double ret=0;
           fp(&n,&(p.values[0]),&m,&ret);

           return ret;
        }

        double Y(double x) const{
                return 1.0113*x/8/(1-x)*(4-x+3*x*log(x)/(1-x));
                }

ex mesondwtest() const{
        const Fermion& f1(meson.q2), f2(meson.q1);
        ex mesonmass=meson.mass;

        Fermion f3=ff3, f4=ff4;
        realsymbol q3("q3"), q4("q4");
        ex s2=pow(mesonmass,2);

                ex v1=0, v2=0;
                ex mq1=mixes.mass(f1),mq2=mixes.mass(f2);
                ex mq3=mixes.mass(f3),mq4=mixes.mass(f4);

                ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;
                scalar_products sp;
                sp.add(q4, q3, (s2-m2q4-m2q3)/2);
                sp.add(q3, q3, m2q3);
                sp.add(q4, q4, m2q4);
                ex q10=(s2+mq1*mq1-mq2*mq2)/(2*sqrt(s2)), lq11=sqrt(q10*q10-mq1*mq1);
                ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq31=sqrt(q30*q30-mq3*mq3);

                ex a;
                a=-(gSr+I*gSi)*s2/(mq1+mq2);
                v1=v1+a*dirac_ONE();
                v2=v2+a.conjugate()*dirac_ONE();
                a=-(gPr+I*gPi)*s2/(mq1+mq2);
                v1=v1+a*dirac_gamma5();
                v2=v2-a.conjugate()*dirac_gamma5();
                ex sl=(dirac_slash(q3,4)+dirac_slash(q4,4));
                a=(gAr+I*gAi);
                v1=v1+a*sl*dirac_gamma5();
                v2=v2+a.conjugate()*sl*dirac_gamma5();

        ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)-mq4*dirac_ONE();
        ex dt=dirac_trace(vq3*v1*vq4*v2).simplify_indexed(sp);
        ex result=expand(dt*4*lq3l/s2/Pi/32);

        lst ltest;
        //ltest.append(conjugate(gL)==pow(abs(gL),2)/gL);
        //ltest.append(conjugate(gR)==pow(abs(gR),2)/gR);
//      ltest.append(conjugate(gS)==pow(abs(gS),2)/gS);
//      ltest.append(conjugate(gP)==pow(abs(gP),2)/gP);
//      ltest.append(conjugate(gA)==pow(abs(gA),2)/gA);

        return pow(meson.decay_factor,2)*collect_common_factors(result.subs(ltest));
        //return expand(ret.subs(lst(exp(-I*wild())==1/exp(I*wild()),sin(wild())==sqrt(1-pow(cos(wild()),2)))));
}
   const Meson meson;
   const Fermion& ff3, ff4;
   shared_ptr<observable> o;
   const realsymbol gSr,gSi, gPr,gPi,gAr,gAi;
   const Mixes mixes;
   FUNCP_CUBA fp;

};



/*

#include "defs.h"


//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)*tanb*M_Ml[i]+(-tanb-1/tanb)*conj(Vnl(GL,i))*Vnl(GL,j)*M_Ml[j]);}

#if UL==1
   #define Nl(i,j)  ((i==j)*(tanb*M_Ml[i]+(-tanb-1/tanb)*M_Ml[i]*(i==GL)))
   #define VnlNl(i,j)  (Vnl[i][j])*M_Ml[j]*(tanb+(-tanb-1/tanb)*(j==GL)))
#else
   #define Nl(i,j)  ((i==j)*tanb*M_Ml[i]+(-tanb-1/tanb)*conj(Vnl[GL][i])*Vnl[GL][j]*M_Ml[j])
   #define VnlNl(i,j)  (Vnl[i][j]*M_Ml[j]*(tanb+(-tanb-1/tanb)*(i==GL)))
#endif

#if UQ==1
   #define Nd(i,j)  (M_Md[j]*(i==j)*(tanb+(-tanb-1/tanb)*(j==GQ)))
   #define Nu(i,j)  (M_Mu[j]*((i==j)*tanb+(-tanb-1/tanb)*Vud[i][GQ]*conj(Vud[j][GQ])))
   #define VudNd(i,j)  (Vud[i][j]*M_Md[j]*(tanb+(-tanb-1/tanb)*(j==GQ)))
   #define NuVud(i,j)  (-M_Mu[i]*Vud[i][j]*(tanb+(-tanb-1/tanb)*(j==GQ)))
#else
   #define Nd(i,j)  (M_Md[j]*((i==j)*tanb+(-tanb-1/tanb)*conj(Vud[GQ][i])*Vud[GQ][j]))
   #define Nu(i,j)  (M_Mu[j]*(i==j)*(tanb+(-tanb-1/tanb)*M_Mu[j]*(j==GQ)))
   #define VudNd(i,j)  (Vud[i][j]*M_Md[j]*(tanb+(-tanb-1/tanb)*(i==GQ)))
   #define NuVud(i,j)  (-M_Mu[i]*Vud[i][j]*(tanb+(-tanb-1/tanb)*(i==GQ)))
#endif


*/
}
#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

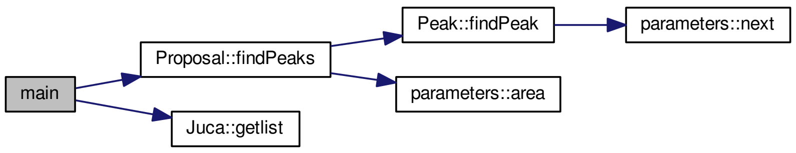
8.13.1.1 int main ( )

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(l);
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

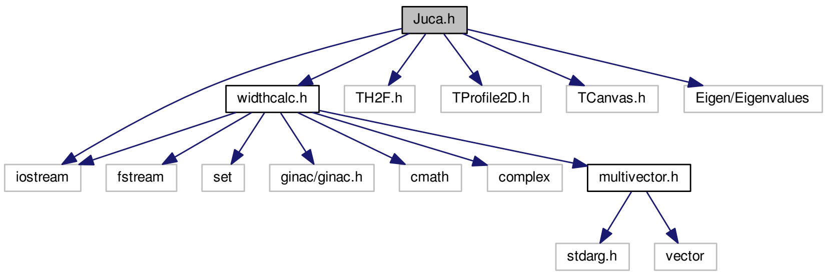
```
#include "MCMC.h"
#include "Juca.h"
#include "TF2.h"
#include "TProfile3D.h"
#include "THStack.h"
#include "TColor.h"
#include "TROOT.h"
#include "TStyle.h"



#include <cln/cln.h>
#include <cln/float.h>


int main(){
        Digits=5;
        cln::cl_inhibit_floating_point_underflow=1;

    //Int_t MyPalette[100];
    Double_t r[]    = {1, 0};
    Double_t g[]    = {1, 0};
    Double_t b[]    = {1, 0};
    Double_t stop[] = {0., 1.0};
    TColor::CreateGradientColorTable(2, stop, r, g,b, 100);

        //TH1F * pdf1=new TH1F("pdf1","pdf1",npoints,10,500);


        //TGraph * chi2=new TGraph(npoints);

        uint npoints=50;


                double llmax=-20,gmax=0;

                Juca* m=new Juca();
        Proposal prop4(m);
        prop4.findPeaks();

        cout<<"gp "<<m->gaussprob(prop4.floatPeak.pr)<<endl;
        lst l=m->getlist(prop4.floatPeak.pr);
        for(uint i=0; i< m->size();i++){
                double mean=m->at(i).calculate(l);
                cout<<i<<" "<<mean<<" "<<sqrt(2*m->at(i).o->loglikelihood(mean))<<endl;
        }
        for(uint i=0; i< prop4.floatPeak.pr.size();i++){
                cout<<i<<" "<<prop4.floatPeak.pr[i].value<<endl;
        }
        delete m;

        return 0;
}

```

8.15 Juca.h File Reference

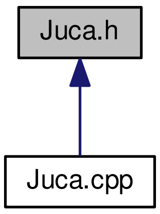
```
#include "widthcalc.h"
```

```
#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

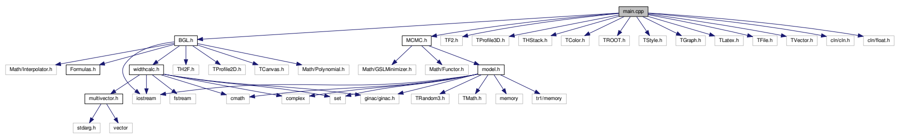
```
#ifndef JUCA_H
#define JUCA_H

#include "widthcalc.h"

#include "TH2F.h"
#include "TProfile2D.h"
#include "TCanvas.h"
#include <iostream>
#include <Eigen/Eigenvalues>

using namespace std;
using namespace Eigen;

```

```
class Juca: public Model{
public:

Juca():Mu("Mu"), Md("Md"), Mc("Mc"), Ms("Ms"), Mt("Mt"), Mb("Mb"), lambda("lambda"), A("A"),rho("rho"),
eta("eta"){

        add("",lambda,new gauss2obs(0.22535,0.00065));
        add("",A,new gauss2obs(0.811, 0.017));
        add("",rho,new gauss2obs(0.131, 0.02));
        add("",eta,new gauss2obs(0.345, 0.014));
        add("",Mu,new gauss2obs(1.27e-3, 0.46e-3));
        add("",Md,new gauss2obs(2.9e-3, 1.22e-3));
        add("",Ms,new gauss2obs(55e-3, 16e-3));
        add("",Mc,new gauss2obs(0.619, 0.084));
        add("",Mb,new gauss2obs(2.89, 0.09));
        add("",Mt,new gauss2obs(171.7, 3.0));
}

~Juca(){}



void add(const char * s, ex pred, observable * ob){
        ex p=collect_common_factors(expand(pred.subs(replacements)));
        push_back(prediction(ob,p));
        }

parameters generateparameters() const{
        parameters p;

        //buu
        p.push_back(freeparameter(-5,0,r));
        //auu
        p.push_back(freeparameter(-5,0,r));
        //add
        p.push_back(freeparameter(-5,0,r));
        //bdd
        p.push_back(freeparameter(-5,0,r));
        //eu
        p.push_back(freeparameter(-2,2,r));
        //gu
        p.push_back(freeparameter(-2,2,r));
        //ed
        p.push_back(freeparameter(-2,2,r));
        //gd
        p.push_back(freeparameter(-2,2,r));
        //nu
        p.push_back(freeparameter(-2,3,r));
        //nd
        p.push_back(freeparameter(-2,3,r));
        //cuu
        //p.push_back(freeparameter(-5,0,r));
        //cdd
        p.push_back(freeparameter(-5,0,r));
        //hu
        //p.push_back(freeparameter(-2,2,r));
        //hd
        p.push_back(freeparameter(-2,2,r));

        return p;
}


lst getlist(const parameters & p) const{


        double buu = pow(10.0,p[0].value), auu = pow(10.0,p[1].value);
        double add = pow(10.0,p[2].value), bdd = pow(10.0,p[3].value);
        double eu =p[4].value, gu = p[5].value;
        double ed = p[6].value, gd = p[7].value;
        //double cuu = pow(10.0,p[10].value);
        double cdd = pow(10.0,p[10].value);
        //double hu =p[12].value;
        double hd= p[11].value;

complex<double> bu = buu*exp(complex<double>(0,gu*M_PI_2)), au = auu*exp(complex<double>(0,eu*M_PI_2));
complex<double> ad = add*exp(complex<double>(0,ed*M_PI_2)), bd = bdd*exp(complex<double>(0,gd*M_PI_2));
//complex<double> cu = cuu*exp(complex<double>(0,hu*M_PI_2));
complex<double> cu = bu;
complex<double> cd = cdd*exp(complex<double>(0,hd*M_PI_2));
//complex<double> cd = bd;

//Matrix3cd X = Matrix3cd::Random(3,3);
Matrix3cd mu,md;
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;
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;
        mu=mu*pow(10.0,p[8].value);
        md=md*pow(10.0,p[9].value);

Matrix3cd Hu = mu*mu.adjoint(), Hd = md*md.adjoint();
SelfAdjointEigenSolver<Matrix3cd> usolver(Hu), dsolver(Hd);
Vector3d Du=usolver.eigenvalues();//.asDiagonal();
Vector3d Dd=dsolver.eigenvalues();//.asDiagonal();
Matrix3cd VLd=dsolver.eigenvectors().adjoint();
Matrix3cd VLu=usolver.eigenvectors().adjoint();
Matrix3cd Vckm=VLu*VLd.adjoint();
double lambda0=sqrt(norm(Vckm(0,1))/(norm(Vckm(0,0))+norm(Vckm(0,1))));
double A0=abs(Vckm(1,2))/abs(Vckm(0,1))/lambda0;
complex<double> rhoeta=-Vckm(0,0)*conj(Vckm(0,2))/Vckm(1,0)/conj(Vckm(1,2));
double rho0=real(rhoeta), eta0=imag(rhoeta);

        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])));
}

const possymbol Mu, Md, Mc, Ms, Mt, Mb, lambda, A;
const realsymbol rho, eta;

lst replacements;
};



#endif
```

8.17 main.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 "TVector.h"
#include <cln/cln.h>
#include <cln/float.h>
```

Include dependency graph for main.cpp:



Functions

- 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

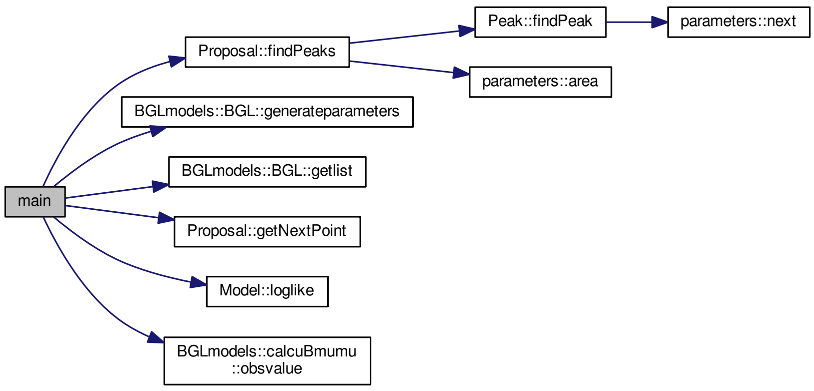
Definition at line 57 of file main.cpp.

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::mmmax, 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->mmmax=300;
00118                 ms[gL][gQ][lup][qup]=m;
00119 
00120                 char name[5]="0000";
```

```
                name[0]+=gL;
                name[1]+=gQ;
                name[2]+=lup;
                name[3]+=qup;

        /*
        TF1 * f1 = new TF1("f1",m,&BGL::BranchingRatio,-3,3,0,"BGL","BranchingRatio");

        TCanvas * c1=new TCanvas("c1","",800,600);
        f1->Draw();
        c1->SaveAs("BR.png");
        TF2 * f2 = new TF2("f2",m,&BGL::topBranchingRatio,-3,3,80,175,0,"BGL","topBranchingRatio");

        TCanvas * c3=new TCanvas("c3","",800,600);
        f2->Draw("colz");
        c3->SaveAs("topBR.png");
        */

        Proposal prop4(m);
        //m->stepsize=1e-4;
        prop4.findPeaks(100,1);
        llmax=log(prop4.floatPeak.lmax);
         tbmax=prop4.floatPeak.pr[0].value;
          McHmax=prop4.floatPeak.pr[1].value;
          MRmax=prop4.floatPeak.pr[2].value;
          MImax=prop4.floatPeak.pr[3].value;

        m->stepsize=1e-4;
        prop4.findPeaks(100);
        double llmax_=log(prop4.floatPeak.lmax);
        if(llmax_>llmax) {llmax=llmax_;
          tbmax=prop4.floatPeak.pr[0].value;
          McHmax=prop4.floatPeak.pr[1].value;
          MRmax=prop4.floatPeak.pr[2].value;
          MImax=prop4.floatPeak.pr[3].value;
        }

        TH2F * limits4=new TH2F("limits4","Likelihood",npoints,init1,final1,npoints,init2,final2);
        TH2F * Bmumu_Bsmumu=new TH2F("Bmumu_Bsmumu","Likelihood",npoints,initBmumu,finalBmumu,npoints,initBsmumu,finalBsmumu);
        TH2F * limits_tb_MR=new TH2F("limits_tb_MR","Likelihood",npoints,init1,final1,npoints,init2,final2);
        TH2F * limits_tb_MI=new TH2F("limits_tb_MI","Likelihood",npoints,init1,final1,npoints,init2,final2);
        TH2F * limits_MR_MI=new TH2F("limits_MR_MI","Likelihood",npoints,init2,final2,npoints,init2,final2);
        TH2F * limits_MR_McH=new TH2F("limits_MR_McH","Likelihood",npoints,init2,final2,npoints,init2,final2);
        TH2F * limits_MI_McH=new TH2F("limits_MI_McH","Likelihood",npoints,init2,final2,npoints,init2,final2);


        for(uint i=0;i<npoints;i++)
                for(uint j=0;j<npoints;j++) {
                        limits4->SetBinContent(i+1,j+1,-1000);
                    Bmumu_Bsmumu->SetBinContent(i+1,j+1,-1000);
                    limits_MR_McH->SetBinContent(i+1,j+1,-1000);
                    limits_MI_McH->SetBinContent(i+1,j+1,-1000);
                        limits_MR_MI->SetBinContent(i+1,j+1,-1000);
                        limits_tb_MR->SetBinContent(i+1,j+1,-1000);
                        plots[i][j]=m->generateparameters();
                        }
        uint steps=40e6;
        //uint steps=npoints*npoints;
        double brtaunu=1;
        for(uint i=steps;i;i--){
                //prop1.getNextPoint();
                //prop2.getNextPoint();
                //prop3.getNextPoint();
                double total=0;
                double gp=0;
                //if(i==steps) cout<<" total "<<m->loglike(prop4.floatPeak.pr)<<endl;
                //else{
                if(i>npoints*npoints){
                        if(i==steps/2) {
                                m->mmmax=1000;
                                m->stepsize=1e-2;
                                prop4.findPeaks(100);
                                llmax_=log(prop4.floatPeak.lmax);
                                 if(llmax_>llmax) {llmax=llmax_;
                                   tbmax=prop4.floatPeak.pr[0].value;
          McHmax=prop4.floatPeak.pr[1].value;
          MRmax=prop4.floatPeak.pr[2].value;
          MImax=prop4.floatPeak.pr[3].value;
        }
                        }
                        if(i<steps) prop4.getNextPoint();
                        total=log(prop4.floatPeak.lmax);
                        //total=m->loglike(prop4.floatPeak.pr);
                        //gp=m->gaussprob(prop4.floatPeak.pr);
                        }
                else{
                        uint x=(i-1)/npoints, y=(i-1)%npoints;
                        prop4.floatPeak.pr[0].value=init1+((x+0.5)*(final1-init1))/npoints;
                        prop4.floatPeak.pr[1].value=init2+((y+0.5)*(final2-init2))/npoints;
                        prop4.floatPeak.pr[2].value=0;
                        prop4.floatPeak.pr[3].value=0;
                        total=m->loglike(prop4.floatPeak.pr);
                        }
                //}


                //gp=m->gaussprob_noT(prop4.floatPeak.pr);
                //double Btaunu=m->loglike(prop1.floatPeak.pr,m->iBtaunu);
                //double BDtaunu=m->loglike(prop2.floatPeak.pr,m->iBDtaunu);
                //double BD2taunu=m->loglike(prop3.floatPeak.pr,m->iBD2taunu);
                double brtaunu0=ex_to<numeric>(m->BR_Htotaunu.subs(m->getlist(prop4.floatPeak.pr).p).evalf()).to_double();
                if(brtaunu0<brtaunu) brtaunu=brtaunu0;
                if(total>llmax) {llmax=total; gmax=gp;
                  tbmax=prop4.floatPeak.pr[0].value;
                  McHmax=prop4.floatPeak.pr[1].value;
                  MRmax=prop4.floatPeak.pr[2].value;
                  MImax=prop4.floatPeak.pr[3].value;
                }



                uint p1=uint((prop4.floatPeak.pr[0].value-init1)/(final1-init1)*npoints);
                double mr=prop4.floatPeak.pr[1].value;
                uint p2=uint((mr-init2)/(final2-init2)*npoints);
                mr+=prop4.floatPeak.pr[2].value;
                uint p3=uint((prop4.floatPeak.pr[2].value+prop4.floatPeak.pr[1].value-init2)/(final2-init2)*npoints);
                mr+=prop4.floatPeak.pr[3].value;
                uint p4=uint((mr-init2)/(final2-init2)*npoints);
                if(p1<npoints && p2<npoints && p3<npoints &&  p4<npoints){
                if(total>likely[p1][p2]){
                        likely[p1][p2]=total;
                        plots[p1][p2].setvalues(prop4.floatPeak.pr);
                        limits4->SetBinContent(p1+1,p2+1,total);
                }
                if(total>likely2[p3][p4]){
                        likely2[p3][p4]=total;
                        //plots[gL][gQ][lup][qup][p1][p2].setvalues(prop4.floatPeak.pr);
                        limits_MR_MI->SetBinContent(p3+1,p4+1,total);
                }

                if(total>likely3[p3][p2]){
                        likely3[p3][p2]=total;
                        //plots[gL][gQ][lup][qup][p1][p2].setvalues(prop4.floatPeak.pr);
                        limits_MR_McH->SetBinContent(p3+1,p2+1,total);
                }

                if(total>likely4[p1][p3]){
                        likely4[p1][p3]=total;
                        //plots[gL][gQ][lup][qup][p1][p2].setvalues(prop4.floatPeak.pr);
                        limits_tb_MR->SetBinContent(p1+1,p3+1,total);
                }
                if(total>likely5[p4][p2]){
                        likely5[p4][p2]=total;
                        //plots[gL][gQ][lup][qup][p1][p2].setvalues(prop4.floatPeak.pr);
                        limits_MI_McH->SetBinContent(p4+1,p2+1,total);
                }
                parameters ppr=m->getlist(prop4.floatPeak.pr);

                double Bmumu=m->cBmumu->obsvalue(ppr)/(1e-10*m->planck/1.519e-12);
                double Bsmumu=m->cBsmumu->obsvalue(ppr)/(1e-9*m->planck/1.516e-12);
                //cout<<Bmumu<<" "<<Bsmumu<<endl;
                uint pB=uint((Bmumu-initBmumu)/(finalBmumu-initBmumu)*npoints);
                uint pBs=uint((Bsmumu-initBsmumu)/(finalBsmumu-initBsmumu)*npoints);
                if(pB<npoints && pBs<npoints)
                if(total>likelyB[pB][pBs]){
                        likely[pB][pBs]=total;
                        Bmumu_Bsmumu->SetBinContent(pB+1,pBs+1,total);
                }

        }else cout<<"OUT!!!"<<endl;

                if(i%100000==0){
                        cout<<"Steps "<<i<<" logtb "<<prop4.floatPeak.pr[0].value<<" logMcH "<<prop4.floatPeak.pr[1].value;
                        cout<<" total "<<total<<" gmax "<<gp<<" "<<brtaunu<<endl;
                        }

                }
        //}

        TFile f((string("h")+string(name)+string(".root")).c_str(),"recreate");
    TVectorD v(5);
        v[0] = llmax;
        v[1]=tbmax;
        v[2]=McHmax;
        v[3]=MRmax;
        v[4]=MImax;
    v.Write("vllmax");
    limits4->Write();
    Bmumu_Bsmumu->Write();
    limits_MR_MI->Write();
    limits_MR_McH->Write();
    limits_MI_McH->Write();
    limits_tb_MR->Write();
        f.Close();

        //for(int gL=2;gL>=0;gL--)
        //for(int gQ=2;gQ>=0;gQ--)
        //for(uint lup=0;lup<2;lup++)
        //for(uint qup=0;qup<2;qup++)
        uint min1=npoints, min2=npoints, min3=npoints;
        uint min11=npoints, min21=npoints, min31=npoints;
        uint min12=npoints, min22=npoints, min32=npoints;

        for(uint i=0;i<npoints;i++)
        for(uint j=0;j<npoints;j++){
                        int binmax=limits4->GetBin(i+1,j+1);
                        double rest=limits4->GetBinContent(binmax);
                        if(rest>=llmax) rest=1;
                        else rest=TMath::Prob(-2*(rest-llmax),2);
                        if(rest>=0.05 && j<min1){min1=j;}
                        if(rest>=0.05 && j<min11 && j>uint(180*npoints/990)){min11=j;}
                        if(rest>=0.05 && j<min12 && j>uint(400*npoints/990)){min12=j;}
                        limits4->SetBinContent(i+1,j+1,rest);

                        rest=Bmumu_Bsmumu->GetBinContent(binmax);
                        if(rest>=llmax) rest=1;
                        else rest=TMath::Prob(-2*(rest-llmax),2);
                        Bmumu_Bsmumu->SetBinContent(i+1,j+1,rest);

                        rest=limits_MR_MI->GetBinContent(binmax);
                        if(rest>=llmax) rest=1;
                        else rest=TMath::Prob(-2*(rest-llmax),2);
                        limits_MR_MI->SetBinContent(i+1,j+1,rest);

                        rest=limits_MR_McH->GetBinContent(binmax);
                        if(rest>=llmax) rest=1;
                        else rest=TMath::Prob(-2*(rest-llmax),2);
                        if(rest>=0.05 && i<min2){min2=i;}
                        if(rest>=0.05 && i<min21 && j>uint(180*npoints/990)){min21=i;}
                        if(rest>=0.05 && i<min22 && j>uint(400*npoints/990)){min22=i;}
                        limits_MR_McH->SetBinContent(i+1,j+1,rest);

                        rest=limits_MI_McH->GetBinContent(binmax);
                        if(rest>=llmax) rest=1;
                        else rest=TMath::Prob(-2*(rest-llmax),2);
                        if(rest>=0.05 && i<min3){min3=i;}
                        if(rest>=0.05 && i<min31 && j>uint(180*npoints/990)){min31=i;}
                        if(rest>=0.05 && i<min32 && j>uint(400*npoints/990)){min32=i;}
                        limits_MI_McH->SetBinContent(i+1,j+1,rest);


                        rest=limits_tb_MR->GetBinContent(binmax);
                        if(rest>=llmax) rest=1;
                        else rest=TMath::Prob(-2*(rest-llmax),2);
                        limits_tb_MR->SetBinContent(i+1,j+1,rest);
                }
                double mmin1=init2+((min1+0.5)*(final2-init2))/npoints;
                double mmin2=init2+((min2+0.5)*(final2-init2))/npoints;
                double mmin3=init2+((min3+0.5)*(final2-init2))/npoints;

                double mmin11=init2+((min11+0.5)*(final2-init2))/npoints;
                double mmin21=init2+((min21+0.5)*(final2-init2))/npoints;
                double mmin31=init2+((min31+0.5)*(final2-init2))/npoints;

                double mmin12=init2+((min12+0.5)*(final2-init2))/npoints;
                double mmin22=init2+((min22+0.5)*(final2-init2))/npoints;
                double mmin32=init2+((min32+0.5)*(final2-init2))/npoints;

                //mass<<name<<" "<<mmin1<<" "<<mmin2<<" "<<mmin3<<endl;

                ofstream maxs((string("maxs_")+string(name)+string(".out")).c_str());
                maxs<<mmin1<<" "<<mmin2<<" "<<mmin3<<endl;
                maxs<<mmin11<<" "<<mmin21<<" "<<mmin31<<endl;
                maxs<<mmin12<<" "<<mmin22<<" "<<mmin32<<endl;
                maxs<<llmax<<endl;
                //for(uint j=0;j<npoints;j++)
        //for(uint i=0;i<npoints;i++){
                //      int binmax=limits4->GetBin(i+1,j+1);
                //      maxs<<"("<<i<<","<<j<<"):"<<limits4->GetBinContent(binmax)<<endl;
                //      }

        maxs.close();

    limits4->SetStats(0);
    limits4->GetXaxis()->SetTitle("log_{10}(tan\\beta)");
    limits4->GetYaxis()->SetTitle("M_{H+} (GeV)");

    Bmumu_Bsmumu->SetStats(0);
    Bmumu_Bsmumu->GetXaxis()->SetTitle("Br(B\\to\\mu\\mu)/10^{-10}");
    Bmumu_Bsmumu->GetYaxis()->SetTitle("Br(B_{s}\\to\\mu\\mu)/10^{-9}");


    limits_MR_MI->SetStats(0);
    limits_MR_MI->GetYaxis()->SetTitle("M_{I} (GeV)");
    limits_MR_MI->GetXaxis()->SetTitle("M_{R} (GeV)");



    limits_MR_McH->SetStats(0);
    limits_MR_McH->GetYaxis()->SetTitle("M_{H+} (GeV)");
    limits_MR_McH->GetXaxis()->SetTitle("M_{R} (GeV)");

    limits_MI_McH->SetStats(0);
    limits_MI_McH->GetYaxis()->SetTitle("M_{H+} (GeV)");
    limits_MI_McH->GetXaxis()->SetTitle("M_{I} (GeV)");

    limits_tb_MR->SetStats(0);
    limits_tb_MR->GetXaxis()->SetTitle("log_{10}(tan\\beta)");
    limits_tb_MR->GetYaxis()->SetTitle("M_{R} (GeV)");


    gStyle->SetOptTitle(0);

   Double_t contours[3];
   contours[0] = 0.003;
   contours[1] = 0.05;
   contours[2] = 0.32;


    double ma=0,me=.2, x0=1,y0=120;

    limits4->SetContour(3, contours);
    //limits4->GetYaxis()->SetLabelOffset(0.02);
    limits4->GetYaxis()->SetLabelSize(0.08);
    limits4->GetYaxis()->SetTitleSize(0.08);
    limits4->GetYaxis()->SetTitleOffset(1.2);
    limits4->GetYaxis()->SetLimits(1,999);
    //limits4->GetXaxis()->SetLabelOffset(0.02);
    limits4->GetXaxis()->SetLabelSize(0.08);
    limits4->GetXaxis()->SetTitleSize(0.08);
    limits4->GetXaxis()->SetTitleOffset(1.2);
    limits4->GetXaxis()->SetLimits(-2.99,2.99);




    gStyle->SetPaperSize(10.,10.);

    TLatex l;
    l.SetTextSize(0.08);
    string ss=qq[qup][gQ]+","+ll[lup][gL];

    TCanvas * c21=new TCanvas("c21","",int(800*(1+ma+me)),int(600*(1+ma+me)));
        c21->SetMargin(me,ma,me,ma);
        c21->SetGrid();
        //limits4->Draw("CONT Z LIST");
    limits4->Draw("CONT Z LIST");
    if(!mssm) l.DrawLatex(x0,y0,ss.c_str());

    c21->SaveAs((string("pdf_")+string(name)+string(".png")).c_str());

    delete c21;

    // Bmumu_Bsmumu->SetContour(3, contours);
    //limits4->GetYaxis()->SetLabelOffset(0.02);
    Bmumu_Bsmumu->GetYaxis()->SetLabelSize(0.08);
    Bmumu_Bsmumu->GetYaxis()->SetTitleSize(0.08);
    Bmumu_Bsmumu->GetYaxis()->SetTitleOffset(1.2);
    Bmumu_Bsmumu->GetYaxis()->SetLimits(0.01,4.99);
    //limits4->GetXaxis()->SetLabelOffset(0.02);
    Bmumu_Bsmumu->GetXaxis()->SetLabelSize(0.08);
    Bmumu_Bsmumu->GetXaxis()->SetTitleSize(0.08);
    Bmumu_Bsmumu->GetXaxis()->SetTitleOffset(1.2);
    Bmumu_Bsmumu->GetXaxis()->SetLimits(0.01,1.99);

    TCanvas * cB=new TCanvas("cB","",int(800*(1+ma+me)),int(600*(1+ma+me)));
        cB->SetMargin(me,ma,me,ma);
        cB->SetGrid();
        //limits4->Draw("CONT Z LIST");
    Bmumu_Bsmumu->Draw("COLZ");
    l.DrawLatex(x0,y0,ss.c_str());

    cB->SaveAs((string("Bmumu_Bsmumu_")+string(name)+string(".png")).c_str());

    delete cB;


    limits_MR_MI->SetContour(3, contours);
    //limits4->GetYaxis()->SetLabelOffset(0.02);
    limits_MR_MI->GetYaxis()->SetLabelSize(0.06);
    limits_MR_MI->GetYaxis()->SetTitleSize(0.06);
    limits_MR_MI->GetYaxis()->SetTitleOffset(1.1);
    limits_MR_MI->GetYaxis()->SetLimits(1,999);
    //limits4->GetXaxis()->SetLabelOffset(0.02);
    limits_MR_MI->GetXaxis()->SetLabelSize(0.06);
    limits_MR_MI->GetXaxis()->SetTitleSize(0.06);
    limits_MR_MI->GetXaxis()->SetTitleOffset(1.1);
    limits_MR_MI->GetXaxis()->SetLimits(1,999);

        TCanvas * c3=new TCanvas("c3","",800,600);
        c3->SetMargin(me,ma,me,ma);
        c3->SetGrid();

        limits_MR_MI->Draw("CONT LIST");

    c3->SaveAs((string("pdf_")+string(name)+string("_MRMI.png")).c_str());

        limits_MR_McH->SetContour(3, contours);
        //limits4->GetYaxis()->SetLabelOffset(0.02);
    limits_MR_McH->GetYaxis()->SetLabelSize(0.06);
    limits_MR_McH->GetYaxis()->SetTitleSize(0.06);
    limits_MR_McH->GetYaxis()->SetTitleOffset(1.1);
    limits_MR_McH->GetYaxis()->SetLimits(1,999);
    //limits4->GetXaxis()->SetLabelOffset(0.02);
    limits_MR_McH->GetXaxis()->SetLabelSize(0.06);
    limits_MR_McH->GetXaxis()->SetTitleSize(0.06);
    limits_MR_McH->GetXaxis()->SetTitleOffset(1.1);
    limits_MR_McH->GetXaxis()->SetLimits(1,999);

        TCanvas * c4=new TCanvas("c4","",800,600);
        limits_MR_McH->Draw("CONT LIST");
    c4->SetMargin(me,ma,me,ma);
        c4->SetGrid();
    c4->SaveAs((string("pdf_")+string(name)+string("_MRMcH.png")).c_str());

        limits_MI_McH->SetContour(3, contours);
        //limits4->GetYaxis()->SetLabelOffset(0.02);
    limits_MI_McH->GetYaxis()->SetLabelSize(0.06);
    limits_MI_McH->GetYaxis()->SetTitleSize(0.06);
    limits_MI_McH->GetYaxis()->SetTitleOffset(1.1);
    limits_MI_McH->GetYaxis()->SetLimits(1,999);
    //limits4->GetXaxis()->SetLabelOffset(0.02);
    limits_MI_McH->GetXaxis()->SetLabelSize(0.06);
    limits_MI_McH->GetXaxis()->SetTitleSize(0.06);
    limits_MI_McH->GetXaxis()->SetTitleOffset(1.1);
    limits_MI_McH->GetXaxis()->SetLimits(1,999);

        TCanvas * c6=new TCanvas("c6","",800,600);
        limits_MI_McH->Draw("CONT LIST");
    c6->SetMargin(me,ma,me,ma);
        c6->SetGrid();
    c6->SaveAs((string("pdf_")+string(name)+string("_MIMcH.png")).c_str());

        TCanvas * c5=new TCanvas("c5","",800,600);
        limits_tb_MR->Draw("colz");

    c5->SaveAs((string("pdf_")+string(name)+string("_tbMR.png")).c_str());


        cout<<"gL "<<gL<<" gQ "<<gQ<<endl;
        cout<<"lup "<<lup<<" qup "<<qup<<endl;
        cout<<"llmax "<<llmax<<" gmax "<<gmax<<endl;


        delete m;
        delete limits4;
        delete Bmumu_Bsmumu;
        delete limits_tb_MR;
        delete limits_tb_MI;
        delete limits_MR_MI;
        delete limits_MI_McH;
        delete limits_MR_McH;

    //mass.close();
        return 0;

}
```

Here is the call graph for this function:



8.18 main.cpp

```
/**
 * @file main.c
 * @author Leonardo Pedro
 * @date May 2014
 * @brief Main file
 *
 * Here typically goes a more extensive explanation of what the header
 * defines. Doxygens tags are words preceeded by either a backslash @\
 * or by an at symbol @@.
 * @see http://www.stack.nl/~dimitri/doxygen/docblocks.html
 * @see http://www.stack.nl/~dimitri/doxygen/commands.html
 */

#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>

using namespace BGLmodels;

/** \mainpage 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 \f$\vec p\f$, we compute the predictions for the considered set of
  observables \f$\vec O_{\mathrm{Th}}(\vec p)\f$.
* Then, using the experimental information \f$\vec O_{\mathrm{Exp}}\f$ available for those observables, we
  build a likelihood function \f$\mathcal L(\vec O_{\mathrm{Exp}}|\vec O_{\mathrm{Th}}(\vec p))\f$
* which gives the probability of obtaining the experimental results \f$\vec O_{\mathrm{\mathrm{Exp}}}\f$
  assuming that the model is correct.
* The likelihood function \f$\mathcal L(\vec O_{\mathrm{Exp}}|\vec O_{\mathrm{Th}}(\vec p))\f$
* encodes all the information on how the model is able to reproduce the observed data all over parameter
  space.
* Nevertheless, the knowledge of \f$\mathcal L(\vec O_{\mathrm{Exp}}|\vec O_{\mathrm{Th}}(\vec p))\f$ 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 \f$\chi^2\equiv -2\log \mathcal L\f$ --
* 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. *
*/


/**
* @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
*/
int main(int argc, char* argv[]){
    // Check the number of parameters
    if(argc < 5){
        std::cerr<<"Usage: "<<argv[0]<<" gL gQ lup qup [mssm]"<<std::endl;
        return 1;}


    int gL=atoi(argv[1]);
    int gQ=atoi(argv[2]);
    int lup=atoi(argv[3]);
    int qup=atoi(argv[4]);
    int mssm=0;
    if(argc>5) mssm=atoi(argv[5]);


   Digits=5;
   cln::cl_inhibit_floating_point_underflow=1;

   string ll[2][3]={{"#nu_{1}","#nu_{2}","#nu_{3}"},{"e","#mu","#tau"}};
   string qq[2][3]={{"u","c","t"},{"d","s","b"}};
   //Int_t MyPalette[100];
   Double_t r[]    = {1, 0.3};
   Double_t g[]    = {1, 0.3};
   Double_t b[]    = {1, 0.3};
   Double_t stop[] = {0., 1.0};
   TColor::CreateGradientColorTable(2, stop, r, g,b, 100);
   //TH1F * pdf1=new TH1F("pdf1","pdf1",npoints,10,500);
   //TGraph * chi2=new TGraph(npoints);

    uint npoints=200;
    double init1=-3, final1=3;
    double init2=10, final2=1000;
    double initBmumu=0, finalBmumu=2;
    double initBsmumu=0, finalBsmumu=5;

        multivector<BGL *,4> ms(0,3,3,2,2);
        multivector<parameters,2> plots(parameters(),npoints,npoints);
        multivector<double,2> likely(-1000,npoints,npoints);
        multivector<double,2> likelyB(-1000,npoints,npoints);
        multivector<double,2> likely2(-1000,npoints,npoints);
        multivector<double,2> likely3(-1000,npoints,npoints);
        multivector<double,2> likely4(-1000,npoints,npoints);
        multivector<double,2> likely5(-1000,npoints,npoints);
        multivector<double,2> likely6(-1000,npoints,npoints);

        //ofstream mass("mass.out");

        //for(int gL=2;gL>=0;gL--)
        //for(int gQ=2;gQ>=0;gQ--)
        //for(int lup=0;lup<2;lup++)
        //for(int qup=0;qup<2;qup++){

//      for(uint qup=0;qup<2;qup++)
//for(uint gL=0;gL<3;gL++)
```

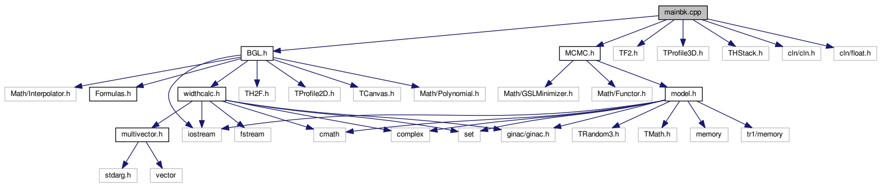
```
//for(uint lup=0;lup<2;lup++)
//for(uint gQ=0;gQ<3;gQ++){
                //if(gL==0 && gQ==2 && lup==0 && qup==1) {t=1; continue;}
                //if(t==0) continue;
        double llmax=-1000,gmax=0, McHmax=1000,MRmax=1000,MImax=1000,tbmax=1;
                BGL* m=new BGL(gL,gQ,lup,qup);
                m->mmmax=300;
                ms[gL][gQ][lup][qup]=m;

                char name[5]="0000";
                name[0]+=gL;
                name[1]+=gQ;
                name[2]+=lup;
                name[3]+=qup;

        /*
        TF1 * f1 = new TF1("f1",m,&BGL::BranchingRatio,-3,3,0,"BGL","BranchingRatio");

        TCanvas * c1=new TCanvas("c1","",800,600);
        f1->Draw();
        c1->SaveAs("BR.png");
        TF2 * f2 = new TF2("f2",m,&BGL::topBranchingRatio,-3,3,80,175,0,"BGL","topBranchingRatio");

        TCanvas * c3=new TCanvas("c3","",800,600);
        f2->Draw("colz");
        c3->SaveAs("topBR.png");
        */

        Proposal prop4(m);
        //m->stepsize=1e-4;
        prop4.findPeaks(100,1);
        llmax=log(prop4.floatPeak.lmax);
         tbmax=prop4.floatPeak.pr[0].value;
          McHmax=prop4.floatPeak.pr[1].value;
          MRmax=prop4.floatPeak.pr[2].value;
          MImax=prop4.floatPeak.pr[3].value;

        m->stepsize=1e-4;
        prop4.findPeaks(100);
        double llmax_=log(prop4.floatPeak.lmax);
        if(llmax_>llmax) {llmax=llmax_;
          tbmax=prop4.floatPeak.pr[0].value;
          McHmax=prop4.floatPeak.pr[1].value;
          MRmax=prop4.floatPeak.pr[2].value;
          MImax=prop4.floatPeak.pr[3].value;
        }

        TH2F * limits4=new TH2F("limits4","Likelihood",npoints,init1,final1,npoints,init2,final2);
        TH2F * Bmumu_Bsmumu=new TH2F("Bmumu_Bsmumu","Likelihood",npoints,initBmumu,finalBmumu,npoints,initBsmumu,finalBsmumu);
        TH2F * limits_tb_MR=new TH2F("limits_tb_MR","Likelihood",npoints,init1,final1,npoints,init2,final2);
        TH2F * limits_tb_MI=new TH2F("limits_tb_MI","Likelihood",npoints,init1,final1,npoints,init2,final2);
        TH2F * limits_MR_MI=new TH2F("limits_MR_MI","Likelihood",npoints,init2,final2,npoints,init2,final2);
        TH2F * limits_MR_McH=new TH2F("limits_MR_McH","Likelihood",npoints,init2,final2,npoints,init2,final2);
        TH2F * limits_MI_McH=new TH2F("limits_MI_McH","Likelihood",npoints,init2,final2,npoints,init2,final2);


        for(uint i=0;i<npoints;i++)
                for(uint j=0;j<npoints;j++) {
                        limits4->SetBinContent(i+1,j+1,-1000);
                    Bmumu_Bsmumu->SetBinContent(i+1,j+1,-1000);
                    limits_MR_McH->SetBinContent(i+1,j+1,-1000);
                    limits_MI_McH->SetBinContent(i+1,j+1,-1000);
                        limits_MR_MI->SetBinContent(i+1,j+1,-1000);
                        limits_tb_MR->SetBinContent(i+1,j+1,-1000);
                        plots[i][j]=m->generateparameters();
                        }
        uint steps=40e6;
        //uint steps=npoints*npoints;
        double brtaunu=1;
        for(uint i=steps;i;i--){
                //prop1.getNextPoint();
                //prop2.getNextPoint();
                //prop3.getNextPoint();
                double total=0;
                double gp=0;
                //if(i==steps) cout<<" total "<<m->loglike(prop4.floatPeak.pr)<<endl;
                //else{
                if(i>npoints*npoints){
                        if(i==steps/2) {
                                m->mmmax=1000;
                                m->stepsize=1e-2;
                                prop4.findPeaks(100);
                                llmax_=log(prop4.floatPeak.lmax);
                                 if(llmax_>llmax) {llmax=llmax_;
                                   tbmax=prop4.floatPeak.pr[0].value;
          McHmax=prop4.floatPeak.pr[1].value;
          MRmax=prop4.floatPeak.pr[2].value;
          MImax=prop4.floatPeak.pr[3].value;
        }
                        }
                        if(i<steps) prop4.getNextPoint();
                        total=log(prop4.floatPeak.lmax);
                        //total=m->loglike(prop4.floatPeak.pr);
                        //gp=m->gaussprob(prop4.floatPeak.pr);
                        }
                else{
                        uint x=(i-1)/npoints, y=(i-1)%npoints;
                        prop4.floatPeak.pr[0].value=init1+((x+0.5)*(final1-init1))/npoints;
                        prop4.floatPeak.pr[1].value=init2+((y+0.5)*(final2-init2))/npoints;
                        prop4.floatPeak.pr[2].value=0;
                        prop4.floatPeak.pr[3].value=0;
                        total=m->loglike(prop4.floatPeak.pr);
                        }
                //}


                //gp=m->gaussprob_noT(prop4.floatPeak.pr);
                //double Btaunu=m->loglike(prop1.floatPeak.pr,m->iBtaunu);
                //double BDtaunu=m->loglike(prop2.floatPeak.pr,m->iBDtaunu);
                //double BD2taunu=m->loglike(prop3.floatPeak.pr,m->iBD2taunu);
                double brtaunu0=ex_to<numeric>(m->BR_Htotaunu.subs(m->getlist(prop4.floatPeak.pr).p).evalf()).to_double();
                if(brtaunu0<brtaunu) brtaunu=brtaunu0;
                if(total>llmax) {llmax=total; gmax=gp;
                  tbmax=prop4.floatPeak.pr[0].value;
                  McHmax=prop4.floatPeak.pr[1].value;
                  MRmax=prop4.floatPeak.pr[2].value;
                  MImax=prop4.floatPeak.pr[3].value;
                }



                uint p1=uint((prop4.floatPeak.pr[0].value-init1)/(final1-init1)*npoints);
                double mr=prop4.floatPeak.pr[1].value;
                uint p2=uint((mr-init2)/(final2-init2)*npoints);
                mr+=prop4.floatPeak.pr[2].value;
                uint p3=uint((prop4.floatPeak.pr[2].value+prop4.floatPeak.pr[1].value-init2)/(final2-init2)*npoints);
                mr+=prop4.floatPeak.pr[3].value;
                uint p4=uint((mr-init2)/(final2-init2)*npoints);
                if(p1<npoints && p2<npoints && p3<npoints &&  p4<npoints){
                if(total>likely[p1][p2]){
                        likely[p1][p2]=total;
                        plots[p1][p2].setvalues(prop4.floatPeak.pr);
                        limits4->SetBinContent(p1+1,p2+1,total);
                }
                if(total>likely2[p3][p4]){
                        likely2[p3][p4]=total;
                        //plots[gL][gQ][lup][qup][p1][p2].setvalues(prop4.floatPeak.pr);
                        limits_MR_MI->SetBinContent(p3+1,p4+1,total);
                }

                if(total>likely3[p3][p2]){
                        likely3[p3][p2]=total;
                        //plots[gL][gQ][lup][qup][p1][p2].setvalues(prop4.floatPeak.pr);
                        limits_MR_McH->SetBinContent(p3+1,p2+1,total);
                }

                if(total>likely4[p1][p3]){
                        likely4[p1][p3]=total;
                        //plots[gL][gQ][lup][qup][p1][p2].setvalues(prop4.floatPeak.pr);
                        limits_tb_MR->SetBinContent(p1+1,p3+1,total);
                }
                if(total>likely5[p4][p2]){
                        likely5[p4][p2]=total;
                        //plots[gL][gQ][lup][qup][p1][p2].setvalues(prop4.floatPeak.pr);
                        limits_MI_McH->SetBinContent(p4+1,p2+1,total);
                }
                parameters ppr=m->getlist(prop4.floatPeak.pr);

                double Bmumu=m->cBmumu->obsvalue(ppr)/(1e-10*m->planck/1.519e-12);
                double Bsmumu=m->cBsmumu->obsvalue(ppr)/(1e-9*m->planck/1.516e-12);
                //cout<<Bmumu<<" "<<Bsmumu<<endl;
                uint pB=uint((Bmumu-initBmumu)/(finalBmumu-initBmumu)*npoints);
                uint pBs=uint((Bsmumu-initBsmumu)/(finalBsmumu-initBsmumu)*npoints);
                if(pB<npoints && pBs<npoints)
                if(total>likelyB[pB][pBs]){
                        likely[pB][pBs]=total;
                        Bmumu_Bsmumu->SetBinContent(pB+1,pBs+1,total);
                }

        }else cout<<"OUT!!!"<<endl;

                if(i%100000==0){
                        cout<<"Steps "<<i<<" logtb "<<prop4.floatPeak.
pr[0].value<<" logMcH "<<prop4.floatPeak.pr[1].value;
                        cout<<" total "<<total<<" gmax "<<gp<<" "<<brtaunu<<endl;
                        }

                }
        //}

        TFile f((string("h")+string(name)+string(".root")).c_str(),"recreate");
    TVectorD v(5);
        v[0] = llmax;
        v[1]=tbmax;
        v[2]=McHmax;
        v[3]=MRmax;
        v[4]=MImax;
    v.Write("vllmax");
    limits4->Write();
    Bmumu_Bsmumu->Write();
    limits_MR_MI->Write();
    limits_MR_McH->Write();
    limits_MI_McH->Write();
    limits_tb_MR->Write();
        f.Close();

        //for(int gL=2;gL>=0;gL--)
        //for(int gQ=2;gQ>=0;gQ--)
        //for(uint lup=0;lup<2;lup++)
        //for(uint qup=0;qup<2;qup++)
        uint min1=npoints, min2=npoints, min3=npoints;
        uint min11=npoints, min21=npoints, min31=npoints;
        uint min12=npoints, min22=npoints, min32=npoints;

        for(uint i=0;i<npoints;i++)
        for(uint j=0;j<npoints;j++){
                        int binmax=limits4->GetBin(i+1,j+1);
                        double rest=limits4->GetBinContent(binmax);
                        if(rest>=llmax) rest=1;
                        else rest=TMath::Prob(-2*(rest-llmax),2);
                        if(rest>=0.05 && j<min1){min1=j;}
                        if(rest>=0.05 && j<min11 && j>uint(180*npoints/990)){min11=j;}
                        if(rest>=0.05 && j<min12 && j>uint(400*npoints/990)){min12=j;}
                        limits4->SetBinContent(i+1,j+1,rest);

                        rest=Bmumu_Bsmumu->GetBinContent(binmax);
                        if(rest>=llmax) rest=1;
                        else rest=TMath::Prob(-2*(rest-llmax),2);
                        Bmumu_Bsmumu->SetBinContent(i+1,j+1,rest);

                        rest=limits_MR_MI->GetBinContent(binmax);
                        if(rest>=llmax) rest=1;
                        else rest=TMath::Prob(-2*(rest-llmax),2);
                        limits_MR_MI->SetBinContent(i+1,j+1,rest);

                        rest=limits_MR_McH->GetBinContent(binmax);
                        if(rest>=llmax) rest=1;
                        else rest=TMath::Prob(-2*(rest-llmax),2);
                        if(rest>=0.05 && i<min2){min2=i;}
                        if(rest>=0.05 && i<min21 && j>uint(180*npoints/990)){min21=i;}
                        if(rest>=0.05 && i<min22 && j>uint(400*npoints/990)){min22=i;}
                        limits_MR_McH->SetBinContent(i+1,j+1,rest);

                        rest=limits_MI_McH->GetBinContent(binmax);
                        if(rest>=llmax) rest=1;
                        else rest=TMath::Prob(-2*(rest-llmax),2);
                        if(rest>=0.05 && i<min3){min3=i;}
                        if(rest>=0.05 && i<min31 && j>uint(180*npoints/990)){min31=i;}
                        if(rest>=0.05 && i<min32 && j>uint(400*npoints/990)){min32=i;}
                        limits_MI_McH->SetBinContent(i+1,j+1,rest);


                        rest=limits_tb_MR->GetBinContent(binmax);
                        if(rest>=llmax) rest=1;
                        else rest=TMath::Prob(-2*(rest-llmax),2);
                        limits_tb_MR->SetBinContent(i+1,j+1,rest);
                }
                double mmin1=init2+((min1+0.5)*(final2-init2))/npoints;
                double mmin2=init2+((min2+0.5)*(final2-init2))/npoints;
                double mmin3=init2+((min3+0.5)*(final2-init2))/npoints;

                double mmin11=init2+((min11+0.5)*(final2-init2))/npoints;
                double mmin21=init2+((min21+0.5)*(final2-init2))/npoints;
                double mmin31=init2+((min31+0.5)*(final2-init2))/npoints;

                double mmin12=init2+((min12+0.5)*(final2-init2))/npoints;
                double mmin22=init2+((min22+0.5)*(final2-init2))/npoints;
                double mmin32=init2+((min32+0.5)*(final2-init2))/npoints;

                //mass<<name<<" "<<mmin1<<" "<<mmin2<<" "<<mmin3<<endl;

                ofstream maxs((string("maxs_")+string(name)+string(".out")).c_str());
                maxs<<mmin1<<" "<<mmin2<<" "<<mmin3<<endl;
                maxs<<mmin11<<" "<<mmin21<<" "<<mmin31<<endl;
                maxs<<mmin12<<" "<<mmin22<<" "<<mmin32<<endl;
                maxs<<llmax<<endl;
                //for(uint j=0;j<npoints;j++)
        //for(uint i=0;i<npoints;i++){
                //      int binmax=limits4->GetBin(i+1,j+1);
                //      maxs<<"("<<i<<","<<j<<"):"<<limits4->GetBinContent(binmax)<<endl;
                //      }

        maxs.close();

    limits4->SetStats(0);
    limits4->GetXaxis()->SetTitle("log_{10}(tan\\beta)");
    limits4->GetYaxis()->SetTitle("M_{H+} (GeV)");

    Bmumu_Bsmumu->SetStats(0);
    Bmumu_Bsmumu->GetXaxis()->SetTitle("Br(B\\to\\mu\\mu)/10^{-10}");
    Bmumu_Bsmumu->GetYaxis()->SetTitle("Br(B_{s}\\to\\mu\\mu)/10^{-9}");


    limits_MR_MI->SetStats(0);
    limits_MR_MI->GetYaxis()->SetTitle("M_{I} (GeV)");
    limits_MR_MI->GetXaxis()->SetTitle("M_{R} (GeV)");



    limits_MR_McH->SetStats(0);
    limits_MR_McH->GetYaxis()->SetTitle("M_{H+} (GeV)");
    limits_MR_McH->GetXaxis()->SetTitle("M_{R} (GeV)");

    limits_MI_McH->SetStats(0);
    limits_MI_McH->GetYaxis()->SetTitle("M_{H+} (GeV)");
    limits_MI_McH->GetXaxis()->SetTitle("M_{I} (GeV)");

    limits_tb_MR->SetStats(0);
    limits_tb_MR->GetXaxis()->SetTitle("log_{10}(tan\\beta)");
    limits_tb_MR->GetYaxis()->SetTitle("M_{R} (GeV)");


    gStyle->SetOptTitle(0);

   Double_t contours[3];
   contours[0] = 0.003;
   contours[1] = 0.05;
   contours[2] = 0.32;


    double ma=0,me=.2, x0=1,y0=120;

    limits4->SetContour(3, contours);
    //limits4->GetYaxis()->SetLabelOffset(0.02);
    limits4->GetYaxis()->SetLabelSize(0.08);
    limits4->GetYaxis()->SetTitleSize(0.08);
    limits4->GetYaxis()->SetTitleOffset(1.2);
    limits4->GetYaxis()->SetLimits(1,999);
    //limits4->GetXaxis()->SetLabelOffset(0.02);
    limits4->GetXaxis()->SetLabelSize(0.08);
    limits4->GetXaxis()->SetTitleSize(0.08);
    limits4->GetXaxis()->SetTitleOffset(1.2);
    limits4->GetXaxis()->SetLimits(-2.99,2.99);




    gStyle->SetPaperSize(10.,10.);

    TLatex l;
    l.SetTextSize(0.08);
    string ss=qq[qup][gQ]+","+ll[lup][gL];

    TCanvas * c21=new TCanvas("c21","",int(800*(1+ma+me)),int(600*(1+ma+me)));
        c21->SetMargin(me,ma,me,ma);
        c21->SetGrid();
        //limits4->Draw("CONT Z LIST");
    limits4->Draw("CONT Z LIST");
    if(!mssm) l.DrawLatex(x0,y0,ss.c_str());

    c21->SaveAs((string("pdf_")+string(name)+string(".png")).c_str());

    delete c21;

    // Bmumu_Bsmumu->SetContour(3, contours);
    //limits4->GetYaxis()->SetLabelOffset(0.02);
    Bmumu_Bsmumu->GetYaxis()->SetLabelSize(0.08);
    Bmumu_Bsmumu->GetYaxis()->SetTitleSize(0.08);
    Bmumu_Bsmumu->GetYaxis()->SetTitleOffset(1.2);
    Bmumu_Bsmumu->GetYaxis()->SetLimits(0.01,4.99);
    //limits4->GetXaxis()->SetLabelOffset(0.02);
    Bmumu_Bsmumu->GetXaxis()->SetLabelSize(0.08);
    Bmumu_Bsmumu->GetXaxis()->SetTitleSize(0.08);
    Bmumu_Bsmumu->GetXaxis()->SetTitleOffset(1.2);
    Bmumu_Bsmumu->GetXaxis()->SetLimits(0.01,1.99);

    TCanvas * cB=new TCanvas("cB","",int(800*(1+ma+me)),int(600*(1+ma+me)));
        cB->SetMargin(me,ma,me,ma);
        cB->SetGrid();
        //limits4->Draw("CONT Z LIST");
    Bmumu_Bsmumu->Draw("COLZ");
    l.DrawLatex(x0,y0,ss.c_str());

    cB->SaveAs((string("Bmumu_Bsmumu_")+string(name)+string(".png")).c_str());

    delete cB;


    limits_MR_MI->SetContour(3, contours);
    //limits4->GetYaxis()->SetLabelOffset(0.02);
    limits_MR_MI->GetYaxis()->SetLabelSize(0.06);
    limits_MR_MI->GetYaxis()->SetTitleSize(0.06);
    limits_MR_MI->GetYaxis()->SetTitleOffset(1.1);
    limits_MR_MI->GetYaxis()->SetLimits(1,999);
    //limits4->GetXaxis()->SetLabelOffset(0.02);
    limits_MR_MI->GetXaxis()->SetLabelSize(0.06);
    limits_MR_MI->GetXaxis()->SetTitleSize(0.06);
    limits_MR_MI->GetXaxis()->SetTitleOffset(1.1);
    limits_MR_MI->GetXaxis()->SetLimits(1,999);

        TCanvas * c3=new TCanvas("c3","",800,600);
        c3->SetMargin(me,ma,me,ma);
        c3->SetGrid();

        limits_MR_MI->Draw("CONT LIST");

    c3->SaveAs((string("pdf_")+string(name)+string("_MRMI.png")).c_str());

        limits_MR_McH->SetContour(3, contours);
        //limits4->GetYaxis()->SetLabelOffset(0.02);
    limits_MR_McH->GetYaxis()->SetLabelSize(0.06);
    limits_MR_McH->GetYaxis()->SetTitleSize(0.06);
    limits_MR_McH->GetYaxis()->SetTitleOffset(1.1);
    limits_MR_McH->GetYaxis()->SetLimits(1,999);
    //limits4->GetXaxis()->SetLabelOffset(0.02);
    limits_MR_McH->GetXaxis()->SetLabelSize(0.06);
    limits_MR_McH->GetXaxis()->SetTitleSize(0.06);
    limits_MR_McH->GetXaxis()->SetTitleOffset(1.1);
    limits_MR_McH->GetXaxis()->SetLimits(1,999);

        TCanvas * c4=new TCanvas("c4","",800,600);
        limits_MR_McH->Draw("CONT LIST");
    c4->SetMargin(me,ma,me,ma);
        c4->SetGrid();
    c4->SaveAs((string("pdf_")+string(name)+string("_MRMcH.png")).c_str());

        limits_MI_McH->SetContour(3, contours);
        //limits4->GetYaxis()->SetLabelOffset(0.02);
    limits_MI_McH->GetYaxis()->SetLabelSize(0.06);
    limits_MI_McH->GetYaxis()->SetTitleSize(0.06);
    limits_MI_McH->GetYaxis()->SetTitleOffset(1.1);
    limits_MI_McH->GetYaxis()->SetLimits(1,999);
    //limits4->GetXaxis()->SetLabelOffset(0.02);
    limits_MI_McH->GetXaxis()->SetLabelSize(0.06);
    limits_MI_McH->GetXaxis()->SetTitleSize(0.06);
    limits_MI_McH->GetXaxis()->SetTitleOffset(1.1);
    limits_MI_McH->GetXaxis()->SetLimits(1,999);

        TCanvas * c6=new TCanvas("c6","",800,600);
        limits_MI_McH->Draw("CONT LIST");
    c6->SetMargin(me,ma,me,ma);
        c6->SetGrid();
    c6->SaveAs((string("pdf_")+string(name)+string("_MIMcH.png")).c_str());
```

```

        TCanvas * c5=new TCanvas("c5","",800,600);
        limits_tb_MR->Draw("colz");

    c5->SaveAs((string("pdf_")+string(name)+string("_tbMR.png")).c_str());


        cout<<"gL "<<gL<<" gQ "<<gQ<<endl;
        cout<<"lup "<<lup<<" qup "<<qup<<endl;
        cout<<"llmax "<<llmax<<" gmax "<<gmax<<endl;


        delete m;
        delete limits4;
        delete Bmumu_Bsmumu;
        delete limits_tb_MR;
        delete limits_tb_MI;
        delete limits_MR_MI;
        delete limits_MI_McH;
        delete limits_MR_McH;

    //mass.close();
        return 0;

}

```

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

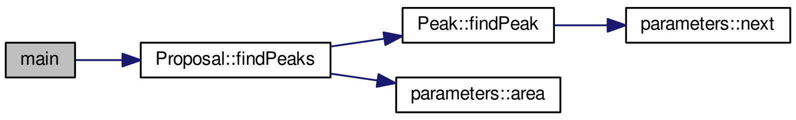
8.19.1.1 int main ( )

Definition at line 8 of file mainbk.cpp.

References Proposal::findPeaks(), Proposal::floatPeak, and Peak::pr.

```
          {
        Digits=5;
        cln::cl_inhibit_floating_point_underflow=1;




        //TH1F * pdf1=new TH1F("pdf1","pdf1",npoints,10,500);


        //TGraph * chi2=new TGraph(npoints);
        double lmax=0;
        double gaussmax=0;
        double tanbmax=0,mcHmax=0, Btaunu=0,BDtaunu=0,BD2taunu=0;
        uint gLm=0,gQm=0,lupm=0,qupm=0;

        for(uint gL=0;gL<3;gL++)
        for(uint gQ=0;gQ<3;gQ++)
        for(uint lup=0;lup<2;lup++)
        for(uint qup=0;qup<2;qup++){
                BGL* m=new BGL(gL,gQ,lup,qup);
                char name[5]="0000";
                name[0]+=gL;
                name[1]+=gQ;
                name[2]+=lup;
                name[3]+=qup;



        /*
        TF1 * f1 = new TF1("f1",m,&BGL::BranchingRatio,-3,3,0,"BGL","BranchingRatio");

        TCanvas * c1=new TCanvas("c1","",800,600);
        f1->Draw();
        c1->SaveAs("BR.png");
        TF2 * f2 = new TF2("f2",m,&BGL::topBranchingRatio,-3,3,80,175,0,"BGL","topBranchingRatio");

        TCanvas * c3=new TCanvas("c3","",800,600);
        f2->Draw("colz");
        c3->SaveAs("topBR.png");
        */
        uint npoints=100;
        //TH2F * pdf=new TH2F("pdf","pdf",npoints,-7/log(10.0),7/log(10.0),npoints,10,500);
        double init=-3, final=3;
        double init2=1, final2=4;

        uint steps=1e4;
        Proposal prop1(m,m->iBtaunu);
        prop1.findPeaks();
        Proposal prop2(m,m->iBDtaunu);
        prop2.findPeaks();
        Proposal prop3(m,m->iBD2taunu);
        prop3.findPeaks();
        Proposal prop4(m,-1);
        prop4.findPeaks();

        tanbmax=pow(10.0,prop4.floatPeak.pr[0].value);
        mcHmax=pow(10.0,prop4.floatPeak.pr[1].value);

        /*

        TH2F * limits1=new TH2F("limits1","Likelihood",npoints,init,final,npoints,init2,final2);
        TH2F * limits2=new TH2F("limits2","Likelihood",npoints,init,final,npoints,init2,final2);
        TH2F * limits3=new TH2F("limits3","Likelihood",npoints,init,final,npoints,init2,final2);
        TH2F * limits4=new TH2F("limits4","Likelihood",npoints,init,final,npoints,init2,final2);
        TH2F * limits5=new TH2F("limits5","Likelihood",npoints,init,final,npoints,init2,final2);
        TProfile2D * like1=new TProfile2D("like1","like",npoints,init,final,npoints,init2,final2);
        TProfile2D * like2=new TProfile2D("like2","like",npoints,init,final,npoints,init2,final2);
        TProfile2D * like3=new TProfile2D("like3","like",npoints,init,final,npoints,init2,final2);
        TProfile2D * like4=new TProfile2D("like4","like",npoints,init,final,npoints,init2,final2);
        THStack hs("hs","test stacked histograms");


        for(uint i=steps;i;i--){
                prop1.getNextPoint();
                prop2.getNextPoint();
                prop3.getNextPoint();
                prop4.getNextPoint();

                double Btaunu=m->loglike(prop1.floatPeak.pr,m->iBtaunu);
                double BDtaunu=m->loglike(prop2.floatPeak.pr,m->iBDtaunu);
                double BD2taunu=m->loglike(prop3.floatPeak.pr,m->iBD2taunu);
                double total=m->loglike(prop4.floatPeak.pr,-1);

                like1->Fill(prop1.floatPeak.pr[0].value, prop1.floatPeak.pr[1].value,Btaunu);
                like2->Fill(prop2.floatPeak.pr[0].value, prop2.floatPeak.pr[1].value,BDtaunu);
                like3->Fill(prop3.floatPeak.pr[0].value, prop3.floatPeak.pr[1].value,BD2taunu);
                like4->Fill(prop4.floatPeak.pr[0].value, prop4.floatPeak.pr[1].value,total);

                if(i%(steps/10)==0) cout<<"Steps "<<i<<endl;
                //pdf1->Fill(prop.floatPeak.pr[1].value);
        }

        for(uint i=0;i<npoints;i++)
                for(uint j=0;j<npoints;j++)
                {
                        double bcmax=0;
                        int binmax=like1->GetBin(i+1,j+1);
                        //for(uint k=0;k<npoints;k++){
                        //int binmax=like1->GetBin(i+1,j+1,j+1);
                        //if(like1->GetBinEntries(bin)){
                        //      double bc=like1->GetBinContent(bin);
                        //      if(bc>bcmax || binmax==-1) {bcmax=bc; binmax=bin;}
                        //      }
                        //}
                        double Btaunu=0;
                        double BDtaunu=0;
                        double BD2taunu=0;
                        double rest=0;
                        double sum=0;

                        if(like1->GetBinEntries(binmax)) {
                                Btaunu=like1->GetBinContent(binmax);
                                //cout<<i<<" "<<j<<" "<<Btaunu<<" ";
                                sum+=Btaunu;
                                Btaunu=TMath::Prob(-2*Btaunu,1);
                                //cout<<Btaunu<<endl;
                                if(Btaunu<0.05) Btaunu=0;
                        }
                        if(like2->GetBinEntries(binmax)) {
                                BDtaunu=like2->GetBinContent(binmax);
                                sum+=BDtaunu;
                                BDtaunu=TMath::Prob(-2*BDtaunu,1);
                                if(BDtaunu<0.05) BDtaunu=0;
                        }
                        if(like3->GetBinEntries(binmax)) {
                                BD2taunu=like3->GetBinContent(binmax);
                                sum+=BD2taunu;
                                BD2taunu=TMath::Prob(-2*BD2taunu,1);
                                if(BD2taunu<0.05) BD2taunu=0;
                        }
                        if((like3->GetBinEntries(binmax) && like2->GetBinEntries(binmax) &&
like1->GetBinEntries(binmax) && sum>-20)){
                                sum=TMath::Prob(-2*sum,3);
                                if(sum>lmax){
                                lmax=sum;
                                tanbmax=pow(10.0,(i+0.5)*1.0/npoints*6-3);
                                mcHmax=pow(10.0,(j+0.5)*1.0/npoints*3+1);
                        }
                        }else sum=0;

                        if(like4->GetBinEntries(binmax)){
                                rest=like4->GetBinContent(binmax);
                                rest=TMath::Prob(-2*rest,m->size());
                        }
                        int scale=100;
                        limits1->SetBinContent(i+1,j+1,Btaunu*scale);
                        limits2->SetBinContent(i+1,j+1,BDtaunu*scale);
                        limits3->SetBinContent(i+1,j+1,BD2taunu*scale);
                        limits4->SetBinContent(i+1,j+1,rest*scale);
                        limits5->SetBinContent(i+1,j+1,sum*scale);
                }

        limits3->GetXaxis()->SetTitle("tan\\beta*MW/MH");
    limits3->GetYaxis()->SetTitle("cotan\\beta*MW/MH");
    limits1->SetStats(0);
    limits2->SetStats(0);
    limits3->SetStats(0);
    limits4->SetStats(0);

    //limits4->SetMarkerStyle(7);
    limits1->SetMarkerColor(1);
    //limits1->SetMarkerStyle(7);
    limits2->SetMarkerColor(2);
    //limits2->SetMarkerStyle(7);
    limits3->SetMarkerColor(3);
    //limits3->SetMarkerStyle(7);
    limits4->SetMarkerColor(9);
    //hs.Add(limits4);
    hs.Add(limits3);
    hs.Add(limits2);
    hs.Add(limits1);

        TCanvas * c2=new TCanvas("c2","",800,600);
        limits4->Draw("colz");
    //hs.Draw("nostack");
    c2->SaveAs((string("pdf_")+string(name)+string(".png")).c_str());

        TCanvas * c3=new TCanvas("c3","",800,600);
        limits1->Draw("colz");
    //hs.Draw("nostack");
    c3->SaveAs((string("pdf1_")+string(name)+string(".png")).c_str());

        TCanvas * c4=new TCanvas("c4","",800,600);
        limits2->Draw("colz");
    //hs.Draw("nostack");
    c4->SaveAs((string("pdf2_")+string(name)+string(".png")).c_str());


        TCanvas * c5=new TCanvas("c5","",800,600);
        limits3->Draw("colz");
    //hs.Draw("nostack");
    c5->SaveAs((string("pdf3_")+string(name)+string(".png")).c_str());

        TCanvas * c6=new TCanvas("c6","",800,600);
        limits5->Draw("colz");
    //hs.Draw("nostack");
    c6->SaveAs((string("pdf123_")+string(name)+string(".png")).c_str());

    TCanvas * c7=new TCanvas("c7","",800,600);
        hs.Draw("nostack");
    c7->SaveAs((string("pdfall_")+string(name)+string(".png")).c_str());
        */
        //cout<<"Lmax "<<lmax<<" "<<TMath::Prob(-2*log(lmax),m->size())<<" GaussMax "<<gaussmax<<endl;

        //cout<<"tanbmax "<<tanbmax<<" McHmax "<<mcHmax<<endl;
        cout<<"gL "<<gL<<" gQ "<<gQ<<endl;
        cout<<"lup "<<lup<<" qup "<<qup<<endl;

        cout<<"Btotaunu "<<m->BtotaunuR.subs(lst(m->tanb==tanbmax,m->McH==mcHmax))<<" exp "<<1.64/0.79<<"
 +/- "<<0.23*1.64/0.79<<endl;
        cout<<"BtoDtaunu "<<m->BtoDtaunuR.subs(lst(m->tanb==tanbmax,m->McH==mcHmax))<<" exp "<<440.0/296<<"
 +/- "<<1.4*58.0/296<<endl;
        cout<<"BtoD2taunu "<<m->BtoD2taunuR.subs(lst(m->tanb==tanbmax,m->McH==mcHmax))<<" exp "<<332.0/252<
<" +/- "<<1.4*24.0/252<<endl;


        delete m;
}

        return 0;

}
```

Here is the call graph for this function:



8.20 mainbk.cpp

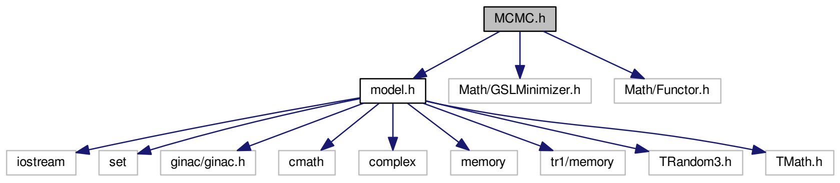
```
#include "MCMC.h"
#include "BGL.h"
#include "TF2.h"
#include "TProfile3D.h"
#include "THStack.h"
#include <cln/cln.h>
#include <cln/float.h>
int main(){
        Digits=5;
        cln::cl_inhibit_floating_point_underflow=1;




        //TH1F * pdf1=new TH1F("pdf1","pdf1",npoints,10,500);


        //TGraph * chi2=new TGraph(npoints);
        double lmax=0;
        double gaussmax=0;
        double tanbmax=0,mcHmax=0, Btaunu=0,BDtaunu=0,BD2taunu=0;
        uint gLm=0,gQm=0,lupm=0,qupm=0;

        for(uint gL=0;gL<3;gL++)
        for(uint gQ=0;gQ<3;gQ++)
        for(uint lup=0;lup<2;lup++)
        for(uint qup=0;qup<2;qup++){
                BGL* m=new BGL(gL,gQ,lup,qup);
                char name[5]="0000";
                name[0]+=gL;
                name[1]+=gQ;
                name[2]+=lup;
                name[3]+=qup;



        /*
        TF1 * f1 = new TF1("f1",m,&BGL::BranchingRatio,-3,3,0,"BGL","BranchingRatio");

        TCanvas * c1=new TCanvas("c1","",800,600);
        f1->Draw();
        c1->SaveAs("BR.png");
        TF2 * f2 = new TF2("f2",m,&BGL::topBranchingRatio,-3,3,80,175,0,"BGL","topBranchingRatio");

        TCanvas * c3=new TCanvas("c3","",800,600);
        f2->Draw("colz");
        c3->SaveAs("topBR.png");
        */
        uint npoints=100;
        //TH2F * pdf=new TH2F("pdf","pdf",npoints,-7/log(10.0),7/log(10.0),npoints,10,500);
        double init=-3, final=3;
        double init2=1, final2=4;

        uint steps=1e4;
        Proposal prop1(m,m->iBtaunu);
        prop1.findPeaks();
        Proposal prop2(m,m->iBDtaunu);
        prop2.findPeaks();
        Proposal prop3(m,m->iBD2taunu);
        prop3.findPeaks();
        Proposal prop4(m,-1);
        prop4.findPeaks();

        tanbmax=pow(10.0,prop4.floatPeak.pr[0].value);
        mcHmax=pow(10.0,prop4.floatPeak.pr[1].value);

        /*

        TH2F * limits1=new TH2F("limits1","Likelihood",npoints,init,final,npoints,init2,final2);
        TH2F * limits2=new TH2F("limits2","Likelihood",npoints,init,final,npoints,init2,final2);
        TH2F * limits3=new TH2F("limits3","Likelihood",npoints,init,final,npoints,init2,final2);
        TH2F * limits4=new TH2F("limits4","Likelihood",npoints,init,final,npoints,init2,final2);
        TH2F * limits5=new TH2F("limits5","Likelihood",npoints,init,final,npoints,init2,final2);
        TProfile2D * like1=new TProfile2D("like1","like",npoints,init,final,npoints,init2,final2);
        TProfile2D * like2=new TProfile2D("like2","like",npoints,init,final,npoints,init2,final2);
        TProfile2D * like3=new TProfile2D("like3","like",npoints,init,final,npoints,init2,final2);
        TProfile2D * like4=new TProfile2D("like4","like",npoints,init,final,npoints,init2,final2);
        THStack hs("hs","test stacked histograms");


        for(uint i=steps;i;i--){
                prop1.getNextPoint();
                prop2.getNextPoint();
                prop3.getNextPoint();
                prop4.getNextPoint();

                double Btaunu=m->loglike(prop1.floatPeak.pr,m->iBtaunu);
                double BDtaunu=m->loglike(prop2.floatPeak.pr,m->iBDtaunu);
                double BD2taunu=m->loglike(prop3.floatPeak.pr,m->iBD2taunu);
                double total=m->loglike(prop4.floatPeak.pr,-1);

                like1->Fill(prop1.floatPeak.pr[0].value, prop1.floatPeak.pr[1].value,Btaunu);
                like2->Fill(prop2.floatPeak.pr[0].value, prop2.floatPeak.pr[1].value,BDtaunu);
                like3->Fill(prop3.floatPeak.pr[0].value, prop3.floatPeak.pr[1].value,BD2taunu);
                like4->Fill(prop4.floatPeak.pr[0].value, prop4.floatPeak.pr[1].value,total);

                if(i%(steps/10)==0) cout<<"Steps "<<i<<endl;
                //pdf1->Fill(prop.floatPeak.pr[1].value);
        }

        for(uint i=0;i<npoints;i++)
                for(uint j=0;j<npoints;j++)
                {
                        double bcmax=0;
                        int binmax=like1->GetBin(i+1,j+1);
                        //for(uint k=0;k<npoints;k++){
                        //int binmax=like1->GetBin(i+1,j+1,j+1);
                        //if(like1->GetBinEntries(bin)){
                        //      double bc=like1->GetBinContent(bin);
                        //      if(bc>bcmax || binmax==-1) {bcmax=bc; binmax=bin;}
                        //      }
                        //}
                        double Btaunu=0;
                        double BDtaunu=0;
                        double BD2taunu=0;
                        double rest=0;
                        double sum=0;

                        if(like1->GetBinEntries(binmax)) {
                                Btaunu=like1->GetBinContent(binmax);
                                //cout<<i<<" "<<j<<" "<<Btaunu<<" ";
                                sum+=Btaunu;
                                Btaunu=TMath::Prob(-2*Btaunu,1);
                                //cout<<Btaunu<<endl;
                                if(Btaunu<0.05) Btaunu=0;
                        }
                        if(like2->GetBinEntries(binmax)) {
                                BDtaunu=like2->GetBinContent(binmax);
                                sum+=BDtaunu;
                                BDtaunu=TMath::Prob(-2*BDtaunu,1);
                                if(BDtaunu<0.05) BDtaunu=0;
                        }
                        if(like3->GetBinEntries(binmax)) {
                                BD2taunu=like3->GetBinContent(binmax);
                                sum+=BD2taunu;
                                BD2taunu=TMath::Prob(-2*BD2taunu,1);
                                if(BD2taunu<0.05) BD2taunu=0;
                        }
                        if((like3->GetBinEntries(binmax) && like2->GetBinEntries(binmax) &&
       like1->GetBinEntries(binmax) && sum>-20)){
                                sum=TMath::Prob(-2*sum,3);
                                if(sum>lmax){
                                lmax=sum;
                                tanbmax=pow(10.0,(i+0.5)*1.0/npoints*6-3);
                                mcHmax=pow(10.0,(j+0.5)*1.0/npoints*3+1);
                        }
                        }else sum=0;

                        if(like4->GetBinEntries(binmax)){
                                rest=like4->GetBinContent(binmax);
                                rest=TMath::Prob(-2*rest,m->size());
                        }
                        int scale=100;
                        limits1->SetBinContent(i+1,j+1,Btaunu*scale);
                        limits2->SetBinContent(i+1,j+1,BDtaunu*scale);
                        limits3->SetBinContent(i+1,j+1,BD2taunu*scale);
                        limits4->SetBinContent(i+1,j+1,rest*scale);
                        limits5->SetBinContent(i+1,j+1,sum*scale);
                }

        limits3->GetXaxis()->SetTitle("tan\\beta*MW/MH");
    limits3->GetYaxis()->SetTitle("cotan\\beta*MW/MH");
    limits1->SetStats(0);
    limits2->SetStats(0);
    limits3->SetStats(0);
    limits4->SetStats(0);

    //limits4->SetMarkerStyle(7);
    limits1->SetMarkerColor(1);
    //limits1->SetMarkerStyle(7);
    limits2->SetMarkerColor(2);
    //limits2->SetMarkerStyle(7);
    limits3->SetMarkerColor(3);
    //limits3->SetMarkerStyle(7);
    limits4->SetMarkerColor(9);
    //hs.Add(limits4);
    hs.Add(limits3);
    hs.Add(limits2);
    hs.Add(limits1);

        TCanvas * c2=new TCanvas("c2","",800,600);
        limits4->Draw("colz");
    //hs.Draw("nostack");
    c2->SaveAs((string("pdf_")+string(name)+string(".png")).c_str());

```

```
        TCanvas * c3=new TCanvas("c3","",800,600);
        limits1->Draw("colz");
    //hs.Draw("nostack");
    c3->SaveAs((string("pdf1_")+string(name)+string(".png")).c_str());

        TCanvas * c4=new TCanvas("c4","",800,600);
        limits2->Draw("colz");
    //hs.Draw("nostack");
    c4->SaveAs((string("pdf2_")+string(name)+string(".png")).c_str());


        TCanvas * c5=new TCanvas("c5","",800,600);
        limits3->Draw("colz");
    //hs.Draw("nostack");
    c5->SaveAs((string("pdf3_")+string(name)+string(".png")).c_str());

        TCanvas * c6=new TCanvas("c6","",800,600);
        limits5->Draw("colz");
    //hs.Draw("nostack");
    c6->SaveAs((string("pdf123_")+string(name)+string(".png")).c_str());

    TCanvas * c7=new TCanvas("c7","",800,600);
        hs.Draw("nostack");
    c7->SaveAs((string("pdfall_")+string(name)+string(".png")).c_str());
        */
        //cout<<"Lmax "<<lmax<<" "<<TMath::Prob(-2*log(lmax),m->size())<<" GaussMax "<<gaussmax<<endl;

        //cout<<"tanbmax "<<tanbmax<<" McHmax "<<mcHmax<<endl;
        cout<<"gL "<<gL<<" gQ "<<gQ<<endl;
        cout<<"lup "<<lup<<" qup "<<qup<<endl;

        cout<<"Btotaunu "<<m->BtotaunuR.subs(lst(m->tanb==tanbmax,m->McH==mcHmax))<<" exp "<<1.64/0.79<<"
 +/- "<<0.23*1.64/0.79<<endl;
        cout<<"BtoDtaunu "<<m->BtoDtaunuR.subs(lst(m->tanb==tanbmax,m->McH==mcHmax))<<" exp "<<440.0/296<<"
 +/- "<<1.4*58.0/296<<endl;
        cout<<"BtoD2taunu "<<m->BtoD2taunuR.subs(lst(m->tanb==tanbmax,m->McH==mcHmax))<<" exp "<<332.0/252<
<" +/- "<<1.4*24.0/252<<endl;


        delete m;
}

        return 0;

}

```

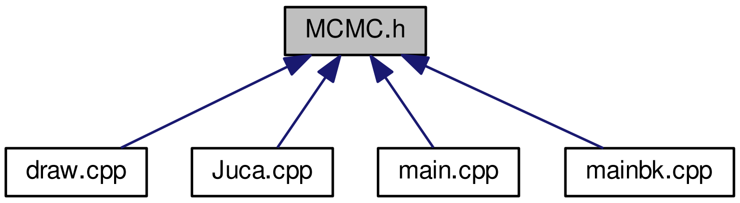
8.21 MCMC.h File Reference

```
#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

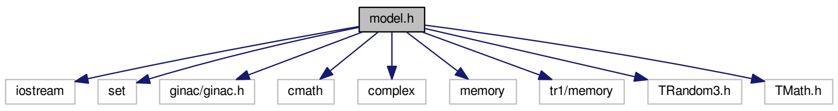
```
#ifndef MCMC_H
#define MCMC_H

#include "model.h"
#include "Math/GSLMinimizer.h"
#include "Math/Functor.h"

using namespace std;

///A class containing the parameters of a maximum of the likelihood function
class Peak {
public:
        Peak(const Model * m,int maxx=0): model(m), pr(m->generateparameters(maxx)), max(maxx){
                lmax=model->likelihood(pr);
                }

        void findPeak(){
           llmax=model->loglike(pr,1,max);
           area=1;
          uint fixed=1e2;
          uint f=fixed;
          double d=1;
          //cout<<"f "<<f<<"llmax "<<llmax<<endl;
                for(uint i=1e5;i;i--){
                        parameters p1(pr);
                        p1.next(model->r,d);

                        double l1=llmax;
                        if(!model->veto(p1,max)){l1=model->loglike(p1,1,max);
                                }
                        if(l1>llmax){pr=p1;  llmax=l1; f=fixed;}
                        else {f--; if(!f) {d/=100; f=fixed; if(d<1e-2) break;}}
                }
           cout<<"d "<<d<<"llmax "<<llmax<<endl;
           if(llmax<-1000) lmax=0;
           else{lmax=exp(llmax);
                area=pr.area();
            larea=area*lmax;
            }

        }
        /*
        void findPeak(){
           llmax=model->loglike(pr);
           area=1;
          uint fixed=1e2;
          uint f=fixed;
          parameters p1(pr);

                for(uint j=1e4;j;j--){
                        for(uint i=0;i<pr.size(); i++){
                           double s=pr[i].step/1e3;
                           pr[i].value+=s;
                           if(pr[i].value>pr[i].max){ s*=-1; pr[i].value+=2*s;}
                           double x=(model->loglike(pr)-llmax)*pr[i].step/1e2;
                           pr[i].value-=s;
                           if(fabs(x)>pr[i].step) x*=pr[i].step/fabs(x);
                           p1[i].value=pr[i].value+x;
                           if(p1[i].value>pr[i].max) p1[i].value=pr[i].max;
                           else if(p1[i].value<pr[i].min) p1[i].value=pr[i].min;
                   }
                   if(pr.dist(p1)<1e-6) {pr.setvalues(p1); f=0; break;}
                   cout<<"Loglike "<<llmax<<" "<<pr.dist(p1)<<endl;
                   pr.setvalues(p1);
                   llmax=model->loglike(pr);
            }

           if(f || llmax<-20) lmax=0;
           else{
                   lmax=exp(llmax);
                area=pr.area();
            larea=area*lmax;
            }

        }

        double RosenBrock(const double *xx )
        {       parameters p=model->generateparameters();
                for(uint i=0; i<p.size();i++) p[i].value=xx[i];

                return -model->loglike(pr);
        }

        void findPeak3(){
           llmax=model->loglike(pr);
           area=1;
          uint fixed=1e2;
          uint f=fixed;
          parameters p1(pr);
          ROOT::Math::GSLMinimizer min( ROOT::Math::kVectorBFGS );

      min.SetMaxFunctionCalls(1000000);
      min.SetMaxIterations(100000);
      min.SetTolerance(0.001);

      ROOT::Math::Functor f(&RosenBrock,pr.size());
      double step[2] = {0.01,0.01};
      double variable[2] = { -1.,1.2};
          char s[3]="x0";

      min.SetFunction(f);
        fo
      // Set the free variables to be minimized!
      min.SetVariable(0,"x",variable[0], step[0]);
      min.SetVariable(1,"y",variable[1], step[1]);

   min.Minimize();

                for(uint j=1e4;j;j--){
                        for(uint i=0;i<pr.size(); i++){
                           double s=pr[i].step/1e3;
                           pr[i].value+=s;
                           if(pr[i].value>pr[i].max){ s*=-1; pr[i].value+=2*s;}
                           double x=(model->loglike(pr)-llmax)*pr[i].step/1e2;
                           pr[i].value-=s;
                           if(fabs(x)>pr[i].step) x*=pr[i].step/fabs(x);
                           p1[i].value=pr[i].value+x;
                           if(p1[i].value>pr[i].max) p1[i].value=pr[i].max;
                           else if(p1[i].value<pr[i].min) p1[i].value=pr[i].min;
                   }
                   if(pr.dist(p1)<1e-6) {pr.setvalues(p1); f=0; break;}
                   cout<<"Loglike "<<llmax<<" "<<pr.dist(p1)<<endl;
                   pr.setvalues(p1);
                   llmax=model->loglike(pr);
            }

           if(f || llmax<-20) lmax=0;
           else{
                   lmax=exp(llmax);
                area=pr.area();
            larea=area*lmax;
            }

        }
        */
        bool adjuststeps(){
                 parameters p1(pr);
                 for(uint i=0;i<pr.size(); i++){
                           double s=p1[i].step;
                           p1[i].value+=s;
                           double x=(lmax-model->likelihood(p1))*2/lmax/s/s;
                           double x0=std::pow(2/(pr[i].max-pr[i].min),2);
                           if(x<x0) return 0;
                          // cout<<"X "<<x<<endl;
                           pr[i].step=1/sqrt(x);
                           p1[i].value-=s;
                           }
           area=pr.area();
           larea=area*lmax;
           return 1;
        }

        const Model * model;
        parameters pr;
        double lmax, llmax;
        double area;
        double larea;
        bool max;
};

///A class containing the parameters of a proposal for the next step in the Markov Chain
class Proposal{
public:

Proposal(const Model * m): model(m), floatPeak(m),proposal(m){}

void findPeaks(uint ns=1, int max=0) {
        //float pmin=-100, pmax=100, s=0.1;
    //floatPeak.s=s;
    //floatPeak.lmax=0;
    //int imax=-1;
    floatPeak=Peak(model,max);
    floatPeak.lmax=0;
    floatPeak.llmax=-1000;
   cout<<"started"<<endl;
        //for(uint i=5e1;i;i--){
        for(uint i=ns;i;i--){
                Peak pp(model,max);
                pp.findPeak();
                if(pp.llmax>-15){
                //for(uint j=0; j< pp.pr.size();j++){
                //cout<<j<<" "<<pp.pr[j].value<<endl;
                //}
                //lst l=model->getlist(pp.pr);
                //for(uint j=0; j< model->size();j++){
                //      double mean=model->at(j).calculate(l);
                //cout<<j<<" "<<mean<<" "<<sqrt(2*model->at(j).o->loglikelihood(mean))<<endl;
                //}
                }
                if(pp.lmax>floatPeak.lmax){
                        cout<<i<<" "<<pp.lmax<<endl;
                        floatPeak.lmax=pp.lmax;
                        floatPeak.pr=pp.pr;
                        }
        }
        floatPeak.area=floatPeak.pr.area();

}

void getProposal(){
        if(model->r->Rndm()<=0.9) {
                proposal.pr=floatPeak.pr;
                proposal.pr.next(model->r);
                return;
                }

        proposal.pr=model->generateparameters();
}

void getNextPoint(){
        getProposal();
        double l1=0;
        l1=model->likelihood(proposal.pr);
        if(model->r->Rndm()<=l1/floatPeak.lmax){
                floatPeak.lmax=l1;
                floatPeak.pr.setvalues(proposal.pr);
        }
```

```
}

const Model * model;

vector<Peak> vPeak;
Peak floatPeak, proposal;
double total;
};

#endif
```

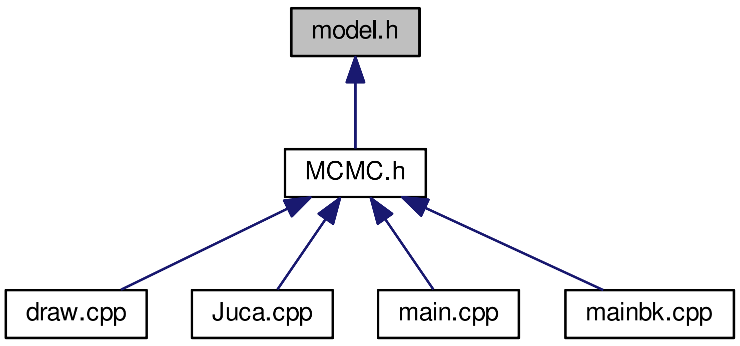
8.23 model.h File Reference

```
#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
```

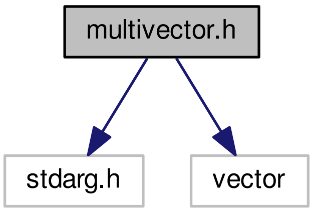
```
using namespace std;
//using namespace tr1;
using namespace GiNaC;
///A class containing the value and uncertainty of an experimental measure
class measure{
public:
        measure(double v=0,double e=0):value(v),error(e){}
        measure operator*(measure m2) const{
                const measure & m1=*this;
                return measure(m1.value*m2.value,sqrt(std::pow(m1.value*m2.error,2)+std::pow(m2.value*m1.error,2)));
                }
        measure operator/(measure m2) const{
                const measure & m1=*this;
                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)));
                }
        double value;
        double error;
        };
///A base class representing an experimental measure
class observable{
public:
 observable(): copies(1) {}
 virtual ~observable(){}

/**\param hipothesis the theoretical hypothesis
 * \return the logarithm of the probability of measuring what was measured,
 * assuming that the hypothesis is true
 */
 virtual double loglikelihood(double hipothesis) const = 0;
 virtual double error(double hipothesis) const = 0;

 //virtual int type() const =0;
 mutable uint copies;
};

///An experimental measure which is an upper limit on a parameter with a given Confidence Level
class limitedobs: public observable{
public:
/**\param limit upper limit on the parameter
 * \param m minimum possible value of the parameter
 * \param p 1-Confidence Level
 * */
 //limitedobs(double limit, double cl=0.9, double m=0): s(fabs(limit-m)/1.282), min(m),lim(limit) {
//       if(cl==0.95) s*=1.282/1.645;
  limitedobs(double limit, double cl=0.9, double m=0): s(fabs(limit-m)/(1.282+sqrt(M_PI_2))), min(m),lim(limit) {
         if(cl==0.95) s*=(1.282+sqrt(M_PI_2))/(1.645+sqrt(M_PI_2));
         }
 ~limitedobs(){}
 double loglikelihood(double hipothesis) const {
         double diff=(hipothesis-min-sqrt(M_PI_2)*s)/s;
         if(diff<0) diff=0;
         return diff*diff/2;
        }
  double error(double hipothesis) const {
        double diff=(hipothesis-min )/s;
        return diff;
        }

 double s,min,lim;
};

///An experimental measure of a parameter which is a mean value and a standard deviation
class gaussobs: public observable{
public:

/**\param mean mean value of the measure
 * \param sigma standard deviation of the measure
 * */
 gaussobs(measure v): m(v.value), s(v.error) {}
 gaussobs(double mean, double sigma): m(mean), s(mean*sigma) {}
 ~gaussobs(){}
 double loglikelihood(double hipothesis) const {
        double diff=(m-hipothesis)/s;
        return diff*diff/2;
        }
  double error(double hipothesis) const {
        double diff=(hipothesis-m)/s;
        return diff;
        }
 const double m, s;
};


```

```
///the same as gaussobs but with a different initializer, such that the uncertainty sigma is absolute
class gauss2obs: public observable{
public:

/**\param mean mean value of the measure
 * \param sigma standard deviation of the measure
 * */
 gauss2obs(measure v): m(v.value), s(v.error) {}
 gauss2obs(double mean, double sigma): m(mean), s(sigma) {}
 ~gauss2obs(){}
 double loglikelihood(double hipothesis) const {
        double diff=(m-hipothesis)/s;
        return diff*diff/2;
        }
 double error(double hipothesis) const {
        double diff=(hipothesis-m)/s;
        return diff;
        }

 double expected()const {return m;}
 //int type() const {return 1;}
 const double m, s;
};

///A parameter which will be fitted in the simulation
class freeparameter{
public:
 /**\param mi minimum possible value for the parameter
  * \param ma maximum possible value for the parameter
  * \param r random number generator
  * */
 freeparameter(double mi, double ma, TRandom3 * r,double ss=1e-2): min(mi), max(ma), value(mi+
(ma-mi)*r->Rndm()), step((ma-mi)*ss) {}

 ///changes randomly the ::value of the parameter, the standard deviation is ::step
 /**\param r random number generator
  * */
 void next(TRandom3 * r,double f=1){
         //double x=r->Gaus()*step;
         value+=r->Gaus()*step*f;
         }
  ///checks if the value of the parameter is between ::min and ::max
  bool isvalid() const {
        return min<=value && value<=max;
  }
  ///probability distribution, to be used by the Markov Chain Monte Carlo simulation
  /**\return \f$(\frac{x-::value}{::step})^2\f$
   * */
  double dist(double x) const {
        return std::pow((x-value)/step,2);
  }
  ///minimum possible value for the parameter
  double min;
  ///maximum possible value for the parameter
  double max;
  ///value of the parameter
  double value;
  ///standard deviation of the random changes of ::value in \ref next(TRandom3 *)
  double step;

};

///A parameter which will be fitted in the simulation
class discreteparameter{
public:
 /**\param mi minimum possible value for the parameter
  * \param ma maximum possible value for the parameter
  * \param r random number generator
  * */
  discreteparameter(int mi, int ma, TRandom3 * r): min(mi), max(ma), value(mi+r->Integer(
ma-mi+1)) {}
  ///minimum possible value for the parameter
  double min;
  ///maximum possible value for the parameter
  double max;
  ///value of the parameter
  double value;
};

///vector of parameters
class parameters: public vector< freeparameter >{
public:

//parameters(){}
///changes randomly the value of the parameters
void next(TRandom3 * r, double f=1){
        for(iterator i=begin();i!=end();i++){
                i->next(r,f);
                }

        //for(uint i=0;i<discrete.size();i++){
                //discrete[i].next(r);
                //}
        }

///checks if all the values are between their minimums and maximums
bool isvalid() const{
        for(const_iterator i=begin();i!=end();i++){
                if(!i->isvalid()) return 0;
                }
        return 1;
        }
///checks if this and another vector of parameters are within 1sigma of distance
double dist(const parameters& p) const{
        double total=0;
        for(uint i=0;i<size();i++){
                total+=at(i).dist(p[i].value);
                }
        return sqrt(total/size());
}

void setvalues(const parameters& p){

        for(uint i=0;i<size();i++){
                at(i).value=p[i].value;
                }
        }

double area() const{
        float a=1;
        for(const_iterator i=begin();i!=end();i++){
                a*=i->step;
                }
        return a;
        }

double gausslikelihood(const parameters & p2) const{
        double l=1;
        for(uint i=0;i<size();i++){
                l*=TMath::Gaus((p2[i].value-at(i).value)/at(i).step);
        }
        return l;
        }

        lst p;
        vector<double> values;
        //vector<discreteparameters> discrete;
};

///Base class to do the calculus of a constraint to the model
class calcu{
        public:
/**\param hipothesis the theoretical hypothesis
 * \return the logarithm of the probability of measuring what was measured,
 * assuming that the hypothesis is true
 */
 virtual double operator()(const parameters & p) const=0;
 //virtual int type() const =0;
};

///class to do the calculus of a constraint based on a GiNaC compiled expression
class calcuba:public calcu{
        public:
        calcuba(observable * ob, const FUNCP_CUBA & e0): calcu(), o(ob), e(e0){}

        double operator()(const parameters & p) const{
                 double ret=1000;
         int pass=1;

        /* try{
                 ret=ex_to<numeric>(e.subs(p.p,subs_options::no_pattern).evalf()).to_double();
         }
         catch(GiNaC::pole_error e){
          pass=0;
          cout<<"Pole error"<<endl;
         }
         catch(...){
          cout<<"Other exception"<<endl;
          exit(1);
         }
         */
         int n=p.values.size(), m=1;
         e(&n,&(p.values[0]),&m,&ret);
         if(pass) ret=o->loglikelihood(ret);
         else ret=1000;

         return ret;
    }

        shared_ptr<observable> o;
        FUNCP_CUBA e;
        };


///class to do the calculus of a constraint based on a GiNaC symbolic expression
class calcuex:public calcu{
        public:
        calcuex(observable * ob, const ex & e0): calcu(), o(ob), e(e0){}
        ~calcuex(){}

        double operator()(const parameters & p) const{
                 double ret=1000;
         int pass=1;
         try{
                 ret=ex_to<numeric>(e.subs(p.p,subs_options::no_pattern).evalf()).to_double();
         }
         catch(GiNaC::pole_error e){
          pass=0;
          cout<<"Pole error"<<endl;
         }
         catch(exception e){
          cout<<e.what()<<endl;
          }
         catch(...){
          cout<<"Other exception"<<endl;
          exit(1);
         }
         if(pass) ret=o->loglikelihood(ret);
         else ret=1000;

         return ret;
    }

    double error(const parameters & p) const{
                 double ret=1000;
         int pass=1;
         try{
                 cout<<e<<endl;
                 cout<<e.subs(p.p)<<endl;

                 ret=ex_to<numeric>(e.subs(p.p,subs_options::no_pattern).evalf()).to_double();
         }
         catch(GiNaC::pole_error er){
          pass=0;
          cout<<"Pole error"<<endl;
         }
         catch(exception er){
          pass=0;

          cout<<er.what()<<endl;
          cout<<e.subs(p.p,subs_options::no_pattern).evalf()<<endl;
          }
         catch(...){
          cout<<"Other exception"<<endl;
          exit(1);
         }
         if(pass) ret=o->error(ret);
         else ret=1000;

         return ret;
    }

shared_ptr<observable> o;
ex e;
};

///theoretical expression for an experimental measure
class prediction{
public:
 prediction(observable * ob, const FUNCP_CUBA & e0): calculate(new calcuba(ob,e0)) {}
 prediction(observable * ob, const ex & e0): calculate(new calcuex(ob,e0)) {}
 prediction(calcu * c): calculate(c) {}
 prediction(const prediction& p): calculate(p.calculate) {}
 ~prediction(){}

 double loglikelihood(const parameters & p) const { return (*calculate)(p);}
 ///theoretical expression for the experimental measure

 shared_ptr<calcu> calculate;
 //vector<ex> es;
};


///Abstract class for a model
class Model: public vector< prediction > {
public:

 Model(): r(new TRandom3(0)){}
 virtual ~Model(){delete r;};
 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 {return !p.
isvalid();}

 ///calculates the probability of getting all the experimental measures if the model describes the reality
/**\param p vector with the values of the free parameters
 */

double likelihood(const parameters & p, bool check=1, int max=0) const{
        if(veto(p,max) && check) return 0;
        double total=loglike(p,0);
        if(total<-1000) return 0;
        return exp(total);
        }

double loglike(const parameters & p, bool check=1, int max=0) const{
        if(veto(p,max) && check) return -1000;
        parameters pp(getlist(p));

        double total=0;
        int n=0;
        for(const_iterator i=begin();i!=end();i++) {
        try{
                n++;
                total+=i->loglikelihood(pp);
                                }
    catch(exception e){
          cout<<n<<e.what()<<endl;
          exit(1);
         }
        //catch(...){
                //cout<<"DD "<<n<<endl;
                //exit(1);
                //}
}

        return -total;
        }

  TRandom3 * r;
};

#endif
```

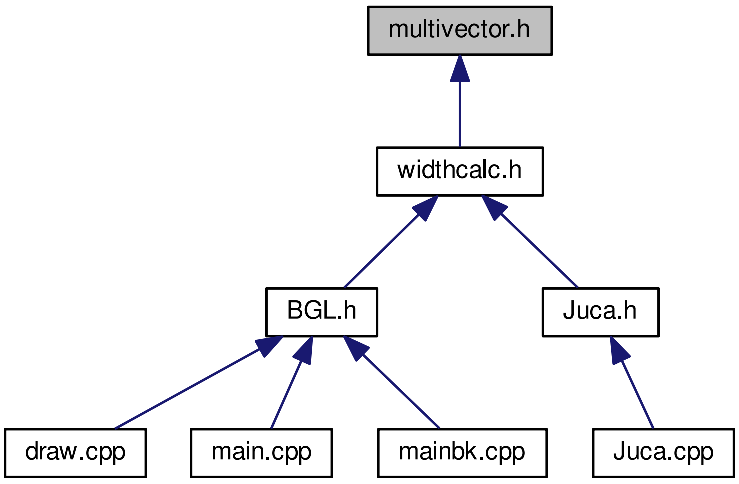
8.25 multivector.h File Reference

```
#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

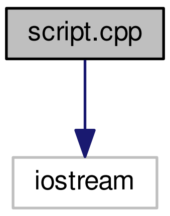
```
#include <stdarg.h>
#include <vector>

namespace std{

/// A vector of vectors of vectors of... (N times) of class T objects.
template < class T, int N>
class multivector: public vector< multivector<T, N-1> >{
public:
    typedef vector< multivector< T,N-1> > v;

    /// Default constructor
    multivector(): v() {}
        /// Copy constructor
        multivector(const multivector& m): v(m){}
///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
 * \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 &)
 * */
multivector(const T& value, ...){
                va_list listPointer;
                va_start(listPointer,value);
                int n=va_arg(listPointer,int);
                v::insert(v::begin(),n,multivector<T,N-1>(value,listPointer));
                va_end(listPointer);
                }

///Auxiliary constructor (recursive)
/**\see multivector(const T&, ...)
 * \see multivector<T,1>
 * */
multivector(const T& value, va_list & listPointer)
                {
                        int n=va_arg(listPointer,int);
                        v::insert(v::begin(),n,multivector<T,N-1>(value,listPointer));
                        }

};

/// Specialization template class of \ref multivector<T,N> for N=1
/**\see \ref multivector<T,N>
 * */
template< class T >
class multivector<T,1>: public vector<T>{
public:
 typedef vector< T > v;
 /// Default constructor
 multivector(): v() {}
 /// Copy constructor
        multivector(const multivector& m): v(m){}
 ///Recommended constructor
 /**\param value the value with which every objects are initialized
 * \param x number of dimensions of the vector
 * */
 multivector(const T& value, int x): v(x,value){}
 ///Auxiliary constructor
/**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 &)
 * */
 multivector(const T& value, va_list & listPointer):
                v(va_arg(listPointer,int), value){}
};

class Matrix: public multivector< ex, 2>{
public:

Matrix(): multivector< ex,2>(0,3,3) {}

Matrix(const Matrix& m): multivector< ex,2>(m) {}
///constructs a symbolic matrix with the symbols names given by the argument
Matrix(const char * m[3][3]): multivector< ex,2>(0,3,3){
        for(uint i=0;i<3;i++)
                for(uint j=0;j<3;j++) at(i)[j]=symbol(m[i][j]);
        }
///constructs a symbolic matrix with the symbols names given by the arguments
Matrix(const char * name,const char ** index1, const char ** index2): multivector< ex,2>(0,3,3){
        for(uint i=0;i<3;i++)
                for(uint j=0;j<3;j++){
                         string res=string(name)+"_{"+string(index1[i])+" "+string(index2[j])+"}";
```

```
                              //cout<<res<<endl;
                              at(i)[j]=symbol(res.c_str());
                              }
        }
///constructs a diagonal matrix
Matrix(ex m1, ex m2, ex m3): multivector< ex,2>(0,3,3) {
        at(0)[0]=m1;
        at(1)[1]=m2;
        at(2)[2]=m3;
}

///constructs a diagonal matrix with all diagonal elements equal
Matrix(ex m1): multivector< ex,2>(0,3,3){
        Matrix();
        at(0)[0]=m1;
        at(1)[1]=m1;
        at(2)[2]=m1;
        }
///constructs a unitary matrix in the standard form
Matrix(ex t12, ex t13, ex t23, ex d13): multivector< ex,2>(0,3,3) {
        Matrix();
        ex c12=cos(t12), c13=cos(t13), c23=cos(t23);
        ex s12=sin(t12), s13=sin(t13), s23=sin(t23);
        ex e13=exp(I*d13);
        ex e13t=ex(1)/e13;

        ex aux[3][3]={
                        {c12*c13,s12*c13,s13*e13t},
                        {-s12*c23-c12*s23*s13*e13,c12*c23-s12*s23*s13*e13,s23*c13},
                        {s12*s23-c12*c23*s13*e13,-c12*s23-s12*c23*s13*e13,c13*c23}
                        };
        for(uint i=0;i<3; i++) at(i).assign(aux[i],aux[i]+3);
        }
///used in the unitary constructor
ex cs(ex t12){
        return (exp(I*t12)+1/exp(I*t12))/2;
        }
///used in the unitary constructor
ex sn(ex t12){
        return -I*(exp(I*t12)-1/exp(I*t12))/2;
        }
///computes the hermitian conjugate of the matrix
Matrix conjugate() const {
        Matrix res;
        for(uint i=0;i<3;i++)
                for(uint j=0;j<3;j++)
                        res[i][j]=at(j)[i].conjugate();

        return res;
}
};

///computes the matrix product
Matrix operator*(const Matrix & m1,const Matrix & m2){
        Matrix res;
        for(uint i=0;i<3;i++)
                for(uint j=0;j<3;j++)
                        for(uint k=0;k<3;k++)
                                res[i][j]=res[i][j]+m1[i][k]*m2[k][j];
        return res;
}

///computes the matrix sum
Matrix operator+(const Matrix & m1,const Matrix & m2){
        Matrix res;
        for(uint i=0;i<3;i++)
                for(uint j=0;j<3;j++)
                                res[i][j]=m1[i][j]+m2[i][j];
        return res;
}



}
```

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 }
```

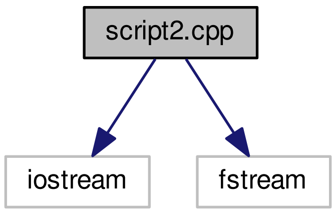
8.28 script.cpp

```
#include <iostream>

using namespace std;
typedef unsigned int uint;
int main(){

string g[3]={"0","1","2"};
string u[3]={"0","1"};

string sg[3]={"1st","2nd","3rd"};
string su[3]={"Down","Up"};

for(uint qup=0;qup<2;qup++)
for(uint gQ=0;gQ<3;gQ++)
{
cout<<"\\pagebreak\n";
for(uint gL=0;gL<3;gL++){
cout<<"\\begin{figure}[!htb]\n\\centering"<<endl;
for(uint lup=0;lup<2;lup++){
cout<<"\\includegraphics[width=0.49\\textwidth]{../pdfs/T_BD/pdf_";
    cout<<g[gL]<<g[gQ]<<u[lup]<<u[qup];
    cout<<".png}"<<endl;
}
cout<<"\\caption{BGL Quarks: gen "<<sg[gQ]<<" FCNC "<<su[qup];
cout<<"; Leptons: gen "<<sg[gL]<<" FCNC chargedlepton/neutrino on the left/right.}"<<endl;
cout<<"\\end{figure}"<<endl<<endl;
}
}
return 0;
}
```

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  }
```

```

 for(uint lup=0;lup<2;lup++)
 for(uint qup=0;qup<2;qup++)
 for(uint gL=0;gL<3;gL++)
 for(uint gQ=0;gQ<3;gQ++){
  cout<<"./update "<<gL<<" "<<gQ<<" "<<lup<<" "<<qup;
  cout<<" > "<<"teste"<<gL<<gQ<<lup<<qup<<".out"<<endl;
 }
return 0;
}
```

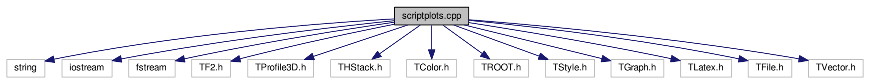
8.30 script2.cpp

```
#include <iostream>
#include <fstream>


using namespace std;
typedef unsigned int uint;
int main(){

string g[3]={"0","1","2"};
string u[3]={"0","1"};

string sg[3]={"1st","2nd","3rd"};
string su[3]={"Down","Up"};

for(uint qup=0;qup<2;qup++){
cout<<"\\pagebreak\n";
cout<<"\\begin{figure}[!htb]\n\\centering"<<endl;
for(uint gL=0;gL<3;gL++)
for(uint lup=0;lup<2;lup++){
cout<<"\\begin{tikzpicture}[every node/.style={anchor=south west,inner sep=0pt},"<<endl;
cout<<"x=0.307692307692\\textwidth,y=0.230769230769\\textwidth,]"<<endl;

for(uint gQ=0;gQ<3;gQ++)
{
                char name[5]="0000";
                name[0]+=gL;
                name[1]+=gQ;
                name[2]+=lup;
                name[3]+=qup;

  ifstream ff((string("pdfs/T_BD/maxs_")+string(name)+string(".out")).c_str());
  if(!ff.is_open()){
                cout<<"ERROR: maxs_*.out not found"<<endl;
                return 1;
        }
  int tx=0, ty=0;
  double xi=0,xxi=0.26,xw=0.384615384615,yi=0.25;
  if(gQ>0) {xi=0.24+gQ; xxi+=gQ; tx=192; xw=0.307692307692;}
  if(gL<2 || lup<1) {ty=144; yi=0.01;}

  double McH=0,MR=0,MI=0;
  ff>>McH>>MR>>MI;
  ff.close();
  int mcH(McH+0.5),mR(MR+0.5),mI(MI+0.5);

  cout<<"\\node at ("<<xi<<",0) {\\includegraphics[trim="<<tx<<" "<<ty<<" 0 0,clip,"<<endl;
  cout<<"width="<<xw<<"\\textwidth]{../pdfs/T_BD/pdf_"<<name<<".png}};"<<endl;
  cout<<"\\node at ("<<xxi<<","<<yi<<") {\\begin{minipage}{0.1\\textwidth}{"<<endl;
  cout<<"\\scriptsize \\begin{align*} M_{H^+}&>"<<mcH<<"\\\\[-4pt]"<<endl;
  cout<<"M_{R^0}&>"<<mR<<"\\\\[-4pt]"<<endl;
  cout<<"M_{I^0}&>"<<mI<<" (GeV)\\end{align*}}\\end{minipage}};"<<endl;


}
cout<<"\\end{tikzpicture}\\\\"<<endl;
}
cout<<"\\caption{BGL Quarks: gen ";
cout<<"; Leptons: gen FCNC chargedlepton/neutrino on the left/right.}"<<endl;
cout<<"\\end{figure}"<<endl<<endl;
}


 for(uint qup=0;qup<2;qup++)
 for(uint lup=0;lup<2;lup++)
 for(uint gL=0;gL<3;gL++)
 for(uint gQ=0;gQ<3;gQ++){
  cout<<"export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH && ./teste "<<gL<<" "<<gQ<<" "<<lup<<" "<<qup;
  cout<<" > "<<"teste"<<gL<<gQ<<lup<<qup<<".out"<<endl;
 }
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++){
```

```

for(uint gL=0;gL<3;gL++)
for(uint lup=0;lup<2;lup++){
f[qup]<<"\\begin{tikzpicture}[every node/.style={anchor=south west,inner sep=0pt},"<<endl;
f[qup]<<"x=0.307692307692\\textwidth,y=0.230769230769\\textwidth,]"<<endl;

for(uint gQ=0;gQ<3;gQ++)
{
                char name[5]="0000";
                name[0]+=gL;
                name[1]+=gQ;
                name[2]+=lup;
                name[3]+=qup;
                char name_[8]="0 0 0 0";
                name_[0]+=gL;
                name_[2]+=gQ;
                name_[4]+=lup;
                name_[6]+=qup;

  //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());

  system((string("cd pdfs/T_BD; ../../draw ./h")+string(name)+string(".root ")+string(name_)).c_str());
  ifstream ff((string("pdfs/T_BD/maxs_")+string(name)+string(".out")).c_str());
  if(!ff.is_open()){
                cout<<"ERROR: maxs_*.out not found"<<endl;
                return 1;
        }
  int tx=0, ty=0;
  double xi=0,xxi=0.26,xw=0.384615384615,yi=0.25;
  yi=0.25;
  if(gQ>0) {xi=0.24+gQ; xxi+=gQ; tx=192; xw=0.307692307692;}
  if(gL<2 || lup<1) {ty=144; yi=0.01;}

  double McH=0,MR=0,MI=0,llmax=-1000,tbmax=0,McHmax=1000,MRmax=1000,MImax=1000;
  double eK__=0,sm__=0,charged__=0,neutral__=0,neutralR__=0,neutralI__=0,all__=0;
  ff>>McH>>MR>>MI;
  if(gQ>0 && qup) {
          ff>>McH>>MR>>MI;
          ff>>McH>>MR>>MI;
  }else{ ff>>llmax>>llmax>>llmax;
    ff>>llmax>>llmax>>llmax;
  }
  ff>>llmax>>tbmax>>McHmax>>MRmax>>MImax;
  //ff>>eK__;
  //ff>>sm__>>charged__>>neutral__>>neutralR__>>neutralI__>>all__;
  ff.close();
  int eK_(eK__*100+0.5);
  int sm_(sm__*100+0.5);
  int charged_(charged__*100+0.5);
  int neutral_(neutral__*100+0.5);
  int neutralR_(neutralR__*100+0.5);
  int neutralI_(neutralI__*100+0.5);
  int all_(all__*100+0.5);
  int mcH(McH+0.5),mR(MR+0.5),mI(MI+0.5);
  double tmax(int(tbmax*100+0.5)/100.0);
  int mcHmax(McHmax+0.5),mRmax(MRmax+0.5),mImax(MImax+0.5);
  f[qup]<<"\\node at ("<<xi<<",0) {\\includegraphics[trim="<<tx<<" "<<ty<<" 0 0,clip,"<<endl;
  f[qup]<<"width="<<xw<<"\\textwidth]{../pdfs/T_BD/pdf_"<<name<<".png}};"<<endl;
  f[qup]<<"\\node at ("<<xxi<<","<<yi<<") {\\begin{minipage}{0.1\\textwidth}{"<<endl;
  f[qup]<<"\\scriptsize \\begin{align*}"<<endl;
  //f[qup]<<tmax<<" & \\ "<<mcHmax<<"\\ "<<mRmax<<"\\ "<<mImax<<"\\\\[-4pt]"<<endl;
  //f[qup]<<sm_<<" & \\ "<<charged_<<"\\ "<<neutral_<<"\\ "<<neutralR_<<"\\ "<<neutralI_<<"\\
 "<<all_<<"\\\\[-4pt]"<<endl;
  //f[qup]<<" & \\ "<<eK_<<"\\\\[-4pt]"<<endl;
  f[qup]<<mcH<<"&<M_{H^+}/\\GeV\\\\[-4pt]"<<endl;
  f[qup]<<mR<<"&<M_{R^0}/\\GeV\\\\[-4pt]"<<endl;
  f[qup]<<mI<<"&<M_{I^0}/\\GeV\\end{align*}}\\end{minipage}};"<<endl;


}
f[qup]<<"\\end{tikzpicture}\\\\"<<endl;
}
f[qup].close();
}

return 0;
}
```

8.32 scriptplots.cpp

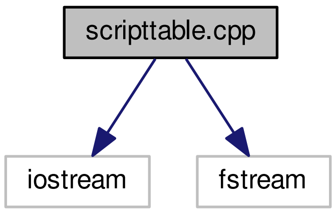
```
#include <string>
```

```
#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"

using namespace std;

int main(){

string g[3]={"0","1","2"};
string u[3]={"0","1"};

string sg[3]={"1st","2nd","3rd"};
string su[3]={"Down","Up"};

ofstream f[2];

f[0].open("draft/large_fig1.tex");
f[1].open("draft/large_fig2.tex");

for(uint qup=0;qup<2;qup++){

for(uint gL=0;gL<3;gL++)
for(uint lup=0;lup<2;lup++){
f[qup]<<"\\begin{tikzpicture}[every node/.style={anchor=south west,inner sep=0pt},"<<endl;
f[qup]<<"x=0.307692307692\\textwidth,y=0.230769230769\\textwidth,]"<<endl;

for(uint gQ=0;gQ<3;gQ++)
{
                char name[5]="0000";
                name[0]+=gL;
                name[1]+=gQ;
                name[2]+=lup;
                name[3]+=qup;
                char name_[8]="0 0 0 0";
                name_[0]+=gL;
                name_[2]+=gQ;
                name_[4]+=lup;
                name_[6]+=qup;

  //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());

  system((string("cd pdfs/T_BD; ../../draw ./h")+string(name)+string(".root ")+string(name_)).c_str());
  ifstream ff((string("pdfs/T_BD/maxs_")+string(name)+string(".out")).c_str());
  if(!ff.is_open()){
                cout<<"ERROR: maxs_*.out not found"<<endl;
                return 1;
        }
  int tx=0, ty=0;
  double xi=0,xxi=0.26,xw=0.384615384615,yi=0.25;
  yi=0.25;
  if(gQ>0) {xi=0.24+gQ; xxi+=gQ; tx=192; xw=0.307692307692;}
  if(gL<2 || lup<1) {ty=144; yi=0.01;}

  double McH=0,MR=0,MI=0,llmax=-1000,tbmax=0,McHmax=1000,MRmax=1000,MImax=1000;
  double eK__=0,sm__=0,charged__=0,neutral__=0,neutralR__=0,neutralI__=0,all__=0;
  ff>>McH>>MR>>MI;
  if(gQ>0 && qup) {
          ff>>McH>>MR>>MI;
          ff>>McH>>MR>>MI;
  }else{ ff>>llmax>>llmax>>llmax;
    ff>>llmax>>llmax>>llmax;
  }
  ff>>llmax>>tbmax>>McHmax>>MRmax>>MImax;
  //ff>>eK__;
  //ff>>sm__>>charged__>>neutral__>>neutralR__>>neutralI__>>all__;
  ff.close();
  int eK_(eK__*100+0.5);
  int sm_(sm__*100+0.5);
  int charged_(charged__*100+0.5);
  int neutral_(neutral__*100+0.5);
  int neutralR_(neutralR__*100+0.5);
  int neutralI_(neutralI__*100+0.5);
  int all_(all__*100+0.5);
  int mcH(McH+0.5),mR(MR+0.5),mI(MI+0.5);
  double tmax(int(tbmax*100+0.5)/100.0);
  int mcHmax(McHmax+0.5),mRmax(MRmax+0.5),mImax(MImax+0.5);
  f[qup]<<"\\node at ("<<xi<<",0) {\\includegraphics[trim="<<tx<<" "<<ty<<" 0 0,clip,"<<endl;
  f[qup]<<"width="<<xw<<"\\textwidth]{../pdfs/T_BD/pdf_"<<name<<".png}};"<<endl;
  f[qup]<<"\\node at ("<<xxi<<","<<yi<<") {\\begin{minipage}{0.1\\textwidth}{"<<endl;
  f[qup]<<"\\scriptsize \\begin{align*}"<<endl;
  //f[qup]<<tmax<<" & \\ "<<mcHmax<<"\\ "<<mRmax<<"\\ "<<mImax<<"\\\\[-4pt]"<<endl;
  //f[qup]<<sm_<<" & \\ "<<charged_<<"\\ "<<neutral_<<"\\ "<<neutralR_<<"\\ "<<neutralI_<<"\\ "<<all_<<"\\\\[-4pt]"<<endl;
  //f[qup]<<" & \\ "<<eK_<<"\\\\[-4pt]"<<endl;
  f[qup]<<mcH<<"&<M_{H^+}/\\GeV\\\\[-4pt]"<<endl;
  f[qup]<<mR<<"&<M_{R^0}/\\GeV\\\\[-4pt]"<<endl;
  f[qup]<<mI<<"&<M_{I^0}/\\GeV\\end{align*}}\\end{minipage}};"<<endl;


}
f[qup]<<"\\end{tikzpicture}\\\\"<<endl;
}
f[qup].close();
}

return 0;
}
```

8.33 scripttable.cpp File Reference

`#include <iostream>`
`#include <fstream>`

Include dependency graph for scripttable.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 scripttable.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 }
```

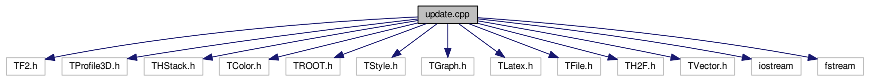
8.34 scripttable.cpp

```
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 resuts 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 resuts in ROOT file1 and file2 producing the output ROOT file

Definition at line 20 of file update.cpp.

```
                                  {
  // Check the number of parameters

  if(argc < 4){
      std::cerr<<"Usage: "<<argv[0]<<" file1 file2 output"<<std::endl;
      return 1;}

      TH2F *limits4,*limits_tb_MR;
      TH2F *limits_MR_MI,*limits_MR_McH,*limits_MI_McH;
      TH2F *limits4_,*limits_tb_MR_;
      TH2F *limits_MR_MI_,*limits_MR_McH_,*limits_MI_McH_;
      TVectorD *vllmax=NULL,*vllmax_=NULL;

      uint npoints=200;

      TFile *f=new TFile(argv[2],"update");
      if(!f->IsOpen()) cout<<"NOFILE"<<endl;
      f->GetObject("vllmax;1",vllmax_);
      f->GetObject("limits4;1",limits4_);
      f->GetObject("limits_tb_MR;1",limits_tb_MR_);
      f->GetObject("limits_MR_MI;1",limits_MR_MI_);
      f->GetObject("limits_MR_McH;1",limits_MR_McH_);
```

```
        f->GetObject("limits_MI_McH;1",limits_MI_McH_);

        if(!vllmax_) cout<<"ERROR"<<endl;

        TFile *f2=new TFile(argv[1],"update");
        if(!f2->IsOpen()) cout<<"NOFILE"<<endl;

        f2->GetObject("vllmax;1",vllmax);
        f2->GetObject("limits4;1",limits4);
        f2->GetObject("limits_tb_MR;1",limits_tb_MR);
        f2->GetObject("limits_MR_MI;1",limits_MR_MI);
        f2->GetObject("limits_MR_McH;1",limits_MR_McH);
        f2->GetObject("limits_MI_McH;1",limits_MI_McH);

        double c=(*vllmax_)[0];
        if(c>(*vllmax)[0]) (*vllmax)[0]=c;
        for(uint i=0;i<npoints;i++)
                for(uint j=0;j<npoints;j++) {
                        c=limits4_->GetBinContent(i+1,j+1);
                        if(c>limits4->GetBinContent(i+1,j+1)) limits4->SetBinContent(i+1,j+1,c);
                        c=limits_tb_MR_->GetBinContent(i+1,j+1);
                        if(c>limits_tb_MR->GetBinContent(i+1,j+1)) limits_tb_MR->SetBinContent(i+1,j+1,c);
                        c=limits_MR_MI_->GetBinContent(i+1,j+1);
                        if(c>limits_MR_MI->GetBinContent(i+1,j+1)) limits_MR_MI->SetBinContent(i+1,j+1,c);
                        c=limits_MR_McH_->GetBinContent(i+1,j+1);
                        if(c>limits_MR_McH->GetBinContent(i+1,j+1)) limits_MR_McH->SetBinContent(i+1,j+1,c);
                        c=limits_MI_McH_->GetBinContent(i+1,j+1);
                        if(c>limits_MI_McH->GetBinContent(i+1,j+1)) limits_MI_McH->SetBinContent(i+1,j+1,c);
                        }
        TFile *f3=new TFile(argv[3],"recreate");
        if(!f3->IsOpen()) cout<<"NOFILE"<<endl;

        vllmax->Write("vllmax");
        limits4->Write();
        limits_MR_MI->Write();
    limits_MR_McH->Write();
    limits_MI_McH->Write();
    limits_tb_MR->Write();

    f3->Close();
        f2->Close();
        f->Close();


        return 0;

}
```

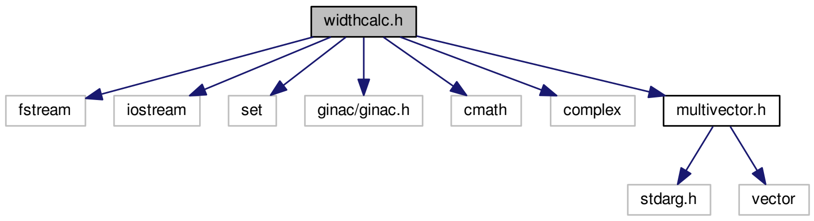
8.36 update.cpp

```
#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>

using namespace std;

/**
* @brief the main function takes the arguments file1 file2 output, merges the resuts in ROOT file1 and
 file2 producing the output ROOT file
*/
int main(int argc, char* argv[]){
    // Check the number of parameters

    if(argc < 4){
        std::cerr<<"Usage: "<<argv[0]<<" file1 file2 output"<<std::endl;
        return 1;}

        TH2F *limits4,*limits_tb_MR;
        TH2F *limits_MR_MI,*limits_MR_McH,*limits_MI_McH;
        TH2F *limits4_,*limits_tb_MR_;
        TH2F *limits_MR_MI_,*limits_MR_McH_,*limits_MI_McH_;
        TVectorD *vllmax=NULL,*vllmax_=NULL;

        uint npoints=200;

        TFile *f=new TFile(argv[2],"update");
        if(!f->IsOpen()) cout<<"NOFILE"<<endl;
        f->GetObject("vllmax;1",vllmax_);
        f->GetObject("limits4;1",limits4_);
        f->GetObject("limits_tb_MR;1",limits_tb_MR_);
        f->GetObject("limits_MR_MI;1",limits_MR_MI_);
        f->GetObject("limits_MR_McH;1",limits_MR_McH_);
        f->GetObject("limits_MI_McH;1",limits_MI_McH_);

        if(!vllmax_) cout<<"ERROR"<<endl;

        TFile *f2=new TFile(argv[1],"update");
        if(!f2->IsOpen()) cout<<"NOFILE"<<endl;

        f2->GetObject("vllmax;1",vllmax);
        f2->GetObject("limits4;1",limits4);
        f2->GetObject("limits_tb_MR;1",limits_tb_MR);
        f2->GetObject("limits_MR_MI;1",limits_MR_MI);
        f2->GetObject("limits_MR_McH;1",limits_MR_McH);
        f2->GetObject("limits_MI_McH;1",limits_MI_McH);

        double c=(*vllmax_)[0];
        if(c>(*vllmax)[0]) (*vllmax)[0]=c;
        for(uint i=0;i<npoints;i++)
                for(uint j=0;j<npoints;j++) {
                        c=limits4_->GetBinContent(i+1,j+1);
                        if(c>limits4->GetBinContent(i+1,j+1)) limits4->SetBinContent(i+1,j+1,c);
                        c=limits_tb_MR_->GetBinContent(i+1,j+1);
                        if(c>limits_tb_MR->GetBinContent(i+1,j+1)) limits_tb_MR->SetBinContent(i+1,j+1,c);
                        c=limits_MR_MI_->GetBinContent(i+1,j+1);
                        if(c>limits_MR_MI->GetBinContent(i+1,j+1)) limits_MR_MI->SetBinContent(i+1,j+1,c);
                        c=limits_MR_McH_->GetBinContent(i+1,j+1);
                        if(c>limits_MR_McH->GetBinContent(i+1,j+1)) limits_MR_McH->SetBinContent(i+1,j+1,c)
;
                        c=limits_MI_McH_->GetBinContent(i+1,j+1);
                        if(c>limits_MI_McH->GetBinContent(i+1,j+1)) limits_MI_McH->SetBinContent(i+1,j+1,c)
;
                        }
        TFile *f3=new TFile(argv[3],"recreate");
        if(!f3->IsOpen()) cout<<"NOFILE"<<endl;

        vllmax->Write("vllmax");
        limits4->Write();
        limits_MR_MI->Write();
    limits_MR_McH->Write();
    limits_MI_McH->Write();
    limits_tb_MR->Write();

    f3->Close();
        f2->Close();
        f->Close();


        return 0;
}

```

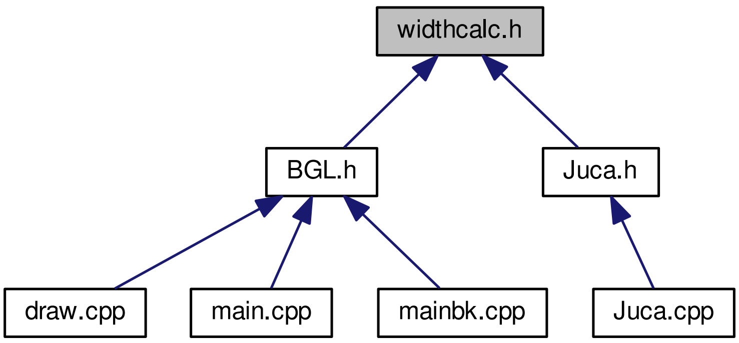
8.37 widthcalc.h File Reference

```
#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

```
#ifndef WIDTHCALC_H
#define WIDTHCALC_H

#define _USE_MATH_DEFINES
#include <fstream>
#include <iostream>
#include <set>
#include <ginac/ginac.h>
#include <cmath>
#include <complex>
#include "multivector.h"


//g++ teste.cpp -o teste -lcln -lginac
using namespace std;
using namespace GiNaC;

/**
* @brief this class calculates decay widths of one lepton to 3 leptons
*/
class widthcalc{

public:

widthcalc(): M2(0,2,2,2,2,2,2,2,2), M22(0,2,2,2,2,2,2,2), s2("s2"), s3("s3"), mq1("mq1"), mq2("mq2"),
mq3("mq3"), mq4("mq4"){

        integral::max_integration_level=100;
        integral::relative_integration_error=1e-3;

        genM2();
        genM22();
}

void genM22(){
        cout<<"Generating M22.dat"<<endl;

        realsymbol mu("mu"), nu("nu"), alpha("alpha"), beta("beta"), rho("rho"), zeta("zeta");
        realsymbol q1("q1"), q2("q2"), q3("q3"), q4("q4");
        realsymbol h1("h1"), h2("h2"), h3("h3"), h4("h4");

        varidx imu(mu,4,0), inu(nu,4,0), irho(rho,4,0);
        varidx ialpha(alpha,4,0), ibeta(beta,4,0), izeta(zeta,4,0);

        varidx jmu(mu,4,1), jnu(nu,4,1), jrho(rho,4,1);

        ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;

        ex q2mu=indexed(q2,imu);
        ex q3mu=indexed(q3,imu);
        ex q4mu=indexed(q4,imu);
        ex q1mu=q2mu+q3mu+q4mu;

        ex s4=m2q1+m2q2+m2q3+m2q4-s2-s3;

        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();
        ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)-mq4*dirac_ONE();

    scalar_products sp;
    //sp.add(q2, q3, (s4-m2q2-m2q3)/2);
    sp.add(q4, q3, (s2-m2q4-m2q3)/2);
    //sp.add(q2, q4, (s3-m2q2-m2q4)/2);

    //sp.add(q2, q2, m2q2);
    sp.add(q3, q3, m2q3);
    sp.add(q4, q4, m2q4);

    //sp.add(h1,h1,-1);
    //sp.add(h2,h2,-1);
    //sp.add(h3,h3,-1);
    //sp.add(h4,h4,-1);
    //sp.add(h1,h1,-1);

    multivector<ex,3> v(0,2,2,2);
        v[0][0][0]=dirac_gammaL(); v[0][0][1]=dirac_gammaR();
        v[0][1][0]=dirac_gammaR(); v[0][1][1]=dirac_gammaL();
        v[1][0][0]=dirac_gammaL(); v[1][0][1]=dirac_gammaL();
        v[1][1][0]=dirac_gammaR(); v[1][1][1]=dirac_gammaR();

        for(uint i=0;i<2;i++)
          for(uint j=0;j<2;j++){
                v[0][i][j]=-v[0][i][j]*s2/(mq1+mq2);
                v[1][i][j]=(dirac_slash(q3,4)+dirac_slash(q4,4))*v[1][i][j];
        }

        //vector<ex> prop(2,0);
        //prop[0]=-s2/(mq1+mq2);
        //prop[1]=indexed(q3,imu.toggle_variance())+indexed(q4,imu.toggle_variance());
        /*
        multivector<ex,2> prop2(0,2,2);
        for(uint i=0;i<2;i++)
          for(uint j=0;j<2;j++){
                                prop2[i][j]=prop[i]*prop[j].subs(mu==nu);
                        }
        */

        //ofstream f("M22.dat");
        for(uint i=0;i<2;i++)
          for(uint j=0;j<2;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++){
                                //cout<<i<<" "<<j<<" "<<k<<" "<<l<<" "<<m<<" "<<n<<endl;
                                //cout<<dirac_trace(vq3*v[i][m][1]*vq4*v[j][n][0].subs(mu==nu))<<endl;

                                ex tmp=dirac_trace(vq3*v[i][m][0]*vq4*v[j][n][1])*int(pow(-1.0,double(k+1+l
)));
                                M22[i][j][k][l][m][n]=tmp.simplify_indexed(sp);
                                //cout<<M22[i][j][k][l][m][n]<<endl<<endl;
                                //f<<M22[i][j][k][l][m][n]<<endl;
                        }
}

void genM2(){
        cout<<"Generating M2.dat"<<endl;

        realsymbol mu("mu"), nu("nu"), alpha("alpha"), beta("beta"), rho("rho"), zeta("zeta");
        realsymbol q1("q1"), q2("q2"), q3("q3"), q4("q4");
        realsymbol h1("h1"), h2("h2"), h3("h3"), h4("h4");

        varidx imu(mu,4,0), inu(nu,4,0), irho(rho,4,0);
        varidx ialpha(alpha,4,0), ibeta(beta,4,0), izeta(zeta,4,0);

        varidx jmu(mu,4,1), jnu(nu,4,1), jrho(rho,4,1);

        ex m2q1=mq1*mq1, m2q2=mq2*mq2, m2q3=mq3*mq3, m2q4=mq4*mq4;

        ex q2mu=indexed(q2,imu);
        ex q3mu=indexed(q3,imu);
        ex q4mu=indexed(q4,imu);
        ex q1mu=q2mu+q3mu+q4mu;



        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();
        ex vq3=dirac_slash(q3,4)+mq3*dirac_ONE(), vq4=dirac_slash(q4,4)-mq4*dirac_ONE();

        ex s4=m2q1+m2q2+m2q3+m2q4-s2-s3;
    scalar_products sp;
    sp.add(q2, q3, (s4-m2q2-m2q3)/2);
    sp.add(q4, q3, (s2-m2q4-m2q3)/2);
    sp.add(q2, q4, (s3-m2q2-m2q4)/2);

    sp.add(q2, q2, m2q2);
    sp.add(q3, q3, m2q3);
    sp.add(q4, q4, m2q4);

    sp.add(h1,h1,-1);
    sp.add(h2,h2,-1);
    sp.add(h3,h3,-1);
    sp.add(h4,h4,-1);

    sp.add(h2,q2,0);
    sp.add(h3,q3,0);
    sp.add(h4,q4,0);

    multivector<ex,3> v(0,2,2,2);
        v[0][0][0]=dirac_gammaL(); v[0][0][1]=dirac_gammaR();
        v[0][1][0]=dirac_gammaR(); v[0][1][1]=dirac_gammaL();
        v[1][0][0]=dirac_gamma(jmu)*dirac_gammaL(); v[1][0][1]=dirac_gamma(jmu)*dirac_gammaL();
        v[1][1][0]=dirac_gamma(jmu)*dirac_gammaR(); v[1][1][1]=dirac_gamma(jmu)*dirac_gammaR();

        multivector<ex,7> traces(0,2,2,2,2,2,2,2);
        for(uint i=0;i<2;i++)
          for(uint j=0;j<2;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 vik=v[i][k][0];
                                ex vim=v[i][m][0].subs(mu==nu);
                                ex vjl=v[j][l][1].subs(mu==alpha);
                                ex vjn=v[j][n][1].subs(mu==beta);

                                traces[i][j][k][l][m][n][0]=dirac_trace(vq2*vik*vq1*vjl)*dirac_trace(vq3*vim*vq4*vjn);
                                traces[i][j][k][l][m][n][1]=-dirac_trace(vq2*vik*vq1*vjl*vq3*vim*vq4*vjn);
                        }

        vector<ex> prop(2,0);
        prop[0]=1;
        prop[1]=lorentz_g(imu,inu);

        multivector<ex,2> prop2(0,2,2);
        for(uint i=0;i<2;i++)
          for(uint j=0;j<2;j++){
                                prop2[i][j]=prop[i]*prop[j].subs(lst(mu==alpha,nu==beta));
                        }

        //ofstream f("M2.dat");
        for(uint i=0;i<2;i++)
          for(uint j=0;j<2;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 o=0;o<2;o++)
                                        {
                                                        //cout<<i<<" "<<j<<" "<<k<<" "<<l<<" "<<m<<endl;
                                                        M2[i][j][k][l][m][n][o]=(traces[i][j][k][l][m][n][o]*prop2[i][j]).simplify_indexed(sp);
                                                        //cout<<M2[i][j][k][l][m][n][o]<<endl<<endl;
                                                        //f<<M2[i][j][k][l][m][n][o]<<endl;
                                                }
}


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 q10=(s2+mq1*mq1-mq2*mq2)/(2*sqrt(s2)), lq1l=sqrt(q10*q10-mq1*mq1);
    ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);
    ex q20=(mq1*mq1+mq2*mq2-s2)/(2*mq1), lq2l=sqrt(q20*q20-mq2*mq2);

    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<<endl;
                                        ex integrand=M2[op[i]][op[j]][k][l][m][n][(r+s)%2];
                                        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)\
                                                                                                                 .eval_integ()/lq1l/sqrt(s2)*lq2l/mq1/mq1);
                                        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(lst(mq1 == 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;
}
ex get_integral_symb(const multivector<ex,4>& a, const vector<ex>& mass, const vector<int>& op, ex m1) const{

        ex q10=(s2+m1*m1)/(2*sqrt(s2)), lq1l=(m1*m1-s2)/(2*sqrt(s2));
    ex q30=sqrt(s2)/2, lq3l=q30;
    ex q20=(m1*m1-s2)/(2*m1), lq2l=q20;

    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<<endl;
                                        ex integrand=M2[op[i]][op[j]][k][l][m][n][(r+s)%2].subs(lst(mq1 == m1, mq2 == 0, mq3 == 0, mq4 == 0));
                                        integrand=expand(integral(s3, m1*m1-2*q10*q30-2*lq1l*lq3l, m1*m1-2*q10*q30+2*lq1l*lq3l, integrand)\
                                                                                                                       .eval_integ()/lq1l/sqrt(s2)*lq2l/m1/m1);
                                        //integrand=integrand.subs(lst(mq1 == m1, mq2 == 0, mq3 == 0, mq4 == 0));
                                        //integrand=integrand.subs(pow(m1,2)/4-s2/2+pow(s2/m1,2)/4==pow((m1-s2/m1)/2,2));


                                        double mm2=0, mm3=0, m4=0;
                                        //if(l) {mm2=m3; mm3=m2;}
                                        ex result=integral(s2,std::pow(mm3+m4,2),pow(m1-mm2,2),integrand).eval_integ()/pow(Pi,3)/512;

                                    ex partial=result*coup/(pow(mass[i],2)*pow(mass[j],2));
                                    total=total+partial;
                        }
                        }
        return total;
}
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 q10=(s2+mq1*mq1-mq2*mq2)/(2*sqrt(s2)), lq1l=sqrt(q10*q10-mq1*mq1);
    ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);
    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, mq1*mq1+mq3*mq3-2*q10*q30-2*lq1l*lq3l, mq1*mq1+mq3*mq3-2*q10*q30+2*lq1l*lq3l, integrand)/lq1l/s2);
                                        ex result=integrand.subs(lst(sqrt(s2) == mm, s2==mm*mm, mq1 == m1, mq2 == m2, mq3 == m3, mq4 == m4))/Pi/128;
                                        ex mi=mass[i];
                                        if(mi.is_zero()) mi=mm;
                                        ex mj=mass[j];
                                        if(mj.is_zero()) mj=mm;
                                    ex partial=result*coup/(pow(mi,2)*pow(mj,2));
                                    //cout<<i<<"-"<<op[i]<<" "<<j<<"-"<<op[j]<<" "<<a[i]*a[j].conjugate()/(pow(mass[i],2)*pow(mass[j],2))<<endl<<endl;

                                    total=total+partial;
                        }
                }

        return total;
}

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{

        ex q10=(s2+mq1*mq1-mq2*mq2)/(2*sqrt(s2)), lq1l=sqrt(q10*q10-mq1*mq1);
    ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);
    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, mq1*mq1+mq3*mq3-2*q10*q30-2*lq1l*lq3l
, mq1*mq1+mq3*mq3-2*q10*q30+2*lq1l*lq3l, integrand)/lq1l/s2);
                                        ex result=integrand.subs(lst(sqrt(s2) == mm, s2==mm*mm, mq1 == m1,
mq2 == m2, mq3 == m3, mq4 == m4))/Pi/128;
                                        ex mi=mass[i];
                                        if(mi.is_zero()) mi=mm;
                                        ex mj=mass[j];
                                        if(mj.is_zero()) mj=mm;
                                    ex partial=result*coup/(pow(mi,2)*pow(mj,2));
                                    //cout<<i<<"-"<<op[i]<<" "<<j<<"-"<<op[j]<<"
 "<<a[i]*a[j].conjugate()/(pow(mass[i],2)*pow(mass[j],2))<<endl<<endl;

                                    total=total+partial;
                        }
                }

        return total;
}
/*
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{

        ex q10=(meson_mass)/(2), lq1l=sqrt(q10*q10-mq1*mq1);
    ex q30=(s2+mq3*mq3-mq4*mq4)/(2*sqrt(s2)), lq3l=sqrt(q30*q30-mq3*mq3);
    ex q20=(mq1*mq1+mq2*mq2-s2)/(2*mq1), lq2l=sqrt(q20*q20-mq2*mq2);


    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++) if(!(a[i][k]*a[j][l]).is_zero()){
                                        //cout<<i<<" "<<j<<" "<<k<<" "<<l<<endl;
                                        ex integrand=M2[op[i]/2][op[j]/2][op[i]%2][op[j]%2][(k+l)%2];
                                        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)\

.eval_integ()/lq1l/sqrt(s2)*lq2l/mq1/mq1);
                                        double mm2=m2, mm3=m3;
                                        if(l) {mm2=m3; mm3=m2;}
                                        double
 result=ex_to<numeric>(integral(s2,pow(mm3+m4,2),pow(m1-mm2,2),integrand.subs(lst(mq1 == m1, mq2 == mm2, mq3 == mm3, mq
                                    ex partial=result*a[i][k]*a[j][l].conjugate()/(pow(mass[i],2)*
pow(mass[j],2));
                                    //cout<<partial<<endl;
                                    total=total+partial;
                        }

        return total;
  }
*/
multivector<ex,7> M2;
multivector<ex,6> M22;
realsymbol s2, s3;
realsymbol mq1, mq2, mq3, mq4;

};



#endif
```

Index