**July 10, 2012**

**Description of BM, Version 4.03.003 (based on version 3.04.033)**

[TODO 5](#_Toc329729157)

[Bug Fix & History 5](#_Toc329729158)

[Version 4.03.003 (June 2011) 5](#_Toc329729159)

[Version 4.02.001 (November 2010) 5](#_Toc329729160)

[Version 4.01.001 (March 2010) 5](#_Toc329729161)

[Version 4.00.000 (January 2 2010) 5](#_Toc329729162)

[Version 3.04.033 (October 2005) 5](#_Toc329729163)

[Version 3.03.022 (April 2005) 5](#_Toc329729164)

[Version 3.02.020 (January 2005) 5](#_Toc329729165)

[Version 3.01.017 (September 2004) 5](#_Toc329729166)

[Versions 3.00.000 - 3.00.015 (March 2004) 6](#_Toc329729167)

[General Comments 6](#_Toc329729168)

[Berreman Common 6](#_Toc329729169)

[Options & Constants: 6](#_Toc329729170)

[UseEulerAngles  False 6](#_Toc329729171)

[CalculateDelta  False 6](#_Toc329729172)

[CalculateBoundarySolution False 6](#_Toc329729173)

[OutputPPPMultiplier  0 6](#_Toc329729174)

[NOT DESCRIBED: UseSolveInSolutionNew 6](#_Toc329729175)

[NOT DESCRIBED: SwapEigenValues 6](#_Toc329729176)

[NOT DESCRIBED: AddOnEigenValuesSort 6](#_Toc329729177)

[UseNumericEigenValues 🡪 True 6](#_Toc329729178)

[mm = 10^-3 6](#_Toc329729179)

[mkm = 10^-6 6](#_Toc329729180)

[nm = 10^-9 7](#_Toc329729181)

[Functions 7](#_Toc329729182)

[JoinRight[lst1, lst2, ... lst4] 7](#_Toc329729183)

[TimePad[s] 7](#_Toc329729184)

[ConvertToTime[t] 7](#_Toc329729185)

[RotationNew[fi,theta,psi,opts] 7](#_Toc329729186)

[Transform[ee, rotn] 7](#_Toc329729187)

[PsiAngle[fita, n1, n2] 7](#_Toc329729188)

[EulerFi[fi,psi,alpha] 7](#_Toc329729189)

[EulerTheta[fi,psi,alpha] 7](#_Toc329729190)

[EulerPsi[fi,psi,alpha] 7](#_Toc329729191)

[FilmLayerNew[Thickness, Epsilon, Mu, Ro] (!!! CHANGED FROM VERSION 3.03 !!!) 7](#_Toc329729192)

[FilmLayerTransform[FilmLayer, Rotation, Optional Reset = True] 7](#_Toc329729193)

[FilmLayerTransformBase[FilmLayer, Rotation] 7](#_Toc329729194)

[FilmLayerThickness[FilmLayer] 8](#_Toc329729195)

[FilmLayerEpsilon[FilmLayer] 8](#_Toc329729196)

[FilmLayerEpsilonBase[FilmLayer] 8](#_Toc329729197)

[FilmLayerMu[FilmLayer] 8](#_Toc329729198)

[FilmLayerMuBase[FilmLayer] 8](#_Toc329729199)

[FilmLayerRo[FilmLayer] 8](#_Toc329729200)

[FilmLayerRoBase[FilmLayer] 8](#_Toc329729201)

[FilmLayerRoT[FilmLayer] 8](#_Toc329729202)

[FilmLayerRoTBase[FilmLayer] 8](#_Toc329729203)

[FilmNew[] 8](#_Toc329729204)

[FilmAddLayer[Film, FilmLayer] 8](#_Toc329729205)

[FilmItem[Film, LayerNo] 8](#_Toc329729206)

[FilmLength[Film] 8](#_Toc329729207)

[FilmTransformAll[Film, Rotation, Reset] 8](#_Toc329729208)

[MediaNew[n1, n2, gamma, Film, Description, nOut, h2, epsilon2, mu2, ro2, epsilon, mu, ro]– (!!! CHANGED FROM VERSION 4.00 and prior to that from version 3.03 !!!) 8](#_Toc329729209)

[MediaFlip[Media] 9](#_Toc329729210)

[MediaUpperRefractionIndex[Media] 9](#_Toc329729211)

[MediaLowerRefractionIndex[Media] 9](#_Toc329729212)

[MediaGamma[Media] 9](#_Toc329729213)

[MediaOutRefractionIndex [Media] 9](#_Toc329729214)

[MediaFilm[Media] 9](#_Toc329729215)

[MediaFilmLength[Media] 9](#_Toc329729216)

[MediaDescription[Media] 9](#_Toc329729217)

[MediaUpperEpsilon[Media], MediaUpperMu[Media], MediaUpperRo[Media], MediaUpperRoT[Media] 9](#_Toc329729218)

[MediaSubstrateThickness[Media] 9](#_Toc329729219)

[MediaLowerEpsilon[Media], MediaLowerMu[Media], MediaLowerRo[Media], MediaLowerRoT[Media] 9](#_Toc329729220)

[IncidentLightNew[lambda, fita, beta, n1, Ampl] – (!!! CHANGED FROM VERSION 4.00 !!!) 9](#_Toc329729221)

[IncidentLightNew[lambda, fita, beta, eh] – (!!! CHANGED FROM VERSION 4.00 !!!) 9](#_Toc329729222)

[IncidentLightNew[lambda, fita, beta, n1] – (!!! ADDED IN VERSION 4.02 !!!) 10](#_Toc329729223)

[IncidentLightLambda[IncidentLight] 10](#_Toc329729224)

[IncidentLightFita[IncidentLight] 10](#_Toc329729225)

[IncidentLightBeta[IncidentLight] 10](#_Toc329729226)

[IncidentLightIsDown[IncidentLight] 10](#_Toc329729227)

[IncidentLightEH[IncidentLight] 10](#_Toc329729228)

[IncidentLightFlip[IncidentLight] 10](#_Toc329729229)

[EpsilonFromN[n1] 10](#_Toc329729230)

[EpsilonFromN[n1,n2,n3] 10](#_Toc329729231)

[Output – Functions and objects 10](#_Toc329729232)

[SolutionNew[Media, IncidentLight, Options] 10](#_Toc329729233)

[EGGetOrder[egvl : {\_, \_, \_, \_}] 10](#_Toc329729234)

[GetSolIncidentLight[sol] 10](#_Toc329729235)

[GetSolReflectedLight[sol] 10](#_Toc329729236)

[GetSolTransmittedLight[sol] 11](#_Toc329729237)

[GetSolIncidentLightE[sol 11](#_Toc329729238)

[GetSolReflectedLightE[sol] 11](#_Toc329729239)

[GetSolTransmittedLightE[sol] 11](#_Toc329729240)

[GetSolIncidentLightH[sol 11](#_Toc329729241)

[GetSolReflectedLightH[sol] 11](#_Toc329729242)

[GetSolTransmittedLightH[sol] 11](#_Toc329729243)

[GetSolIncidentLightD[sol 11](#_Toc329729244)

[GetSolReflectedLightD[sol] 11](#_Toc329729245)

[GetSolTransmittedLightD[sol] 11](#_Toc329729246)

[GetSolIncidentLightB[sol 11](#_Toc329729247)

[GetSolReflectedLightB[sol] 11](#_Toc329729248)

[GetSolTransmittedLightB[sol] 11](#_Toc329729249)

[GetEfromEH[eh] 11](#_Toc329729250)

[GetHfromEH[eh] 11](#_Toc329729251)

[GetSolPPP[sol] 11](#_Toc329729252)

[GetSolDelta[sol] 11](#_Toc329729253)

[SolutionCombine[ehi,ehr,eht,ppp,delta] 11](#_Toc329729254)

[EHFlip[eh] 12](#_Toc329729255)

[GetSolMedia[sol], GetSolIncidentLight[sol], GetSolOptions[sol] 12](#_Toc329729256)

[PPP[eps, mu, ro, rotr, lambda, fita, n1, h] 12](#_Toc329729257)

[GetSolIncidentLightInfo[sol] 12](#_Toc329729258)

[GetSolOptions[sol] 12](#_Toc329729259)

[GetSolCoeff[sol] 12](#_Toc329729260)

[GetSolFreeTerm[sol] 12](#_Toc329729261)

[GetSolEGSys1[sol] 12](#_Toc329729262)

[GetSolEGSys2[sol] 12](#_Toc329729263)

[GetSolEHTEG[sol, idx] 12](#_Toc329729264)

[PoyntingS[ehFld, fldIdx] 12](#_Toc329729265)

[Berreman Direct 13](#_Toc329729266)

[Options & Constants: 13](#_Toc329729267)

[RotateAll True 13](#_Toc329729268)

[ConsecutiveRotation  True 13](#_Toc329729269)

[PrintTimeEstimate  True 13](#_Toc329729270)

[UseThickLastLayer  False 13](#_Toc329729271)

[NoOfAveragingPoints  10 13](#_Toc329729272)

[AveragingPeriods  1 13](#_Toc329729273)

[AveragingType  BDAVGTYPESERIES 13](#_Toc329729274)

[PrintCalculationProgress False 13](#_Toc329729275)

[PrintCalculationDetailsFalse 13](#_Toc329729276)

[TransmittedAnalyzerAngle  0 14](#_Toc329729277)

[TransmittedAnalyzerParallelAmplitude  1 14](#_Toc329729278)

[TransmittedAnalyzerCrossedAmplitude  0 14](#_Toc329729279)

[ReflectedAnalyzerAngle  0 14](#_Toc329729280)

[ReflectedAnalyzerParallelAmplitude  1 14](#_Toc329729281)

[ReflectedAnalyzerCrossedAmplitude  0 14](#_Toc329729282)

[AnalyzerAngleAbsoluteValue  False 14](#_Toc329729283)

[NOT SUPPORTED YET averaging Types: 14](#_Toc329729284)

[UsePlotLabel3D  True 14](#_Toc329729285)

[UseChop  True 14](#_Toc329729286)

[ChopTolerance  10^-7 14](#_Toc329729287)

[ChopPrecision  6 14](#_Toc329729288)

[PrintFunctionDescription  True 14](#_Toc329729289)

[Variables: 14](#_Toc329729290)

[VarListNew[IncidentLightInfo, CommonAnglesInfo] 15](#_Toc329729291)

[VarListAddLayer[VarList, LayerAngleInfo] 15](#_Toc329729292)

[CalcNew[Media, VarList, FuncList, Description, opts] (!!! CHANGED FROM VERSION 3.03 !!!) 15](#_Toc329729293)

[CalcPerform[Calc] 15](#_Toc329729294)

[CalcGetInput[Calc] 15](#_Toc329729295)

[CalcGetOutput[Calc] 15](#_Toc329729296)

[CalcGetFuncNames[Calc] 15](#_Toc329729297)

[CalcGetVarNames[Calc] 15](#_Toc329729298)

[CalcGetNumberOfResults[Calc] 16](#_Toc329729299)

[CalcMedia[Calc] 16](#_Toc329729300)

[CalcDescription[Calc] 16](#_Toc329729301)

[CalcSave[Calc, FileName, OverWrite=True] 16](#_Toc329729302)

[CalcPrintRunTime[Calc] 16](#_Toc329729303)

[CalcPrintTimeEstimate[Calc] 16](#_Toc329729304)

[CalcOptions[Calc] 16](#_Toc329729305)

[CalcPlot[Calc, pltOptsRaw] 16](#_Toc329729306)

[CalcPlotFunc[Calc, Func, FuncName, pltOptsRaw] 16](#_Toc329729307)

[CalcPlot3D[Calc, SwapXY, pltOptsRaw] 16](#_Toc329729308)

[CalcPlot3DFunc[Calc, Func, FuncName, SwapXY, pltOptsRaw] 16](#_Toc329729309)

[CalcFuncPlot[CalcFunc, xVarLst:{xMin, xMax, xName},pltOptsRaw] 16](#_Toc329729310)

[CalcFuncPlotFunc[CalcFunc,xVarLst:{xMin, xMax, xName}, Func, FuncName, pltOptsRaw] ] 16](#_Toc329729311)

[CalcFuncPlot3D[CalcFunc, xVarLst:{xMin, xMax, xName},yVarLst:{yMin, yMax, yName}, pltOptsRaw] ] 16](#_Toc329729312)

[CalcFuncPlot3DFunc[CalcFunc, xVarLst:{xMin, xMax, xName}, yVarLst:{yMin, yMax, yName}, Func, FuncName, pltOptsRaw] 17](#_Toc329729313)

[CalcCollectionNew[BaseDir, BaseFileName, Description] 17](#_Toc329729314)

[CalcCollectionAddCalc[coll, Calc, FileName] 17](#_Toc329729315)

[CalcCollectionPerform[coll] 17](#_Toc329729316)

[CalcCollectionLength[coll] 17](#_Toc329729317)

[CalcCollectionItem[coll, idx] 17](#_Toc329729318)

[CalcCollectionSave[coll, DoPrintOut] 17](#_Toc329729319)

[CalcCollCombine[coll, funcNo, Optional HorizontalVarInputNo = 0, Optional VerticalVarInputNo = 0, Optional OutputHeader = False] 17](#_Toc329729320)

[CalcCollectionPlot[coll, pltOptsRaw] 17](#_Toc329729321)

[CalcCollectionPlot3D[coll, SwapXY, pltOptsRaw] 17](#_Toc329729322)

[SolutionGetTransmittedAnalyzerInfo[sol] 17](#_Toc329729323)

[SolutionGetReflectedAnalyzerInfo[sol] 17](#_Toc329729324)

[AnalyzerInfoGetAngle[anInf] 17](#_Toc329729325)

[AnalyzerInfoGetParallelAmplitude[anInf] 17](#_Toc329729326)

[AnalyzerInfoGetCrossedAmplitude[anInf] 18](#_Toc329729327)

[Field Algebra 19](#_Toc329729328)

[Options & Constants 19](#_Toc329729329)

[FunctionName  "Unknown" 19](#_Toc329729330)

[FunctionDescription  "Unknown" 19](#_Toc329729331)

[NonAverageable  False 19](#_Toc329729332)

[NonAverageableIndex  1 19](#_Toc329729333)

[UseAnalyzer  False 19](#_Toc329729334)

[AbsoluteAzimuth  True 19](#_Toc329729335)

[AzimuthRangeType  0 19](#_Toc329729336)

[AzimuthShiftType  1 19](#_Toc329729337)

[Functions and Function Options. 19](#_Toc329729338)

[GetFunctionName[Func] 19](#_Toc329729339)

[IFull[FullSol] 20](#_Toc329729340)

[RFull[FullSol] 20](#_Toc329729341)

[TFull[FullSol] 20](#_Toc329729342)

[Rx[FullSol] 20](#_Toc329729343)

[Ry[FullSol] 20](#_Toc329729344)

[Tx[FullSol] 20](#_Toc329729345)

[Ty[FullSol] 20](#_Toc329729346)

[Xit[FullSol] 20](#_Toc329729347)

[XitDegree[FullSol] 20](#_Toc329729348)

[Elt[FullSol] 20](#_Toc329729349)

[Xir[FullSol] 20](#_Toc329729350)

[XirDegree[FullSol] 20](#_Toc329729351)

[Elr[FullSol] 21](#_Toc329729352)

[RAnalyzer[FullSol] 21](#_Toc329729353)

[TAnalyzer[FullSol] 21](#_Toc329729354)

[DetM1[FullSol] 21](#_Toc329729355)

[DetM2[FullSol] 21](#_Toc329729356)

[LDetM2[FullSol] 21](#_Toc329729357)

[DetPPP[FullSol] 21](#_Toc329729358)

[DetCoeff::usage="DetCoeff[FullSol]"; 21](#_Toc329729359)

[ReDetCoeff[FullSol] 21](#_Toc329729360)

[ImDetCoeff[FullSol] 21](#_Toc329729361)

[RxAccuracy[FullSol] 21](#_Toc329729362)

[DetM2Accuracy[FullSol] 21](#_Toc329729363)

[DetCoeffAccuracy[FullSol] 22](#_Toc329729364)

[M2EValRe[evi, FullSol] 22](#_Toc329729365)

[M2EValIm[evi, FullSol] 22](#_Toc329729366)

[PoyntingI[xyz,FullSol] 22](#_Toc329729367)

[PoyntingIFull[FullSol] 22](#_Toc329729368)

[PoyntingR[xyz,FullSol] 22](#_Toc329729369)

[PoyntingRFull[FullSol] 22](#_Toc329729370)

[PoyntingT[xyz,FullSol] 22](#_Toc329729371)

[PoyntingTFull[FullSol] 22](#_Toc329729372)

[PoyntingRTFull[FullSol] 22](#_Toc329729373)

[PoyntingIRTzDiff[FullSol] 22](#_Toc329729374)

[PoyntingVector2[evi,xyz,FullSol] 22](#_Toc329729375)

[PoyntingXYPhase2[evi,FullSol] 23](#_Toc329729376)

[PoyntingXY2[evi,FullSol] 23](#_Toc329729377)

[ReEGE2[evi,xyz,FullSol] 23](#_Toc329729378)

[ImEGE2[evi,xyz,FullSol] 23](#_Toc329729379)

[ReEGH2[evi,xyz,FullSol] 23](#_Toc329729380)

[ImEGH2[evi,xyz,FullSol] 23](#_Toc329729381)

[LRFull[FullSol] 23](#_Toc329729382)

[LTFull[FullSol] 23](#_Toc329729383)

[LRx[FullSol] 23](#_Toc329729384)

[LRy[FullSol] 23](#_Toc329729385)

[LTx[FullSol] 23](#_Toc329729386)

[LTy[FullSol] 23](#_Toc329729387)

[EpsComponent[UpDown,ReIm,xyzRow,xyzCol,FullSol] 23](#_Toc329729388)

[MuComponent[UpDown,ReIm,xyzRow,xyzCol,FullSol] 24](#_Toc329729389)

[RoComponent[UpDown,ReIm,xyzRow,xyzCol,FullSol] 24](#_Toc329729390)

[FieldIO 24](#_Toc329729391)

[OutputCopyright[] 24](#_Toc329729392)

[OutputFunc[Calc, fName, OverWrite] 24](#_Toc329729393)

[Berreman Inverse 25](#_Toc329729394)

[Options & Constants 25](#_Toc329729395)

[LambdaMultiplier  nm 25](#_Toc329729396)

[AngleMultiplier  Degree 25](#_Toc329729397)

[DataStartRow  2 25](#_Toc329729398)

[LambdaColumn  3 25](#_Toc329729399)

[FitaColumn  4 25](#_Toc329729400)

[BetaColumn  5 25](#_Toc329729401)

[GammaColumn  6 25](#_Toc329729402)

[OutputStartColumn  9 25](#_Toc329729403)

[OutputColumns {{ strUseAllOutputColumns}} 25](#_Toc329729404)

[MinizationMultiplier 10^9 25](#_Toc329729405)

[UseStepMonitor  True 25](#_Toc329729406)

[UseEvaluationMonitor  False 25](#_Toc329729407)

[UseNMinimize  False 25](#_Toc329729408)

[EpsilonType  BIEPSTYPESTDTWOSTEPTRAPSPARENT 25](#_Toc329729409)

[AngleRoundingMultiplier  100 25](#_Toc329729410)

[UseArbitraryPhase  False 25](#_Toc329729411)

[EpsilonLowerBoundary  1.9, EpsilonUpperBoundary  4 26](#_Toc329729412)

[EpsilonNonDiagLowerBoundary  -0.4, EpsilonNonDiagUpperBoundary  0.4 26](#_Toc329729413)

[MFHlpDataSet  1 26](#_Toc329729414)

[FitaTolerance  (1/10^4) 26](#_Toc329729415)

[EpsilonIMLowerBoundary  -BIIMTOLERANCE, EpsilonIMUpperBoundary  4 26](#_Toc329729416)

[EpsilonIMNonDiagLowerBoundary  -1, EpsilonIMNonDiagUpperBoundary  14 26](#_Toc329729417)

[MFHlpAbsorbingEpsilon  False 26](#_Toc329729418)

[EpsilonStart  EpsilonFromN[1.5,1.9,1.7] 26](#_Toc329729419)

[PrintEpsilon  True 26](#_Toc329729420)

[BIIMTOLERANCE=10^-20 26](#_Toc329729421)

[BIEPSTYPESTDTWOSTEPTRAPSPARENT=1 26](#_Toc329729422)

[BIEPSTYPESTDTWOSTEPABSORBING=101 26](#_Toc329729423)

[Functions. 26](#_Toc329729424)

[InverseSolutionNew[RefrIndList, InptList, FuncList, FuncNameList, Description, options] (changed in version 4.02) 26](#_Toc329729425)

[InverseSolutionRun[InvSol] 26](#_Toc329729426)

[InverseSolutionEpsilon[InvSol] 26](#_Toc329729427)

[InverseSolutionAngles[InvSol] 26](#_Toc329729428)

[MFHlp[Epsilon, InvSol, mfopts] 27](#_Toc329729429)

[ProcessMinSolEps[Epsilon, InvSol] 27](#_Toc329729430)

[EpsilonComp[eps0, deltaEps, nu] ] 27](#_Toc329729431)

[EpsilonFull[eps0, deltaEps, nu, eps13, eps23, eps33] ] 27](#_Toc329729432)

[MFHlpSetEpsilonLimits[Epsilon, opts] ] 27](#_Toc329729433)

[MFHlpSetParLimits[par, minVal, maxVal, minIMVal, maxIMVal] ] 27](#_Toc329729434)

# 

# TODO

FieldAlgebra

PrintFunctionDebugInfo -> False, PrintFunctionDebugInfoLevel -> 0

EllipticityCutOffForAzimyth

Xii::usage = "Xii[FullSol]";

Options[Xii] = {FunctionName -> "Xii",

FunctionDescription -> "Azimuth of Incident light.",

NonAverageable -> True, NonAverageableIndex -> 1};

XiiDegree::usage = "XiiDegree[FullSol]";

Options[XiiDegree] = {FunctionName -> "Xii (Degree)",

FunctionDescription -> "Azimuth of Incident light in Degrees.",

NonAverageable -> True, NonAverageableIndex -> 1};

XiiNew::usage = "XiiNew[FullSol]";

Options[XiiNew] = {FunctionName -> "XiiNew",

FunctionDescription -> "Azimuth of Incident light (new).",

NonAverageable -> True, NonAverageableIndex -> 1};

XiiDegreeNew::usage = "XiiDegreeNew[FullSol]";

Options[XiiDegreeNew] = {FunctionName -> "XiiNew (Degree)",

FunctionDescription ->

"Azimuth of Incident light in Degrees (new).",

NonAverageable -> True, NonAverageableIndex -> 1};

Eli::usage = "Eli[FullSol]";

Options[Eli] = {FunctionName -> "Eli",

FunctionDescription -> "Ellipticity of Incident light.",

NonAverageable -> True, NonAverageableIndex -> 1};

EliNew::usage = "EliNew[FullSol]";

Options[EliNew] = {FunctionName -> "EliNew",

FunctionDescription -> "Ellipticity of Incident light (new).",

NonAverageable -> True, NonAverageableIndex -> 1};

BerremanCommon

PrintCommonDebugInfo -> False, PrintCommonDebugInfoLevel -> 0

IncidentLightNew[lambda\_,fita\_,beta\_,n1\_,Ampl\_, ellipticity\_]

IncidentLightEllipticity[IncidentLight]

BerremanDirect -> UseParallelTable 🡪 True

VarListGetLambda::usage = "";

VarListGetFita::usage = "";

VarListGetBeta::usage = "";

VarListGetGamma::usage = "";

VarListGetEllipticity::usage = "";

VarListGetCommonFi::usage = "";

VarListGetCommonTheta::usage = "";

VarListGeCommonPsi::usage = "";

VarListGetFi::usage = "";

VarListGetTheta::usage = "";

VarListGePsi::usage = "";

VarListNew[ IncidentLightInfo : {{\_, \_, \_, \_, \_}, {\_, \_, \_, \_, \_}, {\_, \_, \_, \_, \_}, {\_, \_, \_, \_, \_}, {\_, \_, \_, \_}}, CommonAnglesInfo : {{\_, \_, \_, \_, \_}, {\_, \_, \_, \_, \_}, {\_, \_, \_, \_, \_}}]

# Bug Fix & History

### Version 4.03.003 (June 2011 – July 2012)

Additional testing was performed to ensure that the new version is working. In particular the issues with the I = R + T were investigated.

### Version 4.02.001 (November 2010)

Additional testing was performed to ensure that the new version is working.

### Version 4.01.001 (March 2010)

Some parameters were moved here and there in preparation for the GUI release.

### Version 4.00.000 (January 2 2010)

This is a technical release based on version 3.04.033 to make the files compatible with Mathematica 7.00. Some small issues in EigenVectors were identified which resulted in introduction of a new parameter in BerremanCommon (UseNumericEigenValues 🡪 True). If value of that parameter == True (default) then eigenvectors are calculated numerically. This allows fixing the issue with some exact values for eigenvectors.

### Version 3.04.033 (October 2005)

A problem in **BerremanCommon**, which was discovered during transition between versions 3.01.017 and 3.02.20 and partially fixed in 3.03.22 was finally identified and fixed. That problem resulted in incorrect decision in a certain area of angles of biaxial and uniaxial crystals. It was due to incorrect determination of up and down eigenvectors in biaxial and uniaxial crystals. Numerous functions were added in FieldAlgebra, initially in an attempt to discover the problem.

### Version 3.03.022 (April 2005)

A problem in **BerremanCommon**, which was discovered during transition between versions 3.01.017 and 3.02.20 was finally fixed, hopefully completely. That problem resulted in incorrect decision in a certain area of angles of biaxial and uniaxial crystals.

### Version 3.02.020 (January 2005)

Version number was incremented throughout all packages to maintain consistency. No major changes were made compared to version 3.01.017. A problem in **BerremanCommon** was discovered, which result in incorrect decision in a certain area of angles of biaxial crystals.

### Version 3.01.017 (September 2004)

1. Fixed **TransformMedia** – in the previous version it did not rotate the media substrate for angle **Gamma**. For isotropic media that bug did not have any complications. However, for anisotropic media that resulted in errors in calculations.
2. Fixed **GetSolIncidentLightInfo[sol]** – by mistake two different functions with the same name were used. That resulted in erroneous calculations of **IFull** for thick plate.
3. Fixed problem in **SolutionNew** resulted from approximate calculations of Eigenvalues. If two eigenvalues were real and two were complex, then sometimes small (about 10-16) imaginary / real parts appeared in real / imaginary eigenvalues. Used **Chop** to get rid of that.
4. Changed **CalcPlot3D** to speed up calculation of several functions.

### Versions 3.00.000 - 3.00.015 (March 2004)

Major rebuilt from previous version 2.xx Modular structure was implemented to allow easier update of portions of the code.

# General Comments

The program consists of several modules. The description of each module along with the functions implemented is presented below.

# Berreman Common

Berreman Common implements the basic feature of Method of Berreman. In effect it allows solving a direct problem of light propagation though a layered media. No assumptions are made. The layered media is considered between two upper and lower) **semi-infinite isotropic** media. The light is considered as a plane wave and it is incident from the upper semi-infinite media.

## Options & Constants:

**For each option a default value is listed. That is if there is no option specified, then a default value is used.**

### UseEulerAngles  False

This option specifies the type of angles to use in rotation transformations. If set to **False** (default) then media transformation angles are used, if set to **True** then Euler angles are used.

### CalculateDelta  False

This option specifies whether or not to calculate 4x4 matrix delta for the media. Default value is False as for multi layered media delta does not have much sense.

### CalculateBoundarySolution False

This option specifies that only a boundary solution should be calculated. Generally used internally.

### OutputPPPMultiplier  0

This option is used internally.

### NOT DESCRIBED: UseSolveInSolutionNew

### NOT DESCRIBED: SwapEigenValues

### NOT DESCRIBED: AddOnEigenValuesSort

### UseNumericEigenValues 🡪 True

If value of this option == True (default) then eigenvectors are calculated numerically (using N[]). This allows to fix some issues in discovered in Mathematica 7.01, which resulted in a crash. If value == False, then the eigenvectors are not evaluated numerically.

### mm = 10^-3

Constant to be used as a multiplier to specify millimeters

### mkm = 10^-6

Constant to be used as a multiplier to specify micrometers

### nm = 10^-9

Constant to be used as a multiplier to specify nanometers

## Functions

### JoinRight[lst1, lst2, ... lst4]

This function is used for joining list to the right side (instead of to the bottom as standard Join does).

### TimePad[s]

This function is used to formatting seconds, minutes of hours to two digits (for example TimePad[1] = “01”)

### ConvertToTime[t]

This function converts real number t (usually obtained by calling to TimeUsed[]) into time format: hh:mm:ss

### RotationNew[fi,theta,psi,opts]

This function created a rotation object, which is later used to perform transformation of other objects, for example, epsilon matrix. The parameters supplied are: **fi, theta, psi** – angles of rotation and **opts** – Options. Only one option is used by **RotationNew**: **UseEulerAngles  False** (default). In that case media rotation angles are used instead of Euler angles. **UseEulerAngles  True**. In that case Euler angles are used for creating a rotation object. It should be noted that Rotation Object is nothing else but 3x3 transformation matrix with the determinant = 1 and some other limitations. Please, refer to the literature for the discussion regarding rotation matrices.

### Transform[ee, rotn]

This function is used to transform 3x3 matrix **ee** using rotation **rotn**. As described above, first we create the rotation object and then we use the rotation object to perform rotation transformation.

### PsiAngle[fita, n1, n2]

This function implements the Snellius law for transmitted light and calculated the angle of propagation for that light. **fita, n1, n2** are the incidence angle and refraction indices of the upper and lower media.

### EulerFi[fi,psi,alpha]

This function calculate Euler angle **Fi** for certain media transformation angles **fi**, **psi**, **alpha**.

### EulerTheta[fi,psi,alpha]

This function calculate Euler angle **Theta** for certain media transformation angles **fi**, **psi**, **alpha**.

### EulerPsi[fi,psi,alpha]

This function calculate Euler angle **Psi** for certain media transformation angles **fi**, **psi**, **alpha**.

### FilmLayerNew[Thickness, Epsilon, Mu, Ro] (!!! CHANGED FROM VERSION 3.03 !!!)

This function creates a film layer object. Thickness is the thickness of the layer. For simplicity a constant **nm** = 10-9 was introduced. That allows writing thicknesses in the form: **100 nm** for a layer of 100 nanometers thick. **Epsilon, Mu, Ro** are 3x3 matrices. If any of **Mu, Ro** are omitted then a standard Mu = 1 and Ro= 0 are used.

### FilmLayerTransform[FilmLayer, Rotation, Optional Reset = True]

This function is used to transform (rotate) the layer, using a rotation object **Rotation**. If **Reset = True**, then media properties (**Epsilon, Mu, Ro, RoT**), which were used when creating a layer by calling to FilmLayerNew are used. If **Reset = False**, then we use the media properties, obtained previously by applying transformations instead of original media properties. This is to allow consecutive transformations. **Usually there is NO NEED TO CALL this function.**

### FilmLayerTransformBase[FilmLayer, Rotation]

This is function is almost the same as **FilmLayerTransform**, but it transforms the originally supplied media properties instead of current ones. **Usually there is NO NEED TO CALL this function.**

### FilmLayerThickness[FilmLayer]

This function returns the value of the film layer thickness for a certain **FilmLayer** object.

### FilmLayerEpsilon[FilmLayer]

This function returns the value of the film layer **current Epsilon** for a certain **FilmLayer** object.

### FilmLayerEpsilonBase[FilmLayer]

This function returns the value of the film layer **base (not transformed) Epsilon** for a certain **FilmLayer** object.

### FilmLayerMu[FilmLayer]

This function returns the value of the film layer **current Mu** for a certain **FilmLayer** object.

### FilmLayerMuBase[FilmLayer]

This function returns the value of the film layer **base (not transformed) Mu** for a certain **FilmLayer** object.

### FilmLayerRo[FilmLayer]

This function returns the value of the film layer **current Ro** for a certain **FilmLayer** object.

### FilmLayerRoBase[FilmLayer]

This function returns the value of the film layer **base (not transformed) Ro** for a certain **FilmLayer** object.

### FilmLayerRoT[FilmLayer]

This function returns the value of the film layer **current RoT** for a certain **FilmLayer** object.

### FilmLayerRoTBase[FilmLayer]

This function returns the value of the film layer **base (not transformed) RoT** for a certain **FilmLayer** object.

### FilmNew[]

This function created empty film object, that is the film, which does not have any layers.

### FilmAddLayer[Film, FilmLayer]

This function takes **Film** and **FilmLayer** as parameters and returns **Film** with the **FilmLayer** added. Please, refer to the examples for further description.

### FilmItem[Film, LayerNo]

This function return **film layer** number **LayerNo** for a **Film**. It should be noted that **Film[[LayerNo]]** or **FilmItem[Film, LayerNo]** - gets layer the same film layer.

### FilmLength[Film]

This function return the number of layers for a **Film**.

### FilmTransformAll[Film, Rotation, Reset]

This function transforms **ALL** layers of the **Film** using the transformation object **Rotation** and options **Reset** = True or False. **Usually there is NO NEED TO CALL this function.**

### MediaNew[n1, n2, gamma, Film, Description, nOut, h2, epsilon2, mu2, ro2, epsilon, mu, ro]– (!!! CHANGED FROM VERSION 4.00 and prior to that from version 3.03 !!!)

This function creates new **Media** object, that is the object fully describing the light propagation task. **n1** and **n2** are the refraction indices of upper and lower semi-infinite media. **Gamma** is the rotation angle of the sample (it used to be a property of the incident light). **Film** is the layered thin film situated between the semi-infinite media. **Description** is **ANY** description of the whole media, just for keeping track of what is being calculated. **nOut** is the refraction index of the output media. This parameter is used ONLY when we are solving the task with the thin film on a thick substrate plate (**BerremanDirect::UseThickLastLayer 🡪 True**). In that case **nOut** is, of course, 1 (that’s air). **h2** is the thickness of a substrate media and it is only used when solving the task with the thin film on a thick substrate plate. It also allows specifying arbitrary optical properties of the substrate media **epsilon2, mu2, ro2** instead of just refraction index **n2** and of upper media **epsilon, mu, ro**. Nevertheless, **n1** and **n2**, still must be supplied and they should be more or less “correct”. Parameters starting from Description are optional and if omitted will be replaced by default values. If any of **Mu, Ro** are omitted then a standard Mu = 1 and Ro= 0 are used.

### MediaFlip[Media]

This function performs a rotation of the media around laboratory x axis by 180 degrees.

### MediaUpperRefractionIndex[Media]

This function returns refraction index of the upper media. That’s **n1**.

### MediaLowerRefractionIndex[Media]

This function returns refraction index of the lower media. That’s **n2**.

### MediaGamma[Media]

This function returns rotation angle gamma.

### MediaOutRefractionIndex [Media]

This function returns refraction index of the output media. That’s **nOut**.

### MediaFilm[Media]

This function returns the Film object of the **Media**.

### MediaFilmLength[Media]

This function is done for simplicity, as it return the **FilmLength** of the **Film** of the **Media**. It is equivalent to **FilmLength[MediaFilm[Media]]**.

### MediaDescription[Media]

This function returns **Description**, as supplied to the call to **MediaNew**.

### MediaUpperEpsilon[Media], MediaUpperMu[Media], MediaUpperRo[Media], MediaUpperRoT[Media]

These functions return Epsilon, Mu, Ro, RoT of the upper media correspondingly. Usually used internally.

### MediaSubstrateThickness[Media]

This function returns thickness of the substrate plate.

### MediaLowerEpsilon[Media], MediaLowerMu[Media], MediaLowerRo[Media], MediaLowerRoT[Media]

These functions return Epsilon, Mu, Ro, RoT of the substrate media correspondingly. Usually used internally.

### IncidentLightNew[lambda, fita, beta, n1, Ampl] – (!!! CHANGED FROM VERSION 4.00 !!!)

This function creates **Incident Light** object for a wavelength **lambda**, incidence angle **fita**, rotation of polarization plane **beta**, refraction index of the upper media **n1**, and Amplitude of the Incident light. Usually used internally, or when calling Solution directly.

**NOTE**: Starting from version 4.01 it no longer includes rotation of the sample **gamm**.

### IncidentLightNew[lambda, fita, beta, eh] – (!!! CHANGED FROM VERSION 4.00 !!!)

This function creates **Incident Light** object for a wavelength **lambda**, incidence angle **fita**, rotation of polarization plane **beta**, rotation of the sample **gamm** (not actually used so far), and six component E H field **eh**. **Usually there is NO NEED TO CALL this function.**

**NOTE**: Starting from version 4.01 it no longer includes rotation of the sample **gamm**.

### IncidentLightNew[lambda, fita, beta, n1] – (!!! ADDED IN VERSION 4.02 !!!)

This function creates **Incident Light** object for a wavelength **lambda**, incidence angle **fita**, rotation of polarization plane **beta**, refraction index of the upper media **n1**, and Unit Amplitude (= 1) of the Incident light. Usually used internally, or when calling Solution directly.

### IncidentLightLambda[IncidentLight]

This function returns wavelength **lambda** of the **Incident Light**.

### IncidentLightFita[IncidentLight]

This function returns incidence angle **fita** of the **Incident Light**.

### IncidentLightBeta[IncidentLight]

This function returns rotation of polarization plane **beta** of the **Incident Light**.

### IncidentLightIsDown[IncidentLight]

This function returns **True** if **IncidentLight** is going down and **False** if it is going up. **Usually there is NO NEED TO CALL this function.**

### IncidentLightEH[IncidentLight]

This function returns **6 component E H field** of the **Incident Light**. **Usually there is NO NEED TO CALL this function.**

### IncidentLightFlip[IncidentLight]

This function rotates **IncidentLight** around laboratory x axis by 180 degrees. **Usually there is NO NEED TO CALL this function.**

### EpsilonFromN[n1]

This function creates isotropic 3x3 matrix Epsilon for a refraction index **n1**.

### EpsilonFromN[n1,n2,n3]

This function creates diagonal 3x3 matrix Epsilon for refraction indices **n1, n2, n3**.

## Output – Functions and objects

### SolutionNew[Media, IncidentLight, Options]

This function creates a **Solution** object for given **Media**, **Incident Light** and **Options**. By calling the **SolutionNew** we tell Mathematica to solve the light propagation problem and return the results in the Solution object. The solution object has the following standard form: **sol = {EHIFull, EHRFull, EHTFull, PPP, delta, Media, IncidentLight, opts }**, where EHIFull – is **6 component E H field** of the **Incident Light**, EHRFull – is **6 component E H field** of the **Reflected Light**, EHTFull – is **6 component E H field** of the **Transmitted Light**, PPP –is combined PPP 4x4 matrix, **delta** is 4x4 delta matrix of the media, **Media, IncidentLight, opts** are the media, incident light and options used in calculations. Each of the EH fields has the following structure: **eh ={Ex, Ey, Ez, Hx, Hy, Hz}**.

### EGGetOrder[egvl : {\_, \_, \_, \_}]

This function was first introduced in version 3.03.22 in order to take care about ordering of eigenvalues of Matrix M. This function is used internally in module **BerremanCommon** and also in module **FieldAlgebra** in order to get properly ordered eigenvalues of matrix M.

### GetSolIncidentLight[sol]

This function returns **6 component E H field of Incident Light** for a given **Solution sol**. It is equivalent to **sol[[1]]**.

### GetSolReflectedLight[sol]

This function returns **6 component E H field of Reflected Light** for a given **Solution sol**. It is equivalent to **sol[[2]]**.

### GetSolTransmittedLight[sol]

This function returns **6 component E H field of Transmitted Light** for a given **Solution sol**. It is equivalent to **sol[[3]]**.

### GetSolIncidentLightE[sol

### GetSolReflectedLightE[sol]

### GetSolTransmittedLightE[sol]

### GetSolIncidentLightH[sol

### GetSolReflectedLightH[sol]

### GetSolTransmittedLightH[sol]

### GetSolIncidentLightD[sol

### GetSolReflectedLightD[sol]

### GetSolTransmittedLightD[sol]

### GetSolIncidentLightB[sol

### GetSolReflectedLightB[sol]

### GetSolTransmittedLightB[sol]

### GetEfromEH[eh]

### GetHfromEH[eh]

### GetSolPPP[sol]

This function returns **4x4 matrix PPP** for a given **Solution sol**. It is equivalent to **sol[[4]]**.

### GetSolDelta[sol]

This function returns **4x4 matrix delta** for a given **Solution sol**. It is equivalent to **sol[[5]]**.

### SolutionCombine[ehi,ehr,eht,ppp,delta]

This function is used internally to combine **6 component E H field of Incident, Reflecter, and Transmitted Light, PPP and dalta** into a solution object. **Usually there is NO NEED TO CALL this function.**

### EHFlip[eh]

This function rotates **6 component E H field** around laboratory x axis by 180 degrees. **Usually there is NO NEED TO CALL this function.**

### GetSolMedia[sol], GetSolIncidentLight[sol], GetSolOptions[sol]

These function returns **Media**, **Incident Light** and **options** for a given solution. Used primarily in FieldAlgebra to obtain access to these objects, when necessary.

### PPP[eps, mu, ro, rotr, lambda, fita, n1, h]

This function allows to calculate PPP for a given media optical properties.

### GetSolIncidentLightInfo[sol]

This function returns Incident Light Info structure (please, DO NOT confuse with Incident Light – EH field ).

### GetSolOptions[sol]

This function returns options for a given solution.

### GetSolCoeff[sol]

This function returns 4x4 matrix of coefficients of an equation for transmittance and reflectance coefficients (eigenvector multipliers): PPP . (EHI + EHR) - EHT=0

### GetSolFreeTerm[sol]

This function returns 4 vector of free terms of an equation for transmittance and reflectance coefficients (eigenvector multipliers): PPP . (EHI + EHR) - EHT=0

### GetSolEGSys1[sol]

This function returns eigen system for the upper media in the form: {{EigenValue1, EigenValue 2, EigenValue 3, EigenValue 4},{ EigenVector1, EigenVector2, EigenVector3, EigenVector4}}

### GetSolEGSys2[sol]

This function returns eigen system for the lower media in the form: {{EigenValue1, EigenValue 2, EigenValue 3, EigenValue 4},{ EigenVector1, EigenVector2, EigenVector3, EigenVector4}, EHTFullEG1, EHTFullEG2, EHTFullEG3, EHTFullEG4} where EHTFullEG***i*** is full 6 component electromagnetic field corresponding to ***i***th eigenvalue/engenvector.

### GetSolEHTEG[sol, idx]

This function returns full 6 component electromagnetic field corresponding to eigenvalue/engenvector # idx of a lower media.

### PoyntingS[ehFld, fldIdx]

This is a service function, which calculates component ***fldIdx*** of a Poynting vector for a given electromagnetic field ***ehFld***.

# Berreman Direct

Berreman Direct extends the basic features of Method of Berreman. The most important additions are the notation of **Calculation** (**Calc** object), which allows to perform multiple calculations for the same media simultaneously, account for influence of thick substrate plate (option **UseThickLastLayer** and various related options), notation of **Calculation Collection** (**CalcCollection** object), which further extends the capabilities of a single calculation and various services functions.

## Options & Constants:

**For each option a default value is listed. That is if there is no option specified, then a default value is used.**

### RotateAll True

if set to **True** then **the same rotation** is used for **ALL layers** in the film. If set to **False**, then each layer in the film can be rotated differently.

### ConsecutiveRotation  True

if set to **True**, then rotation by angle gamma (rotation of the sample as a whole) is treated as a consecutive rotation. If set to **False**, then gamma is added to the original transformation angles (fi, theta, psi) so that to come up with a single rotation object. The results obtained by using both of these methods MUST BE EQUAL TO EACH OTHER. If they are NOT, then there is a mistake somewhere.

### PrintTimeEstimate  True

if set to True, then the program will print time estimate for the calculations.

### UseThickLastLayer  False

if set to **False**, then the program treats all layers as thin and solves the direct light propagation problem for upper and lower semi-infinite media and layered thin film in the middle. If set to **True**, then the program considers the influence of a thick substrate plate on the solution. The behavior of the program in that case depends on the option setting **AveragingType**. Please, refer to the description of that setting for further comments.

### NoOfAveragingPoints  10

this setting is used only when **UseThickLastLayer  True**. In that case it determines how many averaging points will be used for taking into account thick substrate plane. It should be noted that for different averaging methods (option **AveragingType**) that value can be ignored or treated differently. Please, refer to the description of **AveragingType** setting for further comments.

### AveragingPeriods  1

this setting is used for certain averaging methods only when **UseThickLastLayer  True**. Currently it is not supported by the implemented methods and should be safely ignored.

### AveragingType  BDAVGTYPESERIES

this setting is used only when **UseThickLastLayer  True**. Default value of that setting is **BDAVGTYPESERIES**. That setting uses the following approach. First, the problem is solved for incident light and upper and lower semi-infinite media with refraction indices of **n1** and **n2** and intermediate layered think film. Then the transmitted light is treated as “incident” for the boundary **n2** (lower semi-infinite media) / **nOut** (output media). Then the light reflected from the boundary **n2** / **nOut** is treated as incident light for the “flipped” media and a solution is obtained again for transmitted and reflected light. It should be noted that transmitted light in that case is actually an addition to the reflected light of the initial problem and reflected light is an incident light for the next step (reflection / transmission on the boundary **n2** / **nOut**). Such series is generated for **NoOfAveragingPoints \* 2** solutions. These solutions then later used to obtain the values of the output functions (for example **Rpar**).

### PrintCalculationProgress False

if set to **True**, then the program prints some calculation progress.

### PrintCalculationDetailsFalse

if set to **True**, then the program prints some additional calculation details. Currently it is not supported yet.

### TransmittedAnalyzerAngle  0

this setting specifies angle of analyzer for transmitted light.

### TransmittedAnalyzerParallelAmplitude  1

this setting specifies “parallel” transmittance of analyzer for transmitted light.

### TransmittedAnalyzerCrossedAmplitude  0

this setting specifies “crossed” transmittance of analyzer for transmitted light.

### ReflectedAnalyzerAngle  0

this setting specifies angle of analyzer for reflected light.

### ReflectedAnalyzerParallelAmplitude  1

this setting specifies “parallel” transmittance of analyzer for reflected light.

### ReflectedAnalyzerCrossedAmplitude  0

this setting specifies “crossed” transmittance of analyzer for reflected light.

### AnalyzerAngleAbsoluteValue  False

if set to **True**, then angles of BOTH analyzers are treated as having absolute value, if set to **False** (default), then the angles are relative to the angle of rotation of polarization plane of incident light.

### NOT SUPPORTED YET averaging Types:

Wavelength averaging: **AveragingType  BDAVGTYPESUMOFTWOPOINTS** – sum of two points / 2, **AveragingType  BDAVGTYPEUSEAVGPERIODS** – averaging by **NoOfAveragingPoints** & using **AveragingPeriods**, **AveragingType  BDAVGTYPEUSEAVGPERIODSEXACTLAMBDA** - same as **BDAVGTYPEUSEAVGPERIODS**, but lambda is exactly as supplied.

### UsePlotLabel3D  True

This option is used to specify whether to use plot label on top of the 3D plots (UsePlotLabel3D  True) instead of the name of a z axis. It is useful if the name of the function is long..

### UseChop  True

This option is used to specify that all numerical data used for plotting should be rounded with the precision of **ChopTolerance**. It replaces approximate real numbers in function that are close to zero by the exact integer 0 and drops all digits, which are less than **ChopTolerance** for not close to zero number. If **UseChop** is set to false then no rounding is done.

### ChopTolerance  10^-7

This is a default value for rounding up. Please, refer to **UseChop** for description how it is used.

### ChopPrecision  6

Currently NOT used.

### PrintFunctionDescription  True

This option is used to specify that function description should be printed before rendering the graphs of functions.

## Variables:

Each variable in the calculations has identical form **Var = {StartValue, EndValue, Step, VariableName, Multiplier}**, where StartValue is the starting value of the variable, EndValue is the ending value if the variable, Step is the step of changes, VariableName is the name of the variable to be used in output and Multiplier is used to multiply each value of the variable on it. For example, **lambdaVar ={550, 600, 5, “Lambda”, nm}** – that means that we introduce the variable for wavelength (lambda) with the starting value of 550, ending value of 600 and step of 5, variable name “Lambda” and multiplier **nm**. As discussed above, a constant **nm** = 10-9 was introduced to simplify the code. This is similar to built-in function **Degree**, which allows conversion from radians to degrees in the same way as **nm** allows conversion from meters to nanometers. An example of angle variable is: **fitaVar = {0, 90, 5, "Fita", Degree}**.

### VarListNew[IncidentLightInfo, CommonAnglesInfo]

this function creates new **VarList** object. **IncidentLightInfo** is structure, which holds the information about incident light variables. It has the following form: **IncidentLightInfo = {lambdaVar, fitaVar, betaVar, gammaVar}**,where each xxxVar has structure as described above and specifies the range of changes of Incident light parameters (wavelength – **lambdaVar**, incidence angle – **fitaVar**, rotation of polarization plane angle – **betaVar**, rotation of the sample angle – **gammaVar**). **CommonAnglesInfo** is the structure, which holds the information about common rotations (that’s when option **RotateAll  True**). **CommonAnglesInfo = {fiVar, thetaVar, psiVar}**. Each of the **fiVar, thetaVar, psiVar** is a variable object (see above). Please, refer to the description of Rotation object for meaning of each angle.

### VarListAddLayer[VarList, LayerAngleInfo]

this function adds layer angles info starting from the top layer (adjacent to Upper Refraction Index (n1). **LayerAngleInfo** = **{fiVar, thetaVar, psiVar}**. Each of the **fiVar, thetaVar, psiVar** is a variable object (see above). Each subsequent call to **VarListAddLayer** adds layer angle info for the subsequent layer until the last one. Number of LayerAngleInfo, that is calls to **VarListAddLayer** MUST be equal to the number of layers in the film. It should be noted that if option **RotateAll  True**, then ONLY **CommonAnglesInfo** structure is used for rotating the layers. In that case each of the variables in LayerAngleInfo MUST have ONLY one point, that is, for example, **fiVar = {0, 0, 5, "Fi", Degree}**. Please, refer to examples for further discussion.

### CalcNew[Media, VarList, FuncList, Description, opts] (!!! CHANGED FROM VERSION 3.03 !!!)

this function creates new Calc object for a given **Media**, list of variables – **VarList**, list of functions to be calculated (**FuncList**), description of the calculation – **Description**, and options – **opts**. Each function in FulcList takes only one parameter, that the Solution (please, refer to the description of the function **SolutionNew**) and returns one number. Please, refer to examples for further discussion. Note that list of names of the functions (**FuncNameList**) was removed in version 3.04 to simplify the coding. Functions, which take only one parameter (FullSolution) should be submitted by their name in the function list. Functions, which take more than one parameters (for example, **PoyntingVector2[evi,xyz,FullSol]**) should be submitted with the last parameter (FullSol) removed and the rest explicitly specified (for example, **FuncNameList ={ PoyntingVector2[2, 3] }** – to specify that **z (3rd)** component of a **second (2nd)** eigen vector should be plotted).

### CalcPerform[Calc]

this function takes **Calc** object created by call to **CalcNew**, performs calculations for the whole grid of input parameters and calculates the output functions (specified by **FuncList** in call to **CalcNew**).

### CalcGetInput[Calc]

this function returns a table (grid) of input variables. Generally there is no need to call this function as it is being used by output functions to store the data in the file.

### CalcGetOutput[Calc]

this function returns a table (grid) of output function values (for each function name in FuncList. Generally there is no need to call this function as it is being used by output functions to store the data in the file.

### CalcGetFuncNames[Calc]

this function returns a table (grid) of output function names, that’s FuncNameList. Generally there is no need to call this function as it is being used by output functions to store the data in the file.

### CalcGetVarNames[Calc]

this function returns a table (grid) of input variable names. Generally there is no need to call this function as it is being used by output functions to store the data in the file.

### CalcGetNumberOfResults[Calc]

this function returns the number of output results. Generally there is no need to call this function as it is being used by output functions to store the data in the file.

### CalcMedia[Calc]

this function returns the Media, which was used in calculations. Generally there is no need to call this function as it is being used by output functions to store the data in the file.

### CalcDescription[Calc]

this function returns the Description, which was used in calculations. Generally there is no need to call this function as it is being used by output functions to store the data in the file.

### CalcSave[Calc, FileName, OverWrite=True]

this function saves the results of performed calculation **Calc** into file **FileName**.

### CalcPrintRunTime[Calc]

this function prints total processor run time for the performed calculation.

### CalcPrintTimeEstimate[Calc]

this function prints time estimate for the calculation. This is important if there are many points involved and it is necessary to know approximately when the program will finish calculations. If option **PrintTimeEstimate  True** then the program calls this function automatically.

### CalcOptions[Calc]

this function returns options for calculation.

### CalcPlot[Calc, pltOptsRaw]

this function plots 2D graphs of ALL functions specified in calculation. The first variable, which has non zero range is used as x variable for each function. For the remaining variables starting values are used. **pltOptsRaw** are plot options. Please refer to description of function **Plot** for details.

### CalcPlotFunc[Calc, Func, FuncName, pltOptsRaw]

this function plots a function with a certain name. Usually used internally.

### CalcPlot3D[Calc, SwapXY, pltOptsRaw]

this function plots 3D graphs of ALL functions specified in calculation. The first TWO variables, which has non zero range are used as x and y variables for each function. For the remaining variables starting values are used. **SwapXY** – if set to True, the x and y axes are swapped. **pltOptsRaw** are plot options. Please refer to description of function **Plot3D** for details.

### CalcPlot3DFunc[Calc, Func, FuncName, SwapXY, pltOptsRaw]

this function plots 3D function with a certain name. Usually used internally.

### CalcFuncPlot[CalcFunc, xVarLst:{xMin, xMax, xName},pltOptsRaw]

this function plots 2D graphs of ALL functions specified in calculation. The main difference from **CalcPlot** is that this function uses a callback function **CalcFunc[x]**. For each value of x callback function must supply a ready calculation. The primary purpose of this function is to plot complicated graphs, for example dependence on refraction index or one of refraction indices, etc.

### CalcFuncPlotFunc[CalcFunc,xVarLst:{xMin, xMax, xName}, Func, FuncName, pltOptsRaw] ]

this function plots a function with a certain name. Usually used internally.

### CalcFuncPlot3D[CalcFunc, xVarLst:{xMin, xMax, xName},yVarLst:{yMin, yMax, yName}, pltOptsRaw] ]

this function plots 3D graphs of ALL functions specified in calculation. The main difference from **CalcPlot** is that this function uses a callback function **CalcFunc[x, y]**. For each value of x and y callback function must supply a ready calculation. The primary purpose of this function is to plot complicated graphs, for example dependence on refraction index or one of refraction indices, etc.

### CalcFuncPlot3DFunc[CalcFunc, xVarLst:{xMin, xMax, xName}, yVarLst:{yMin, yMax, yName}, Func, FuncName, pltOptsRaw]

this function plots 3D function with a certain name. Usually used internally.

### CalcCollectionNew[BaseDir, BaseFileName, Description]

this function creates new Collection of Calculations object. **CalcCollection** object holds several calculations and allows performing things, which a single calculation cannot do. For example, if one wants to perform a calculation for different epsilon, then a **CalcCollection** is needed. **BaseDir** is the base output directory. It MUST end with “**\**”.**BaseFileName** is the base file name. For each calculation in the collection an “\_” and calculation number will be added to produce the output file name.

### CalcCollectionAddCalc[coll, Calc, FileName]

this function adds the calculation **Calc** to collection **coll**. If **FileName** is blank then automatic file name generation will be used, as described in **CalcCollectionNew**. Otherwise, **FileName** will be used.

### CalcCollectionPerform[coll]

this function takes **coll** object and performs calculations for the ALL calculations in the collection.

### CalcCollectionLength[coll]

this function returns the number of calculations in the collection **coll**.

### CalcCollectionItem[coll, idx]

this function returns calculation number **idx** from collection **coll**.

### CalcCollectionSave[coll, DoPrintOut]

this function saves collection if **DoPrintOut = False** and prints collection if **DoPrintOut = True**.

### CalcCollCombine[coll, funcNo, Optional HorizontalVarInputNo = 0, Optional VerticalVarInputNo = 0, Optional OutputHeader = False]

this function is used to combine the results of Calculation Collection into a single “mesh”, that is **N x M matrix** of output values. The typical use is as follows. We set up a calc collection, where in each calculation only one variable is changes (identically among all calculations, of course). Then among calculations, another variable is changed. Then by calling to **CalcCombine** we can get a 3D surface of output values. Currently ONLY parameters **coll** and **funcNo** are supported. **coll** is, of course, the Calculation Collection and **funcNo** is the function number in the Function List.

### CalcCollectionPlot[coll, pltOptsRaw]

this function plots 2D graphs of ALL functions specified in each calculation of the collection.

### CalcCollectionPlot3D[coll, SwapXY, pltOptsRaw]

this function plots 3D graphs of ALL functions specified in each calculation of the collection.

### SolutionGetTransmittedAnalyzerInfo[sol]

this function returns **AnalyzerInfo** object of transmitted light for a given solution **sol**.

### SolutionGetReflectedAnalyzerInfo[sol]

this function returns **AnalyzerInfo** object of reflected light for a given solution **sol**.

### AnalyzerInfoGetAngle[anInf]

this function returns **ABSOLUTE** angle of rotation of Analyzer for a given **AnalyzerInfo** object.

### AnalyzerInfoGetParallelAmplitude[anInf]

this function returns “parallel” transmittance of Analyzer for a given **AnalyzerInfo** object.

### AnalyzerInfoGetCrossedAmplitude[anInf]

this function returns “crossed” transmittance of Analyzer for a given **AnalyzerInfo** object.

The following functions are used internally and are not described in this manual.

**DateTimeToString[cdt]**

**SolutionAverageNew[Media, IncidentLight, opts]**

**AveragingFunc[f, solavg]**

**TransformMedia[Media, varlist, opts]**

# Field Algebra

Field Algebra implements basic output functions to be used in calculations. They can serve as examples of how to create additional output functions, if necessary. Each of the output functions MUST take only one parameter Solution object as input and return one number as output. **!!! PLEASE, USE THE CONTENT OF THIS MODULE AS AN EXAMPLE OF HOW TO CREATE THE OUPUT FUNCTIONS !!!**

## Options & Constants

### FunctionName  "Unknown"

each function must specify option for function name. Default value is ***Unknown***.

### FunctionDescription  "Unknown"

each function must specify option for function description. Default value is ***Unknown***.

### NonAverageable  False

function MAY specify that it is should not be used in averaging (for the sum of series averaging type). If that value is set to true, then also **NonAverageableIndex** (below) needs to be specified.

### NonAverageableIndex  1

if **NonAverageable** is set to **True**, then **NonAverageableIndex** MUST BE specified. For reflected light based functions correct value is **1**, for transmitted light based functions correct value is **2**.

### UseAnalyzer  False

function MAY specify that it uses analyzer in its calculations by setting this option to **True**.

### AbsoluteAzimuth  True

if set to **False** then from the value of azimuth the program subtracts the angle of rotation of polarization plane of incident light **beta**. If set to **True** (default) then **beta** is not subtracted.

### AzimuthRangeType  0

the value of **0** means that azimuth range is set from **0 to 90 degrees** shifted by **Azimuth Shift**, the value of **1** means that azimuth range is set from **0 to 180 degrees** shifted by **Azimuth Shift**.

### AzimuthShiftType  1

the value of **0** means that **Azimuth Shift** is **0 degrees**, the value of **1** means that **Azimuth Shift** is **45 degrees**, the value of **2** means that **Azimuth Shift** is **90 degrees**.

## Functions and Function Options.

Below are functions along with their options. They are self explanatory. In addition before rendering the graph of function the program (by default) will print the description of each function. Functions, which take only one parameter (FullSolution) should be submitted by their name in the function list. Functions, which take more than one parameters (for example, **PoyntingVector2[evi,xyz,FullSol]**) should be submitted with the last parameter (FullSol) removed and the rest explicitly specified (for example, **FuncNameList ={ PoyntingVector2[2, 3] }** – to specify that **z (3rd)** component of a **second (2nd)** eigen vector should be plotted).

### GetFunctionName[Func]

This function is used internally to extract the correct function name from its corresponding symbol. For simple function is uses FunctionName /. Options[Func] replacement rule to get the name. For multi component functions (for example, PoyntingVector2[EigenValueIndex, xyz, Solution] it uses specially written for each such function ***FunctionNameString***, which construct the proper name.

### IFull[FullSol]

**Options[IFull]={FunctionName "IFull", FunctionDescription "Full intensity of Incident light."};**

This function returns full intensity of incident light.

### RFull[FullSol]

**Options[RFull]={FunctionName "RFull", FunctionDescription "Full intensity of Reflected light."};**

This function returns full intensity of reflected light.

### TFull[FullSol]

**Options[TFull]={FunctionName "TFull", FunctionDescription "Full intensity of Transmitted light."};**

This function returns full intensity of transmitted light.

### Rx[FullSol]

**Options[Rx]={FunctionName "Rx", FunctionDescription "Intensity of Reflected light going into X component."};**

This function returns xz component of intensity of reflected light.

### Ry[FullSol]

**Options[Ry]={FunctionName  "Ry", FunctionDescription "Intensity of Reflected light going into Y component."};**

This function returns y component of intensity of reflected light.

### Tx[FullSol]

**Options[Tx]={FunctionName "Tx", FunctionDescription "Intensity of Transmitted light going into X component."};**

This function returns xz component of intensity of transmitted light.

### Ty[FullSol]

**Options[Ty]={FunctionName  "Ty", FunctionDescription "Intensity of Transmitted light going into Y component."};**

This function returns y component of intensity of transmitted light.

### Xit[FullSol]

**Options[Xit]={FunctionName  "Xit", FunctionDescription "Azimyth of Transmitted light.", NonAverageable  True,NonAverageableIndex  2};**

### XitDegree[FullSol]

**Options[XitDegree]={FunctionName  "Xit (Degree)", FunctionDescription "Azimyth of Transmitted light in Degrees.", NonAverageable  True,NonAverageableIndex  2};**

### Elt[FullSol]

**Options[Elt]={FunctionName  "Elt", FunctionDescription "Ellipticity of Transmitted light.", NonAverageable  True,NonAverageableIndex  2};**

### Xir[FullSol]

**Options[Xir]={FunctionName  "Xir", FunctionDescription "Azimyth of Reflected light.", NonAverageable  True,NonAverageableIndex  1};**

### XirDegree[FullSol]

**Options[XirDegree]={FunctionName  "Xir (Degree)", FunctionDescription "Azimyth of Reflected light in Degrees.", NonAverageable  True,NonAverageableIndex  1};**

### Elr[FullSol]

**Options[Elr]={FunctionName  "Elr", FunctionDescription "Ellipticity of Reflected light.", NonAverageable  True,NonAverageableIndex  1};**

### RAnalyzer[FullSol]

**Options[RAnalyzer]={FunctionName  "RAnalyzer", FunctionDescription "Intensity of Reflected light passed through analyzer.", UseAnalyzer  True};**

### TAnalyzer[FullSol]

**Options[TAnalyzer]={FunctionName  "TAnalyzer", FunctionDescription "Intensity of Transmitted light passed through analyzer.", UseAnalyzer  True};**

### DetM1[FullSol]

**Options[DetM1]={FunctionName  "Det[M1]", FunctionDescription "Determinant of Matrix M of the upper media.", NonAverageable  True,NonAverageableIndex  1};**

### DetM2[FullSol]

**Options[DetM2]={FunctionName  "Det[M2]", FunctionDescription "Determinant of Matrix M of the lower media.", NonAverageable  True,NonAverageableIndex  1};**

### LDetM2[FullSol]

**Options[LDetM2]={FunctionName  "Log[Det[M2]]", FunctionDescription "Log of Determinant of Matrix M of the lower media.", NonAverageable  True,NonAverageableIndex  1};**

### DetPPP[FullSol]

**Options[DetPPP]={FunctionName  "Det[PPP]", FunctionDescription "Det[PPP].", NonAverageable  True,NonAverageableIndex  1};**

### DetCoeff::usage="DetCoeff[FullSol]";

**Options[DetCoeff]={FunctionName  "Det[Coeff]", FunctionDescription "Det[Coeff].", NonAverageable  True,NonAverageableIndex  1};**

### ReDetCoeff[FullSol]

**Options[ReDetCoeff]={FunctionName  "Re[Det[Coeff]]", FunctionDescription "Re[Det[Coeff]].", NonAverageable  True,NonAverageableIndex  1};**

### ImDetCoeff[FullSol]

**Options[ImDetCoeff]={FunctionName  "Im[Det[Coeff]]", FunctionDescription "Im[Det[Coeff]].", NonAverageable  True,NonAverageableIndex  1};**

### RxAccuracy[FullSol]

**Options[RxAccuracy]={FunctionName  "RxAccuracy", FunctionDescription "Accuracy of Rx.", NonAverageable  True,NonAverageableIndex  1};**

### DetM2Accuracy[FullSol]

**Options[DetM2Accuracy]={FunctionName  "DetM2Accuracy", FunctionDescription "Accuracy of Det[M2].", NonAverageable  True,NonAverageableIndex  1};**

### DetCoeffAccuracy[FullSol]

**Options[DetCoeffAccuracy]={FunctionName  "DetCoeffAccuracy", FunctionDescription "Accuracy of Det[Coeff].", NonAverageable  True,NonAverageableIndex  1};**

### M2EValRe[evi, FullSol]

**Options[M2EValRe]={FunctionName  "Re[EVal[M2]]", FunctionDescription "Re of EigenValues of Matrix M of the lower media.", NonAverageable  True,NonAverageableIndex  1};**

### M2EValIm[evi, FullSol]

**Options[M2EValIm]={FunctionName  "Im[EVal[M2]]", FunctionDescription "Im of EigenValues of Matrix M of the lower media.", NonAverageable  True,NonAverageableIndex  1};**

### PoyntingI[xyz,FullSol]

**Options[PoyntingI]={FunctionName  "PoyntingI", FunctionDescription "Poynting vector component of Incident light."};**

### PoyntingIFull[FullSol]

**Options[PoyntingIFull]={FunctionName  "PoyntingIFull", FunctionDescription "Value of full Poynting Vector of incident light."};**

### PoyntingR[xyz,FullSol]

**Options[PoyntingR]={FunctionName  "PoyntingR", FunctionDescription "Poynting vector component of Reflected light."};**

### PoyntingRFull[FullSol]

**Options[PoyntingRFull]={FunctionName  "PoyntingRFull", FunctionDescription "Value of full Poynting Vector of reflected light."};**

### PoyntingT[xyz,FullSol]

**Options[PoyntingT]={FunctionName  "PoyntingT", FunctionDescription "Poynting vector component of Transmitted light."};**

### PoyntingTFull[FullSol]

**Options[PoyntingTFull]={FunctionName  "PoyntingTFull", FunctionDescription "Value of full Poynting Vector of transmitted light."};**

### PoyntingRTFull[FullSol]

**Options[PoyntingRTFull]={FunctionName  "PoyntingRTFull", FunctionDescription "PoyntingRTFull."};**

### PoyntingIRTzDiff[FullSol]

**Options[PoyntingIRTzDiff]={FunctionName  "PoyntingIRTzDiff", FunctionDescription "Sum of z components of Poynting Vector of incident, reflected and transmitted light."};**

### PoyntingVector2[evi,xyz,FullSol]

**Options[PoyntingVector2]={FunctionName  "PoyntingVector2", FunctionDescription "Poynting vector component of eigen vector of the lower media."};**

### PoyntingXYPhase2[evi,FullSol]

**Options[PoyntingXYPhase2]={FunctionName  "PoyntingXYPhase2", FunctionDescription "Poynting vector component phase (azimuth) in xy plane of eigen vector of the lower media."};**

### PoyntingXY2[evi,FullSol]

**Options[PoyntingXY2]={FunctionName  "PoyntingXY2", FunctionDescription "Poynting vector xy component of eigen vector of the lower media."};**

### ReEGE2[evi,xyz,FullSol]

**Options[ReEGE2]={FunctionName  "ReEGE2", FunctionDescription "Real part of Electric field of eigen vector of the lower media."};**

### ImEGE2[evi,xyz,FullSol]

**Options[ImEGE2]={FunctionName  "ImEGE2", FunctionDescription "Imaginary part of Electric field of eigen vector of the lower media."};**

### ReEGH2[evi,xyz,FullSol]

**Options[ReEGH2]={FunctionName  "ReEGH2", FunctionDescription "Real part of Magnetic field of eigen vector of the lower media."};**

### ImEGH2[evi,xyz,FullSol]

**Options[ImEGH2]={FunctionName  "ImEGH2", FunctionDescription "Imaginary part of Magnetic field of eigen vector of the lower media."};**

### LRFull[FullSol]

**Options[LRFull]={FunctionName "Log[RFull]", FunctionDescription "Logarithm of full intensity of Reflected light."};**

### LTFull[FullSol]

**Options[LTFull]={FunctionName "Log[TFull]", FunctionDescription "Logarithm of full intensity of Transmitted light."};**

### LRx[FullSol]

**Options[LRx]={FunctionName "Log[Rx]", FunctionDescription "Logarithm of Intensity of Reflected light going into X component."};**

### LRy[FullSol]

**Options[LRy]={FunctionName  "Log[Ry]", FunctionDescription "Logarithm of Intensity of Reflected light going into Y component."};**

### LTx[FullSol]

**Options[LTx]={FunctionName "Log[Tx]", FunctionDescription "Logarithm of Intensity of Transmitted light going into X component."};**

### LTy[FullSol]

**Options[LTy]={FunctionName  "Log[Ty]", FunctionDescription "Logarithm of Intensity of Transmitted light going into Y component."};**

### EpsComponent[UpDown,ReIm,xyzRow,xyzCol,FullSol]

**Options[EpsComponent]={FunctionName  "EpsComponent", FunctionDescription "Component of Matrix Epsilon.", NonAverageable  True,NonAverageableIndex  1};**

This function allows to output various components of Matrix Epsilon. The first parameter **UpDown** specifies what matrix will be used. **UpDown = -1** will output upper media, **UpDown = -2** will output lower media, and **UpDown > 0** will output the matrix from layer number **UpDown**. The second parameter **ReIm** is used to specify whether real (**ReIm = 1**) or Imaginary (**ReIm = 2**) parts of the matrix Epsilon to calculate (output). Parameters 3 and 4 (**xyzRow, xyzCol**) are used to specify which element of the matrix to output. For example **EpsComponent[-2, 1, 3, 3]** means to ouput **Real** part (**ReIm = 1**) of **zz** component (**xyzRow = 3, xyzCol = 3**) of the matrix Epsilon of the **lower media** (**UpDown = -2**) and **EpsComponent[4, 1, 2, 3]** means to ouput **Real** part (**ReIm = 1**) of **yz** component (**xyzRow = 2, xyzCol = 3**) of the matrix Epsilon of the **4th layer** (**UpDown = 4**).

### MuComponent[UpDown,ReIm,xyzRow,xyzCol,FullSol]

**Options[MuComponent]={FunctionName  "MuComponent", FunctionDescription "Component of Matrix Mu.", NonAverageable  True,NonAverageableIndex  1};**

All the parameters have the same meaning as in **EpsComponent[UpDown, ReIm, xyzRow, xyzCol, FullSol]**.

### RoComponent[UpDown,ReIm,xyzRow,xyzCol,FullSol]

**Options[RoComponent]={FunctionName  "RoComponent", FunctionDescription "Component of Matrix Ro.", NonAverageable  True,NonAverageableIndex  1};**

All the parameters have the same meaning as in **EpsComponent[UpDown, ReIm, xyzRow, xyzCol, FullSol]**.

# FieldIO

FieldIO implements some basic input / output functions for calculation. Currently there are only a few functions in that module.

### OutputCopyright[]

this function output copyright information about the program.

### OutputFunc[Calc, fName, OverWrite]

this function saves the calculation Calc into file **fName**.

# Berreman Inverse

Berreman Inverse module implements solving of inverse problem, that is determining optical parameters of the media based on the experimental data. Currently there are only a few functions in that module.

## Options & Constants

### LambdaMultiplier  nm

sets wavelength lambda multiplier in input data (default is nm)

### AngleMultiplier  Degree

sets angle multiplier in input data (default is Degree)

### DataStartRow  2

sets starting row in input data (default is 2, that is the first row is ignored)

### LambdaColumn  3

sets lambda (wavelength) column in input data

### FitaColumn  4

sets fita (incidence angle) column in input data

### BetaColumn  5

sets beta (rotation of polarization plane of incident light) column in input data

### GammaColumn  6

sets gamma (rotation of the sample angle) column in input data

### OutputStartColumn  9

sets output column in input data

### OutputColumns {{ strUseAllOutputColumns}}

sets output columns collection in input data. Please, refer to an example.

### MinizationMultiplier 10^9

used internally

### UseStepMonitor  True

if set to **True**, then on each minimization step output will be produced.

### UseEvaluationMonitor  False

if set to **True**, then on each minimization evaluation output will be produced.

### UseNMinimize  False

if set to True, then on **NMinimize** will be used, otherwise **FindMinimum** will be used.

### EpsilonType  BIEPSTYPESTDTWOSTEPTRAPSPARENT

this setting determines the type of epsilon being sought. Please, refer to epsilon type constants below.

### AngleRoundingMultiplier  100

No comments on this option so far

### UseArbitraryPhase  False

This option is currently ignored

### EpsilonLowerBoundary  1.9, EpsilonUpperBoundary  4

sets lower and upper boundary of allowed real epsilon values for diagonal elements.

### EpsilonNonDiagLowerBoundary  -0.4, EpsilonNonDiagUpperBoundary  0.4

sets lower and upper boundary of allowed real epsilon values for non diagonal elements.

### MFHlpDataSet  1

This option is used internally

### FitaTolerance  (1/10^4)

This option is used internally

### EpsilonIMLowerBoundary  -BIIMTOLERANCE, EpsilonIMUpperBoundary  4

sets lower and upper boundary of allowed imaginary epsilon values for diagonal elements.

### EpsilonIMNonDiagLowerBoundary  -1, EpsilonIMNonDiagUpperBoundary  14

sets lower and upper boundary of allowed imaginary epsilon values for non diagonal elements.

### MFHlpAbsorbingEpsilon  False

This option is used internally

### EpsilonStart  EpsilonFromN[1.5,1.9,1.7]

sets starting value of epsilon

### PrintEpsilon  True

if set to true then MFHlp will print epsilon every time when it is called.

### BIIMTOLERANCE=10^-20

This option is used internally

### BIEPSTYPESTDTWOSTEPTRAPSPARENT=1

this constants specifies that epsilon is transparent and that standard two step method should be used in inverse solution.

### BIEPSTYPESTDTWOSTEPABSORBING=101

this constants specifies that epsilon is absorbing and that standard two step method should be used in inverse solution.

## Functions.

### InverseSolutionNew[RefrIndList, InptList, FuncList, FuncNameList, Description, options] (changed in version 4.02)

This function initializes inverse solution. Please, refer to the example for the details. NOTE: In version 4.02 **FuncNameList** parameter was removed as it is no longer needed.

### InverseSolutionRun[InvSol]

This function runs inverse solution.

### InverseSolutionEpsilon[InvSol]

This function returns dielectric susceptibility matrix epsilon for a solved inverse solution (in diagonal form).

### InverseSolutionAngles[InvSol]

This function returns angles of rotation of dielectric susceptibility matrix epsilon for a solved inverse solution (from diagonal form).

### MFHlp[Epsilon, InvSol, mfopts]

This function returns minimization functional for a given epsilon and experimental data. Usually used internally.

### ProcessMinSolEps[Epsilon, InvSol]

This function is used internally to obtain angles of rotation of matrix epsilon.

### EpsilonComp[eps0, deltaEps, nu] ]

This function is used internally.

### EpsilonFull[eps0, deltaEps, nu, eps13, eps23, eps33] ]

This function is used internally.

### MFHlpSetEpsilonLimits[Epsilon, opts] ]

This function is used internally.

### MFHlpSetParLimits[par, minVal, maxVal, minIMVal, maxIMVal] ]

This function is used internally.