

CrysFML

Table of contents

Welcome	23
Authors	23
Structure of CrysFML	24
Installing and Compiling CrysFML	25
Linux	25
MacOS	25
Windows	26
Compiling and Running the Examples	26
CFML4C	27
Cryscal_TR	27
Cryst_Calculator_Console	27
Magnetism	28
Metrics	28
Hkl_Gen	28
PowderPattern	29
Simbo-Enermag	29
SpaceGroups	30
StructureFactors	31
Structures_GlobalOptimization	32
Twins	35
Modules of CrysFML	36
Level 0	39
CFML_GlobalDeps	40
Numeric Parameters	40
CP	41
DEps	41
DP	41
Eps	41
Pi	41
SP	41
To_Deg	41
To_Rad	41
TPi	42
Operative System Parameters	42
Ops	42
Ops_Name	42
Ops_Sep	42
Functions	43
Directory_Exists	43
Level 1	43
CFML_FFT	44
Variables	44
Points_Interval_Type	44
Functions	45
Convol	45
Convol_Peaks	45
F_FFT	46

FFT	47
Subroutines	47
HFFT	47
SFFT	48
CFML_IO_Messages	48
Variables	49
Win_Console	49
Subroutines	49
Close_Scroll_Window	50
Error_Message	50
Info_Message	50
Print_Message	50
Question_Message	50
Stop_Message	51
Wait_Message	51
Warning_Message	51
Write_Scroll_Text	51
CFML_LSQ_TypeDef	52
Parameters	52
Max_Free_Par	52
Variables	52
LSQ_Conditions_Type	52
LSQ_State_Vector_Type	53
LSQ_Data_Type	53
CFML_Math_General	54
Variables	55
Err_MathGen	56
Err_MathGen_Mess	56
Functions	56
AcosD	57
AsinD	57
Atan2D	57
AtanD	57
BessJ0	57
BessJ1	57
BessJ	57
Co_Linear	58
Co_Prime	58
CosD	58
Equal_Matrix	58
Equal_Vector	59
Euclidean_Norm	59
Factorial	59
IMaxLoc	59
IMinLoc	60
Locate	60
Modulo_Lat	60
Norm	60
Negligible	61
OuterProd	61

PGCD	61
PPCM	61
Pythag	61
Scalar	62
SinD	62
TanD	62
Trace	62
ZBelong	63
Subroutines	63
Co_Prime_Vector	63
Determinant	64
Diagonalize_SH	64
First_Derivative	64
In_Sort	65
Init_Err_MathGen	65
Invert_Matrix	65
Linear_Dependent	65
LU_Backsub	66
LU_Decompo	66
MatInv	67
Points_in_Line2D	67
Rank	67
RTan	67
Second_Derivative	68
Set_EPSG	68
Set_EPSG_Default	68
SmoothingVec	68
Sort	68
Sort.Strings	69
Spline	69
Splint	69
SVDCMP	70
Swap	70
CFML_PowderProfiles_CW	70
Functions	71
Back_To_Back_Exp	71
Exponential	72
Gaussian	72
Hat	72
Ikeda_Carpenter	73
Lorentzian	73
PseudoVoigt	73
Split_PseudoVoigt	74
TCH_PVoigt	74
Subroutines	74
Back_To_Back_Exp_Der	74
Exponential_Der	74
Gaussian_Der	75
Hat_Der	75
Ikeda_Carpenter_Der	75

Lorentzian_Der	75
PseudoVoigt_Der	76
Split_PseudoVoigt_Der	76
TCH_PVoigt_Der	76
CFML_PowderProfiles_Finger	76
Variables	77
Init_ProfVal	77
Subroutines	77
Init_Prof_Val	77
Prof_Val	77
CFML_PowderProfiles_TOF	78
Variables	78
Deriv_TOF_Type	79
LorComp	79
Functions	79
Erfc	79
Erfcp	79
Subroutines	80
TOF_Carpenter	80
TOF_Jorgensen	80
TOF_Jorgensen_Vondreele	80
CFML_Random_Generators	81
Variables	82
Err_Random	82
Err_Random_Mess	82
Subroutines	82
Init_Err_Random	82
Random_Beta	82
Random_Binomial1	83
Random_Binomial2	83
Random_Cauchy	84
Random_Chisq	84
Random_Exponential	84
Random_Gamma	84
Random_Gamma1	85
Random_Gamma2	85
Random_Inv_Gauss	85
Random_MVNorm	86
Random_Neg_Binomial	86
Random_Normal	86
Random_Order	87
Random_Poisson	87
Random_T	87
Random_Von_Mises	87
Random_Weibull	87
Seed_Random_Number	88
CFML_Spherical_Harmonics	88
Variables	88
Err_Spher	89
Err_Spher_Mess	89

Functions	89
Cubic_Harm_Ang	89
Cubic_Harm_UcVec	89
Int_Slater_Bessel	90
Real_Spher_Harm_Ang	90
Real_Spher_Harm_UcVec	91
Real_Spher_HarmCharge_UcVec	91
Subroutines	91
Init_Err_Spher	91
Pikout_LJ_Cubic	91
SphJn	92
CFML_String_Utils	92
Variables	93
Err_Text_Type	93
Err_String	93
Err_String_Mess	94
Ierr_FMT	94
Mess_FindFMT	94
Functions	94
Equal_Sets_Text	95
L_Case	95
Pack_String	95
Strip_String	95
U_Case	95
Subroutines	95
CutST	96
FindFMT	96
Frac_Trans_1Dig	97
Frac_Trans_2Dig	97
Get_BaseName	97
Get_DirName	98
Get_Fraction_1Dig	98
Get_Fraction_2Dig	98
Get_LogUnit	98
Get_Separator_Pos	98
GetNum	99
GetNum_STD	99
GetWord	99
Inc_LineNum	100
Init_Err_String	100
Init_FindFMT	100
Lcase	100
Number_Lines	100
NumCol_From_NumFMT	100
Read_Key_Str	101
Read_Key_StrVal	101
Read_Key_Value	101
Read_Key_ValueSTD	102
Reading_Lines	102
SetNum_STD	103

Ucase	103
Level 2	103
CFML_Diffraction_Patterns	104
Variables	104
Diffraction_Pattern_Type	104
Err_DiffPatt	106
Err_DiffPatt_Mess	106
Functions	106
Calc_FWHM_Peak	106
Subroutines	106
Add_Diffraction_Patterns	106
Allocate_Diffraction_Pattern	107
Calc_Background	107
Delete_Noisy_Points	107
Init_Err_DiffPatt	108
Purge_Diffraction_Pattern	108
Read_Background_File	108
Read_Pattern	108
Write_Pattern_FreeFormat	109
Write_Pattern_XYSig	109
CFML_Math_3D	110
Variables	111
Err_Math3D	111
Err_Math3D_Mess	111
Functions	111
Cross_Product	111
Determ_A	111
Determ_V	112
Invert_A	112
Polyhedron_Volume	112
Rotate_OX	112
Rotate_OY	113
Rotate_OZ	113
VecLength	113
Subroutines	113
Get_Cart_From_Cylin	113
Get_Cart_From_Spher	114
Get_Centroid_Coord	114
Get_Cylindr_Coord	114
Get_Plane_From_Points	114
Get_Spheric_Coord	115
Init_Err_Math3D	115
Matrix_DiagEigen	115
Matrix_Inverse	115
Resolv_Sist_1x2	116
Resolv_Sist_1x3	116
Resolv_Sist_2x2	116
Resolv_Sist_2x3	117
Resolv_Sist_3x3	117
Set_Eps	117

Set_Eps_Default	118
CFML_Optimization_General	118
Variables	118
Opt_Conditions_Type	118
Err_Optim	120
Err_Optim_Mess	120
Subroutines	120
CG_Quasi_Newton	121
CSendes_Global	121
Init_Err_Optim	122
Init_Opt_Conditions	122
Local_Min_DFP	123
Local_Min_Rand	124
Local_Optimize	124
Nelder_Mead_Simplex	125
Set_Opt_Conditions	126
Write_Optimization_Conditions	126
CFML_Optimization_LSQ	126
Variables	127
Err_LSQ	127
Err_LSQ_Mess	127
Info_LSQ_Mess	128
Functions	128
FChiSQ	128
Subroutines	128
Info_LSQ_Output	128
Levenberg_Marquardt_Fit	129
Marquardt_Fit	132
CFML_Scattering_Chemical_Tables	134
Parameters	135
Num_Chem_Info	135
Num_Delta_Fp	135
Num_Mag_Form	135
Num_Mag_J2	135
Num_Mag_J4	136
Num_Mag_J6	136
Num_Xray_Form	136
Variables	136
Anomalous_SC_Type	136
Chem_Info_Type	137
Magnetic_Form_Type	137
Xray_Form_Type	137
Xray_Wavelength_Type	138
Anomalous_SCFac	138
Chem_Info	138
Magnetic_Form	138
Magnetic_J2	138
Magnetic_J4	138
Magnetic_J6	139
Xray_Form	139

Xray_Wavelengths	139
Subroutines	139
Get_Atomic_Mass	140
Get_ChemSymb	140
Get_Covalent_Radius	140
Get_Fermi_Length	140
Get_Ionic_Radius	140
Remove_Chem_Info	141
Remove_Delta_Fp_Fpp	141
Remove_Magnetic_Form	141
Remove_Xray_Form	141
Set_Chem_Info	141
Set_Delta_Fp_Fpp	141
Set_Magnetic_Form	141
Set_Xray_Form	141
CFML_Symmetry_Tables	141
Parameters	143
BC_D6H	143
BC_OH	144
DepMat	144
IntSymD6H	145
IntSymOH	145
Kov_D6H	146
Kov_OH	146
Latt	146
Laue_Class	147
Ltr_A	147
Ltr_B	147
Ltr_C	148
Ltr_F	148
Ltr_I	148
Ltr_R	148
MagMat	149
ML_D6H	149
ML_OH	150
Mod6	150
Point_Group	150
Sys_Cry	151
X_D6H	151
X_OH	151
Zak_D6H	152
Zak_OH	152
Variables	153
Spgr_Info_Type	153
Table_Equiv_Type	153
Wyck_Info_Type	154
Err_SymTab	154
Err_SymTab_Mess	154
Spgr_Info	154
System_Equiv	154

Wyckoff_Info	154
Subroutines	155
Get_Generators	155
Remove_Spgr_Info	155
Remove_System_Equiv	155
Remove_Wyckoff_Info	155
Set_Spgr_Info	155
Set_System_Equiv	156
Set_Wyckoff_Info	156
Level 3	156
CFML_Bond_Tables	157
Variables	157
Bond_Length_Table	157
Err_Bond	158
Err_Bond_Mess	158
Subroutines	158
Get_Bonds_Table	158
Init_Err_Bond	158
Remove_Bonds_Table	158
Set_Bonds_Table	159
CFML_Crystal_Metrics	159
Variables	160
Crystal_Cell_Type	160
Twofold_Axes_Type	161
Err_Crys	161
Err_Crys_Mess	161
Functions	161
Cart_U_Vector	162
Cart_Vector	162
Convert_B_Betas	162
Convert_B_U	162
Convert_Betas_B	162
Convert_Betas_U	163
Convert_U_B	163
Convert_U_Betas	163
Get_Betas_from_Biso	163
Rot_Matrix	163
U_Equiv	164
Subroutines	164
Change_Setting_Cell	164
Get_Conventional_Cell	164
Get_Cryst_Family	165
Get_Deriv_Orth_Cell	165
Get_Primitive_Cell	165
Get_Transform_Matrix	165
Get_Twofold_Axes	166
Init_Err_Crys	166
Niggli_Cell	166
Set_Crystal_Cell	168
Write_Crystal_Cell	168

CFML_Crystallographic_Symmetry	168
Parameters	170
Cubic	170
HexaG	171
Monoc	171
Num_Spgr_Info	171
Orthor	171
Tetra	171
Trigo	171
Variables	171
Space_Group_Type	172
Sym_Oper_Type	172
Wyck_Pos_Type	173
Wyckoff_Type	173
Lat_Ch	173
Err_Symm	173
Err_Symm_Mess	174
Hexa	174
InLat	174
Ltr	174
NLat	174
SpaceG	174
Functions	174
ApplySO	175
Axes_Rotation	175
Get_Laue_Num	175
Get_Multip_Pos	175
Get_Occ_Site	175
Get_PointGroup_Num	175
Is_New_OP	176
Lattice_Trans	176
Spgr_Equal	176
Sym_Prod	176
Subroutines	176
DecodMatMag	177
Get_Centring_Vectors	178
Get_Crystal_System	178
Get_HallSymb_From_Gener	178
Get_Lattice_Type	178
Get_Laue_PG	179
Get_Laue_Str	179
Get_Orbit	179
Get_PointGroup_Str	179
Get_SO_From_Fix	180
Get_SO_From_Gener	180
Get_SO_From_Hall	181
Get_SO_From_HMS	181
Get_Stabilizer	182
Get_String_Resolv	182
Get_SubOrbits	182

Get_SymEl	183
Get_SymKov	183
Get_SymSymb	183
Get_T_SubGroups	184
Init_Err_Symm	184
Inverse_Symm	184
LatSym	184
Read_MSymm	184
Read_SymTrans_Code	185
Read_XSym	185
SearchOP	185
Set_SpaceGroup	186
Set_Spg_Mult_Table	186
Setting_Change	186
Similar_Transf_SG	187
Sym_B_Relations	187
Sym_Prod_St	188
Symmetry_Symbol	188
Write_SpaceGroup	189
Write_Sym	189
Write_SymTrans_Code	189
Write_Wyckoff	189
Wyckoff_Orbit	190
CFML_ILL_Instrm_Data	190
Variables	191
Basic_NumC_Type	191
Basic_NumI_Type	192
Basic_NumR_Type	192
Calibration_Detector_Type	192
Diffractometer_Type	193
Generic_Numor_Type	194
ILL_Data_Record_Type	194
Powder_Numor_Type	196
SXTAL_Numor_Type	196
SXTAL_Orient_Type	197
Current_Instrm	197
Current_Orient	198
Cycle_Number	198
Err_ILLData	198
Err_ILLData_Mess	198
ILL_Data_Directory	198
ILL_Temp_Directory	198
Instrm_Directory	198
Machine_Name	198
Year_ILLData	198
Subroutines	198
Allocate_Numors	199
Define_Uncompress_Program	199
Get_Absolute_Data_Path	200
Get_Next_YearCycle	200

Get_Single_Frame_2D	200
Init_Err_ILLData	201
Initialize_Numor	201
Initialize_Data_Directory	201
PowderNumors_To_DiffPattern	201
Read_Calibration_File	202
Read_Current_Instrm	202
Read_Numor	202
Set_Current_Orient	202
Set_Default_Instrument	203
Set_ILL_Data_Directory	203
Set_Instrm_Directory	203
Update_Current_Instrm_UB	203
Write_Current_Instrm_Data	204
Write_HeaderInfo_Numor	204
Write_Numor_Info	204
Level 4	204
CFML_Atom_TypeDef	205
Variables	206
Atom_Type	206
Atom_List_Type	207
Atoms_Cell_Type	207
MAtom_Type	208
MAtom_List_Type	209
Err_ATMD	209
Err_ATMD_Mess	210
Functions	210
Equiv_ATOM	210
WRT_Lab	210
Subroutines	210
Allocate_Atom_List	210
Allocate_Atoms_Cell	211
Allocate_MAtom_List	211
ATList1_ExtenCell_ATList2	211
Atom_List_To_Cell	212
Atom_Uequi_List	212
Atoms_Cell_To_List	212
Copy_Atom_List	212
Deallocate_Atom_List	212
Desallocate_Atoms_Cell	213
Deallocate_MAtom_List	213
Init_Atom_Type	213
Init_Err_ATMD	213
Init_MAtom_Type	213
Merge_Atoms_Peaks	213
Multi	214
Write_Atom_List	214
Write_Atoms_CFL	214
Write_CFL	215
CFML_EoS	215

Parameters	216
NCol_Data_Max	216
N_EoSPar	216
Variables	216
EoS_Data_Type	217
EoS_Data_List_Type	217
EoS_Type	218
EoS_List_Type	219
Err_EoS	219
Err_EoS_Mess	219
Functions	219
Alpha_Cal	220
dKdT_Cal	220
Get_Pressure	220
Get_Pressure_Esd	220
Get_Temperature	220
Get_Volume	221
Get_Volume_S	221
K_Cal	221
Kp_Cal	221
Kpp_Cal	221
Pressure_F	221
Strain	222
Strain_EoS	222
Subroutines	222
Allocate_EoS_Data_list	223
Allocate_EoS_List	223
Deallocate_EoS_Data_List	223
Deallocate_EoS_List	223
Deriv_Partial_P	223
EoS_Cal	224
EoS_Cal_Esd	224
FfCal_Dat	224
FfCal_Dat_Esd	225
FfCal_EoS	225
Init_EoS_Data_Type	225
Init_EoS_Thermal	225
Init_EoS_Type	226
Init_Err_EoS	226
Read_EoS_DataFile	226
Set_EoS_Names	226
Set_EoS_Use	226
Set_Kp_Kpp_Cond	227
Write_Info_EoS	227
CFML_Geometry_SXTAL	227
Variables	228
PSD_Val_Type	228
SXD_Val_Type	229
Err_SXTGeom	229
Err_SXTGeom_Mess	229

PSD	230
SXD	230
Subroutines	230
Angs_4C_Bisecting	231
CalAng	231
Calc_Om_Chi_Phi	231
Calc_Psi	232
Cell_Fr_UB	232
Chi_Mat	232
D19PSD	233
DSpace	233
Equatorial_Chi_Phi	233
FixDNu	233
Flat_Cone_VertDet	234
GenB	234
GenUB	234
Get_Angs_NB	235
Get_DSpacing_Theta	235
Get_GaOmNu_FrChiPhi	235
Get_WaveGaNu_FrZ4	236
Get_Z1_D9Angls	236
Get_Z1_From_Pixel	236
Normal	236
Normal_Beam_Angles	237
Phi_Mat	237
PSD_Convert	237
Psi_Mat	238
RefVec	238
S4CNB	238
Set_PSD	239
SNB4C	239
SXDPSPD	239
Triple	240
Z1FrFC	240
Z1FrMD	240
Z1FrNB	240
Z1FrZ2	241
Z1FrZ3	241
Z1FrZ4	241
Z2FrZ1	242
Z3FrZ1	242
Z4FrGN	242
Z4FrZ1	242
CFML_Reflections_Utils	243
Variables	244
Reflect_Type	244
Reflection_Type	244
Reflection_List_Type	244
Err_Refl	245
Err_Refl_Mess	245

HKL_Ref_Conditions	245
Functions	245
AsU_HKL	245
Get_HEquiv_AsU	246
Get_MaxNumRef	246
HKL_Absent	246
HKL_Equal	246
HKL_Equiv	247
HKL_Mult	247
HKL_R	247
HKL_S	247
Unit_Cart_HKL	248
Subroutines	248
HKL_Equiv_List	248
HKL_Gen	248
HKL_Gen_SXTAL	249
HKL_RP	249
HKL_Uni	250
Init_Err_Refl	250
Init_RefList	250
Search_Extinctions	250
Write_AsU	251
Write_RefList_Info	251
Level 5	251
CFML_Geometry_Calc	251
Variables	253
Coordination_Type	253
Point_List_Type	253
Coord_Info	254
Err_Geom	254
Err_Geom_Mess	254
Functions	254
Angle_Dihedral	254
Angle_Mod	255
Angle_UV	255
Coord_Mod	255
Distance	256
Matrix_PhiTheChi	256
Matrix_RX	256
Matrix_RY	257
Matrix_RZ	257
Subroutines	257
Allocate_Coordination_Type	258
Allocate_Point_List	258
Calc_Dist_Angle	258
Calc_Dist_Angle_Sigma	258
Deallocate_Coordination_Type	259
Deallocate_Point_List	259
Distance_And_Sigma	259
Get_Euler_From_Fract	259

Get_PhiTheChi	260
Get_Transf_List	260
Init_Err_Geom	260
P1_Dist	261
Print_Distances	261
Set_Orbits_InList	261
Set_TDist_Coordination	261
Set_TDist_Partial_Coordination	262
CFML_Propagation_Vectors	262
Variables	263
Group_K_Type	263
Functions	263
HK_Equiv	263
K_Equiv	264
K_Equiv_Minus_K	264
Subroutines	264
K_Star	264
Write_Group_K	264
CFML_Structure_Factors	265
Variables	265
Err_SFac	265
Err_SFac_Mess	265
Subroutines	265
Calc_HKL_StrFactor	266
Calc_StrFactor	266
Init_Calc_StrFactors	267
Init_Calc_HKL_StrFactors	267
Init_Structure_Factors	267
Modify_SF	268
Structure_Factors	268
Write_Structure_Factors	268
Level 6	269
CFML_BVS_Energy_Calc	269
Parameters	270
BVS_Anions	270
BVS_Anions_N	270
BVS_Anions_RIon	270
BVS_Species_N	271
Variables	271
Atoms_Conf_List_Type	271
BVS_Par_Type	272
BVS_Table	272
Err_Conf	272
Err_Conf_Mess	272
Subroutines	272
Allocate_Atoms_Conf_List	273
Calc_BVS	273
Calc_Map_BVS	273
Cost_BVS	274
Cost_BVS_CoulombRep	274

Deallocate_Atoms_Conf_List	274
Deallocate_BVS_Table	275
Init_Err_Conf	275
Set_BVS_Table	275
Set_Table_D0_B	275
Species_on_List	275
CFML_Maps_Calculations	275
Parameters	276
Max_Points	276
Variables	276
Cube_Info_Type	277
Cube_Info	277
Err_Maps	277
Err_Maps_Mess	277
Functions	277
Index_Cube	277
Vertice_Point	278
Vertices_Cube	278
VPoint_in_Cube	278
VPoint_in_Line	279
VPoint_in_Square	279
Subroutines	280
Calculate_Contour2D	280
Calculate_Mesh	281
Init_Err_Maps	281
Load_ExtendedMap	281
Load_Section	281
Search_Peaks	282
Set_Cube_Info	282
Statistic_Map	282
CFML_Molecular_Crystals	283
Variables	284
Molecule_Type	284
Molecular_Crystal_Type	285
Err_Molec	286
Err_Molec_Mess	286
Subroutines	286
Cartesian_To_Fractional	287
Cartesian_To_Spherical	287
Cartesian_To_ZMatrix	287
Empiric_Formula	288
Fix_Orient_Cartesian	288
Fix_Reference	288
Fractional_To_Cartesian	289
Fractional_To_Spherical	289
Fractional_To_ZMatrix	289
Init_Err_Molec	290
Init_Molecule	290
MolCrys_To_AtomList	290
Molec_To_AtomList	290

Read_Free_Atoms	290
Read_Molecule	291
Set_Euler_Matrix	292
Spherical_To_Cartesian	292
Spherical_To_Fractional	293
Spherical_To_ZMatrix	293
Write_Free_Atoms	293
Write_Molecular_Crystal	293
Write_Molecule	294
ZMatrix_To_Cartesian	294
ZMatrix_To_Fractional	294
ZMatrix_To_Spherical	294
Level 7	294
CFML_IO_Formats	295
Variables	296
File_List_Type	296
Interval_Type	296
Job_Info_Type	296
Err_Form	297
Err_Form_Mess	297
Subroutines	297
File_To_FileList	298
Get_Job_Info	298
Get_Phases_File	299
Init_Err_Form	299
Read_Atom	299
Read_Cell	299
Read_CIF_Atom	300
Read_CIF_Cell	300
Read_CIF_ChemicalName	300
Read_CIF_Cont	300
Read_CIF_Hall	301
Read_CIF_HM	301
Read_CIF_Lambda	301
Read_CIF_Symm	302
Read_CIF_Title	302
Read_CIF_Z	302
Read_File_Atom	302
Read_File_Cell	303
Read_File_Lambda	303
Read_File_RNGSinTIL	304
Read_File_Spg	304
Read_File_Transf	304
Read_SHX_Atom	305
Read_SHX_Cell	305
Read_SHX_Cont	306
Read_SHX_FVar	306
Read_SHX_Latt	306
Read_SHX_Symm	306
Read_SHX_Titl	307

Read_UVals	307
ReadN_Set_XTAL_Structure	307
Write_CIF_Powder_Profile	308
Write_CIF_Template	308
Write_SHX_Template	308
Level 8	309
CFML_Keywords_Code_Parser	309
Parameters	310
Code_Nam	310
Key_Code	311
Variables	311
Angle_Restraint_Type	312
Distance_Restraint_Type	312
Torsion_Restraint_Type	313
Ang_Rest	313
Dis_Rest	313
Err_RefCodes	313
Err_RefCodes_Mess	313
NP_Cons	313
NP_Max	313
NP_Refi	313
NP_Rest_Ang	314
NP_Rest_Dis	314
NP_Rest_Tor	314
Tor_Rest	314
V_BCon	314
V_Bounds	314
V_List	314
V_Name	314
V_Vec	314
V_Shift	315
Subroutines	315
Allocate_RestParam	315
Allocate_VParam	315
Get_RestAng_Line	315
Get_RestDis_Line	315
Get_RestTor_Line	316
Init_Err_RefCodes	316
Init_RefCodes	316
Read_RefCodes_File	316
VState_To_AtomsPar	317
Write_Info_RefCodes	317
Write_Info_RefParams	317
Write_Restraints_ObsCalc	317
CFML_Magnetic_Symmetry	318
Variables	318
Magnetic_Domain_Type	318
Magnetic_Group_Type	319
MagSymm_K_Type	319
MSym_Oper_Type	320

Err_MagSym	321
Err_MagSym_Mess	321
Functions	321
ApplyMSO	321
Subroutines	321
Init_Err_MagSym	321
Init_MagSymm_K_Type	321
ReadN_Set_Magnetic_Structure	322
Set_Shubnikov_Group	322
Write_Magnetic_Structure	322
Write_Shubnikov_Group	322
CFML_Simulated_Annealing	323
Parameters	323
NP_CONF	324
NP_SAN	324
Variables	324
MultiState_Vector_Type	324
SimAnn_Conditions_Type	325
State_Vector_Type	325
Err_SAN	326
Err_SAN_Mess	326
Subroutines	326
SAnn_Opt_MultiConf	326
Set_SimAnn_Cond	327
Set_SimAnn_MStateV	327
Set_SimAnn_StateV	327
SimAnneal_Gen	328
SimAnneal_MultiConf	328
Write_SimAnn_Cond	329
Write_SimAnn_MStateV	329
Write_SimAnn_StateV	329
Level 9	330
CFML_Magnetic_Structure_Factors	330
Variables	331
MagH_Type	331
MagH_List_Type	331
MagHD_Type	332
MagHD_List_Type	332
Err_MSFac	332
Err_MSFac_Mess	332
Subroutines	333
Calc_Mag_Interaction_Vector	333
Calc_Magnetic_STRF_MIV	333
Calc_Magnetic_STRF_MIV_Dom	333
Gen_Satellites	334
Init_Err_MSFac	334
Init_Mag_Structure_Factors	334
Mag_Structure_Factors	334
Modify_MSF	335
Write_Mag_Structure_Factors	335

CFML_Polarimetry	335
Variables	336
Polar_Calc_Type	336
Polar_Calc_List_Type	337
Polar_Info_Type	337
Polar_Obs_Type	338
Polar_Obs_List_Type	338
Subroutines	338
Calc_Polar_Dom	338
Set_Polar_Info	339
Write_Polar_Info	339
Write_Polar_Line	339

Welcome

Crystallographic Fortran Modules Library

Version: 5.0

The Crystallographic Fortran Modules Library (**CrysFML**) is a set of Fortran 90/95 modules containing procedures of interest in Crystallographic applications.

This set of modules has been, and is still being, developed by us in order to facilitate the design and the development of crystallographic computing programs.

Many of the algorithms and procedures of the library come from adaptations and modifications of existing codes of different sources. We make the source code available publicly without any licence (we have to time to think in legal stuff). We hope that academic groups interested in cooperative scientific software development will take some benefit of the library and we expect that additional individuals will contribute to the development of the library. If somebody is interested in working in the library at the level of developer we can add his/her name in the list of developer by contacting us by e-mail.

The **CrysFML** library is distributed in the hope that it will be useful, but **without any warranty** of being free of internal errors. In no event will the authors be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the library (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the library to operate with any other programs).

Authors

Authors



Juan Rodríguez-Carvajal

Institut Laue Langevin
Diffraction Group
6, rue Jules Horowitz
BP 156 - 38042 Grenoble Cedex 9,
FRANCE
E-mail: jrc@ill.fr



Javier González-Platas

Dpto. Fisica Fundamental II
Universidad de La Laguna
Avda. Astrofísico Fco. Sanchez s/n
E-38204 La Laguna, Tenerife, SPAIN
E-mail: jplatas@ull.es

Collaborators

Structure of CrysFML

The present distribution of **CrysFML** have the following directory structure:

Src	CFGL CFML Scripts	Linux MacOS Windows
<Compiler> <i>Absoft</i> <i>G95</i> <i>GFortran</i> <i>Intel</i> <i>Lahey</i>	LibC LibGL LibR (*) LibW	
Help (**)		
Html (**)	files	
Program_Examples	CFML4C Cryscal_TR Cryst_Calculator_Console Magnetism Metrics HKL_Gen PowderPattern Simbo-Enermag SpaceGroups StructureFactors Structures_GlobalOptimization Twins	

(*) Using Lahey compiler and RealWin library (obsolete)

(**) Windows Help format (chm)

(***) Html format for use in Linux, MacOS, Windows

Installing and Compiling CrysFML

The installation of **CrysFML** depends of your operating system. Presently we have tested it in the following operating systems:

- [Linux](#)
- [MacOS](#)
- [Windows](#)

Linux

Please, follow the following steps to install the **CrysFML** library in your Linux System.

1. Create a new directory to install **CrysFML** (<CRYSFMLDIR>)
For example: \$HOME/CrysFML
2. Download the last version of **CrysFML** from the [ILL forge site](#).
You can do it directly from the site or using a **svn** program as [RapidSVN](#) or [KDESvn](#) that is integrated in the KDE file system.
3. After downloading the Library you should have at <CRYSFMLDIR>, **Src**, **Help** and **Program_Examples** subdirectories.
Then, go to **Src** directory.
4. Check that you have a file called **makecrys** with execute permission.
5. Run the script file with the name of the command invoking your compiler. More info if you run the script file without arguments. At present **CrysFML** exist in two versions:
 - **Console**
Run: **makecrys <compiler-command>**
 - **Graphical**
in this case, **CrysFML** will be linked together with the [Winteracter Library](#)
Run: **makecrys <compiler-command> all**
6. If the script runs properly you'll have **CrysFML** compiled in the form of a library and the list of module files (*.mod) in <CRYSFMLDIR>/Compiler/LibC
Where **Compiler** can be: *Absoft, intel, Lahey, G95, GFortran*

If you have compiled and linked with the Winteracter library, you will get two additional subdirectories: **LibW** and **LibGL**

MacOS

Please, follow the following steps to install the **CrysFML** library in your MacOS X System.

1. Create a new directory to install the **CrysFML** library (<CRYSFMLDIR>)
For example: \$HOME/CrysFML
2. Download the last version of **CrysFML** from the [ILL forge site](#).
You can do it directly from the site or using a **svn** program as [ZigZig](#), [RapidSVN](#) or [SvnX](#).
3. After downloading the Library you should have at <CRYSFMLDIR>, **Src**, **Help** and **Program_Examples**

subdirectories.

Then, go to **Src** directory.

4. Check that you have a file called **makecrys** with execute permission.
5. Run the script file with the name of the command invoking your compiler. More info if you run the script file without arguments. At present **CrysFML** exist in two versions:
 - o **Console**
Run: **makecrys <compiler-command>**
 - o **Graphical**
in this case, **CrysFML** will be linked together with the [Winteracter Library](#)
Run: **makecrys <compiler-command> all**
6. If the script runs properly you'll have **CrysFML** compiled in the form of a library and the list of module files (*.mod) in <CRYSFMLDIR>/Compiler/LibC
Where *Compiler* can be: *Absoft, intel, G95, GFortran*

If you have compiled and linked with the Winteracter library, you will get two additional subdirectories: *LibW* and *LibGL*

Windows

Please, follow the following steps to install the **CrysFML** library in your Windows System.

1. Create a new directory to install the **CrysFML** library (<CRYSFMLDIR>)
For example: C:\CrysFML
 2. Download the last version of **CrysFML** from the [ILL forge site](#).
You can do it directly from the site or using a **svn** program as [Tortoise](#).
 3. After downloading the Library you should have at <CRYSFMLDIR>, **Src**, **Help** and **Program_Examples** subdirectories.
Then, go to **Src** directory.
 4. Check that you have a file called **makecrys.bat**
 5. Run the batch file with the name of the command invoking your compiler. More info if you run the batch file without arguments. At present **CrysFML** exist in two versions:
 - o **Console**
Run: **makecrys <compiler-command>**
 - o **Graphical**
in this case, **CrysFML** will be linked together with the [Winteracter Library](#)
Run: **makecrys <compiler-command> all**
 6. If the batch file runs properly you'll have **CrysFML** compiled in the form of a library and the list of module files (*.mod) in <CRYSFMLDIR>\Compiler\LibC
Where *Compiler* can be: *Absoft, G95, GFortran, Intel, Lahey*
- If you have compiled and linked with the Winteracter library, you will get two additional subdirectories: *LibW* and *LibGL*

Compiling and Running the Examples

Some examples programs are distributed together with the **CrysFML** library in order to facilitate the

understanding of how you can make programs. At the moment, the examples are:

- [CFML4C](#)
- [Cryscal TR](#)
- [Cryst Calculator Console](#)
- [Magnetism](#)
- [Metrics](#)
- [Hkl Gen](#)
- [PowderPattern](#)
- [Simbo-Enermag](#)
- [SpaceGroups](#)
- [StructureFactors](#)
- [Structures_GlobalOptimization](#)
- [Twins](#)

CFML4C

This is a sample to use **CFML** in C language.

Building the executable file

Run the script (for Linux or MacOS). The user must have installed the GFortran and GCC compilers.

make_xtl

Running the program

./xtl test

Cryscal TR

Console program for crystallographic calculations from Thierry Roisnel.

Building the executable file

Run the batch file (for Windows) located in scripts directory

make_cryscal lf95| g95 |all

Running the program

The program is invoked at the prompt using the name of the executable file.

Cryst_Calculator_Console

Initial example about the use of CrysFML to do some crystallographic calculations.

Building the executable file

Run the batch file (for Windows) or the script (for Linux or MacOS)

make_cryscal <compiler-command>

Running the program

The program is invoked at the prompt using the name of the executable file.

Magnetism

Examples about the use of **CrysFML** for calculations of Magnetic structure factors and interaction

MagPolar program calculates magnetic structure factors, and magnetic interaction, vectors from magnetic structures by reading a *.CFL file and
Magnetic S-domains and chirality domains are considered.

MagRef calculates magnetic structure factors, and magnetic interaction, vectors from magnetic structures by reading a *.CFL file

Building the executable file

Run the batch file (for Windows)

```
make_magref g95 | gfortran | ifort | lf95
make_magpolar3d g95 | gfortran | ifort | lf95
```

Running the program

The program is invoked at the prompt using the name of the executable file.

Metrics

Get_Conven_Cell is a program to calculate the conventional unit cell parameters from an input unit cell using **CFML**.

Building the executable file

Run the batch file (for Windows) or the script (for Linux or MacOS)

```
make_convcell <compiler-command>
```

Running the program

The program is invoked at the prompt using the name of the executable file.

Hkl_Gen

Hkl_Gen is a program that generates HKL reflections using a minimal information

Building the executable file

Run the batch file (for Windows) or the script (for Linux or MacOS)

```
make_hkl_gen <compiler-command>
```

Running the program

The program is invoked at the prompt using the name of the executable file generated in the previous step.

The program calculate a list of unique reflections and for it the user will have to introduce the space group, cell parameters, wavelength and the range on $\sin\theta/\lambda$. All reflections calculated will be written in output file.

```
C:\CrysFML\Program_Examples\hkl_gen>hkl_gen
=====
List of unique reflections
=====
=> Space Group <HM/Hall symbol or number>: p b c a
=> Cell parameters <a,b,c>: 10.5 8.3 13.4
=> Give the wavelength: 0.71073
=> Interval in Sin_Theta/Lambda <2 reals, if val1 < 0 => steps>: 1 0.5
=> Maximum number of reflections: 1761
=> Name of the output file: test.hkl
```

PowderPattern

There are two examples about the calculation of powder pattern reading CIF or CFL file.

Simple_Calc_Powder is a very simple program for calculating powder patterns by reading a CIF or a CFL file while **Calc_Powder** is an extended version of the previous program. In this case the user can use it for generating a super-cell and perturb the atom positions to see the effect on the powder diffraction pattern.

Building the executable file

Run the batch file (for Windows) or the script (for Linux or MacOS)

```
make_simple_calc_powder <compiler-command>
make_calc_powder <compiler-command>
```

Running the program

The program is invoked at the prompt using the name of the executable file.

Simbo-Enermag

Simbo program generate files with neighbouring information around magnetic atoms for simulation purposes.

Enermag program calculate the classical magnetic energy for a set of k-vectors and exchange parameters. The input file is created by the program **Simbo**.

Phase_Diagram is a console program helps to visualise a magnetic phase diagram as a function of

exchange interactions using **GFourier** from the **FullProf Suite**.

The program reads a file (with extension *.res) coming from **Enermag** (when used for generating a phase diagram). A maximum of three variable exchange interactions are taken into account. And generates some information in the screen as well as a binary file that can be read from **GFourier**

Building the executable file

Run the batch file (for Windows) to compile the programs

```
make_Simbo      <compiler-command>
make_EnerMag    <compiler-command>
make_Phas_Diag <compiler-command>
```

Running the program

The program is invoked at the prompt using the name of the executable file.

SpaceGroups

Program that give all information related with the Space group.

Building the executable file

Run the batch file (for Windows) or the script (for Linux or MacOS)

```
make_space_group_info <compiler-command>
```

Running the program

The program is invoked at the prompt using the name of the executable file generated in the previous step: space_group_info.

To the question about the space group the user can answer with the number of the space group, the Hermann-Mauguin symbol or the Hall symbol.

After striking the <enter> key the program shows all information about the space group.

```

C:\> CrysFML\Program_Examples\SpaceGroups>space_group_info
=> Enter a space group:
=> Space Group <HM/Hall symbol or number>: p 21 21 21

Information on Space Group:

=> Number of Space group: 19
=> Hermann-Mauguin Symbol: P 21 21 21
=> Hall Symbol: P 2ac 2ab
=> Table Setting Choice:
=>     Setting Type: IT <Generated from Hermann-Mauguin symbol>
=>     Crystal System: Orthorhombic
=>     Laue Class: mmm
=>     Point Group: 222
=>     Bravais Lattice: P
=>     Lattice Symbol: oP
=> Reduced Number of S.O.: 4
=> General multiplicity: 4
=> Centrosymmetry: Acentric
=> Generators <exc. -1&L>: 2
=>     Asymmetric unit: 0.000 <= x <= 0.500
                           0.000 <= y <= 0.500
                           0.000 <= z <= 1.000
=> Centring vectors: 0

=> List of all Symmetry Operators and Symmetry Symbols

=> SYMM< 1>: x,y,z                                     Symbol: 1
=> SYMM< 2>: x+1/2,-y+1/2,-z                         Symbol: 2 <1/2,0,0> x,1/4,0
=> SYMM< 3>: -x,y+1/2,-z+1/2                        Symbol: 2 <0,1/2,0> 0,y,1/4
=> SYMM< 4>: -x+1/2,-y,z+1/2                        Symbol: 2 <0,0,1/2> 1/4,0,z
=> Enter a space group:
=> Space Group <HM/Hall symbol or number>:

```

Example showing the screen where Space_Group_info is ran

StructureFactors

Simple program to do Structure factors calculations using **CrysFML**.

Building the executable file

Run the batch file (for Windows) or the script (for Linux or MacOS)

```
make_calc_sfac <compiler-command>
```

Running the program

The program is invoked at the prompt using the name of the executable file: **calc_sfac**

```

Intel(R) Visual Fortran Compiler for applications running on IA-32, Version 10.1.025 build environment
C:\CrysFML\Program_Examples\StructureFactors>calc_sfac

----- PROGRAM STRUCTURE FACTORS -----
----- Version 0.2 November-2008-----
***** Calculates structure factors reading a *.CFL or a *.CIF file ****
***** <JRC- November 2008 >

=> Code of the file xx.cif<cfl> <give xx>: mfe_sfac
=> Maximum sinTheta/Lambda: 0.5
Normal End of: PROGRAM STRUCTURE FACTORS
Results in File: mfe_sfac.sfa

C:\CrysFML\Program_Examples\StructureFactors>

```

The input file requires a minimal information to do the calculations. Here an example for use with this program.

```

mfe_sfac - Bloc de notas
Archivo Edición Formato Ver Ayuda
Title NiFePO5
!           a      b      c   alpha   beta   gamma
Cell  7.1882  6.3924  7.4847  90.000  90.000  90.000
!           Space Group
Spgr  P n m a
!
!           x      y      z      B      occ   Spin Charge
Atom  Ni   NI  0.0000  0.0000  0.0000  0.74   0.5    2.0   2.0
Atom  Fe   FE  0.1443  0.2500  0.7074  0.63   0.5    5.0   3.0
Atom  P    P   0.3718  0.2500  0.1424  0.79   0.5    0.0   5.0
Atom  O1   O   0.3988  0.2500  0.64585 0.71   0.5    0.0  -2.0
Atom  O2   O   0.19415 0.2500  0.0253  0.70   0.5    0.0  -2.0
Atom  O3   O   0.0437  0.2500  0.4728  0.83   0.5    0.0  -2.0
Atom  O4   O   0.3678  0.0566  0.2633  0.77   1.0    0.0  -2.0

```

Structures_GlobalOptimization

More elaborate example using CrysFML about simulated annealing procedure.

Building the executable file

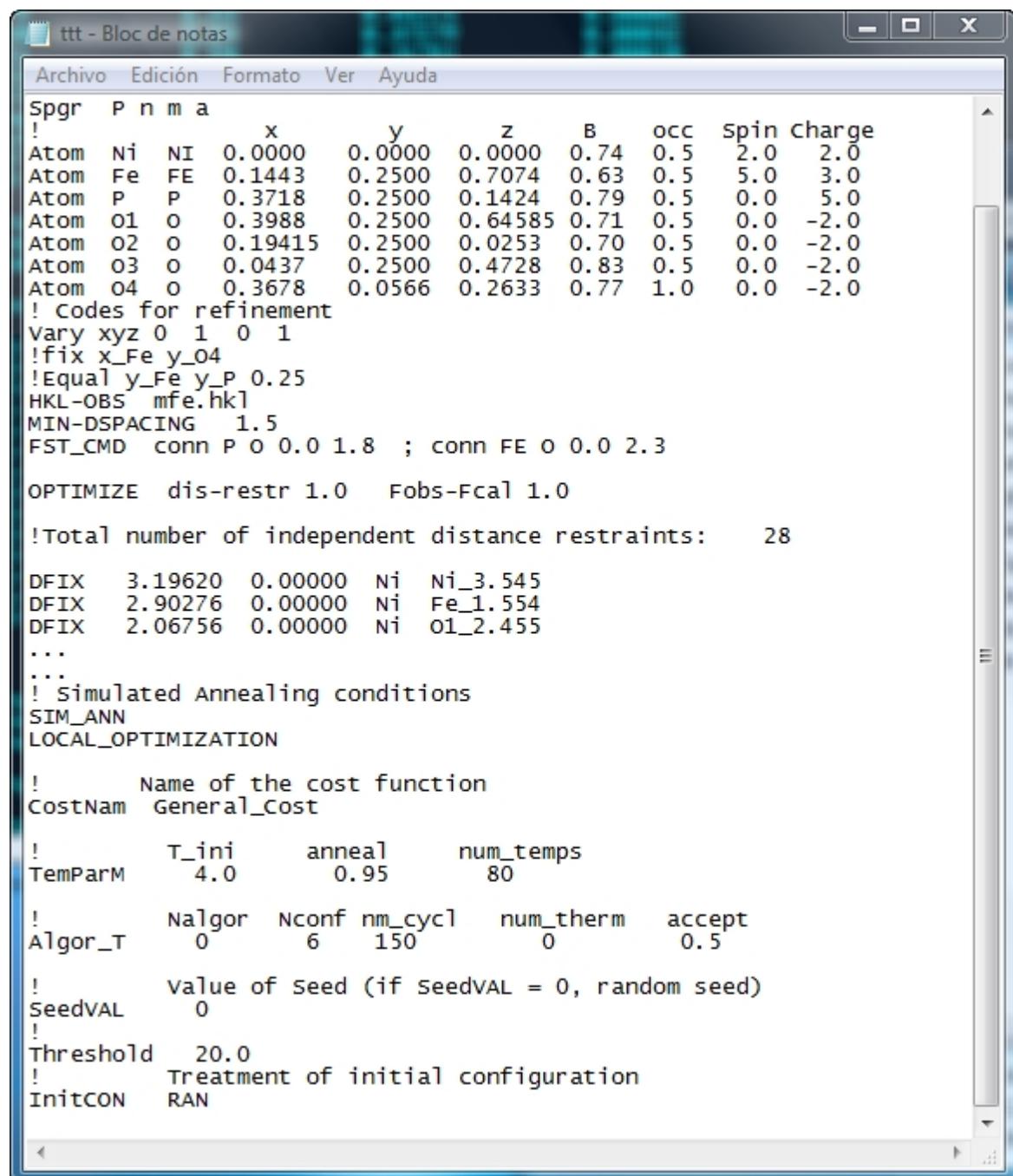
Run the batch file (for Windows) or the script (for Linux or MacOS)

```
make_optim_gen <compiler-command>
```

Running the program

The program is invoked at the prompt using the name of the executable file generated in the previous step. The user need a file containing the reflections and an input file to introduce the information for calculations.

This is a piece of general example to do Simulated annealing refinement



```

Spgr P n m a
!
Atom Ni NI 0.0000 0.0000 0.0000 0.74 0.5 2.0 2.0
Atom Fe FE 0.1443 0.2500 0.7074 0.63 0.5 5.0 3.0
Atom P P 0.3718 0.2500 0.1424 0.79 0.5 0.0 5.0
Atom O1 O 0.3988 0.2500 0.64585 0.71 0.5 0.0 -2.0
Atom O2 O 0.19415 0.2500 0.0253 0.70 0.5 0.0 -2.0
Atom O3 O 0.0437 0.2500 0.4728 0.83 0.5 0.0 -2.0
Atom O4 O 0.3678 0.0566 0.2633 0.77 1.0 0.0 -2.0
! Codes for refinement
Vary xyz 0 1 0 1
!fix x_Fe y_O4
!Equal y_Fe y_P 0.25
HKL-OBS mfe.hkl
MIN-DSPACING 1.5
FST_CMD conn P o 0.0 1.8 ; conn FE o 0.0 2.3
OPTIMIZE dis-restr 1.0 Fobs-Fcal 1.0
!
!Total number of independent distance restraints: 28
DFIX 3.19620 0.00000 Ni Ni_3.545
DFIX 2.90276 0.00000 Ni Fe_1.554
DFIX 2.06756 0.00000 Ni O1_2.455
...
!!!
! Simulated Annealing conditions
SIM_ANN
LOCAL_OPTIMIZATION
!
! Name of the cost function
CostNam General_Cost
!
TemParM T_ini anneal num_temps
        4.0      0.95     80
!
Algor_T Nalgor Nconf nm_cycl num_therm accept
        0         6       150          0      0.5
!
SeedVAL value of seed (if SeedVAL = 0, random seed)
SeedVAL 0
!
Threshold 20.0
!
! Treatment of initial configuration
InitCON RAN

```

output information is showed during execution about the Cost-Function parameter

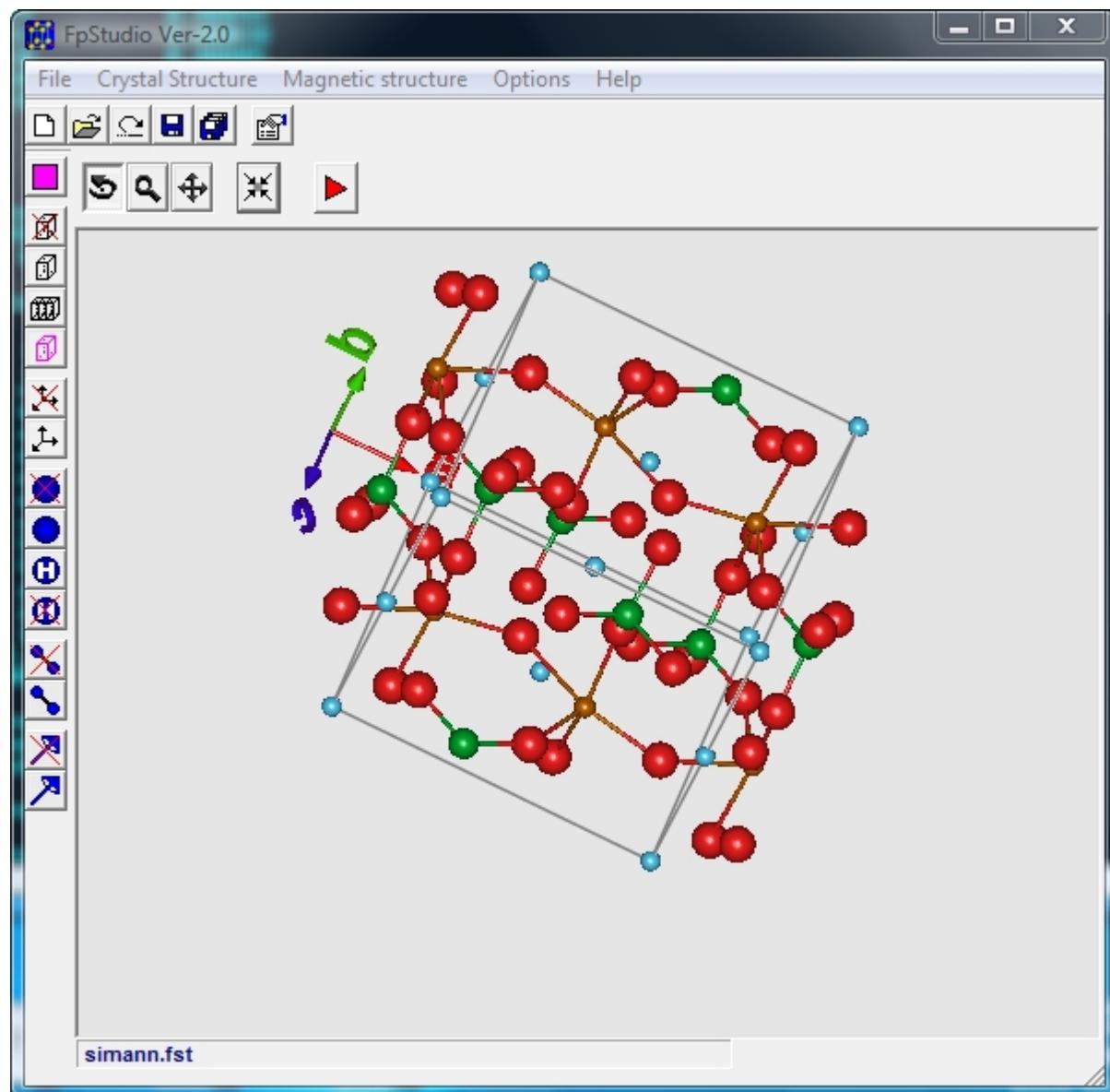
```

Intel(R) Visual Fortran Compiler for applications running on IA-32, Version 10.1.025 build environment - optim_general

Conf: 6 <xAcc>: 3.846 <Step< 6>>; 1.000 <General_Cos>; 0.1937024E+10 -> Current Cost: 6155750.
=> New Temp: 3.00000 NT: 2 Number of function evaluations: 11700
Conf: 1 <xAcc>: 0.564 <Step< 1>>; 0.357 <General_Cos>; 882200.8 -> Current Cost: 655988.7
Conf: 2 <xAcc>: 0.667 <Step< 2>>; 0.360 <General_Cos>; 6856786. -> Current Cost: 3001399.
Conf: 3 <xAcc>: 1.179 <Step< 3>>; 0.358 <General_Cos>; 4065894. -> Current Cost: 1005399.
Conf: 4 <xAcc>: 0.462 <Step< 4>>; 0.357 <General_Cos>; 1496079. -> Current Cost: 1043395.
=> Configuration # 4 converged => dead in the algorithm!
Conf: 5 <xAcc>: 0.872 <Step< 5>>; 0.353 <General_Cos>; 3460475. -> Current Cost: 1688901.
Conf: 6 <xAcc>: 0.923 <Step< 6>>; 0.356 <General_Cos>; 3094112. -> Current Cost: 1584664.
=> New Temp: 3.61000 NT: 3 Number of function evaluations: 23400
Conf: 1 <xAcc>: 0.513 <Step< 1>>; 0.120 <General_Cos>; 380755.3 -> Current Cost: 242374.3
Conf: 2 <xAcc>: 1.949 <Step< 2>>; 0.121 <General_Cos>; 1291904. -> Current Cost: 415390.8
Conf: 3 <xAcc>: 0.821 <Step< 3>>; 0.122 <General_Cos>; 419151.7 -> Current Cost: 286831.5
Conf: 5 <xAcc>: 0.974 <Step< 5>>; 0.120 <General_Cos>; 1093414. -> Current Cost: 532279.5
Conf: 6 <xAcc>: 1.282 <Step< 6>>; 0.121 <General_Cos>; 848189.4 -> Current Cost: 454145.9
=> New Temp: 3.42950 NT: 4 Number of function evaluations: 33150
Conf: 1 <xAcc>: 1.282 <Step< 1>>; 0.040 <General_Cos>; 81746.66 -> Current Cost: 13002.95
Conf: 2 <xAcc>: 1.282 <Step< 2>>; 0.042 <General_Cos>; 179626.6 -> Current Cost: 40077.65
Conf: 3 <xAcc>: 1.128 <Step< 3>>; 0.041 <General_Cos>; 124649.0 -> Current Cost: 55365.27
Conf: 5 <xAcc>: 1.538 <Step< 5>>; 0.040 <General_Cos>; 227700.0 -> Current Cost: 57244.26
Conf: 6 <xAcc>: 2.205 <Step< 6>>; 0.041 <General_Cos>; 160018.6 -> Current Cost: 32484.09
=> New Temp: 3.25802 NT: 5 Number of function evaluations: 42900
Conf: 1 <xAcc>: 0.923 <Step< 1>>; 0.014 <General_Cos>; 3864.500 -> Current Cost: 1445.568
Conf: 2 <xAcc>: 1.744 <Step< 2>>; 0.014 <General_Cos>; 12217.14 -> Current Cost: 776.6317
Conf: 3 <xAcc>: 1.744 <Step< 3>>; 0.014 <General_Cos>; 29900.89 -> Current Cost: 15788.89
Conf: 5 <xAcc>: 1.846 <Step< 5>>; 0.014 <General_Cos>; 20865.14 -> Current Cost: 3503.815
Conf: 6 <xAcc>: 1.231 <Step< 6>>; 0.014 <General_Cos>; 11937.93 -> Current Cost: 4276.629
=> New Temp: 3.09512 NT: 6 Number of function evaluations: 52650
Conf: 1 <xAcc>: 1.077 <Step< 1>>; 0.005 <General_Cos>; 966.3631 -> Current Cost: 591.5047
Conf: 2 <xAcc>: 0.821 <Step< 2>>; 0.005 <General_Cos>; 324.6593 -> Current Cost: 44.75232
Conf: 3 <xAcc>: 2.256 <Step< 3>>; 0.005 <General_Cos>; 3931.831 -> Current Cost: 415.4786
Conf: 5 <xAcc>: 1.538 <Step< 5>>; 0.005 <General_Cos>; 1748.476 -> Current Cost: 647.0912
Conf: 6 <xAcc>: 1.538 <Step< 6>>; 0.005 <General_Cos>; 2078.794 -> Current Cost: 586.0333
=> New Temp: 2.94037 NT: 7 Number of function evaluations: 62400
Conf: 1 <xAcc>: 1.538 <Step< 1>>; 0.002 <General_Cos>; 232.5383 -> Current Cost: 101.6228
Conf: 2 <xAcc>: 1.538 <Step< 2>>; 0.002 <General_Cos>; 26.45060 -> Current Cost: 18.39630
Conf: 3 <xAcc>: 2.051 <Step< 3>>; 0.002 <General_Cos>; 223.2486 -> Current Cost: 126.6048
Conf: 5 <xAcc>: 2.154 <Step< 5>>; 0.002 <General_Cos>; 248.3771 -> Current Cost: 73.92747
Conf: 6 <xAcc>: 1.949 <Step< 6>>; 0.002 <General_Cos>; 138.6501 -> Current Cost: 59.13053
=> New Temp: 2.79335 NT: 8 Number of function evaluations: 72150

```

And also the structure can be investigated at the same time if you have the **FP_Studio** program



Twins

Search_TwinLaws program calculate possible twin laws from special metrics using the method of A. Santoro (Acta Cryst A30, 224 (1974)).

The input data are simply the unit cell parameters and the symbol of the space group, and the keyword *TOL* (for tolerance) followed by its value.

If the value is greater than 1 it is supposed to be given in percentage and a division by 100 is performed. The input file should have the extension
CFL

Building the executable file

Run the batch file (for Windows) or the script (for Linux or MacOS)

```
make_twin <compiler-command>
```

Running the program

The program is invoked at the prompt using the name of the executable file.

Modules of CrysFML

CrysFML library presents different levels of complexity, but this is not important for the end user.

Level 0

Concept	Module Name	Purpose
Constants...	CFML_GlobalDeps	Global parameters on the <i>CrysFML</i> library

Level 1

Concept	Module Name	Purpose
Mathematics...	CFML_FFT CFML_LSQ_TypeDef CFML_Math_General	FFT calculations Definitions of Types for LSQ routines General mathematic utilities for use in Crystallography, Solid State Physics and Chemistry.
	CFML_Random_Generators	Random number generators for different kind of statistical distributions
	CFML_Spherical_Harmonics	Spherical Harmonics routines
Messages...	CFML_IO_Messages	Input/Output messages using <i>CrysFML</i>
Profiles...	CFML_PowderProfiles_CW CFML_PowderProfiles_Finger CFML_PowderProfiles_TOF	Calculation of peak profile functions Routines for calculations of asymmetry due to axial divergence (Finger, Cox and Jephcoat) Contains variables and procedures used by programs aiming to handle T.O.F. powder diffraction patterns
Strings...	CFML_String_Utils	Manipulation of strings with alphanumeric characters

Level 2

Concept	Module Name	Purpose
Chemical Tables...	<u>CFML_Scattering_Chemical_Tables</u>	Tabulated information about atomic chemical and scattering data
Mathematics...	<u>CFML_Math_3D</u>	Simple mathematics general utilities for 3D Systems
Optimization...	<u>CFML_Optimization_General</u>	Module implementing several algorithms for global and local optimization
	<u>CFML_Optimization_LSQ</u>	Module implementing Marquard algorithm for non-linear least-squares
Patterns...	<u>CFML_Diffraction_Patterns</u>	Diffraction Patterns data structures and procedures for reading different powder diffraction formats.
Symmetry Tables...	<u>CFML_Symmetry_Tables</u>	Tabulated information on Crystallographic Symmetry

Level 3

Concept	Module Name	Purpose
Bonds Tables...	<u>CFML_Bond_Tables</u>	Contain a simple subroutine providing the list of the usual bonds between atoms
Crystal Metrics...	<u>CFML_Crystal_Metrics</u>	Define crystallographic types and to provide automatic crystallographic metrics operations
instrumentation on ILL...	<u>CFML_ILL_Instrm_Data</u>	Procedures to access the (single crystals) instrument output data base at ILL

Symmetry information...	CFML_Crystallographic_Symmetry	Contain nearly everything needed for handling symmetry in Crystallography.
-------------------------	--	--

Level 4

Concept	Module Name	Purpose
Atoms...	CFML_Atom_TypeDef	Module defining different data structures concerned with atoms
EoS	CFML_EoS	Equation of State Calculations
Geometry...	CFML_Geometric_SXTAL	Module for geometrical calculations in single crystal instruments
Reflections...	CFML_Reflections_Utils	Procedures handling operation with Bragg reflections

Level 5

Concept	Module Name	Purpose
Geometry...	CFML_Geometry_Calc	Geometry Calculations
Propagation vectors...	CFML_Propagation_Vectors	Procedures handling operations with propagation/modulation vectors
Structure Factors...	CFML_Structure_Factors	Structure Factors Calculations

Level 6

Concept	Module Name	Purpose
Configurations...	CFML_BVS_Energy_Calc	Procedures related to calculations of energy or configuration properties depending on the crystal structure: BVS, Energy,...

Maps...	<u>CFML Maps Calculations</u>	Procedures related to operations on arrays describing maps
Molecular...	<u>CFML Molecular Crystals</u>	Types and procedures related to molecules in crystals

Level 7

Concept	Module Name	Purpose
Formats...	<u>CFML IO Formats</u>	Procedures for handling different formats for input/output

Level 8

Concept	Module Name	Purpose
Refinement...	<u>CFML Keywords Code Parser</u>	Refinable Codes parser
Magnetic Symmetry...	<u>CFML Magnetic Symmetry</u>	Procedures handling operations with Magnetic Symmetry and Magnetic Structures
Simulated Annealing...	<u>CFML Simulated Annealing</u>	Module for Global Optimization using Simulated Annealing

Level 9

Concept	Module Name	Purpose
Magnetic Structure Factors...	<u>CFML Magnetic Structure Factors</u>	Magnetic Structure Factors Calculations
Polarimetry...	<u>CFML Polarimetry</u>	Procedures to calculate the polarization tensor as measured using CRYOPAD

Level 0

Concept	Module Name	Purpose
Constants...	CFML_GlobalDeps	Global parameters on the <i>CrysFML</i> library

CFML_GlobalDeps

Precision parameters for **CrysFML** library and Operating System information.

Numeric parameters

- CP
 - DEps
 - DP
 - Eps
 - SP
 - Pi
 - To_Deg
 - To_Rad
 - TPi

Operative system parameters

- Ops
 - Ops Name
 - Ops Sep

Functions

- Directory Exists

Fortran Filenames

Windows:

CFML_GlobalDeps_Windows.f90
CFML_GlobalDeps_Windows_Intel.f90

(to be used for all Fortran compilers except intel)
(to be used with intel Fortran Compiler)

Linux:

CFML_GlobalDeps_Linux.f90
CFML_GlobalDeps_Linux_Intel.f90

MacOS:

CFML_GlobalDeps_MacOS.f90
CFML_GlobalDeps_MacOS_Intel.f90

Numeric Parameters

- CP

- [DEps](#)
- [DP](#)
- [Eps](#)
- [SP](#)
- [Pi](#)
- [To_Deg](#)
- [To_Rad](#)
- [TPi](#)

CP

Integer, Parameter :: CP

Define the current precision

(**Default:** Simple precision)

DEps

Real (Kind=DP), Parameter :: DEps

Epsilon value parameters in double precision

DP

Integer, Parameter :: DP

Define the double precision for real variables

Eps

Real (Kind=CP), Parameter :: Eps

Epsilon value parameter in current precision

Pi

Real (Kind=DP), Parameter :: Pi

Real parameter containing the value of π in double precision

SP

Integer, Parameter :: SP

Define the simple precision for real variables

To_Deg

Real (Kind=DP), Parameter :: To_Deg

Real parameter containing the conversion factor from Radians to Degrees

To_Rad

Real (Kind=DP), Parameter :: To_Rad

Real parameter containing the conversion factor from Degrees to Radians

TPi

Real (Kind=DP), Parameter :: TPi

Real parameter containing the value of 2π

Operative System Parameters

- [Ops](#)
- [Ops_Name](#)
- [Ops_Sep](#)

Ops

Integer, Parameter :: Ops

Integer parameter that define the operative system that you are using. The values are:

Value	Operative System
1	Windows
2	Linux
3	MacOS

Ops_Name

Character (Len=*), Parameter :: Ops_Name

String containing the name of the Operative System.

Value	Operative System
Windows	Windows
Linux	Linux
MacOS	MacOS

Ops_Sep

Character (Len=*), Parameter :: Ops_Sep

String containing the character to define the directory separator.

Value	Operative System
\	Windows
/	Linux, MacOS

Functions

- [Directory_Exists](#)

[Directory_Exists](#)

Logical Function Directory_Exists (DirName)

Character (Len=*)	Intent(in)	DirName	String containing the name of the directory
-------------------	------------	---------	---

Return `.TRUE.` or `.FALSE.` depending if the directory name in the variable `DirName` exists or not.

Level 1

Concept	Module Name	Purpose
Mathematics...	CFML_FFT	FFT calculations
	CFML_LSQ_TypeDef	Type definitions for LSQ routines
	CFML_Math_General	General mathematic utilities for use in Crystallography, Solid State Physics and Chemistry.
	CFML_Random_Generators	Random number generators for different kind of statistical distributions
	CFML_Spherical_Harmonics	Spherical Harmonics routines
Messages...	CFML_IO_Messages	input / output general messages
Profiles...	CFML_PowderProfiles_CW	Calculation of peak profile functions
	CFML_PowderProfiles_Finger	Routines for calculations of asymmetry due to axial divergence (Finger, Cox and Jephcoat)
	CFML_PowderProfiles_TOF	Contains variables and procedures used by programs aiming to handle T.O.F. powder diffraction patterns
Strings...	CFML_String_Utils	Manipulation of strings with alphanumeric characters

CFML_FFT

Module for multivariate Fast Fourier Transform calculations

Variables

- [Points_Interval_Type](#)

Functions

- [Convol](#)
- [Convol_Peaks](#)
- [F_FFT](#)
- [FFT](#)

Subroutines

- [HFFT](#)
- [SFFT](#)

Fortran Filename

CFML_FFTs.f90

Variables

- [Points_Interval_Type](#)

Points_Interval_Type

	Variable	Definition
Type :: Points_Interval_Type		
Integer	Np	Number of points
Real(Kind=CP)	Low	Lower range value
Real(Kind=CP)	High	Higher range value
End Type Points_Interval_Type		

Functions

- [Convol](#)
- [Convol_Peaks](#)
- [F_FFT](#)
- [FFT](#)

Convol

Real Function Convol (F, PF, G, PG, Interval)

Defined Function F		F	
Real(Kind=CP), Dimension (:)	Intent(in)	PF	Parameters of the function F
Defined Function G		G	
Real(Kind=CP), Dimension (:)	Intent(in)	PG	Parameters of the function G
Type (Points_Interval_Type)	Intent(in)	Interval	Give the number of points and the limits of the interval for calculation.

With

Function F (X, ParF)

Real(Kind=CP)	Intent(in)	X	
Real(Kind=CP), Dimension (:)	Intent(in)	ParF	

End Function F

and

Function G (X, ParG)

Real(Kind=CP)	Intent(in)	X	
Real(Kind=CP), Dimension (:)	Intent(in)	ParG	

End Function G

Return a real vector of dimension **interval%NP** with the convolution of the user-provided centered at x=0 of peak functions **F** and **G**. The convolution function is normalized to unit area.

Example:

`h = convol(Pseudo_Voigt, P_PV, Hat, P_Hat, My_interval)`

generates my_interval%np values `h(i)`, $i=1, \dots, \text{my_interval}%\text{np}$ corresponding to the convolution of a pseudo-Voigt function with a hat function

[Convol_Peaks](#)

Real Function Convol_Peaks (F, PF, G, PG, WD, NP)

Defined Function F		F	
--------------------	--	---	--

Real(Kind=CP), Dimension (:)	Intent(in)	PF	Parameters of the function F (starting with FWHM)
Defined Function G		G	
Real(Kind=CP), Dimension (:)	Intent(in)	PG	Parameters of the function G (starting with FWHM)
Real(Kind=CP)	Intent(in)	WD	Number of times a FWHM of the f-function to calculate range
Integer	Intent(in)	NP	Number of points (it is increased internally up to the closest power of 2)

With

Function F (X, ParF)

Real(Kind=CP)	Intent(in)	X	
Real(Kind=CP), Dimension (:)	Intent(in)	ParF	

End Function F

and

Function G (X, ParG)

Real(Kind=CP)	Intent(in)	X	
Real(Kind=CP), Dimension (:)	Intent(in)	ParG	

End Function G

Return a real vector of dimension **NP** with the convolution of the user-provided centered at x=0 of peak functions **F** and **G**. The convolution function is normalized to unit area.

The definition interval [a,b] of the peaks is calculated as:

$$\begin{aligned} a &= -b \\ b &= WD * \text{FWHM} = WD * PF(1) \end{aligned}$$

Example:

h = convol_peaks(Pseudo_Voigt, P_PV, Hat, P_Hat, 15.0, 150)

generates 150 values h(i), i=1,150 corresponding to the convolution of a pseudo-Voigt function with a hat function

F_FFT

Complex Function F_FFT (Array, Mode)

Complex, Dimension (:)	Intent(in)	Array	Complex vector containing real parts of transform
Character (Len=*) , Optional	Intent(in)	Mode	= inv backward transform. ≠ inv forward transform for the rest

This function is similar to subroutine [SFFT](#) and it is useful only when one want to retain the original array. It is a slight modification of a complex split radix FFT routine presented by C.S. Burrus.

NOTE: There is no control of the error consisting in giving a dimension that is not a power of two. It is the

responsibility of the user to provide a complex array of dimension equal to a power of 2.

Example:

`FX = F_FFT(X)`

`Y = F_FFT(FY,"inv")`

FFT

Complex Function FFT (Array, Dim, Inv)

Complex, Dimension (:) or Complex, Dimension (:,:) or Complex, Dimension (:,:,:) or ... or Complex, Dimension (:,:,:,:,:,:)	Intent (in)	Array	Complex array
Integer, Dimension (:), Optional	Intent (in)	Dim	array containing the dimensions to be transformed
Logical, Optional	Intent (in)	Inv	= .False. Forward transformation (Default) = .True. inverse transformation will be performed.

Multivariate Fast Fourier Transform (from 1 to up 7 dimensions). It is an implementation of Singleton's mixed-radix algorithm, RC Singleton, Stanford Research institute, Sept. 1968.

NOTE: Transformation results will always be scaled by the square root of the product of sizes of each dimension in Dim.

Example:

Let A be a L*M*N three dimensional complex array. Then result = fft(A) will produce a three dimensional transform, scaled by $\sqrt{L \cdot M \cdot N}$.

`result = fft(A, dim=(/1,3/))` will transform with respect to the first and the third dimension, scaled by $\sqrt{L \cdot M \cdot N}$.
`result = fft(result, inv=.true.)` should (approximately) reproduce A.

Subroutines

- [HFFT](#)
- [SFFT](#)

HFFT

Subroutine HFFT (Array, IfSet, IfErr)

Complex, Dimension (:)	Intent(in out)	Array	Contains the complex 3D array to be transformed
Integer	Intent(in)	IfSet	= 1 or 2 inverse Fourier Transform

			=-1 or -2 Fourier Transform
Integer	Intent(out)	IfErr	Flags to error. 0 for no error.

Performs Discrete Complex Fourier Transforms on a complex three dimensional array. This subroutine is to be used for complex, 3-dimensional arrays in which each dimension is a power of 2. The maximum m(i) must not be less than 3 or greater than 20,

For **IfSet** = -1, or -2, the Fourier transform of complex array a is obtained.

$$X(J1, J2, J3) = \sum_{K1=0}^{N1-1} \sum_{K2=0}^{N2-1} \sum_{K3=0}^{N3-1} A(K1, K2, K3) W1^{L1} W2^{L2} W3^{L3}$$

where Wi is the n(i) root of unit and L1=K1*J1, L2=K2*J2, L3=K3*J3.

For **IfSet** = +1, or +2, the inverse Fourier transform a of complex array x is obtained.

$$A(K1, K2, K3) = \frac{1}{N1 N2 N3} \sum_{J1=0}^{N1-1} \sum_{J2=0}^{N2-1} \sum_{J3=0}^{N3-1} X(J1, J2, J3) W1^{-L1} W2^{-L2} W3^{-L3}$$

SFFT

Subroutine SFFT (Array, Mode, IfErr)

Complex, Dimension (:)	Intent(in out)	Array	Real array containing real parts of transform
Character (Len=*), Optional	Intent(in)	Mode	= inv backward transform. ≠ inv forward transform for the rest
Integer, Optional	Intent(out)	IfErr	Flags to error. 0 for no error

The forward transform computes:

$$X(k) = \sum_{j=0}^{N-1} x(j) \cdot e^{(-i2jk\pi/N)}$$

The backward transform computes

$$x(j) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) \cdot e^{(2ijk\pi/N)}$$

CFML_IO_Messages

Input/Output general messages for **CrysFML** library

It is convenient to use these intermediate procedures instead of fortran **write(*,*)** or **print***, because it is much more simple to modify a program for making a GUI. Usually GUI tools and libraries need special calls to dialog boxes for screen messages.

Variables

- [Win_Console](#)

Subroutines

- [Close_Scroll_Window](#)
- [Error_Message](#)
- [info_Message](#)
- [Print_Message](#)
- [Question_Message](#)
- [Stop_Message](#)
- [Wait_Message](#)
- [Warning_Message](#)
- [Write_Scroll_Text](#)

Fortran Filenames

Console:

CFML_IO_Mess.f90

Realwin:

CFML_IO_MessRW.f90

Winteracter:

CFML_IO_MessWin.f90

Variables

- [Win_Console](#)

Win_Console

Integer :: Win_Console

Integer value that identify a scroll window when **Winteracter** library is used.

NOTE: Only available using **Winteracter** library

Subroutines

- [Close_Scroll_Window](#)
- [Error_Message](#)
- [info_Message](#)
- [Print_Message](#)
- [Question_Message](#)
- [Stop_Message](#)
- [Wait_Message](#)
- [Warning_Message](#)

- [Write_Scroll_Text](#)

[Close_Scroll_Window](#)

Subroutine Close_Scroll_Window()

Close the scroll window if still open.

NOTE: Only available using *Winteracter* library

[Error_Message](#)

Subroutine Error_Message(Mess, lunit, Routine, Fatal)

Character (Len=*)	Intent(in)	Mess	String containing the error information
Integer, Optional	Intent(in)	lunit	Write information on unit=lunit
Character (Len=*) , Optional	Intent(in)	Routine	Name of the subroutine where occurs the error
Logical, Optional	Intent(in)	Fatal	Flag to stop the program or not

Print an error message on the screen and/or in unit defined by *lunit* variable if it is present

[Info_Message](#)

Subroutine Info_Message(Mess, lunit, Scroll_Window)

Character (Len=*)	Intent(in)	Mess	String Info message
Integer, Optional	Intent(in)	lunit	Write information on unit=lunit
Integer, Optional	Intent(in)	Scroll_Window	Write information on Scroll Window

Print an Information message on the screen and/or in the unit defined by *lunit* if present.

NOTE: The last option is only available using *RealWin* library

[Print_Message](#)

Subroutine Print_Message(Mess)

Character (Len=*)	Intent(in)	Mess	Print information
--------------------------	-------------------	------	-------------------

Print an message on the screen.

NOTE: Only available for *Console* version

[Question_Message](#)

Subroutine Question_Message(Mess, Title)

Character (Len=*)	Intent(in)	Mess	string containing the message
Character (Len=*) , Optional	Intent(in)	Title	Title on Dialog

Show a question dialog on the screen

NOTE: Only available using **Winteracter** library

Stop_Message

Subroutine Stop_Message(Mess, Title)

Character (Len=*)	Intent(in)	Mess	string containing the message
Character (Len=*) , Optional	Intent(in)	Title	Title on Dialog

Show a stop dialog on the screen

NOTE: Only available using **Winteracter** library

Wait_Message

Subroutine Wait_Message(Mess)

Character (Len=*)	Intent(in)	Mess	String a message and the wait an answer or action by the user
--------------------------	-------------------	------	---

Similar to Pause for Console version

NOTE: Only available for **Console** version

Warning_Message

Subroutine Warning_Message(Mess, lunit)

Character (Len=*)	Intent(in)	Mess	String containing the message
Integer, Optional	Intent(in)	lunit	Write information on unit=lunit

Show a warning dialog on the screen

NOTE: Only available using **Winteracter** library

Write_Scroll_Text

Subroutine Write_Scroll_Text(Mess, ICmd)

Character (Len=*)	Intent(in)	Mess	String to print
Integer, Optional	Intent(in)	ICmd	Define the type of the Editor Window opened = 0 Editor with command line = 1 Editor without command line

Print the string in a actual scroll window /default terminal/Editor window. The procedure will open a scroll window or editor if it wasn't opened before.

NOTE: The last option is only available using **Winteracter** library

CFML_LSQ_TypeDef

Definitions for LSQ routines into **CrysFML** library

Parameters

- [Max_Free_Par](#)

Variables

- [LSQ_Conditions_Type](#)
- [LSQ_Data_Type](#)
- [LSQ_State_Vector_Type](#)

Fortran Filename

CFML_LSQ_TypeDef.f90

Parameters

- [Max_Free_Par](#)

Max_Free_Par

Integer, Parameter :: Max_Free_Par=809

Maximum number of free parameters for use on [CFML_Optimization_LSQ](#)

Variables

- [LSQ_Conditions_Type](#)
- [LSQ_State_Vector_Type](#)
- [LSQ_Data_Type](#)

LSQ_Conditions_Type

	Variable	Definition
Type :: LSQ_Conditions_Type		
Logical	Constr	if true box constraint of PERCENT are applied to parameters

Logical	Reached	if true convergence was reached in the algorithm
Integer	CorrMax	Value of correlation in % to output
Integer	NFEv	Number of function evaluations (output component, useful for assessing LM algorithm)
Integer	NJEv	Number of Jacobian evaluations
Integer	ICyc	Number of cycles of refinement
Integer	NPVar	Number of effective free parameters of the model
Integer	Iw	indicator for weighting scheme = 1 w=1/yc
Real(Kind=CP)	Tol	Tolerance value for applying stopping criterion in LM algorithm
Real(Kind=CP)	Percent	%value of maximum variation of a parameter w.r.t. the intial value before fixing it
End Type		
LSQ_Conditions_Type		

Definition of a derived type encapsulating all necessary to establish the conditions for running the LSQ algorithm

LSQ_State_Vector_Type

	Variable	Definition
Type :: LSQ_State_Vector_Type		
Integer	NP	Total number of model parameters <= Max_Free_Par
Real(Kind=CP), Dimension(Max_Free_Par)	PV	Vector of parameters
Real(Kind=CP), Dimension(Max_Free_Par)	SPV	Vector of standard deviations
Real(Kind=CP), Dimension(Max_Free_Par)	DPV	Vector of derivatives at a particular point
Integer, Dimension(Max_Free_Par)	Code	Pointer for selecting variable parameters
Character (Len=40), Dimension (Max_Free_Par)	NamPar	Names of parameters
End Type LSQ_State_Vector_Type		

Definition of derived type encapsulating the vector state defining a set of parameter for calculating the model function and running the LSQ algorithm.

LSQ_Data_Type

	Variable	Definition
Type :: LSQ_Data_Type		
Integer	NObs	Total number of observations
Integer	Iw	indicator for type of values contained in component SW
Real(Kind=CP), Dimension(:,), Allocatable	X	Vector containing a relevant quantity for each observation (x-coordinate ...)

Real(Kind=CP), Dimension(:), Allocatable	Y	Vector containing the observed values
Real(Kind=CP), Dimension(:), Allocatable	Sw	if lw=0 Vector containing the standard deviation of observations if lw=1 Weight factors for least squares refinement
Real(Kind=CP), Dimension(:), Allocatable	Yc	Vector containing the calculated values
End Type LSQ_Data_Type		

Defining a derived type encapsulating the observed and calculated data as well as the weighting factors. It is responsibility of the calling program to allocate the components before calling the Marquardt_Fit subroutine.

CFML_Math_General

General utilities of mathematics for use in Crystallography, Solid State Physics and Chemistry.

Variables

- [Err_MathGen](#)
- [Err_MathGen_Mess](#)

Functions

Arrays Functions

- [Co_Linear](#)
- [Co_Prime](#)
- [Equal_Matrix](#)
- [Equal_Vector](#)
- [Euclidean_Norm](#)
- [IMaxLoc](#)
- [IMinLoc](#)
- [Locate](#)
- [Modulo_Lat](#)
- [Norm](#)
- [OuterProd](#)
- [Scalar](#)
- [Trace](#)
- [ZBelong](#)

Scalar Functions

- [Factorial](#)
- [Negligible](#)
- [PGCD](#)
- [PPCM](#)
- [Pythag](#)

Special Functions

- [BessJ0](#)
- [BessJ1](#)

- [BessJ](#)

Trigonometric Functions

- [AcosD](#)
- [AsinD](#)
- [Atan2D](#)
- [AtanD](#)
- [CosD](#)
- [SinD](#)
- [TanD](#)

Subroutines

- [Co_Prime_Vector](#)
- [Determinant](#)
- [Diagonalize_SH](#)
- [First_Derivative](#)
- [In_Sort](#)
- [Init_Err_MathGen](#)
- [Invert_Matrix](#)
- [Linear_Dependent](#)
- [LU_Backsub](#)
- [LU_Decompo](#)
- [MatInv](#)
- [Points_in_Line2D](#)
- [Rank](#)
- [RTan](#)
- [Second_Derivative](#)
- [Set_EPSG](#)
- [Set_EPSG_Default](#)
- [SmoothingVec](#)
- [Sort](#)
- [Sort.Strings](#)
- [Spline](#)
- [Splint](#)
- [SVDCMP](#)
- [Swap](#)

Fortran Filename

CFML_Math_Gen.f90

Variables

- [Err_MathGen](#)

- [Err_MathGen_Mess](#)

Err_MathGen

Logical :: Err_MathGen

This variable is set to .TRUE. if an error occurs in procedures belonging to this module

Err_MathGen_Mess

Character (Len=150) :: Err_MathGen_Mess

Variable containing information about the last error occurred in the procedures belonging to this module

Functions

Arrays Functions

- [Co_Linear](#)
- [Co_Prime](#)
- [Equal_Matrix](#)
- [Equal_Vector](#)
- [Euclidean_Norm](#)
- [IMaxLoc](#)
- [IMinLoc](#)
- [Locate](#)
- [Modulo_Lat](#)
- [Norm](#)
- [OuterProd](#)
- [Scalar](#)
- [Trace](#)
- [ZBelong](#)

Scalar Functions

- [Factorial](#)
- [Negligible](#)
- [PGCD](#)
- [PPCM](#)
- [Pythag](#)

Special Functions

- [BessJ0](#)
- [BessJ1](#)
- [BessJ](#)

Trigonometric Functions

- [AcosD](#)
- [AsinD](#)
- [Atan2D](#)
- [AtanD](#)
- [CosD](#)
- [SinD](#)

- [TanD](#)

AcosD

Real Function AcosD (X)

Real(Kind=SP / DP)	Intent(in)	X	Value
---------------------------	-------------------	---	-------

Elemental function that gives the inverse of cosine in degrees

AsinD

Real Function AsinD (X)

Real(Kind=SP / DP)	Intent(in)	X	Value
---------------------------	-------------------	---	-------

Elemental function that gives the inverse of sine in degrees

Atan2D

Real Function Atan2D (Y, X)

Real(Kind=SP / DP)	Intent(in)	Y	Value
Real(Kind=SP / DP)	Intent(in)	X	Value

Elemental function that gives the arctangent of y/x in degrees

AtanD

Real Function AtanD (X)

Real(Kind=SP / DP)	Intent(in)	X	Value
---------------------------	-------------------	---	-------

Elemental function that gives the arctangent in degrees

BessJ0

Real Function BessJ0 (X)

Real(Kind=CP)	Intent(in)	X	Value
----------------------	-------------------	---	-------

Elemental function that gives the value of the Bessel function J0(x)

BessJ1

Real Function BessJ1 (X)

Real(Kind=CP)	Intent(in)	X	Value
----------------------	-------------------	---	-------

Elemental function that gives the value of the Bessel function J1(x)

BessJ

Real Function BessJ (N, X)

Integer	Intent(in)	N	Order N of the Bessel function
Real(Kind=CP)	Intent(in)	X	Value

Returns the value of the Bessel function $J_n(x)$ for any real x and $n \geq 2$

Co_Linear

Logical Function Co_Linear (A, B, N)

Complex, Dimension(:)	Intent(in)	A	Vector of dimension N
Complex, Dimension(:)	Intent(in)	B	Vector of dimension N
Integer	Intent(in)	N	Dimension of A and B vectors

or

Integer, Dimension(:)	Intent(in)	A	Vector of dimension N
Integer, Dimension(:)	Intent(in)	B	Vector of dimension N
Integer	Intent(in)	N	Dimension of A and B vectors

or

Real(Kind=CP), Dimension(:)	Intent(in)	A	Vector of dimension N
Real(Kind=CP), Dimension(:)	Intent(in)	B	Vector of dimension N
Integer	Intent(in)	N	Dimension of A and B vectors

Logical function that returns the .TRUE. value if the vectors **A** and **B** are co-linear. That's means that vectors have the same direction.

Co_Prime

Logical Function Co_Prime(V, IMax)

Integer, Dimension(:)	Intent(in)	V	Vector of Numbers
Integer	Intent(in)	IMax	Maximun prime number to be tested

Provides the value .TRUE. if the vector **V** contains co-primes integers: there is no common divisor for all the Integers.

Only the first 1000 prime numbers are tested. The value of **IMAX** the the maximum prime number to be tested (**IMAX** <=7919)

CosD

Real Function CosD (X)

Real(Kind=SP / DP)	Intent(in)	X	Value
--------------------	------------	---	-------

Elemental function that gives the cosine value when the argument is provided in degrees

Equal_Matrix

Logical Function Equal_Matrix (A, B, N)

Integer, Dimension(:, :)	Intent(in)	A	Array of dimension N x N
Integer, Dimension(:, :)	Intent(in)	B	Array of dimension N x N
Integer	Intent(in)	N	Dimension of A and B arrays

or

Real(Kind=CP), Dimension(:, :)	Intent(in)	A	Array of dimension N x N
Real(Kind=CP), Dimension(:, :)	Intent(in)	B	Array of dimension N x N
Integer	Intent(in)	N	Dimension of A and B arrays

Logical function that returns the .TRUE. value if the array **A** is equal to array **B**

Equal_Vector

Logical Function Equal_Vector (A, B, N)

Integer, Dimension(:)	Intent(in)	A	Vector of dimension N
Integer, Dimension(:)	Intent(in)	B	Vector of dimension N
Integer	Intent(in)	N	Dimension of A and B vectors

or

Real(Kind=CP), Dimension(:)	Intent(in)	A	Vector of dimension N
Real(Kind=CP), Dimension(:)	Intent(in)	B	Vector of dimension N
Integer	Intent(in)	N	Dimension of A and B vectors

Logical function that returns the .TRUE. value if the array **A** is equal to array **B**

Euclidean_Norm

Real Function Euclidean_Norm (N, V)

Integer	Intent(in)	N	Dimension of vector X
Real(Kind=CP), Dimension(:)	Intent(in)	V	Vector

This function calculates safely the Euclidean norm of a vector:

$$\|v\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$$

Factorial

Integer Function Factorial (N)

Integer	Intent(in)	N	Value
----------------	-------------------	---	-------

Returns the factorial of the number N

IMaxLoc

Integer / Real Function IMaxLoc (A)

Integer, Dimension(:)	Intent(in)	A	Vector of dimension N
------------------------------	-------------------	---	-----------------------

or

Real(Kind=CP), Dimension(:)	Intent(in)	A	Vector of dimension N
------------------------------------	-------------------	---	-----------------------

Return the index indicating the position of the maximum value of a vector

IMinLoc

Integer / Real Function IMinLOC (A)

Integer, Dimension(:)	Intent(in)	A	Vector of dimension N
------------------------------	-------------------	---	-----------------------

or

Real(Kind=CP), Dimension(:)	Intent(in)	A	Vector of dimension N
------------------------------------	-------------------	---	-----------------------

Return the index indicating the position of the minimum value of a vector

Locate

Integer Function Locate (XX, N, X)

Integer, Dimension(:)	Intent(in)	XX	Vector of dimension N
Integer	Intent(in)	N	Dimension of XX
Integer	Intent(in)	X	Value

or

Real(Kind=CP), Dimension(:)	Intent(in)	XX	Vector of dimension N
Integer	Intent(in)	N	Dimension of XX
Real(Kind=CP)	Intent(in)	X	Value

Function for locating the index j of an array **XX(N)** satisfying that **XX(J) <= X < XX(J+1)**

Modulo_Lat

Integer / Real Function Modulo_Lat (U)

Integer, Dimension(:)	Intent(in)	U	Vector of free dimension
------------------------------	-------------------	---	--------------------------

or

Real(Kind=CP), Dimension(:)	Intent(in)	U	Vector of free dimension
------------------------------------	-------------------	---	--------------------------

Return an integer or real vector with components in the interval [0,1)

Norm

Integer / Real Function Norm (X, G)

Integer, Dimension(:)	Intent(in)	X	Vector
------------------------------	-------------------	---	--------

Real(Kind=CP), Dimension(:,:)	Intent(in)	G	Metric array
--------------------------------------	-------------------	---	--------------

or

Real(Kind=CP), Dimension(:)	Intent(in)	X	Vector
Real(Kind=CP), Dimension(:,:)	Intent(in)	G	Metric array

Calculate the norm of a vector using a particular metric

Negligible

Logical Function Negligible (V)

Complex	Intent(in)	V	Value
----------------	-------------------	---	-------

or

Real(Kind=CP)	Intent(in)	V	Value
----------------------	-------------------	---	-------

Elemental function that provides the value .TRUE. if the complex / real number V is less than **EPS** value

OuterProd

Real Function OuterProd (A, B)

Real(Kind=SP / CP), Dimension(:)	Intent(in)	A	Vector of free dimension
Real(Kind=SP / CP), Dimension(:)	Intent(in)	B	Vector of free dimension

Function that return an array **C** (size(**A**) x size(**B**)) containing the components of the tensorial product of two vectors

PGCD

Integer Function PGCD (I, J)

Integer	Intent(in)	I	Value
Integer	Intent(in)	J	Value

Return the maximum common divisor of two integers

PPCM

Integer Function PPCM (I, J)

Integer	Intent(in)	I	Value
Integer	Intent(in)	J	Value

Return the minimum common multiple of two integers

Pythag

Real Function Pythag (A,B)

<code>Real(Kind=SP / DP)</code>	<code>Intent(in)</code>	A	Value
<code>Real(Kind=SP / DP)</code>	<code>Intent(in)</code>	B	Value

Computes the square root of ($A^2 + B^2$) without destructive underflow or overflow

Scalar

Integer / Real Function Scalar (X, Y, G)

<code>Integer, Dimension(:)</code>	<code>Intent(in)</code>	X	Vector
<code>Integer, Dimension(:)</code>	<code>Intent(in)</code>	Y	Vector
<code>Real(Kind=CP), Dimension(:, :)</code>	<code>Intent(in)</code>	G	Metric array

or

<code>Real(Kind=CP), Dimension(:)</code>	<code>Intent(in)</code>	X	Vector
<code>Real(Kind=CP), Dimension(:)</code>	<code>Intent(in)</code>	Y	Vector
<code>Real(Kind=CP), Dimension(:, :)</code>	<code>Intent(in)</code>	G	Metric array

Return the scalar product of two vectors including metrics

SinD

Real Function SinD (X)

<code>Real(Kind=SP / DP)</code>	<code>Intent(in)</code>	X	Value
---------------------------------	-------------------------	---	-------

Elemental function that gives the sine value when the argument is provided in degrees

TanD

Real Function TanD (X)

<code>Real(Kind=SP / DP)</code>	<code>Intent(in)</code>	X	Value
---------------------------------	-------------------------	---	-------

Elemental function that give the tangent value when the argument is provided in degrees

Trace

Complex / Integer / Real Function Trace (A)

<code>Complex, Dimension(:, :, :)</code>	<code>Intent(in)</code>	A	Array of dimension N x N
--	-------------------------	---	--------------------------

or

<code>Integer, Dimension(:, :, :)</code>	<code>Intent(in)</code>	A	Array of dimension N x N
--	-------------------------	---	--------------------------

or

<code>Real(Kind=CP), Dimension(:, :, :)</code>	<code>Intent(in)</code>	A	Array of dimension N x N
--	-------------------------	---	--------------------------

Function that provides the trace of a complex/integer or real matrix

ZBelong

Logical Function Zbelong (V)

Real(Kind=CP), Dimension(:,::)	Intent(in)	V	Array of dimension N x N
---------------------------------------	-------------------	----------	--------------------------

or

Real(Kind=CP), Dimension(:)	Intent(in)	V	Vector of dimension N
------------------------------------	-------------------	----------	-----------------------

or

Real(Kind=CP)	Intent(in)	V	Value
----------------------	-------------------	----------	-------

Logical function that provides the value **.TRUE.** if a real number, vector or array **V** is close enough (within **EPS**) to an integer.

Subroutines

- [Co_Prime_Vector](#)
- [Determinant](#)
- [Diagonalize_SH](#)
- [First_Derivative](#)
- [In_Sort](#)
- [Init_Err_MathGen](#)
- [Invert_Matrix](#)
- [Linear_Dependent](#)
- [LU_Backsub](#)
- [LU_Decompo](#)
- [MatInv](#)
- [Points_in_Line2D](#)
- [Rank](#)
- [RTan](#)
- [Second_Derivative](#)
- [Set_EPSG](#)
- [Set_EPSG_Default](#)
- [SmoothingVec](#)
- [Sort](#)
- [Sort.Strings](#)
- [Spline](#)
- [Splint](#)
- [SVDCMP](#)
- [Swap](#)

[Co_Prime_Vector](#)**Subroutine Co_Prime_Vector (V, Cop, F)**

Integer, Dimension (:)	Intent(in)	V	Input integer vector
-------------------------------	-------------------	----------	----------------------

Integer, Dimension (:)	Intent(out)	Cop	Co-prime output vector
Integer, Optional	Intent(out)	F	Common multiplicative factor

Calculates the co-prime vector (Cop) parallel to the input vector (V)

Determinant

Subroutine Determinant (A, N, Determ)

Complex, Dimension (:,:)	Intent(in)	A	Array (N x N)
Integer	Intent(in)	N	Dimension for Square Matrix
Real(Kind=CP)	Intent(out)	Determ	$\text{Det}(\text{Re}[A^2]) + \text{Det}(\text{Im}[A^2])$

or

Real(Kind=CP), Dimension (:,:)	Intent(in)	A	Array (N x N)
Integer	Intent(in)	N	Dimension for Square Matrix
Real(Kind=CP)	Intent(out)	Determ	Determinant of A

Subroutine that calculates the determinant of a real or integer square matrix and a pseudo-determinant for the complex square matrix.

NOTE: The calculated value is only useful for linear dependency purposes. It tell us if the complex matrix is singular or not.

Diagonalize_SH

Subroutine Diagonalize_SH (A, N, E_Val, E_Vect)

Complex, Dimension (:,:)	Intent(in)	A	Array (N x N)
Integer	Intent(in)	N	Dimension for Square Matrix
Real(Kind=CP), Dimension (:)	Intent(out)	E_Val	Eigen values sorted in descending order
Complex, Dimension (:,:), Optional	Intent(out)	E_Vect	Eigenvectors

or

Real(Kind=CP), Dimension (:,:)	Intent(in)	A	Array (N x N)
Integer	Intent(in)	N	Dimension for Square Matrix
Real(Kind=CP), Dimension (:)	Intent(out)	E_Val	Eigen values sorted in descending order
Real(Kind=CP), Dimension (:,:), Optional	Intent(out)	E_Vect	Eigenvectors

Procedure for diagonalizes Symmetric/Hermitian matrices.

First_Derivative

Subroutine First_Derivative (X, Y, N, D2Y, D1Y)

Real(Kind=CP), Dimension (:)	Intent(in)	X	Vector of N points
Real(Kind=CP), Dimension (:)	Intent(in)	Y	$Y_i = F(X)$

Integer	Intent(in)	N	Dimension of vectors X, Y
Real(Kind=CP), Dimension (:)	Intent(in)	D2Y	Vector containing second derivatives at the given points
Real(Kind=CP), Dimension (:)	Intent(out)	D1Y	Vector containing first derivatives at the given points

Subroutine that calculates the first derivate values of a vector of N points

In_Sort

Subroutine In_Sort(ID, N, P, Q)

Integer, Dimension (:)	Intent(in)	ID	Vector to be sorted
Integer	Intent(in)	N	Number items in the vector
Integer, Dimension (:)	Intent(in)	P	initial pointer from a previous related call
Integer, Dimension (:)	Intent(out)	Q	Final pointer doing the sort of id

Subroutine to order in ascending mode the integer vector ID.

Init_Err_MathGen

Subroutine Init_Err_MathGen ()

Subroutine that initializes errors flags in **CFML_Math_General** module.

Invert_Matrix

Subroutine Invert_Matrix (A, B, Singular, Perm)

Real(Kind=CP), Dimension (:,:)	Intent(in)	A	input Array
Real(Kind=CP), Dimension (:,:)	Intent(in)	B	output array containing A^{-1}
Logical	Intent(out)	Singular	.TRUE. is the input array is singular
Integer, Dimension (:), Optional	Intent(out)	Perm	Hold the row permutation performed during procedure

Subroutine to invert a real matrix using LU decomposition.

Linear_Dependent

Subroutine Linear_Dependent (A, NA, B, NB, MB, Info)

Complex, Dimension (:)	Intent(in)	A	input Vector
Integer	Intent(in)	NA	Dimension of A
Complex, Dimension (:,:)	Intent(in)	B	input Array B(NB,MB)
Integer	Intent(in)	NB	Number of rows of B
Integer	Intent(in)	MB	Number of columns of B
Integer, Dimension (:), Optional	Intent(out)	Info	.TRUE. is A is linear dependent

or

Integer, Dimension (:)	Intent(in)	A	input Vector
Integer	Intent(in)	NA	Dimension of A
Integer, Dimension (:,:)	Intent(in)	B	input Array B(NB,MB)
Integer	Intent(in)	NB	Number of rows of B
Integer	Intent(in)	MB	Number of columns of B
Integer, Dimension (:), Optional	Intent(out)	Info	.TRUE. is A is linear dependent

or

Real(Kind=CP), Dimension (:)	Intent(in)	A	input Vector
Integer	Intent(in)	NA	Dimension of A
Real(Kind=CP), Dimension (:,:)	Intent(in)	B	input Array B(NB,MB)
Integer	Intent(in)	NB	Number of rows of B
Integer	Intent(in)	MB	Number of columns of B
Integer, Dimension (:), Optional	Intent(out)	Info	.TRUE. is A is linear dependent

This subroutine provides a .TRUE. value if the vector **A** is linear dependent of the vectors constituting the rows (columns) of the matrix **B**.

The problem is equivalent to determine the rank (in algebraic sense) of the composite matrix $C(NB+1, MB) = (B|A)$ or $C(NB, MB+1) = (B|A)$. In the first case it is supposed that $NA = MB$ and in the second $NA = NB$ and the rank of B is $\min(NB, MB)$.

If $NA \neq NB$ and $NA \neq MB$ an error condition is generated. The function uses floating arithmetic for all types.

NOTE: The actual dimension of vector A should be $NA = \max(NB, MB)$.

LU_Backsub

Subroutine LU_Backsub (A, Indx, B)

Real(Kind=CP), Dimension (:,:)	Intent(in)	A	input Array
Integer, Dimension (:)	Intent(in)	Indx	Permutation vector returned by LU_Decompo
Real(Kind=CP), Dimension (:)	Intent(in out)	B	in: Right-hand-side vector out: Solutions of the linear system

Subroutine that solves the set of N linear equations $\mathbf{A} \cdot \mathbf{X} = \mathbf{B}$

A and Indx are not modified by this routine and can be left in place for successive calls with different right-hand sides B

NOTE: Here the matrix A (N,N) is not as the original matrix A, but rather as its LU decomposition, determined by the routine [LU_Decompo](#).

LU_Decompo

Subroutine LU_Decompo (A, D, Singular, Indx)

Real(Kind=CP), Dimension (:,:)	Intent(in out)	A	in: input Array out: Matrix U in its upper triangular part (plus diagonal) and in the lower triangular part
---------------------------------------	---------------------------	---	--

				contains the nontrivial part of matrix L.
Real(Kind=CP)	Intent(out)	D		D is output as +/-1 depending on whether the number of row interchanges was even or odd, respectively
Logical	Intent(out)	Singular		.TRUE. if A is singular
Integer, Dimension (:), Optional	Intent(out)	Indx		Permutation vector

Subroutine to make the LU decomposition of an input matrix **A**.

MatInv

Subroutine MatInv (A, N)

Real(Kind=CP), Dimension (:, :)	Intent(in out)	A	in: out:	input Array A^{-1}
Integer	Intent(in)	N		Dimension of A

Subroutine for inverting a real square matrix. The input matrix is replaced in output with its inverse

Points_in_Line2D

Subroutine Points_in_Line2D(X1, XN, N, XP)

Real(Kind=CP), Dimension (2)	Intent(in)	X1	Point 1 in 2D
Real(Kind=CP), Dimension (2)	Intent(in)	XN	Point N in 2D
Integer	Intent(in)	N	Number of Total points
Real(Kind=CP), Dimension (:)	Intent(out)	XP	Vector of N points

Calculate N points belonging to the line defined by X1 and XN with equal distance between them. XP is a vector containing X_1, X_2, \dots, X_N points.

Rank

Subroutine Rank (A, Tol, R)

Real (Kind=DP /SP), Dimension (:, :)	Intent(in)	A	input Array
Real (Kind=DP /SP)	Intent(in)	Tol	Tolerance value
Integer	Intent(out)	R	Rank of A

Compute the rank (in algebraic sense) of the rectangular matrix **A**.

RTan

Subroutine RTan (Y, X, Ang, Deg)

Real(Kind=DP / SP)	Intent(in)	Y	Value
Real(Kind=DP / SP)	Intent(in)	X	Value
Real(Kind=DP / SP)	Intent(out)	Ang	Value
Character (Len=*) , Optional	Intent(in)	Deg	Value

Returns the arctangent (Y/X), in the argument **Ang**, in the quadrant where the signs $\sin(Ang)$ and $\cos(Ang)$ are those of **Y** and **X**.

If **Deg** is present, then **Ang** is provided in degrees

Second_Derivative

Subroutine Second_Derivative (X, Y, N, D2Y)

Real(Kind=CP), Dimension (:)	Intent(in)	X	Vector of N points
Real(Kind=CP), Dimension (:)	Intent(in)	Y	$Y_i=F(X_i)$
Integer	Intent(in)	N	Dimension of vectors X, Y
Real(Kind=CP), Dimension (:)	Intent(out)	D2Y	Vector containing second derivatives at the given points

Computes the second derivate of a vector of N points

Set_EPSG

Subroutine Set_EPSG (NEW_EPS)

Real(Kind=CP)	Intent(in)	NEW_EPS	Value
----------------------	-------------------	---------	-------

Sets an internal **EPS** parameters on this module to the value **NEW_EPS**

Set_EPSG_Default

Subroutine Set_EPSG_Default ()

Sets the internal **EPS** variable belong to this module to the default value.

Default: ($\text{EPS}=10^{-5}$)

SmoothingVec

Subroutine SmoothingVec (Y, N, NIter, Ys)

Real(Kind=CP), Dimension (:)	Intent(in out)	Y	in: out:	input Vector Vector smoothed if Ys is not present in the call of this routine
Integer	Intent(in)	N		Number of Points
Integer	Intent(in)	NIter		Number of Iterations
Real(Kind=CP), Dimension (:), Optional	Intent(out)	Ys		If present, Vector smoothed

Smooth a set of values contained in a vector of N points

Sort

Subroutine Sort (A, N, Indx)

Integer, Dimension (:)	Intent(in)	A	input vector
Integer	Intent(in)	N	Dimension of A
Integer, Dimension (:)	Intent(out)	Indx	Vector containing the initial index

or

Real(Kind=CP), Dimension (:)	Intent(in)	A	input vector
Integer	Intent(in)	N	Dimension of A
Integer, Dimension (:)	Intent(out)	Indx	Vector containing the initial index

Sort a vector such the **A** (**indx** (j)) is in ascending order for j=1,2,...,N.

Sort_Strings

Subroutine Sort_Strings (A)

Character (Len=*) , Dimension (:)	Intent(in out)	A	in: input Vector of Strings out: Vector of strings ordered
--	---------------------------	---	---

Sorts a vector of strings. The original vector is replaced by the ordered one on output.

Spline

Subroutine Spline (X, Y, N, YP1, YPN, Y2)

Real(Kind=CP), Dimension (:)	Intent(in)	X	input vector
Real(Kind=CP), Dimension (:)	Intent(in)	Y	input vector
Integer	Intent(in)	N	Dimension of X and Y
Real(Kind=CP)	Intent(in)	YP1	Derivate of Point 1
Real(Kind=CP)	Intent(in)	YPN	Derivate of Point N
Real(Kind=CP), Dimension (:)	Intent(out)	Y2	Vector containing second derivatives at the given points

Computes the spline of N points

Splint

Subroutine Splint (X, Y, Y2, N, Xp, Yp)

Real(Kind=CP), Dimension (:)	Intent(in)	X	input vector
Real(Kind=CP), Dimension (:)	Intent(in)	Y	input vector $Y_i=F(X_i)$
Real(Kind=CP), Dimension (:)	Intent(in)	Y2	Vector containing second derivatives at the given points
Integer	Intent(in)	N	Dimension of X, Y, Y2
Real(Kind=CP)	Intent(in)	Xp	Point to evaluate
Real(Kind=CP)	Intent(out)	Yp	interpolated value

Compute the spline interpolation **Yp** at the point **Xp**

SVDCMP**Subroutine SVDCMP (A, W, V)**

Real (Kind=SP / DP), Dimension (:,:)	Intent(in out)	A	in: Matrix A(M,N) out: Matrix U
Real (Kind=SP / DP), Dimension (:)	Intent(out)	W	The diagonal matrix of singular values W is output as the N-dimensional vector W
Real (Kind=SP / DP), Dimension (:,:)	Intent(out)	V	Array V(N,N)

Compute the computes its singular value decomposition, $\mathbf{A} = \mathbf{U} \cdot \mathbf{W} \cdot \mathbf{V}^T$

Swap**Subroutine Swap (A, B, Mask)**

Complex / Integer / Real (Kind=CP)	Intent(in out)	A	in: A out: B
Complex / Integer / Real (Kind=CP)	Intent(in out)	B	in: B out: A
Logical, Optional	Intent(out)	Mask	.TRUE. if it is present

or

Complex / Integer / Real (Kind=CP), Dimension (:)	Intent(in out)	A	in: A out: B
Complex / Integer / Real (Kind=CP), Dimension (:)	Intent(in out)	B	in: B out: A
Logical, Optional	Intent(out)	Mask	.TRUE. if it is present

or

Complex / Integer / Real (Kind=CP), Dimension (:,:)	Intent(in out)	A	in: A out: B
Complex / Integer / Real (Kind=CP), Dimension (:,:)	Intent(in out)	B	in: B out: A
Logical, Optional	Intent(out)	Mask	.TRUE. if it is present

Swap the contents of **A** and **B**

CFML_PowderProfiles_CW

Peak profile calculations for constant wavelength

Functions

- [Back To Back Exp](#)
- [Exponential](#)
- [Gaussian](#)

- [Hat](#)
- [Ikeda_Carpenter](#)
- [Lorentzian](#)
- [PseudoVoigt](#)
- [Split_PseudoVoigt](#)
- [TCH_PVoigt](#)

Subroutines

- [Back_To_Back_Exp_Der](#)
- [Exponential_Der](#)
- [Gaussian_Der](#)
- [Hat_Der](#)
- [Ikeda_Carpenter_Der](#)
- [Lorentzian_Der](#)
- [PseudoVoigt_Der](#)
- [Split_PseudoVoigt_Der](#)
- [TCH_PVoigt_Der](#)

Fortran Filename

CFML_Profile_Functs.f90

Functions

- [Back_To_Back_Exp](#)
- [Exponential](#)
- [Gaussian](#)
- [Hat](#)
- [Ikeda_Carpenter](#)
- [Lorentzian](#)
- [PseudoVoigt](#)
- [Split_PseudoVoigt](#)
- [TCH_PVoigt](#)

Back_To_Back_Exp

Real Function Back_To_Back_Exp(X, Par)

Real(Kind=CP)	Intent(in)	X	
Real(Kind=CP), Dimension(:)	Intent(in)	Par	PAR(1)= α ; PAR(2)= β

The Back_to_Back exponential function is defined as

$$BB(x) = \begin{cases} \frac{1}{2} \frac{\alpha\beta}{\alpha+\beta} \exp(\alpha x) & x < 0 \\ \frac{1}{2} \frac{\alpha\beta}{\alpha+\beta} \exp(-\beta x) & x \geq 0 \end{cases}$$

Exponential

Real Function Exponential(X, Par)

Real(Kind=CP)	Intent(in)	X	
Real(Kind=CP), Dimension(:)	Intent(in)	Par	PAR(1)=α

The exponential function is defined as

$$Exp(x) = \begin{cases} 0 & x < 0 \\ \alpha \exp(-\alpha x) & x \geq 0 \end{cases}$$

Gaussian

Real Function Gaussian(X, Par)

Real(Kind=CP)	Intent(in)	X	
Real(Kind=CP), Dimension(:)	Intent(in)	Par	PAR(1)=H

The Gaussian function is:

$$G(x) = a_G \exp(-b_G x^2)$$

$$a_G = \frac{2}{H} \sqrt{\frac{\ln 2}{\pi}} \quad b_G = \frac{4 \ln 2}{H^2}$$

where H is the FWHM.

Hat

Real Function Hat(X, Par)

Real(Kind=CP)	Intent(in)	X	
Real(Kind=CP), Dimension(:)	Intent(in)	Par	PAR(1)=H

The HAT function is defined as

$$Hat(x) = \begin{cases} 1/H & -H/2 < x < H/2 \\ 0 & otherwise \end{cases}$$

[Ikeda_Carpenter](#)

Real Function Ikeda_Carpenter(X, Par)

Real(Kind=CP)	Intent(in)	X	
Real(Kind=CP), Dimension(:)	Intent(in)	Par	PAR(1)= α ; PAR(2)= β ; PAR(3)=R

The Ikeda-Carpenter function is for $x > 0$ as

$$IK(x) = \frac{1}{2}\alpha^3 \left[(1-R)x^2 \exp(-\alpha x) + 2R\beta \frac{1}{(\alpha-\beta)} \left\{ \exp(-\beta x) - \exp(-\alpha x) \left[1 + \left(1 + \frac{1}{2}x(\alpha-\beta) \right) (\alpha-\beta)x \right] \right\} \right]$$

[Lorentzian](#)

Real Function Lorentzian(X, Par)

Real(Kind=CP)	Intent(in)	X	
Real(Kind=CP), Dimension(:)	Intent(in)	Par	PAR(1)=H

The Lorentzian function is

$$L(x) = \frac{a_L}{1+b_L x^2}$$

$$a_L = \frac{2}{\pi H} \quad b_L = \frac{4}{H^2}$$

where H is the *FWHM*.

[PseudoVoigt](#)

Real Function PseudoVoigt(X, Par)

Real(Kind=CP)	Intent(in)	X	
Real(Kind=CP), Dimension(:)	Intent(in)	Par	PAR(1)=H; PAR(2)= η

The PseudoVoigt function is

$$pV(x) = \eta L(x) + (1-\eta)G(x) \quad 0 \leq \eta \leq 1$$

where $L(x)$ and $G(x)$ are a [Lorentzian](#) and [Gaussian](#) functions with the same $FWHM(H)$.

Split_PseudoVoigt

Real Function Split_PseudoVoigt(X, Par)

Real(Kind=CP)	Intent(in)	X	
Real(Kind=CP), Dimension(:)	Intent(in)	Par	PAR(1)=H1; PAR(2)=H2 PAR(3)=η1; PAR(4)=η2

The Split PseudoVoigt is defined as

TCH_PVoigt

Real Function TCH_PVoigt (X, Par)

Real(Kind=CP)	Intent(in)	X	
Real(Kind=CP), Dimension(:)	Intent(in)	Par	PAR(1)=H1; PAR(2)=H2 PAR(3)=η1; PAR(4)=η2

The TCH_PVoigt is defined as

Subroutines

- [Back_To_Back_Exp_Der](#)
- [Exponential_Der](#)
- [Gaussian_Der](#)
- [Hat_Der](#)
- [Ikeda_Carpenter_Der](#)
- [Lorentzian_Der](#)
- [PseudoVoigt_Der](#)
- [Split_PseudoVoigt_Der](#)
- [TCH_PVoigt_Der](#)

Back_To_Back_Exp_Der

Subroutine Back_To_Back_Exp_Der (X, Par, BB_Val, DPar)

Real(Kind=CP)	Intent(in)	X	
Real(Kind=CP), Dimension(:)	Intent(in)	Par	PAR(1)=α; PAR(2)=β
Real(Kind=CP)	Intent(out)	BB_Val	
Real(Kind=CP), Dimension(:), Optional	Intent(out)	DPar	1=DerX; 2=Derα; 3=Derβ

Calculation of the value of the function on X and the partial derivate of the function according to the parameters.

Exponential_Der

Subroutine Exponential_Der (X, Par, Ex_Val, DPar)

Real(Kind=CP)	Intent(in)	X	
Real(Kind=CP), Dimension(:)	Intent(in)	Par	PAR(1)= α
Real(Kind=CP)	Intent(out)	Ex_Val	
Real(Kind=CP), Dimension(:), Optional	Intent(out)	DPar	1=DerX; 2=Der α

Calculation of the value of the function on X and the partial derivate of the function according to the parameters.

Gaussian_Der

Subroutine Gaussian_Der (X, Par, Gauss_Val, DPar)

Real(Kind=CP)	Intent(in)	X	
Real(Kind=CP), Dimension(:)	Intent(in)	Par	PAR(1)=H
Real(Kind=CP)	Intent(out)	Gauss_Val	
Real(Kind=CP), Dimension(:), Optional	Intent(out)	DPar	1=DerX; 2=DerH

Calculation of the value of the function on X and the partial derivate of the function according to the parameters.

Hat_Der

Subroutine Hat_Der (X, Par, H_Val, DPar)

Real(Kind=CP)	Intent(in)	X	
Real(Kind=CP), Dimension(:)	Intent(in)	Par	PAR(1)=H
Real(Kind=CP)	Intent(out)	H_Val	
Real(Kind=CP), Dimension(:), Optional	Intent(out)	DPar	1=DerX; 2=DerH

Calculation of the value of the function on x and the partial derivate of the function according to the parameters.

Ikeda_Carpenter_Der

Subroutine Ikeda_Carpenter_Der(X, Par, IK_Val, DPar)

Real(Kind=CP)	Intent(in)	X	
Real(Kind=CP), Dimension(:)	Intent(in)	Par	PAR(1)= α ; PAR(2)= β ; PAR(3)=R
Real(Kind=CP)	Intent(out)	IK_Val	
Real(Kind=CP), Dimension(:), Optional	Intent(out)	DPar	1=DerX, 2=Der α , 3=Der β , 4=DerR

Calculation of the value of the function on X and the partial derivate of the function according to the parameters.

Lorentzian_Der

Subroutine Lorentzian_Der (X, Par, Lor_Val, DPar)

Real(Kind=CP)	Intent(in)	X	
Real(Kind=CP), Dimension(:)	Intent(in)	Par	PAR(1)=H
Real(Kind=CP)	Intent(out)	Lor_Val	
Real(Kind=CP), Dimension(:), Optional	Intent(out)	DPar	1=DerX, 2=DerH

Calculation of the value of the function on x and the partial derivate of the function according to the parameters

PseudoVoigt_Der

Subroutine PseudoVoigt_Der (X, Par, PV_Val, DPar)

Real(Kind=CP)	Intent(in)	X	
Real(Kind=CP), Dimension(:)	Intent(in)	Par	PAR(1)=H; PAR(2)= η
Real(Kind=CP)	Intent(out)	PV_Val	
Real(Kind=CP), Dimension(:), Optional	Intent(out)	DPar	1=DerX, 2=DerH, 3=Der η

Calculation of the value of the function on X and the partial derivate of the function according to the parameters

Split_PseudoVoigt_Der

Subroutine Split_PseudoVoigt_Der (X, Par, PV_Val, DPar)

Real(Kind=CP)	Intent(in)	X	
Real(Kind=CP), Dimension(:)	Intent(in)	Par	PAR(1)=H1; PAR(2)=H2 PAR(3)= η 1; PAR(4)= η 2
Real(Kind=CP)	Intent(out)	PV_Val	
Real(Kind=CP), Dimension(:), Optional	Intent(out)	DPar	1=DerX, 2=DerH1, 3=DerH2, 4=Der η 1, 5=Der η 2

Calculation of the value of the function on x and the partial derivate of the function according to the parameters

TCH_PVoigt_Der

Subroutine TCH_PVoigt_Der (X, Par, PV_Val, DPar)

Real(Kind=CP)	Intent(in)	X	
Real(Kind=CP), Dimension(:)	Intent(in)	Par	PAR(1)=HG; PAR(2)=HL
Real(Kind=CP)	Intent(out)	PV_Val	
Real(Kind=CP), Dimension(:), Optional	Intent(out)	DPar	1=DerX, 2=DerHG, 3=DerHL

Calculation of the value of the function on X and the partial derivate of the function according to the parameters

CFML_PowderProfiles_Finger

Procedures for asymmetry calculations due to axial divergence using the method of Finger, Cox and Jephcoat (*J. Appl. Cryst.* (1992), 27, 892)

Variables

- [Init_ProfVal](#)

Subroutines

- [Init_Prof_Val](#)
- [Prof_Val](#)

Fortran Filename

CFML_Profile_Finger.f90

Variables

- [Init_ProfVal](#)

`Init_ProfVal`

LOGICAL :: Init_ProfVal

.TRUE. if the values for the abscissas and weights of the Gauss-Legendre N-point quadrature formula have been precomputed using routine `Init_Prof_Val` and internally stored.

Subroutines

- [Init_Prof_Val](#)
- [Prof_Val](#)

`Init_Prof_Val`

Subroutine Init_Prof_Val()

Routine that calculate the values for the abscissas and weights of the Gauss-Legendre N-point quadrature formula have been precomputed using routine Gauleg and the data are stored internally. When this routine is called then the variable `Init_ProfVal` is set to .TRUE.

`Prof_Val`

Subroutine Prof_Val (Eta, Gamma, S_L, D_L, TwoTH, TwoTH0, DPrdT, DPrdG, DPrdE, DPrdS, DPrdD, ProfVal, Use_Asym)

<code>Real(Kind=CP)</code>	<code>Intent(in)</code>	<code>Eta</code>	Mixing coefficient between Gaussian and Lorentzian
<code>Real(Kind=CP)</code>	<code>Intent(in)</code>	<code>Gamma</code>	FWHM
<code>Real(Kind=CP)</code>	<code>Intent(in)</code>	<code>S_L</code>	Source width/detector distance
<code>Real(Kind=CP)</code>	<code>Intent(in)</code>	<code>D_L</code>	Detector width/detector distance

Real(Kind=CP)	Intent(in)	TwoTH	Point at which to evaluate the profile
Real(Kind=CP)	Intent(in)	TwoTH0	Two theta value for peak
Real(Kind=CP)	Intent(out)	DPrdT	derivative of profile wrt TwoTH0
Real(Kind=CP)	Intent(out)	DPrdG	derivative of profile wrt Gamma
Real(Kind=CP)	Intent(out)	DPrdE	derivative of profile wrt Eta
Real(Kind=CP)	Intent(out)	DPrdS	derivative of profile wrt S_L
Real(Kind=CP)	Intent(out)	DPrdD	derivative of profile wrt D_L
Real(Kind=CP)	Intent(out)	ProfVal	
Logical	Intent(in)	Use_Asym	.TRUE. if asymmetry to be used

Return the value of Profile.

NOTE: Asymmetry due to axial divergence using the method of Finger, Cox and Jephcoat, J. Appl. Cryst. 27, 892, 1992.

CFML_PowderProfiles_TOF

This module contains variables and procedures used by programs aiming to handle **T.O.F.** powder diffraction patterns.

Variables

- [Deriv_TOF_Type](#)
- [LorComp](#)

Functions

- [Erfc](#)
- [Erfcp](#)

Subroutines

- [TOF_Carpenter](#)
- [TOF_Jorgensen](#)
- [TOF_Jorgensen_Vondreele](#)

Fortran Filename

CFML_Profile_TOF.f90

Variables

- [Deriv_TOF_Type](#)
- [LorComp](#)

Deriv_TOF_Type

	Variable	Definition
Type :: Deriv_TOF_Type		
Real(Kind=CP)	Alfa	omega_a DOmega/Dalpha
Real(Kind=CP)	Beta	omega_b DOmega/Dbeta
Real(Kind=CP)	Dt	omega_t DOmega/Ddt (dt=TOFi-TOF (Bragg))
Real(Kind=CP)	Sigma	omega_s DOmega/Dsigma (for tof_Jorgensen function)
Real(Kind=CP)	Gamma	omega_g DOmega/Dgamma (for tof_Jorgensen_VonDreele function)
Real(Kind=CP)	Eta	omega_e DOmega/Deta (for tof_Jorgensen_VonDreele function)
Real(Kind=CP)	Kappa	omega_e DOmega/kappa (for tof_Carpenter function)
End Type Deriv_TOF_Type		

LorComp

Logical :: LorComp

This variable is set to .TRUE. if there are Lorentzian components

Functions

- [Erfc](#)
- [Erfcp](#)

Erfc

Real Function Erfc(X)

Real(Kind=SP / DP)	Intent(in)	X	
--------------------	------------	---	--

Complementary error function

Erfcp

Real Function Erfcp(X)

Real(Kind=SP / DP)	Intent(in)	X	
--------------------	------------	---	--

Derivate of the complementary error function

Subroutines

- [TOF_Carpenter](#)
- [TOF_Jorgensen](#)
- [TOF_Jorgensen_Vondreele](#)

[TOF_Carpenter](#)

Subroutine TOF_Carpenter (Dt, D, Alfa, Beta, Gamma, Eta, Kappa, TOF_Theta, TOF_Peak, Deriv)

Read(Kind=CP)	Intent(in)	Dt	TOF(channel i) -TOF(Bragg position)
Read(Kind=CP)	Intent(in)	D	d-spacing of the peak in A
Read(Kind=CP)	Intent(in)	Alfa	units microsecs ⁻¹
Read(Kind=CP)	Intent(in)	Beta	units microsecs ⁻¹
Read(Kind=CP)	Intent(in)	Gamma	units microsecs
Read(Kind=CP)	Intent(in)	Eta	Mixing coefficient calculated using TCH
Read(Kind=CP)	Intent(in)	Kappa	Mixing coefficient of the Ikeda-Carpenter function
Read(Kind=CP)	Intent(in)	TOF_Theta a	This is the value of $2\sin(\theta)$
Read(Kind=CP)	Intent(out)	TOF_Pea k	
Type(Deriv_TOF_Type), Optional	Intent(out)	Deriv	present if derivatives are to be calculated

Calculate Profile of TOF according to Carpenter

[TOF_Jorgensen](#)

Subroutine TOF_Jorgensen (Dt, Alfa, Beta, Sigma, TOF_Peak, Deriv)

Read(Kind=CP)	Intent(in)	Dt	TOF(channel i) -TOF(Bragg position)
Read(Kind=CP)	Intent(in)	Alfa	units microsecs ⁻¹
Read(Kind=CP)	Intent(in)	Beta	units microsecs ⁻¹
Read(Kind=CP)	Intent(in)	Sigma	units microsecs ²
Read(Kind=CP)	Intent(out)	TOF_Pea k	
Type(Deriv_TOF_Type), Optional	Intent(out)	Deriv	present if derivatives are to be calculated

Calculate Profile of TOF according to Jorgensen

[TOF_Jorgensen_Vondreele](#)

Subroutine TOF_Jorgensen_Vondreele (Dt, Alfa, Beta, Gamma, Eta, TOF_Peak, Deriv)

Read(Kind=CP)	Intent(in)	Dt	TOF(channel i) -TOF(Bragg position)
-------------------------------	----------------------------	----	-------------------------------------

Read(Kind=CP)	Intent(in)	Alfa	units microsecs ⁻¹
Read(Kind=CP)	Intent(in)	Beta	units microsecs ⁻¹
Read(Kind=CP)	Intent(in)	Gamma	units microsecs
Read(Kind=CP)	Intent(in)	Eta	Mixing coefficient calculated using TCH
Read(Kind=CP)	Intent(out)	TOF_Peak	
Type(Deriv_TOF_Type), Optional	Intent(out)	Deriv	present if derivatives are to be calculated

Calculate Profile of TOF according to Jorgensen_Vondreele

CFML_Random_Generators

Module for random number generation for different distributions

Variables

- [Err_Random](#)
- [Err_Random_Mess](#)

Subroutines

- [Init_Error_Random](#)
- [Random_Beta](#)
- [Random_Binomial1](#)
- [Random_Binomial2](#)
- [Random_Cauchy](#)
- [Random_Chisq](#)
- [Random_Exponential](#)
- [Random_Gamma](#)
- [Random_Gamma1](#)
- [Random_Gamma2](#)
- [Random_Inv_Gauss](#)
- [Random_MVNorm](#)
- [Random_Neg_Binomial](#)
- [Random_Normal](#)
- [Random_Order](#)
- [Random_Poisson](#)
- [Random_T](#)
- [Random_Von_Mises](#)
- [Random_Weibull](#)
- [Seed_Random_Number](#)

Fortran Filename

CFML_Random.f90

Variables

- [Err_Random](#)
- [Err_Random_Mess](#)

Err_Random

LOGICAL :: Err_Random

This variable is set to **.TRUE.** if an error occurs in procedures belonging to this module

Err_Random_Mess

Character(Len=150) :: Err_Random_Mess

This variable contains information about the last error occurred in the procedures belonging to this module

Subroutines

- [Init_Err_Random](#)
- [Random_Beta](#)
- [Random_Binomial1](#)
- [Random_Binomial2](#)
- [Random_Cauchy](#)
- [Random_Chisq](#)
- [Random_Exponential](#)
- [Random_Gamma](#)
- [Random_Gamma1](#)
- [Random_Gamma2](#)
- [Random_Inv_Gauss](#)
- [Random_MVNorm](#)
- [Random_Neg_Binomial](#)
- [Random_Normal](#)
- [Random_Order](#)
- [Random_Poisson](#)
- [Random_T](#)
- [Random_Von_Mises](#)
- [Random_Weibull](#)
- [Seed_Random_Number](#)

Init_Err_Random

Subroutine Init_Err_Random ()

Initializes general error variables [Err_Random](#) and [Err_Random_Mess](#)

Random_Beta

Subroutine Random_Beta (AA, BB, First, FN_VAL)

Real(Kind=CP)	Intent(in)	AA	Shape parameter from distribution ($0 < \text{real}$)
-------------------------------	----------------------------	----	---

Real(Kind=CP)	Intent(in)	BB	shape parameter from distribution ($0 < \text{real}$)
Logical	Intent(in)	First	.TRUE. for the first call
Real(Kind=CP)	Intent(out)	FN_VAL	

Generate a random variate in [0,1] from a beta distribution with density proportional to $\beta^{(AA-1)} * (1-\beta)^{(BB-1)}$ using Cheng's log logistic method.

NOTE: Reference to the book from Dagpumar, J. *Principles of random variate generation*, Clarendon Press, Oxford, 1988. ISBN 0-19-852202-9

Random_Binomial1

Subroutine Random_Binomial1 (N, P, First, Ival)

Integer	Intent(in)	N	Number of Bernoulli Trials (1 <= Integer)
Real(Kind=CP)	Intent(in)	P	Bernoulli Success Probability (0 <= Real <= 1)
Logical	Intent(in)	First	.TRUE. for the first call using the current parameter values .FALSE. if the values of (n,p) are unchanged from last call
Integer	Intent(out)	Ival	

Subroutine that generates a random binomial variate using C.D.Kemp's method. This algorithm is suitable when many random variables are required with the same parameter values for n & p

NOTE: Reference Kemp, C.D. (1986). 'A modal method for generating binomial variables", Commun. Statist. - Theor. Meth. 15(3), 805-813.

Random_Binomial2

Subroutine Random_Binomial2 (N, PP, First, Ival)

Integer	Intent(in)	N	The number of trials in the binomial distribution from which a random deviate is to be generated
Real(Kind=CP)	Intent(in)	PP	The probability of an event in each trial of the binomial distribution from which a random deviate is to be generated.
Logical	Intent(in)	First	.TRUE. for the first call to perform initialization .FALSE. for further calls using the same pair of parameter values (N, PP)
Integer	Intent(out)	Ival	

Subroutine that generates a single random deviate from a binomial distribution whose number of trials is N and whose probability of an event in each trial is PP.

Random_Cauchy

Subroutine Random_Cauchy (FN_VAL)

Real(Kind=CP)	Intent(out)	FN_VA L	The probability of an event in each trial of the binomial distribution from which a random deviate is to be generated.
----------------------	--------------------	------------	--

Generates a single random deviate from the standard Cauchy distribution.

Random_Chisq

Subroutine Random_Chisq (Ndf, First, FN_VAL)

Integer	Intent(in)	Ndf	Integer that represent degree of freedom
Logical	Intent(in)	First	.TRUE. for the first call using the current parameter values
Real(Kind=CP)	Intent(out)	FN_VA L	

Generates a random variate from the chi-squared distribution with **Ndf** degrees of freedom

Random_Exponential

Subroutine Random_Exponential (FN_VAL)

Real(Kind=CP)	Intent(out)	FN_VA L	
----------------------	--------------------	------------	--

Generates a random variate in $[0, \infty)$ from a negative exponential distribution with density proportional to $\exp(-\text{random_exponential})$, using inversion.

NOTE: Reference to the book from Dagpunar, J. *Principles of random variate generation*, Clarendon Press, Oxford, 1988. ISBN 0-19-852202-9

Random_Gamma

Subroutine Random_Gamma (S, First, FN_VAL)

Real(Kind=CP)	Intent(in)	S	Shape parameter of distribution ($0.0 < \text{real}$)
Logical	Intent(in)	First	.TRUE. for the first call using the current parameter values
Real(Kind=CP)	Intent(out)	FN_VA L	

Generates a random gamma variate

NOTE: Reference to the book from Dagpunar, J. *Principles of random variate generation*, Clarendon Press, Oxford, 1988. ISBN 0-19-852202-9

Random_Gamma1

Subroutine Random_Gamma1 (S, First, FN_VAL)

Real(Kind=CP)	Intent(in)	S	Shape parameter of distribution (0.0 < real)
Logical	Intent(in)	First	.TRUE. for the first call using the current parameter values
Real(Kind=CP)	Intent(out)	FN_VAL	

Generates a random variate in $[0, \infty)$ from a gamma distribution with density proportional to $\text{gamma}^{**}(s-1) * \exp(-\text{gamma})$, based upon best's t distribution method.

NOTE: Reference to the book from Dagpunar, J. *Principles of random variate generation*, Clarendon Press, Oxford, 1988. ISBN 0-19-852202-9

Random_Gamma2

Subroutine Random_Gamma2 (S, First, FN_VAL)

Real(Kind=CP)	Intent(in)	S	Shape parameter of distribution (0.0 < real)
Logical	Intent(in)	First	.TRUE. for the first call using the current parameter values
Real(Kind=CP)	Intent(out)	FN_VAL	

Generates a random variate in $[0, \infty)$ from a gamma distribution with density proportional to $\text{gamma2}^{**}(s-1) * \exp(-\text{gamma2})$, using a switching method.

NOTE: Reference to the book from Dagpunar, J. *Principles of random variate generation*, Clarendon Press, Oxford, 1988. ISBN 0-19-852202-9

Random_Inv_Gauss

Subroutine Random_Inv_Gauss (H, B, First, FN_VAL)

Real(Kind=CP)	Intent(in)	H	Parameter of distribution ($0 \leq \text{real}$)
Real(Kind=CP)	Intent(in)	B	Parameter of distribution ($0 \leq \text{real}$)
Logical	Intent(in)	First	.TRUE. for the first call using the current parameter values
Real(Kind=CP)	Intent(out)	FN_VAL	

Generates a random variate in $[0, \infty)$ from a reparameterised generalised inverse gaussian (gig) distribution with density proportional to $\text{gig}^{**}(h-1) * \exp(-0.5*b*(\text{gig}+1/\text{gig}))$ using a ratio method.

NOTE: Reference to the book from Dagpunar, J. *Principles of random variate generation*, Clarendon Press, Oxford, 1988. ISBN 0-19-852202-9

Random_MVNorm

Subroutine Random_MVNorm (N, H, D, F, First, X, ler)

Integer	Intent(in)	N	Number of variates in vector (input, Integer >= 1)
Real(Kind=CP), Dimension(N)	Intent(in)	H	Vector of means
Real(Kind=CP), Dimension(N*(N +1)/2)	Intent(in)	D	Variance matrix ($j \geq i$)
Real(Kind=CP), Dimension(N*(N +1)/2)	Intent(in)	F	Parameter of distribution ($0 < \text{real}$)
Logical	Intent(in)	First	.TRUE. if this is the first call of the routine or if the distribution has changed since the last call of the routine. .FALSE. otherwise
Real(Kind=CP), Dimension(N)	Intent(out)	X	Delivered vector
Integer	Intent(out)	ler	= 1 if the input covariance matrix is not +ve definite = 0 otherwise

Generates an n variate random normal vector using a Cholesky decomposition.

NOTE: Reference to the book from Dagpunar, J. *Principles of random variate generation*, Clarendon Press, Oxford, 1988. ISBN 0-19-852202-9

Random_Neg_Binomial

Subroutine Random_Neg_Binomial (Sk, P, lval)

Real(Kind=CP)	Intent(in)	Sk	Number of failures required the "power" parameter of the negative binomial ($0 < \text{real}$)
Real(Kind=CP)	Intent(in)	P	Bernoulli success probability ($0 < \text{real} < 1$)
Integer	Intent(out)	lval	

Generates a random negative binomial variate using unstored inversion and/or the reproductive property.

NOTE: Reference to the book from Dagpunar, J. *Principles of random variate generation*, Clarendon Press, Oxford, 1988. ISBN 0-19-852202-9

Random_Normal

Subroutine Normal (FN_VAL)

Real(Kind=CP)	Intent(out)	FN_VAL	
----------------------	--------------------	--------	--

Returns a normally distributed pseudo-random number with zero mean and unit variance. The algorithm uses the ratio of uniforms method of A.J. Kinderman and J.F. Monahan augmented with quadratic bounding curves.

Random_Order

Subroutine Random_Order (Order, N)

Integer	Intent(in)	N	Number of Integers
Integer, Dimension(N)	Intent(out)	Order	Vector of random integers

Generate a random ordering of the Integers 1 ... n.

Random_Poisson

Subroutine Random_Poisson (Mu, Genpoi)

Real(Kind=CP)	Intent(in)	Mu	
Integer	Intent(out)	Genpoi	

Generates a single random deviate from a Poisson distribution with mean mu.

Random_T

Subroutine Random_T(M, FN_VAL)

Integer	Intent(in)	M	Degrees of freedom of distribution (1 <= Integer)
Real(Kind=CP)	Intent(out)	FN_VAL	

Generates a random variate from a t distribution using kinderman and monahan's ratio method.

Random_Von_Mises

Subroutine Random_Von_Mises(K, First, FN_VAL)

Real(Kind=CP)	Intent(in)	K	Parameter of the von Mises distribution
Logical	Intent(in)	First	set to .TRUE. the first time that the subroutine is called
Real(Kind=CP)	Intent(out)	FN_VAL	

Generate a Von Mises Distribution

Random_Weibull

Subroutine Random_Weibull(A, FN_VAL)

Real(Kind=CP)	Intent(in)	A	
Real(Kind=CP)	Intent(out)	FN_VAL	

Generates a random variate from the Weibull distribution with probability density as

$$f(x) = \alpha x^{\alpha-1} e^{-x^2}$$

[Seed_Random_Number](#)

Subroutine Seed_Random_Number (I_Input, I_Output)

Integer, Optional	Intent(in)	I_Input	Unit number for input
Integer, Optional	Intent(in)	I_Output	Unit number for output

The seed is read from the **I_Input** unit if present and from keyboard if not.

The output messages is directed to **I_Output** unit if it is present or in the screen in default.

CFML_Spherical_Harmonics

Module containing Spherical Harmonics routines

Variables

- [Err_Spher](#)
- [Err_Spher_Mess](#)

Functions

- [Cubic_Harm_Ang](#)
- [Cubic_Harm_UcVec](#)
- [Int_Slater_Bessel](#)
- [Real_Spher_Harm_Ang](#)
- [Real_Spher_Harm_UcVec](#)
- [Real_Spher_HarmCharge_UcVec](#)

Subroutines

- [Init_Err_Spher](#)
- [Pikout_LJ_Cubic](#)
- [SphJn](#)

Fortran Filename

CFML_Spher_Harm.f90

Variables

- [Err_Spher](#)
- [Err_Spher_Mess](#)

Err_Spher

LOGICAL :: Err_Spher

This variable is set to .TRUE. if an error occurs in procedures belonging to this module

Err_Spher_Mess

Character (Len=150) :: Err_Spher_Mess

This variable contains information about the last error occurred in the procedures belonging to this module

Functions

- [Cubic_Harm_Ang](#)
- [Cubic_Harm_UcVec](#)
- [Int_Slater_Bessel](#)
- [Real_Spher_Harm_Ang](#)
- [Real_Spher_Harm_UcVec](#)
- [Real_Spher_HarmCharge_UcVec](#)

Cubic_Harm_Ang

Real Function Cubic_Harm_Ang(L, M, THETA, PHI)

Integer	Intent(in)	L	
Integer	Intent(in)	M	
Real(Kind=CP)	Intent(in)	THETA	
Real(Kind=CP)	Intent(in)	PHI	

$$K_{lj}(\theta, \varphi) = \sum_{nr} k_{nrj}^l y_{lnr}(\theta, \varphi)$$

Calculation of the cubic harmonics given in Table 3 of reference M.Kara & K. Kurki-Suonio, Acta Cryt. A37, 201 (1981).

NOTE: Only up to tenth order.

Cubic_Harm_UcVec

Real Function Cubic_Harm_UcVec(L, M, U)

Integer	Intent(in)	L	
Integer	Intent(in)	M	
Real(Kind=CP), Dimension(3)	Intent(in)	U	

Calculation of the cubic harmonics given in Table 3 of reference M.Kara & K. Kurki-Suonio, Acta Cryt. A37, 201 (1981).

NOTE: Only up to tenth order. A control of errors is included. For **m3m** symmetry, calculations include up to L=20 M=2 using the coefficients from F.M. Mueller and M.G. Priestley, Phys Rev 148, 638 (1966)

Int_Slater_Bessel

Real Function Int_Slater_Bessel(N, L, Z, S)

Integer	Intent(in)	N	
Integer	Intent(in)	L	
Real(Kind=CP)	Intent(in)	Z	
Real(Kind=CP)	Intent(in)	S	

Returns the integral:

$$\int_0^{\infty} r^{n+2} \exp(-\psi r) \cdot jl(sr) \cdot dr$$

where jl is the spherical Bessel function of order l . Only $-1 \leq n$ and $0 \leq l \leq n+1$

Real_Spher_Harm_Ang

Real Function Real_Spher_Harm_Ang(L, M, P, Theta, Phi)

Integer	Intent(in)	L	index $l \geq 0$
Integer	Intent(in)	M	index $m \leq l$
Integer	Intent(in)	P	+1: Cosine -1: Sine
Real(Kind=CP)	Intent(in)	Theta	Spherical coordinate in degree
Real(Kind=CP)	Intent(in)	Phi	Spherical coordinate in degree

Return the value of

$$y_{lm\pm}(\theta, \varphi) = \frac{1}{N_{lm}} P_l^m(\cos \theta) \begin{cases} \cos m\varphi \\ \sin m\varphi \end{cases}$$

where

$$N_{lm}^2 = (1 + \delta_{m0}) \frac{2\pi}{2l+1} \frac{(l+m)!}{(l-m)!}$$

$$P_l^0(z) = P_l(z) = \frac{1}{2^l l!} \frac{d^l}{dz^l} (z^2 - 1)^l$$

$$P_l^m(z) = (1 - z^2)^{m/2} \frac{d^m}{dz^m} P_l(z)$$

NOTE: M.Kara & K. Kurki-Suonio, Acta Cryt. A37, 201 (1981)

Real_Spher_Harm_UcVec

Real Function Real_Spher_Harm_UcVec(L, M, P, U)

Integer	Intent(in)	L	index l >= 0
Integer	Intent(in)	M	index m <= l
Integer	Intent(in)	P	+1: Cosine -1: Sine
Real(Kind=CP), Dimension(3)	Intent(in)	U	Unit vector in cartesian coordinates

Return the value of $Ylm(u)$.

NOTE: M.Kara & K. Kurki-Suonio, Acta Cryt. A37, 201 (1981)

Real_Spher_HarmCharge_UcVec

Real Function Real_Spher_HarmCharge_UcVec(L, M, P, U)

Integer	Intent(in)	L	index l >= 0
Integer	Intent(in)	M	index m <= l
Integer	Intent(in)	P	+1: Cosine -1: Sine
Real(Kind=CP), Dimension(3)	Intent(in)	U	Unit vector in cartesian coordinates

Return the value of $C_{l,m} Y_{l,m}(u)$ where $C_{l,m}$ are selected so that $\int [abs(D_{l,m}) d\Omega] = 2 \cdot d(l,0)$

Subroutines

- [Init_Err_Spher](#)
- [Pikout_LJ_Cubic](#)
- [SphJn](#)

[Init_Err_Spher](#)

Subroutine Init_Err_Spher()

Subroutine that initializes errors flags in **CFML_Spherical_Harmonics** module.

[Pikout_LJ_Cubic](#)

Subroutine Pikout_LJ_Cubic(Group, LJ, NCoef, Lun)

Character(Len=*)	Intent(in)	Group	
Integer, Dimension(2,11)	Intent(out)	LJ	
Integer	Intent(out)	NCoef	
Integer, Optional	Intent(in)	Lun	

Picking out rules for indices of cubic harmonics for the 5 cubic groups.

NOTE: Only up to tenth order Given in Table 4 of reference M.Kara & K. Kurki-Suonio, Acta Cryt. A37, 201 (1981)

SphJn

Subroutine SphJn(N, X, NM, JN, DJN)

Integer	Intent(in)	N	Order of Jn(x)
Real(Kind=DP)	Intent(in)	X	Argument of Jn
Integer	Intent(out)	NM	Highest order computed
Real(Kind=DP), Dimension(0:N)	Intent(out)	JN	Array with spherical Bessel functions Jn(x)
Real(Kind=DP), Dimension(0:N)	Intent(out)	DJN	Array with derivatives Jn'(x)

Compute Spherical Bessel functions Jn(x) and their derivatives

CFML_String_Utillities

Module containing procedures for manipulation of strings with alphanumeric characters

Variables

- [Err_String](#)
- [Err_String_Mess](#)
- [Err_Text_Type](#)
- [Ierr_FMT](#)
- [Mess_FindFMT](#)

Functions

- [Equal_Sets_Text](#)
- [L_Case](#)
- [Pack_String](#)
- [Strip_String](#)
- [U_Case](#)

Subroutines

- [CutST](#)
- [FindFMT](#)
- [Frac_Trans_1Dig](#)
- [Frac_Trans_2Dig](#)
- [Get_BaseName](#)
- [Get_DirName](#)

- [Get_Fraction_1Dig](#)
- [Get_Fraction_2Dig](#)
- [Get_LogUnit](#)
- [Get_Separator_Pos](#)
- [GetNum](#)
- [GetNum_STD](#)
- [GetWord](#)
- [Inc_LineNum](#)
- [Init_Err_String](#)
- [Init_FindFMT](#)
- [Lcase](#)
- [Number_Lines](#)
- [NumCol_From_NumFMT](#)
- [Read_Key_Str](#)
- [Read_Key_StrVal](#)
- [Read_Key_Value](#)
- [Read_Key_ValueSTD](#)
- [Reading_Lines](#)
- [SetNum_STD](#)
- [Ucase](#)

Fortran Filename

CFML_String_Util.f90

Variables

- [Err_Text_Type](#)
- [Err_String](#)
- [Err_String_Mess](#)
- [Ierr_FMT](#)
- [Mess_FindFMT](#)

[Err_Text_Type](#)

	Variable	Definition
Type :: Err_Text_Type		
Integer	NLines	Number of lines
Character (Len=132), Dimension(5)	Txt	Error information
End Type Err_Text_Type		

Definition of this special variable used for the [FindFMT](#) procedure

[Err_String](#)

Logical :: Err_String

This variable is set to **.TRUE.** if an error occurs in procedures belonging to this module

[Err_String_Mess](#)

Character (Len=150) :: Err_String_Mess

This variable contains information about the last error occurred in the procedures belonging to this module

[Ierr_FMT](#)

Integer :: Ierr_FMT

This variable contains information about the error on [FindFMT](#) procedure.

The error code is according to next information:

Code Meaning

- 2 FORTRAN read error
- 1 End of file
- 0 No Error
- 1 Empty format descriptor (0 field)
- 2 Data string read error
- 3 Integer field found real
- 4 Begged dot, sign or "e" character
- 5 invalid character in an Integer field
- 6 invalid field in format descriptor
- 7 invalid character in a numeric field
- 8 0 character in current field
- 9 Format string length exceeded
- 10 Separator missing
- 11 incomplete E or D format
- 12 incomplete number

[Mess_FindFMT](#)

TYPE (Err_Text_Type) :: Mess_FindFMT

This variable is a text composed of a maximum of 5 lines to inform about position or error in free format reading when [FindFMT](#) procedure is used

[Functions](#)

- [Equal_Sets_Text](#)
- [L_Case](#)
- [Pack_String](#)
- [Strip_String](#)

- [U_Case](#)

[Equal_Sets_Text](#)

Logical Function Equal_Sets_Text (Text1, N1, Text2, N2)

Character (Len=*) , Dimension (:)	Intent(in)	Text1	String vector
Integer	Intent(in)	N1	Number of lines on TEXT1
Character (Len=*) , Dimension (:)	Intent(in)	Text2	String vector
Integer	Intent(in)	N2	Number of lines on TEXT2

The function is true if the two sets of text have the same lines in whatever order. Two lines are equal only if they have the same length and all their component characters are equal and in the same order.

[L_Case](#)

Character Function L_Case (Text)

Character (Len=*)	Intent(in)	TEXT	String
-------------------	------------	------	--------

Function that converts to lower case the **Text** variable

[Pack_String](#)

Character Function Pack_String (Text)

Character (Len=*)	Intent(in)	Text	String
-------------------	------------	------	--------

Function that packs a string. This means that the function provides a string without empty spaces

[Strip_String](#)

Character Function Strip_String (String, To_String)

Character (Len=*)	Intent(in)	String	String
Character (Len=*)	Intent(in)	To_String	Word from where string is cutted

Function that return a string without from **To_String** up the end.

[U_Case](#)

Character Function U_Case (Text)

Character (Len=*)	Intent(in)	Text	String
-------------------	------------	------	--------

Function that converts to upper case the **Text** variable

[Subroutines](#)

- [CutST](#)
- [FindFMT](#)

- [Frac_Trans_1Dig](#)
- [Frac_Trans_2Dig](#)
- [Get_BaseName](#)
- [Get_DirName](#)
- [Get_Fraction_1Dig](#)
- [Get_Fraction_2Dig](#)
- [Get_LogUnit](#)
- [Get_Separator_Pos](#)
- [GetNum](#)
- [GetNum_STD](#)
- [GetWord](#)
- [Inc_LineNum](#)
- [Init_Err_String](#)
- [Init_FindFMT](#)
- [Lcase](#)
- [Number_Lines](#)
- [NumCol_From_NumFMT](#)
- [Read_Key_Str](#)
- [Read_Key_StrVal](#)
- [Read_Key_Value](#)
- [Read_Key_ValueSTD](#)
- [Reading_Lines](#)
- [SetNum_STD](#)
- [Ucase](#)

CutST

Subroutine CutST (Line1, Nlong1, Line2, Nlong2)

Character (Len=*)	Intent(in out)	Line1	in:	input string
			out	input string without the first word
			:	
Integer, Optional	Intent(out)	Nlong1		If present, give the length of Line1

Character (Len=*) , Optional	Intent(out)	Line2		If present, the first word of string on input
Integer, Optional	Intent(out)	Nlong2		If present, give the length of Line2

Subroutine that removes the first word of the input String

FindFMT

Subroutine FindFMT (Lun, ALine, FMTFields, FMTString, Idebug)

Integer	Intent(in)	Lun		Logical unit number
Character (Len=*)	Intent(in out)	ALine	in: out	String to be decoded input string without the first word
			:	
Character (Len=*)	Intent(in)	FMTFields		Description of the format fields (e.g. IIFIF)
Character (Len=*)	Intent (out)	FMTString		format of the line (e.g. (I5,I1,F8.0,I4,F7.0,))
Integer, Optional	Intent(in)	Idebug		Logical unit number for writing the input file. If

			Zero then no writing is performed
--	--	--	-----------------------------------

This routine emulates the free format data input, according to

READ (*unit*=String1, *fmt*='(a,i,2f,...)') AString, I1, R1, R2,...

but with additional error checking. Thus, given a description of the expected fields **FindFMT** returns the format of the line to be decoded.

Valid field descriptors are: **I** (Integer), **A** (free A format), 1 to 5 for A1 to A5

NOTE: This routine have an associated an error code **Ierr_FMT**. If occurs an error then also an error message is generated and written to the public variable **Mess_FindFMT**

Frac_Trans_1Dig

Subroutine Frac_Trans_1Dig (V, Charf)

Real(Kind=CP), Dimension(3)	Intent(in)	V	Vector
Character (Len=*)	Intent(out)	Charf	String

Subroutine returning a string describing a 3D translation vector written in fractional form as quotient of 1-digit Integers with sign.

Example:

input: V= (0.25, -0.4, 0.3333)
output: CHARF="(1/4,-2/5,1/3)"

Frac_Trans_2Dig

Subroutine Frac_Trans_2Dig (V, Charf)

Real(Kind=CP), Dimension(3)	Intent(in)	V	Vector
Character (Len=*)	Intent(out)	Charf	String

Subroutine returning a string describing a 3D translation vector written in fractional form as quotient of 2-digit Integers with sign.

Example:

input: V= (0.3, -0.4, -5.5)
output: CHARF="(3/10,-2/5,-11/2)"

Get_BaseName

Subroutine Get_BaseName (Filename, ChSep, Basename)

Character (Len=*)	Intent(in)	Filename	input string containing pathname
Character (Len=*)	Intent(in)	ChSep	Normally it will be '\' or '.' or '/'
Character (Len=*)	Intent(out)	Basename	The final component of the input Pathname

Subroutine returning a base name

[Get_DirName](#)**Subroutine Get_DirName (Filename, Directory)**

Character (Len=*)	Intent(in)	Filename	input string containing full path
Character (Len=*)	Intent(out)	Directory	String containing only the Path

Subroutine returning the directory for **Filename**

[Get_Fraction_1Dig](#)**Subroutine Get_Fraction_1Dig (V, Fracc)**

Real(Kind=CP)	Intent(in)	V	Real number
Character (Len=*)	Intent(out)	Fracc	String

Subroutine that gets a string with the most simple fraction that uses single digits in numerator and denominator.

[Get_Fraction_2Dig](#)**Subroutine Get_Fraction_2Dig (V, Fracc)**

Real(Kind=CP)	Intent(in)	V	Real number
Character (Len=*)	Intent(out)	Fracc	String

Subroutine that gets a string with the most simple fraction that uses up to two digits in numerator and denominator.

[Get_LogUnit](#)**Subroutine Get_LogUnit (Lun)**

Integer	Intent(out)	Lun	First logical unit available
----------------	--------------------	-----	------------------------------

Subroutine providing the number of the first logical unit that is not opened. Useful for getting a logical unit to a file that should be opened on the fly.

[Get_Separator_Pos](#)**Subroutine Get_Separator_Pos (Line, Car, Pos, Ncar)**

Character (Len=*)	Intent(in)	Line	input String
Character (Len=1)	Intent(in)	Car	Separator character
Integer, Dimension(:)	Intent(out)	Pos	Vector with positions of Car in Line
Integer	Intent(out)	Ncar	Number of appearance of Car in Line

Determines the positions of the separator character **Car** in string **Line** and generates the vector **Pos** containing the positions. The number of times the character **Car** appears in **Line** is stored in **Ncar**. The separator **Car** is not counted within substrings of **Line** that are written within quotes.

Example:

```
line =' 23, "List, of, authors", this book, year=1989'
...
call Get_Separator_Pos(line,',',pos,ncar)
...
```

Then this routine provides

```
POS=(/4, 25, 36, 0, .../)
NCAR=3
```

GetNum

Subroutine GetNum (Line, Vet, Ivet, Iv)

Character (Len=*)	Intent(in)	Line	input String to convert
Real(Kind=CP), Dimension (:)	Intent(out)	Vet	Vector of real numbers
Integer, Dimension (:)	Intent(out)	Ivet	Vector of Integer numbers
Integer	Intent(out)	Iv	Number of numbers in VET / IVET

Subroutine that converts a string to numbers and write on **VET/IVET** if real/Integer.
Control of errors is possible by inquiring the global variables [Err_String](#) and [Err_Mess_String](#)

GetNum_STD

Subroutine GetNum_STD (Line, Value, STD, Ic)

Character (Len=*)	Intent(in)	Line	input String to convert
Real(Kind=CP), Dimension (:)	Intent(out)	Value	Vector of real numbers
Real(Kind=CP), Dimension (:)	Intent(out)	STD	Vector of standard deviation values
Integer	Intent(out)	Ic	Number of components of vector Value

Subroutine that converts a string to numbers with standard deviation with format: XFFFF(S). Control of errors is possible by inquiring the global variables [Err_String](#) and [Err_Mess_String](#).

GetWord

Subroutine GetWord (Line, Dire, Iv)

Character (Len=*)	Intent(in)	Line	input String to convert
Character (Len=*), Dimension (:)	Intent(out)	Dire	Vector of words
Integer	Intent(out)	Iv	Number of components of vector DIRE

Subroutine that determines the number of words in the input string and generates a character vector with separated words.

Control of errors is possible by inquiring the global variables [Err_String](#) and [Err_Mess_String](#)

Inc_LineNum

Subroutine Inc_LineNum (Line_N)

Integer	Intent(in)	Line_N	Number of Lines that need increases
----------------	-------------------	--------	-------------------------------------

Subroutine that determines increments the current line number used in [FindFMT](#)

Init_Err_String

Subroutine Init_Err_String ()

Subroutine that initializes general error variables [Err_String](#) and [Err_String_Mess](#)

Init_FindFMT

Subroutine Init_FindFMT (NLine)

Integer, Optional	Intent(in)	NLine	Number of the line
--------------------------	-------------------	-------	--------------------

Subroutine that initializes the subroutine [FindMT](#) and [Mess_FindFMT](#) is initialized to zero lines. The current line in the file is also to initialized to zero or put to the value NLine if the optional argument is present

Lcase

Subroutine Lcase (Line)

Character (Len=*)	Intent(in out)	Line	in: input string out input line converted to lower case :
--------------------------	---------------------------	------	---

Subroutine that converts to lower case the string in the argument

Number_Lines

Subroutine Number_Lines (Filename, N)

Character (Len=*)	Intent(in)	Filename	Name of the input file
Integer	Intent(out)	N	Number of lines in the file

Subroutine that gives the number of lines contained in a file. If the file is opened, a rewind command is performed.

NumCol_From_NumFMT

Subroutine NumCol_From_NumFMT (Text, N_Col)

Character (Len=*)	Intent(in)	Text	input format string
Integer	Intent(out)	N_Col	Integer number of columns

Subroutine that provides the number of columns spanned by a numeric format field F,I,G,E

Read_Key_Str

Subroutine Read_Key_Str (Filevar, Nline_Ini, Nline_End, Keyword, String)

Character (Len=*) , Dimension (:)	Intent(in)	Filevar		input vector of Strings
Integer	Intent(inout)	Nline_Ini	in: out :	initial position to search Current position in search
Integer	Intent(in)	Nline_End		Define the final position to search
Character (Len=*)	Intent(in)	Keyword		Word to search
Character (Len=*)	Intent(out)	String		Rest of the input string where KEYWORD is contained.

Subroutine that read a string on **Filevar** starting with a particular **Keyword** between lines **Nline_Ini** and **Nline_End**.

Read_Key_StrVal

Subroutine Read_Key_StrVal (Filevar, Nline_Ini, Nline_End, Keyword, String, Vet, Ivet, Iv)

Character (Len=*) , Dimension (:)	Intent(in)	Filevar		input vector of Strings
Integer	Intent(inout)	Nline_Ini	in: out :	initial position to search Current position in search
Integer	Intent(in)	Nline_End		Define the final position to search
Character (Len=*)	Intent(in)	Keyword		Word to search
Character (Len=*)	Intent(out)	String		Rest of the input string where KEYWORD is contained.
Real(Kind=CP) , Dimension (:) , Optional	Intent(out)	Vet		Vector for real numbers
Integer, Dimension (:) , Optional	Intent(out)	Ivet		Vector for Integer numbers
Integer, Optional	Intent(out)	Iv		Number of numbers on VET / IVET

Subroutine that read a string on **Filevar** starting with a particular **Keyword** between lines **Nline_Ini** and **Nline_End**. If the string contains numbers they are read and put into **VET** / **IVET**. The variable **String** contains the input string without the **KEYWORD**.

Read_Key_Value

Subroutine Read_Key_Value (Filevar, Nline_Ini, Nline_End, Keyword, Vet, Ivet, Iv)

Character (Len=*) , Dimension (:)	Intent(in)	Filevar		input vector of Strings
---	-------------------	---------	--	-------------------------

Integer	Intent(inout)	Nline_Ini	in: out :	initial position to search Current position in search
Integer	Intent(in)	Nline_End		Define the final position to search
Character (Len=*)	Intent(in)	Keyword		Word to search
Real(Kind=CP), Dimension (:), Optional	Intent(out)	Vet		Vector for real numbers
Integer, Dimension (:), Optional	Intent(out)	Ivet		Vector for Integer numbers
Integer, Optional	Intent(out)	lv		Number of numbers on VET / IVET

Subroutine that read parameters on **Filevar** starting with a particular **Keyword** between lines **Nline_Ini** and **Nline_End**. If the string contains numbers they are read and put into **VET** / **IVET**.

Read_Key_ValueSTD

Subroutine Read_Key_ValueSTD (Filevar, Nline_Ini, Nline_End, Keyword, Vet1, Vet2, lv)

Character (Len=*), Dimension (:)	Intent(in)	Filevar		input vector of Strings
Integer	Intent(inout)	Nline_Ini	in: out :	initial position to search Current position in search
Integer	Intent(in)	Nline_End		Define the final position to search
Character (Len=*)	Intent(in)	Keyword		Word to search
Real(Kind=CP), Dimension (:)	Intent(out)	Vet1		Vector for real numbers
Real(Kind=CP), Dimension (:)	Intent(out)	Vet2		Vector for standard deviations numbers
Integer	Intent(out)	lv		Number of numbers on VET1 and VET2

Subroutine that read parameters and standard deviation on **FILEVAR** starting with a particular **KEYWORD** between lines **Nline_Ini** and **Nline_End**.

Reading_Lines

Subroutine Reading_Lines (Filename, Nlines, Filevar)

Character (Len=*)	Intent(in)	Filename	Name of the input file
Integer	Intent(in)	Nlines	Number of lines to read
Character (Len=*), Dimension(:)	Intent(out)	Filevar	String vector

Subroutine that reads **NLines** of the input file and put the information on **Filevar**. If the file was opened, then a rewind command is performed.

SetNum_STD**Subroutine SetNum_STD (Value, STD, Line)**

Real(Kind=CP)	Intent(in)	Value	Real number
Real(Kind=CP)	Intent(in)	STD	Standar deviation
Character (Len=*)	Intent(out)	Line	String with format XFFFF(S)

Subroutine that writes in **Line** a real number with standard deviation between parenthesis

Ucase**Subroutine Ucase (Line)**

Character (Len=*)	Intent(in out)	Line	in: input string out input line converted to upper case :
--------------------------	---------------------------	------	---

Subroutine that converts to upper case the string in the argument

Level 2

Concept	Module Name	Purpose
<i>Chemical Tables...</i>	<u>CFML_Scattering_Chemical_Tables</u>	Tabulated information about atomic chemical and scattering data
<i>Mathematics...</i>	<u>CFML_Math_3D</u>	Simple mathematics general utilities for 3D Systems
<i>Optimization...</i>	<u>CFML_Optimization_General</u>	Module implementing several algorithms for global and local optimization
	<u>CFML_Optimization_LSQ</u>	Module implementing Marquard algorithm for non-linear least-squares
<i>Patterns...</i>	<u>CFML_Diffraction_Patterns</u>	Diffraction Patterns data structures and procedures for reading different powder diffraction formats.
<i>Symmetry Tables...</i>	<u>CFML_Symmetry_Tables</u>	Tabulated information on Crystallographic Symmetry

CFML_Diffraction_Patterns

Module containing procedures related with Diffraction Patterns information

Variables

- [Diffraction_Pattern_Type](#)
- [Err_DiffPatt](#)
- [Err_DiffPatt_Mess](#)

Functions

- [Calc_FWHM_Peak](#)

Subroutines

- [Add_Diffraction_Patterns](#)
- [Allocate_Diffraction_Pattern](#)
- [Calc_Background](#)
- [Delete_Noisy_Points](#)
- [Init_Err_DiffPatt](#)
- [Purge_Diffraction_Pattern](#)
- [Read_Background_File](#)
- [Read_Pattern](#)
- [Write_Pattern_FreeFormat](#)
- [Write_Pattern_XYSig](#)

Fortran Filename

CFML_Diffpatt.f90

Variables

- [Diffraction_Pattern_Type](#)
- [Err_DiffPatt](#)
- [Err_DiffPatt_Mess](#)

Diffraction_Pattern_Type

	Variable	Definition
Type ::		

Diffraction_Pattern_Type		
Character (Len=180)	Title	Identification of the pattern
Character (Len=20)	Diff_Kind	Type of radiation
Character (Len=20)	Scat_Var	x-space: 2θ , TOF, Q, s, d-spacing, $\sin\theta/\lambda$
Character (Len=20)	Instr	File type
Character (Len=512)	Filename	File name
Real (Kind=CP)	Xmin	
Real (Kind=CP)	Xmax	
Real (Kind=CP)	Ymin	
Real (Kind=CP)	Ymax	
Real (Kind=CP)	Scal	
Real (Kind=CP)	Monitor	
Real (Kind=CP)	Step	
Real (Kind=CP)	TSsamp	Sample Temperature
Real (Kind=CP)	TSset	Setting Temperature (wished temperature)
Integer	NPts	Number of points
Logical	CT_Step	Constant step
Logical	GY	Logical constants for Graphics
Logical	GYcalc	
Logical	GBgr	
Logical	GSigma	
Logical	AL_X	Logicals for Allocations
Logical	AL_Y	
Logical	AL_Ycalc	
Logical	AL_Bgr	
Logical	AL_Sigma	
Logical	AL_IStat	
Real (Kind=CP), Dimension(3)	Conv	Wavelengths or Dtt1, Dtt2 for converting to Q,d, etc
Real (Kind=CP), Dimension(:, Allocatable)	X	Scattering variable (2theta...)
Real (Kind=CP), Dimension(:, Allocatable)	Y	Experimental intensity
Real (Kind=CP), Dimension(:, Allocatable)	Sigma	observations VARIANCE (it is the square of sigma!)
Integer, Dimension(:, Allocatable)	IStat	information about the point "i"
Real (Kind=CP), Dimension(:, Allocatable)	Ycalc	Calculated intensity
Real (Kind=CP), Dimension(:, Allocatable)	Bgr	Background
Integer, Dimension(:, Allocatable)	Nd	Number of detectors contributing at the point "i"
End Type Diffraction_Pattern_Type		

Err_DiffPatt

Logical :: Err_DiffPatt

This variable is set to .TRUE. if an error occurs in procedures belonging to this module.

Err_DiffPatt_Mess

Character (Len=150) :: Err_DiffPatt_Mess

This variable contains information about the last error occurred in the procedures belonging to this module

Functions

- [Calc_FWHM_Peak](#)

Calc_FWHM_Peak

Real Function Calc_FWHM_Peak (Pat, Xi, Yi, Ybi, RLim)

Type(Diffraction_Pattern_Type)	Intent(in)	Pat	Pattern profile
Real(Kind=CP)	Intent(in)	Xi	X value on point i
Real(Kind=CP)	Intent(in)	Yi	Y value on point i
Real(Kind=CP)	Intent(in)	Ybi	Y value for Background on point i
Real(Kind=CP), Optional	Intent(in)	Rlim	Limit range in X units to search the point

Function that calculate the FWHM of a peak situated on (xi, yi) . Then the routine search the Y_m value in the range $(xi-rlim, xi+rlim)$ to obtain the FWHM.

The function return a negative values if an error is occurred during calculation.

Subroutines

- [Add_Diffraction_Patterns](#)
- [Allocate_Diffraction_Pattern](#)
- [Calc_Background](#)
- [Delete_Noisy_Points](#)
- [Init_Err_DiffPatt](#)
- [Purge_Diffraction_Pattern](#)
- [Read_Background_File](#)
- [Read_Pattern](#)
- [Write_Pattern_FreeFormat](#)
- [Write_Pattern_XYSig](#)

Add_Diffraction_Patterns

Subroutine Add_Diffraction_Patterns (Patterns, N, Active, Pat, VNorm)

Type (Diffraction_Pattern_Type), Dimension(:)	Intent(in)	Pattern s	Patterns
Integer	Intent(in)	N	Number of elements on Patterns object

Logical, Dimension(:)	Intent(in)	Active	Vector for active or not the Pattern on the List
Type(Diffraction_Pattern_Type)	Intent(out)	Pat	Pattern created
Real, optional	Intent(in)	VNorm	Normalization value in the created Pattern

Add all active Patterns

Allocate_Diffraction_Pattern

Subroutine Allocate_Diffraction_Pattern (Pat, Npts)

Type(Diffraction_Pattern_Type)	Intent(in out)	Pat	Pattern
Integer	Intent(in)	Npts	Number of points for this Pattern

Allocate the object **Pat** of type [Diffraction_Pattern_Type](#)

Calc_Background

Subroutine Calc_Background (Pat, Ncyc, Np, Xmin, Xmax)

Type(Diffraction_Pattern_Type)	Intent(in out)	Pat	Pattern
Integer	Intent(in)	Ncyc	Number of Iterations for Background calculations
Integer	Intent(in)	Np	Number of points to define the average
Real(Kind=CP), Optional	Intent(in)	Xmin	Lower limit for Background calculation
Real(Kind=CP), Optional	Intent(in)	Xmax	Upper limit for Background calculation

Calculate a Background using an iterative process according to Brückner, S. (2000). J. Appl. Cryst., 33, 977-979.

Delete_Noisy_Points

Subroutine Delete_Noisy_Points (Pat, NoisyP, FileInfo)

Type(Diffraction_Pattern_Type)	Intent(in out)	Pat	Pattern
Integer	Intent(out)	NoisyP	Number of noisy points found
Logical, optional	Intent(in)	FileInfo	If .true. then create a file containing information about the noisy points

Delete noisy points in a Pattern.

The procedure actual consider that a point is a noisy point when:

1. Point "i" is non monotonic
2. Dif1 < .or. Dif2 < sc

where:

$$sc = 8.0 * \sqrt{(ci_2 + ci_1 + cd_1 + cd_2) / 4.0}$$

$$Dif1 = cc - 2.0 * ci_1 + ci_2$$

$$Dif2 = cc - 2.0 * cd_1 + cd_2$$

and

ci2:	counts at	left-left position
ci1:	counts at	left position
cc:	counts at	current position
cd1:	counts at	right position
cd2:	counts at	right-right position

Init_Err_DiffPatt

Subroutine Init_Err_DiffPatt ()

Subroutine that initializes errors flags in **CFML_Diffraction_Patterns** module.

Purge_Diffraction_Pattern

Subroutine Purge_Diffraction_Pattern (Pat, Mode)

Type(Diffraction_Pattern_Type)	Intent(in out)	Pat	Pattern
Character (Len=*)	Intent(in)	Mode	

Deallocate components of the object **Pat**, of type [Diffraction_Pattern_Type](#) depending on the value of the **Mode** string.

At present the following MODE values are available:

MODE	Value
DATA	Purge SIGMA, YCALC, BGR, ISTAT
DATAS	Purge YCALC, BGR, ISTAT
RIETV	Purge ISTAT
GRAPH	Purge YCALC, BGR
PRF	Purge SIGMA

Read_Background_File

Subroutine Read_Background_File (BCK_File, BCK_Mode, Dif_Pat)

Character (Len=*)	Intent(in)	BCK_File	Name of the file
Character (Len=*)	Intent(in)	BCK_Mode	Options are: POL -> Polynomial INT -> Interpolation
Type(Diffraction_Pattern_Type)	Intent(in out)	Dif_Pat	Pattern

Read background from a file

Read_Pattern

Subroutine Read_Pattern (Filename, Dif_Pat, Mode)

Character (Len=*)	Intent(in)	Filena	Name of the file
-------------------	------------	--------	------------------

	me	
Type(Diffraction_Pattern_Type)	Intent(in out)	Dif_Pat Pattern
Character (Len=*) , Optional	Intent(in)	Mode D1B D20 NLS D1A D2B DMC Socabim XYSIGMA GSAS GSASTOF PANalytical (.csv, .udf, .xrdml)

Or

Subroutine Read_Pattern (Filename, Dif_Pat, NumPat, Mode)

Character (Len=*)	Intent(in)	Filena me	Name of the file
Type (Diffraction_Pattern_Type), Dimension(:)	Intent(in out)	Dif_Pat	Pattern
Integer	Intent(out)	NumPa t	Number of Patterns
Character (Len=*) , Optional	Intent(in)	Mode	Actual options are: XYSIGMA ISIS GSAS

Read one or several Patterns from a Filename

Write_Pattern_FreeFormat

Subroutine Write_Pattern_FreeFormat (Filename, Pat)

Character (Len=*)	Intent(in)	Filena me	Name of the file
Type(Diffraction_Pattern_Type)	Intent(in)	Pat	Pattern

Write a pattern in Free format (according to INSTRM=0 in the FullProf Program) in file **Filename**

Write_Pattern_XYSig

Subroutine Write_Pattern_XYSig (Filename, Pat)

Character (Len=*)	Intent(in)	Filena me	Name of the file
Type(Diffraction_Pattern_Type)	Intent(in)	Pat	Pattern

Write a pattern in X,Y,Sigma format in file **Filename**

CFML_Math_3D

Simple mathematics general utilities for 3D Systems

Variables

- [Err_Math3D](#)
- [Err_Math3D_Mess](#)

Functions

- [Cross_Product](#)
- [Determinant_A](#)
- [Determinant_V](#)
- [Invert_A](#)
- [Polyhedron_Volume](#)
- [Rotate_OX](#)
- [Rotate_OY](#)
- [Rotate_OZ](#)
- [VecLength](#)

Subroutines

- [Get_Cart_From_Cylin](#)
- [Get_Cart_From_Spher](#)
- [Get_Centroid_Coord](#)
- [Get_Cylindr_Coord](#)
- [Get_Plane_From_Points](#)
- [Get_Spheric_Coord](#)
- [Init_Error_Math3D](#)
- [Matrix_DiagEigen](#)
- [Matrix_Inverse](#)
- [Resolv_Sist_1x2](#)
- [Resolv_Sist_1x3](#)
- [Resolv_Sist_2x2](#)
- [Resolv_Sist_2x3](#)
- [Resolv_Sist_3x3](#)
- [Set_Eps](#)
- [Set_Eps_Default](#)

Fortran Filename

CFML_Math_3D.f90

Variables

- [Err_Math3D](#)
- [Err_Math3D_Mess](#)

Err_Math3D

Logical :: Err_Math3D

This variable is set to .TRUE. if an error occurs in procedures belonging to this module

Err_Math3D_Mess

Character (Len=150) :: Err_Math3D_Mess

This variable contains information about the last error occurred in the procedures belonging to this module

Functions

- [Cross_Product](#)
- [Determ_A](#)
- [Determ_V](#)
- [Invert_A](#)
- [Polyhedron_Volume](#)
- [Rotate_OX](#)
- [Rotate_OY](#)
- [Rotate_OZ](#)
- [VecLength](#)

Cross_Product

Integer / Real Function Cross_Product(U, V)

Integer, Dimension (3)	Intent(in)	U	Vector
Integer, Dimension (3)	Intent(in)	V	Vector

or

Real(Kind=SP / DP), Dimension (3)	Intent(in)	U	Vector
Real(Kind=SP / DP), Dimension (3)	Intent(in)	V	Vector

Calculates the cross product of vectors U and V. All vectors are given in cartesian components.

Determ_A

Integer / Real Function Determ_A (A)

Integer, Dimension (3,3)	Intent(in)	A	Array
---------------------------------	-------------------	----------	-------

or

Real(Kind=CP), Dimension (3,3)	Intent(in)	A	Array
---------------------------------------	-------------------	---	-------

Calculates the determinant of an Integer/real 3x3 matrix

Determ_V

Integer / Real Function Determ_V(A, B, C)

Integer, Dimension (3)	Intent(in)	A	Vector
Integer, Dimension (3)	Intent(in)	B	Vector
Integer, Dimension (3)	Intent(in)	C	Vector

or

Real(Kind=CP), Dimension (3)	Intent(in)	A	Vector
Real(Kind=CP), Dimension (3)	Intent(in)	B	Vector
Real(Kind=CP), Dimension (3)	Intent(in)	C	Vector

Calculates the determinant of the components of three vectors

Invert_A

Real Function Invert_A (A)

Real(Kind=SP / DP), Dimension (3,3)	Intent(in)	A	Array
--	-------------------	---	-------

Calculate de inverse of a real 3x3 matrix. If the routine fails, then a 0.0 matrix is returned.

Polyhedron_Volume

Real Function Polyhedron_Volume(NV, Vert, Cent)

Integer	Intent(in)	NV	Number of vertices of Polyhedra
Real(Kind=CP), Dimension (:,:)	Intent(in)	Vert	Cartesian coordinates of vertices. First index (1:NV), Second index 3
Real(Kind=CP), Dimension (3)	Intent(in)	Cent	Cartesian coordinates for a central point

Procedure to calculate the volume of polyhedral with NV vertices.

Note: It is based on Volcal program of L. W. Finger.

Rotate_OX

Real Function Rotate_OX(X, Angle)

Real(Kind=CP), Dimension (3)	Intent(in)	X	Vector
Real(Kind=CP)	Intent(in)	Angle	Angle (Degrees)

Rotation a 3D point on X axis about **Angle** degrees

Rotate_OY

Real Function ROTATE_OY(Y, ANGLE)

Real(Kind=CP), Dimension (3)	Intent(in)	Y	Vector
Real(Kind=CP)	Intent(in)	Angle	Angle (Degrees)

Rotation a 3D point on Y axis about **Angle** degrees

Rotate_OZ

Real Function Rotate_OZ (Z, Angle)

Real(Kind=CP), Dimension (3)	Intent(in)	Z	Vector
Real(Kind=CP)	Intent(in)	Angle	Angle (Degrees)

Rotation a 3D point on Z axis about **Angle** degrees

VecLength

Real Function VecLength(A, B)

Real(Kind=CP), Dimension (3,3)	Intent(in)	A	
Real(Kind=CP), Dimension (3)	Intent(in)	B	

Length of vector **B** when **A** is the Crystallographic to orthogonal matrix length

Subroutines

- [Get_Cart_From_Cylin](#)
- [Get_Cart_From_Spher](#)
- [Get_Centroid_Coord](#)
- [Get_Cylindr_Coord](#)
- [Get_Plane_From_Points](#)
- [Get_Spheric_Coord](#)
- [Init_Err_Math3D](#)
- [Matrix_DiagEigen](#)
- [Matrix_Inverse](#)
- [Resolv_Sist_1x2](#)
- [Resolv_Sist_1x3](#)
- [Resolv_Sist_2x2](#)
- [Resolv_Sist_2x3](#)
- [Resolv_Sist_3x3](#)
- [Set_Eps](#)
- [Set_Eps_Default](#)

[Get_Cart_From_Cylin](#)

Subroutine Get_Cart_From_Cylin (Rho, Phi, Zeta, X0, Mode)

Real(Kind=SP / DP)	Intent(in)	Rho	
Real(Kind=SP / DP)	Intent(in)	Phi	
Real(Kind=SP / DP)	Intent(in)	Zeta	
Real(Kind=SP / DP), Dimension(3)	Intent(out)	X0	
Character (Len=*) , Optional	Intent(in)	Mode	

Determine the Cartesian coordinates from cylindrical coordinates. If Mode='D' the angle phi is provided in Degrees.

Get_Cart_From_Spher

Subroutine Get_Cart_From_Spher (R, Theta, Phi, X0, Mode)

Real(Kind=SP / DP)	Intent(in)	R	
Real(Kind=SP / DP)	Intent(in)	THETA	
Real(Kind=SP / DP)	Intent(in)	PHI	
Real(Kind=SP / DP), Dimension(3)	Intent(out)	X0	
Character (Len=*) , Optional	Intent(in)	MODE	

Determine the Cartesian coordinates from spherical coordinates. If Mode='D' the angle phi is provided in Degrees.

Get_Centroid_Coord

Subroutine Get_Centroid_Coord (Cn, Atm_Cart, Centroid, Baricenter)

Integer	Intent(in)	Cn	Coordination Number
Real(Kind=CP), Dimension(:, :)	Intent(in)	Atm_Cart	Cartesian coordinates of atoms
Real(Kind=CP), Dimension(3)	Intent(out)	Centroid	Centroid point in Cartesian coordinates
Real(Kind=CP), Dimension(3)	Intent(out)	Baricenter	Baricenter point in Cartesian coordinates

Procedure to calculate Centroid and Baricenter of Polyhedral according to Tonci Balic-Zunic (Acta Cryst. B52, 1996, 78-81; Acta Cryst. B54, 1998, 766-773)

Get_Cylindr_Coord

Subroutine Get_Cylindr_Coord (X0, Rho, Phi, Zeta, Mode)

Real(Kind=SP / DP), Dimension(3)	Intent(in)	X0	
Real(Kind=SP / DP)	Intent(out)	Rho	
Real(Kind=SP / DP)	Intent(out)	Phi	
Real(Kind=SP / DP)	Intent(out)	Zeta	
Character (Len=*) , Optional	Intent(in)	Mode	

Determine the cylindrical coordinates from Cartesian coordinates. If Mode='D' the angle phi is provided in Degrees.

Get_Plane_From_Points

Subroutine Get_Plane_From_Points (P1, P2, P3, A, B, C, D)

Real(Kind=CP), Dimension(3)	Intent(in)	P1	
Real(Kind=CP), Dimension(3)	Intent(in)	P2	
Real(Kind=CP), Dimension(3)	Intent(in)	P3	
Real(Kind=CP)	Intent(out)	A	
Real(Kind=CP)	Intent(out)	B	
Real(Kind=CP)	Intent(out)	C	
Real(Kind=CP)	Intent(out)	D	

Calculate the implicit form of a Plane in 3D as $A * X + B * Y + C * Z + D = 0$

Get_Spheric_Coord

Subroutine Get_Spheric_Coord (X0, Ss, Theta, Phi, Mode)

Real(Kind=SP / DP), Dimension(3)	Intent(in)	X0	
Real(Kind=SP / DP)	Intent(in)	Ss	
Real(Kind=SP / DP)	Intent(in)	Theta	
Real(Kind=SP / DP)	Intent(out)	Phi	
Character (Len=*) , Optional	Intent(in)	Mode	

Determine the spheric coordinates from rectangular coordinates. If Mode='D' the angles will be done in Degrees.

Init_Err_Math3D

Subroutine Init_Err_Math3D ()

Subroutine that initializes errors flags in **CFML_Math_3D** module.

Matrix_DiagEigen

Subroutine Matrix_DiagEigen(A, V, C)

Real(Kind=CP), Dimension(3,3)	Intent(in)	A	
Real(Kind=CP), Dimension(3)	Intent(out)	V	
Real(Kind=CP), Dimension(3,3)	Intent(out)	C	

Diagonalize the matrix **A**, put eigenvalues in **V** and eigenvectors in **C**

Matrix_Inverse

Subroutine Matrix_Inverse(A, B, Ifail)

Real(Kind=CP), Dimension(3,3)	Intent(in)	A	input array
Real(Kind=CP), Dimension(3,3)	Intent(out)	B	inverse of input array $\mathbf{B}=\mathbf{A}^{-1}$
Integer	Intent(out)	Ifail	0: OK 1: Fail

Inverts a 3x3 Matrix

Resolv_Sist_1x2

Subroutine Resolv_Sist_1x2(W, T, TS, X, IX)

Integer, Dimension(2)	Intent(in)	W	input vector
Real(Kind=CP)	Intent(in)	T	input value
Real(Kind=CP), Dimension(2)	Intent(out)	TS	Fixed value of solution
Real(Kind=CP), Dimension(2)	Intent(out)	X	Fixed value for x_1 and x_2
Integer, Dimension(2)	Intent(out)	IX	1:X 2:Y 3:Z

Resolve the system though the solutions be undetermined

$$W_{11} X_1 + W_{12} X_2 = T$$

and the solution is

$$X_{\text{sol}}(i) = TS(i) + X(i) IX(i)$$

Resolv_Sist_1x3

Subroutine Resolv_Sist_1x3(W, T, TS, X, IX)

Integer, Dimension(3)	Intent(in)	W	input vector
Real(Kind=CP)	Intent(in)	T	input value
Real(Kind=CP), Dimension(3)	Intent(out)	TS	Fixed value of solution
Real(Kind=CP), Dimension(3)	Intent(out)	X	Fixed value for x_1, x_2 and x_3
Integer, Dimension(3)	Intent(out)	IX	1:X 2:Y 3:Z

Resolve the system though the solutions be undetermined

$$W_{11} X_1 + W_{12} X_2 + W_{13} X_3 = T$$

and the solution is

$$X_{\text{sol}}(i) = TS(i) + X(i) IX(i)$$

Resolv_Sist_2x2

Subroutine Resolv_Sist_2x2(W, T, TS, X, IX)

Integer, Dimension(2,2)	Intent(in)	W	input vector
Real(Kind=CP), Dimension(2)	Intent(in)	T	input value
Real(Kind=CP), Dimension(2)	Intent(out)	TS	Fixed value of solution
Real(Kind=CP), Dimension(2)	Intent(out)	X	Fixed value for x_1 and x_2
Integer, Dimension(2)	Intent(out)	IX	1:X 2:Y 3:Z

Resolve the system though the solutions be undetermined

$$W_{11} X_1 + W_{12} X_2 = T_1$$

$$W_{21} X_1 + W_{22} X_2 = T_2$$

and the solution is

$$X_{\text{sol}}(i) = TS(i) + X(i) \cdot IX(i)$$

Resolv_Sist_2x3

Subroutine Resolv_Sist_2x3(W, T, TS, X, IX)

Integer, Dimension(2,3)	Intent(in)	W	input vector
Real(Kind=CP), Dimension(2)	Intent(in)	T	input value
Real(Kind=CP), Dimension(3)	Intent(out)	TS	Fixed value of solution
Real(Kind=CP), Dimension(3)	Intent(out)	X	Fixed value for x_1, x_2 and x_3
Integer, Dimension(3)	Intent(out)	IX	1:X 2:Y 3:Z

Resolve the system though the solutions be undetermined

$$W_{11} X_1 + W_{12} X_2 + W_{13} X_3 = T_1$$

$$W_{21} X_1 + W_{22} X_2 + W_{23} X_3 = T_2$$

and the solution is

$$X_{\text{sol}}(i) = TS(i) + X(i) \cdot IX(i)$$

Resolv_Sist_3x3

Subroutine Resolv_Sist_3x3(W, T, TS, X, IX)

Integer, Dimension(3,3)	Intent(in)	W	input vector
Real(Kind=CP), Dimension(3)	Intent(in)	T	input value
Real(Kind=CP), Dimension(3)	Intent(out)	TS	Fixed value of solution
Real(Kind=CP), Dimension(3)	Intent(out)	X	Fixed value for x_1, x_2 and x_3
Integer, Dimension(3)	Intent(out)	IX	1:X 2:Y 3:Z

Resolve the system though the solutions be undetermined

$$W_{11} X_1 + W_{12} X_2 + W_{13} X_3 = T_1$$

$$W_{21} X_1 + W_{22} X_2 + W_{23} X_3 = T_2$$

$$W_{31} X_1 + W_{32} X_2 + W_{33} X_3 = T_3$$

and the solution is

$$X_{\text{sol}}(i) = TS(i) + X(i) \cdot IX(i)$$

Set_Eps

Subroutine Set_Eps (NewEps)

Real(Kind=CP)	Intent(in)	NewEp	
----------------------	-------------------	-------	--



Sets an internal **Eps** variable on **CFML_MATH_3D** module to the value **NewEps**

[Set_Eps_Default](#)

Subroutine Set_Eps_Default ()

Sets the internal **Eps** variable to the default value

Default: 10^{-5}

CFML_Optimization_General

Module containing several algorithms for global and local optimization.

Variables

- [Opt_Conditions_Type](#)
- [Err_Optim](#)
- [Err_Optim_Mess](#)

Subroutines

- [CG_Quasi_Newton](#)
- [CSendes_Global](#)
- [Init_Err_Optim](#)
- [Init_Opt_Conditions](#)
- [Local_Min_DFP](#)
- [Local_Min_Rand](#)
- [Local_Optimize](#)
- [Nelder_Mead_Simplex](#)
- [Set_OptT_Conditions](#)
- [Write_Optimization_Conditions](#)

Fortran Filename

CFML_Optimization.f90

Variables

- [Opt_Conditions_Type](#)
- [Err_Optim](#)
- [Err_Optim_Mess](#)

[Opt_Conditions_Type](#)

	Variable	Definition
Type :: Opt_Conditions_Type		
Character(Len=20)	Method	String describing the Method
Integer	NMeth	Type of method used
Integer	NPar	Number of free parameters
Integer	MXFun	Maximum number function calls
Integer	IOut	Printing parameter
Integer	Loops	Useful for SIMPLEX method
Integer	IQuad	Useful for SIMPLEX method. If iquad/= 0 fitting to a quadratic
Integer	NFlag	Flag value states which condition caused the exit of the optimization subroutine
Integer	IFun	Total number of function and gradient evaluations
Integer	Iter	Total number of search directions used in the algorithm
Real(Kind=CP)	Eps	Convergence parameter
Real(Kind=CP)	Acc	User supplied estimate of machine accuracy
End Type Opt_Conditions_Type		

Values for **Method** are:

Value	Description
Conjugate_Gradient	
BFGS_Quasi_Newton	
Simplex	
DFP_NO_Derivatives	
Global_CSendes	
Local_Random	
UniRandi	

Values for **NMeth** are:

Value	Description
0	Conjugate Gradient
1	BFGS method

Values for **IOut** are:

Value	Description
0	No printing for Quasi_Newton & Conjugate Gradient Partial printing for Simplex (<0 no printing)
>0	Printing each iout iterations/evaluations

Values for **NFlag** are:

Value Description

- 0 The algorithm has converged
- 1 The maximum number of function evaluations have been used
- 2 The linear search has failed to improve the function value. This is the usual exit if either the function or the gradient is incorrectly coded.
- 3 The search vector was not a descent direction. This can only be caused by round off, and may suggest that the convergence criterion is too strict.

Values for **Eps** are:

Value Description

- 10^{-6} Convergence occurs when the norm of the gradient is less than or equal to EPS times the maximum of one and the norm of the vector X

Values for **Acc** are:

Value Description

- 10^{-20} is a user supplied estimate of machine accuracy. A linear search is unsuccessfully terminated when the norm of the step size becomes smaller than ACC. in practice, ACC= 10^{-20} has proved satisfactory. This is the default value.
- 10^{-6} For Simplex method the meaning is different (see EPS parameter) and this should be changed to 10^{-6}

This Type has been introduced to simplify the call to optimization procedures. It contains the optimization parameters useful for different algorithms. All Integer components are initialized to zero and the real components are initialized as indicated below.

A variable of this type should be defined by the user and all their input parameters (in) must be provided before calling the procedures. On output from the procedure the (out) items are provided for checking.

Err_Optim

Logical :: Err_Optim

This variable is set to .TRUE. if an error occurs in procedures belonging to this module

Err_Optim_Mess

Character (Len=150) :: Err_Optim_Mess

This variable contains information about the last error occurred in the procedures belonging to this module

Subroutines

- [CG_Quasi_Newton](#)
- [CSendes_Global](#)
- [Init_Err_Optim](#)
- [Init_Opt_Conditions](#)
- [Local_Min_DFP](#)
- [Local_Min_Rand](#)
- [Local_Optimize](#)
- [Nelder_Mead_Simplex](#)

- [Set_OptT_Conditions](#)
- [Write_Optimization_Conditions](#)

CG_Quasi_Newton

Subroutine CG_Quasi_Newton (Model_Funct, N, X, F, G, C, Ipr)

Defined Subroutine Model_Funct		Model_Funct		
Integer	Intent(in)	N		The number of variables in the function to be minimized.
Real(Kind=CP), Dimension (N)	Intent(in out)	X	in: out:	Must contain an initial estimate supplied by the user X will hold the best estimate to the minimizer obtained
Real(Kind=CP)	Intent(out)	F		Contain the lowest value of the object function obtained
Real(Kind=CP), Dimension (N)	Intent(out)	G		G =(g(1),...g(n)) will contain the elements of the gradient of F evaluated at the point contained in X=(x(1),...x(N))
Type (Opt_Conditions_Type)	Intent(in out)	C		Conditions for the algorithm
Integer, Optional	Intent(in)	Ipr		Logical unit for printing if the parameter C% lout /= 0.

where

Subroutine Model_Funct (N, X, F, G)

Integer	Intent(in)	N	Number of free parameters
Real(Kind=CP), Dimension (:)	Intent(in)	X	Variables
Real(Kind=CP)	Intent(out)	F	Value of Model
Real(Kind=CP), Dimension (:)	Intent(out)	G	Gradiente of F

End Subroutine Model_Funct

CG_Quasi_Newton minimizes an unconstrained nonlinear scalar valued function of a vector variable X either by the BFGS variable metric algorithm or by a beale restarted conjugate gradient algorithm.

(BFGS: Broyden, Fletcher, Goldfarb and Shanno. ACM TRANSACTIONS ON MATHEMATICAL SOFTWARE 6 (DECEMBER 1980), 618-622).

CSendes_Global

Subroutine CSendes_Global(Model_Funct, Mini, Maxi, NParm, NSampl, NSel, NSig, X0, NC, F0, Ipr, Mode)

Defined Subroutine Model_Funct		Model_Funct		Dummy name of the objective function to be optimized
Real(Kind=CP), Dimension (:)	Intent(in)	Mini		Vector of length NPARM containing the lower bounds
Real(Kind=CP), Dimension (:)	Intent(in)	Maxi	in: out:	Vector of length NPARM containing the upper bounds
Integer	Intent(in)	NParm		Number of Parameters

Integer	Intent(in out)	NSampl		Number of sample points to be drawn uniformly in one cycle. Suggested value is 100*NPARM
Integer	Intent(in out)	NSel		Number of best points selected from the actual sample. The suggested value is twice the expected number of local minima.
Integer	Intent(in)	NSig		The accuracy required in the parameter estimates. This convergence criterion is satisfied if on two successive iterations the parameter estimates agree, component by component, to nsig digits. The suggested value is 6.
Real(Kind=CP), Dimension (:,:)	Intent(in out)	X0		output matrix (NPARM x NC) containing NC local minimizers found
Integer	Intent(out)	NC		Number of different Local Minimizers Found
Real(Kind=CP), Dimension (:)	Intent(in out)	F0		output vector of NC objective function values. F0(I) Belongs to the parameters X0(1,I), X0(2,I), ..., X0(NPARM,I)
Integer	Intent(in)	Ipr		Printing information
Integer, Optional	Intent(in)	Mode		If present the routine LOCAL_Min_DFP is replaced by LOCAL_Min_RAND

where

Subroutine Model_Funct (NParm, X, F)

Integer	Intent(in)	NParm	Number of free parameters
Real(Kind=CP), Dimension (:)	Intent(in)	X	Variables
Real(Kind=CP)	Intent(out)	F	Value of Model

End Subroutine Model_Funct

Global optimization procedure using the Boender-Timmer-Rinnoy Kan algorithm (Local Search Method)

Global optimization is a part of nonlinear optimization, it deals with problems with (possibly) several local minima. The presented method is stochastic (i.e. not deterministic). The framework procedure, the CSendes_Global routine gives a computational evidence, that the best local minimum found is with high probability the global minimum. This routine calls a local search routine, and a routine for generating random numbers.

Let $F(X)$ be a real function of NParm parameters and we are looking for parameter values $X(I)$ from the given intervals $[Min(I), MAX(I)]$ for each $I = 1, 2, \dots, NPARM$. The problem is to determine such a point X^* , that the function value $F(X)$ is greater than or equal to $F(X^*)$ for every X in the NPARM-dimensional interval specified by $Min(I)$'s and $MAX(I)$'s.

Init_Err_Optim

Subroutine Init_Err_Optim ()

Subroutine that initializes errors flags in **CFML_Optimization_General** module.

Init_Opt_Conditions

Subroutine Init_Opt_Conditions (Opt)

Type(Opt_Conditions_Type)	Intent(out)	OPT	Opt Conditions
----------------------------------	--------------------	-----	----------------

Initialize the variable **OPT**. Default values are:

Parameter	Value
METHOD	SIMPLEX
NMETH	0
NPAR	0
MXFUN	1000
lout	2000
LOOPS	1
IQUAD	0
NFLAG	0
IFUN	0
ITER	0
EPS	10^{-6}
ACC	10^{-20}

Local_Min_DFP

Subroutine Local_Min_DFP(Model_Funct, N, X, F, C, Mini, Maxi, lpr)

Defined Subroutine Model_Funct		Model_Funct		
Integer	Intent(in)	N		The number of variables in the function to be minimized.
Real(Kind=CP), Dimension (:)	Intent(in out)	X	in: out:	Must contain an initial estimate supplied by the user The Final Parameter Estimates As Determined By Local
Real(Kind=CP)	Intent(out)	F		The value of the Function at the final parameter estimates
Type (Opt_Conditions_Type)	Intent(in out)	C		Conditions for the algorithm
Real(Kind=CP), Dimension (:)	Intent(in)	Mini		Lower bounds of the parameters
Real(Kind=CP), Dimension (:)	Intent(in)	Maxi		Upper bounds of the parameters
Integer, Optional	Intent(in)	lpr		Logical unit for printing if the parameter C% lout /= 0.

Subroutine Model_Funct (N, X, F)

Integer	Intent(in)	N	Number of free parameters
Real(Kind=CP), Dimension (:)	Intent(in)	X	Variables
Real(Kind=CP)	Intent(out)	F	Value of Model
End Subroutine Model_Funct			

Provides the minimum of a function of N variables using a Quasic-Newton Method. If there is no stable minimum in the given region the algorithm may fail.

The important parameters for the algorithm are stored in the C-variable on input the components C%Eps and C%MxFun are needed (a call to Init_Opt_Conditions is enough to provide sensible values). On output the component C%ifun is updated.

Local_Min_Rand

Subroutine Local_Min_Rand (Model_Funct, N, X, F, C, Mini, Maxi)

Defined Subroutine Model_Funct		Model_Funct		
Integer	Intent(in)	N		The number of variables in the function to be minimized.
Real(Kind=CP), Dimension (:)	Intent(in out)	X	in: out:	Must contain an initial estimate supplied by the user The Final Parameter Estimates As Determined By Local
Real(Kind=CP)	Intent(out)	F		The value of the Function at the final parameter estimates
Type (Opt_Conditions_Type)	Intent(in out)	C		Conditions for the algorithm
Real(Kind=CP), Dimension (:), Optional	Intent(in)	Mini		Lower bounds of the parameters
Real(Kind=CP), Dimension (:), Optional	Intent(in)	Maxi		Upper bounds of the parameters

Subroutine Model_Funct (N, X, F)

Integer	Intent(in)	N	Number of free parameters
Real(Kind=CP), Dimension (:)	Intent(in)	X	Variables
Real(Kind=CP)	Intent(out)	F	Value of Model

End Subroutine Model_Funct

Local_Optimize

Subroutine Local_Optimize (Model_Funct, X, F, C, G, Mini, Maxi, V, Ipr)

Defined Subroutine Model_Funct		Model_Funct		
Real(Kind=CP), Dimension (:)	Intent(in out)	X	in: out:	Must contain an initial estimate supplied by the user X will hold the best estimate to the minimizer obtained
Real(Kind=CP)	Intent(out)	F		Contain the lowest value of the object function obtained
Type (Opt_Conditions_Type)	Intent(in out)	C		Conditions for the algorithm
Real(Kind=CP), Dimension (:), Optional	Intent(in out)	G		G =(g(1),...g(n)) will contain the elements of the gradient of F evaluated at the point contained in X=(x(1),...x(N)) For SIMPLEX it contains the step values
Real(Kind=CP), Dimension (:), Optional	Intent(in out)	Mini		Lower range
Real(Kind=CP), Dimension (:), Optional	Intent(in out)	Maxi		Upper range

Real(Kind=CP), Dimension (:), Optional	Intent(out)	V		For SIMPLEX it contains the sigma of parameters
Integer, Optional	Intent(in)	Ipr		Logical unit for printing if the parameter C% lout /= 0.

Subroutine Model_Funct (N, X, F, G)

Integer	Intent(in)	N	Number of free parameters
Real(Kind=CP), Dimension (:)	Intent(in)	X	Variables
Real(Kind=CP)	Intent(out)	F	Value of Model
Real(Kind=CP), Dimension (:), Optional	Intent(out)	G	Gradiente of F

End Subroutine Model_Funct

Wraper for selection an optimization method of the function Model_Funct.

The list of free parameters are provided in the vector X (in out), the value of the function F, and eventually the gradient G, are output variables. The optimization conditions in the variable C should be provided for selecting the optimization algorithm

Nelder_Mead_Simplex**Subroutine Nelder_Mead_Simplerx (Model_Funct, Nop, P, Step, Var, Func, C, Ipr)**

Defined Subroutine Model_Funct		Model_Funct		
Integer	Intent(in)	Nop		The number of variables in the function to be minimized.
Real(Kind=CP), Dimension (:)	Intent(in out)	P	in: out:	starting values of parameters final values of parameters
Real(Kind=CP), Dimension (:)	Intent(in out)	Step	in: out:	initial step sizes final step sizes
Real(Kind=CP), Dimension (:)	Intent(out)	Var		Contains the diagonal elements of the inverse of the information matrix
Real(Kind=CP)	Intent(out)	Func		The function value corresponding to the final parameter values.
Type (Opt_Conditions_Type)	Intent(in out)	C		Conditions for the algorithm
Integer, Optional	Intent(in)	Ipr		Logical unit for printing if the parameter C% lout /= 0.

Subroutine Model_Funct (N, X, F, G)

Integer	Intent(in)	N	Number of free parameters
Real(Kind=CP), Dimension (:)	Intent(in)	X	Variables
Real(Kind=CP)	Intent(out)	F	Value of Model
Real(Kind=CP), Dimension (:), Optional	Intent(out)	G	Gradiente of F

End Subroutine Model_Funct

Procedure for function minimization using the SIMPLEX method.

Optimization Conditions type with the following components:

Parameter	Description
C%MxFun	The maximum number of function evaluations allowed. Say, 20 times the number of parameters
C%IOut	< 0 No printing = 0 Printing of parameter values and the function value after initial evidence of convergence > 0 As for C%lout = 0 plus progress reports after every C%lout evaluations, plus printing for the initial simplex
C%Eps	Stopping criterion. The criterion is applied to the standard deviation of the values of FUNC at the points of the simplex
C%Loops	The stopping rule is applied after every NLOOP function evaluations. Normally NLOOP should be slightly greater than NOP, say NLOOP = 2*NOP.
C%IQuad	= 1 If fitting of a quadratic surface is required = 0 If not
	The fitting of a quadratic surface is strongly recommended, provided that the fitted function is continuous in the vicinity of the minimum. It is often a good indicator of whether a premature termination of the search has occurred.
C%Acc	criterion for expanding the simplex to overcome rounding errors before fitting the quadratic surface. The simplex is expanded so that the function values at the points of the simplex exceed those at the supposed minimum by at least an amount SIMP
IC%NFlag	= 0 for successful termination = 1 If maximum no. of function evaluations exceeded = 2 If information matrix is not +ve semi-definite = 3 if NOP < 1 = 4 if C%LOOPS < 1

Other considerations:

P, Step and Var (if C%Iquad = 1) must have dimension at least **Nop** in the calling program.

For details, see Nelder & Mead, The Computer Journal, January 1965. Programmed by D.E.Shaw, CSIRO, Division of Mathematics & Statistics P.O. BOX 218, Lindfield, N.S.W. 2070

[Set_Opt_Conditions](#)

Subroutine Set_Opt_Conditions (N, File_Lines, Opt)

Integer	Intent(in)	N	Logical unit for writing
Character(Len=*) , Dimension(:)	Intent(in)	File_Lines	info
Type(Opt_Conditions_Type)	Intent(out)	Opt	Variable

Get the optimization conditions from a list of text lines obtained from the input file

[Write_Optimization_Conditions](#)

Subroutine Write_Optimization_Conditions (lpr, C)

Integer	Intent(in)	lpr	Logical unit for writing
Type(Opt_Conditions_Type)	Intent(in)	C	Opt Conditions

Subroutine for writing in unit **lpr** the [Opt_Conditions_Type](#) variable **C**

[CFML_Optimization_LSQ](#)

Module implementing several algorithms for non-linear least-squares. At present only the Levenberg-

Marquardt method is implemented.

There are two high level procedures contained in CFML_Optimization_LSQ based in the Levenberg-Marquardt method. The first procedure, called [Marquardt_Fit](#), is a simple implementation of the method and the second one is a Fortran 90 version of the MINPACK Fortran 77 LMXXX subroutines, accessible through the general name [Levenberg_Marquardt_Fit](#). The second one is, in principle, more robust for general LSQ problems.

Variables

- [Err_LSQ](#)
- [Err_LSQ_Mess](#)
- [Info_LSQ_Mess](#)

Functions

- [FChiSQ](#)

Subroutines

- [Info_LSQ_Output](#)
- [Levenberg_Marquardt_Fit](#)
- [Marquardt_Fit](#)

Fortran Filename

CFML_Optimization.f90

Variables

- [Err_LSQ](#)
- [Err_LSQ_Mess](#)
- [Info_LSQ_Mess](#)

[Err_LSQ](#)

Logical :: Err_LSQ

This variable is set to .TRUE. if an error occurs in procedures belonging to this module

[Err_LSQ_Mess](#)

Character (Len=150) :: Err_LSQ_Mess

This variable contains information about the last error occurred in the procedures belonging to this module

Info_LSQ_Mess

Character (Len=150) :: Info_LSQ_Mess

Character variable containing the information message associated to the exit parameter Info of the Levenberg_Marquardt_Fit procedure.

Functions

- [FChiSQ](#)

FChiSQ

Real Function FChiSQ (Nfr, NObs, Y, W, Yc)

Integer	Intent(in)	Nfr	
Integer	Intent(in)	NObs	
Real(Kind=CP), Dimension (:)	Intent(in)	Y	
Real(Kind=CP), Dimension (:)	Intent(in)	W	
Real(Kind=CP), Dimension (:)	Intent(in)	Yc	

Evaluate reduced χ^2

Subroutines

- [Info_LSQ_Output](#)
- [Levenberg_Marquardt_Fit](#)
- [Marquardt_Fit](#)

Info_LSQ_Output

Subroutine Info_LSQ_Output (Chi2, FL, NObs, X, Y, Yc, W, Lun, C, VS, Out_Obscal)

Real(Kind=CP)	Intent(in)	Chi2	Final χ^2
Real(Kind=CP)	Intent(in)	FL	Final Marquardt lambda
Real(Kind=CP)	Intent(in)	NObs	Number of data points
Real(Kind=CP), Dimension(:)	Intent(in)	X	Array with X of Data points
Real(Kind=CP), Dimension(:)	Intent(in)	Y	Array with Y of Data points
Real(Kind=CP), Dimension(:)	Intent(in)	Yc	Array with calculated data points
Real(Kind=CP), Dimension(:)	Intent(in)	W	Array with weight factors
Integer	Intent(in)	Lun	Logical unit for output
Type (LSQ_Conditions_Type)	Intent(in)	C	Conditions of the refinement
Type (LSQ_State_Vector_Type)	Intent(in)	VS	State vector (parameters of the model)
Character (Len=*), Optional	Intent(in)	Out_Obscal	If present the vectors X,Y,Yc, $\sigma=\sqrt{1/w}$ are output in a file called LM_fit.xy

Subroutine for output information at the end of refinement procedure

Levenberg_Marquardt_Fit

Subroutine Levenberg_Marquardt_Fit (Model_Funct, M, N, X, FVec, Tol, Info, lwa)

Defined Subroutine Model_Funct		Model_Funct		Name of the subroutine
Integer	Intent(in)	M		Positive number of functions
Integer	Intent(in)	N		Positive number of variables (n <= m)
Real(Kind=CP), Dimension (:)	Intent(in out)	X	in: out:	Vector of length N initial estimate of the solution vector Final estimate of the solution vector
Real(Kind=CP), Dimension (:)	Intent(out)	FVec		Vector of length M contains the functions evaluated at the output X.
Real(Kind=CP)	Intent(in)	Tol		Termination occurs when both the actual and predicted relative reductions in the sum of squares are at most TOL. Therefore, TOL measures the relative error desired in the sum of squares.
Integer	Intent(out)	Info	< 0 = 0 = 1 = 2 = 3 = 4 = 5 = 6 = 7	If the user has terminated execution Improper input parameters. Relative error in the sum of squares is at most TOL Algorithm estimates that the relative error between x and the solution is at most tol. Conditions for info = 1 and info = 2 both hold. FVEC is orthogonal to the columns of the jacobian to machine precision. Number of calls to fcn has reached or exceeded maxfev. Tol is too small. no further reduction in the sum of squares is possible. Tol is too small. no further improvement in the approximate solution x is possible.
Integer, Dimension(:)	Intent(out)	lwa		is an Integer work array of length n

Or

Subroutine Levenberg_Marquardt_Fit (Model_Funct, M, C, VS, Chi2, InfOut, Residuals)

Defined Subroutine Model_Funct		Model_Funct	Name of the subroutine calculating YC(i) for point X(i)
Integer	Intent(in)	M	Number of observations
Type (LSQ_Conditions_Type)	Intent(in out)	C	Conditions of refinement
Type (LSQ_State_Vector_Type)	Intent(in out)	VS	State vector for the model calculation
Real(Kind=CP)	Intent(out)	Chi2	Final reduced Chi-2
Character(Len=*)	Intent(out)	InfOut	information about the refinement
Real(Kind=CP), Dimension(:), Optional	Intent(out)	Residuals	Residuals vector

and

Subroutine Model_Funct (M, N, X, FVec, IFlag)

Integer	Intent(in)	M	Number of observations
Integer	Intent(in)	N	Number of Free parameters
Real(Kind=CP), Dimension (:)	Intent(in)	X	Array with the values of free parameters: X (1:N)
Real(Kind=CP), Dimension (:)	Intent(in out)	FVec	Array of residuals FVEC=(y-yc)/sig : FVEC (1:M)
Integer	Intent(in out)	IFlag	=1 calculate only FVEC without changing FJAC =2 calculate only FJAC keeping FVEC fixed

End Subroutine Model_Funct

or

Subroutine Levenberg_Marquardt_Fit (Model_Funct, M, N, X, FVec, FJac, Tol, Info, lwa)

Defined Subroutine Model_Funct		Model_Funct	Name of the subroutine
Integer	Intent(in)	M	Positive number of functions
Integer	Intent(in)	N	Positive number of variables (n <= m)
Real(Kind=CP), Dimension (:)	Intent(in out)	X	Vector of length N initial estimate of the solution vector out: Final estimate of the solution vector
Real(Kind=CP), Dimension (:)	Intent(out)	FVec	Vector of length M contains the functions evaluated at the output X
Real(Kind=CP), Dimension (:, :)	Intent(in out)	FJac	Is an output m by n array. the upper n by n submatrix of fjac contains an upper triangular matrix r with diagonal elements of non increasing magnitude such that $p^T * (jac * jac)^T = r^T * r,$ where p is a permutation matrix and jac is the final calculated Jacobian. Column j of p is column ipvt(j) (see below) of the identity matrix. The lower trapezoidal part of fjac contains information generated during the computation of r.
Real(Kind=CP)	Intent(in)	Tol	Termination occurs when both the actual and predicted relative reductions in the sum of squares are at most TOL. Therefore, TOL measures the relative error desired in the sum of squares.

Integer	Intent(out)	Info	< 0 = 0 = 1 = 2 = 3 = 4 = 5 = 6 = 7	If the user has terminated execution Improper input parameters. Relative error in the sum of squares is at most TOL Algorithm estimates that the relative error between x and the solution is at most tol. Conditions for info = 1 and info = 2 both hold. FVEC is orthogonal to the columns of the jacobian to machine precision. Number of calls to fcn has reached or exceeded maxfev. Tol is too small. no further reduction in the sum of squares is possible. Tol is too small. no further improvement in the approximate solution x is possible.
Integer, Dimension(:)	Intent(out)	Iwa		is an Integer work array of length n

or

Subroutine Levenberg_Marquardt_Fit (Model_Funct, M, C, VS, Chi2, CalDer, InfOut, Residuals)

Defined Subroutine Model_Funct		Model_Funct	Name of the subroutine calculating YC(i) for point X(i)
Integer	Intent(in)	M	Number of observations
Type (LSQ_Conditions_Type)	Intent(in out)	C	Conditions of refinement
Type (LSQ_State_Vector_Type)	Intent(in out)	VS	State vector for the model calculation
Real(Kind=CP)	Intent(out)	Chi2	Final reduced Chi-2
Logical	Intent(in)	CalDer	logical (should be .true.) used only for purposes of making unambiguous the generic procedure
Character(Len=*)	Intent(out)	InfOut	information about the refinement
Real(Kind=CP), Dimension(:), Optional	Intent(out)	Residuals	Residuals vector

and

Subroutine Model_Funct (M, N, X, FVec, FJac, IFlag)

Integer	Intent(in)	M	Number of observations
Integer	Intent(in)	N	Number of Free parameters
Real(Kind=CP), Dimension (:)	Intent(in)	X	Array with the values of free parameters: X (1:N)
Real(Kind=CP), Dimension (:)	Intent(in out)	FVec	Array of residuals FVEC=(y-yc)/sig : FVEC (1:M)
Real(Kind=CP), Dimension (:,:)	Intent(out)	FJac	Jacobian DFVEC/DX(i,j)=DFVEC(i)/DX(j): FJAC(1:m,1:n)
Integer	Intent(in out)	IFlag	=1 calculate only FVEC without changing FJAC =2 calculate only FJAC keeping FVEC

fixed

End Subroutine Model_Funct

The purpose of this routine is to minimize the sum of the squares of M nonlinear functions in N variables by a modification of the Levenberg-Marquardt algorithm. The user must provide a subroutine which calculates the functions. The Jacobian is then calculated by a forward-difference approximation.

Marquardt_Fit**Subroutine Marquardt_Fit (Model_Funct, X, Y, W, Yc, NObs, C, VS, Ipr, Chi2, Scroll_Lines)**

Defined Subroutine Model_Funct		Model_Func t	Name of the subroutine calculating Yc(i) for point X(i)
Real(Kind=CP), Dimension (:)	Intent(in)	X	Vector of x-values
Real(Kind=CP), Dimension (:)	Intent(in)	Y	Vector of observed y-values
Real(Kind=CP), Dimension (:)	Intent(in out)	W	Vector of weights-values (1/variance)
Real(Kind=CP), Dimension (:)	Intent(out)	Yc	Vector of calculated y-values
Integer	Intent(in)	NObs	Number of effective components of X, Y, W, YC
Type (LSQ_Conditions_Type)	Intent(in out)	C	Conditions for the algorithm
Type (LSQ_State_Vector_Type)	Intent(in out)	VS	State vector for the model calculation
Integer	Intent(in)	Ipr	Logical unit for writing
Real(Kind=CP)	Intent(out)	Chi2	Reduced Chi-2
Character(Len=*), Dimension(:), Optional	Intent(out)	Scroll_Lines	If present, part of the output is stored in this text for treatment in the calling program

and**Subroutine Model_Funct (Iv, Xv, Ycalc, Aa, Der)**

Integer	Intent(in)	Iv	Number of the component "i"
Real(Kind=CP)	Intent(in)	Xv	Value of X(i)
Real(Kind=CP)	Intent(out)	Ycalc	Value of yc at point x(i)
Real(Kind=CP), Dimension (:)	Intent(in)	Aa	Vector of parameters
Real(Kind=CP), Dimension (:), Optional	Intent(out)	Der	Derivatives of the function w.r.t. free parameters

End Subroutine Model_Funct**or****Subroutine Marquardt_Fit (Model_Funct, D, C, VS, Ipr, Chi2, Scroll_Lines)**

Defined Subroutine Model_Funct		Model_Func t	Name of the subroutine calculating YC(i) for point X(i)
---	--	-----------------	--

Type(LSQ_Data_Type)	Intent(in out)	D	LSQ Data Type
Type(LSQ_Conditions_Type)	Intent(in out)	C	Conditions for the algorithm
Type(LSQ_State_Vector_Type)	Intent(in out)	VS	State vector for the model calculation
Integer	Intent(in)	Ipr	Logical unit for writing
Real(Kind=CP)	Intent(out)	Chi2	Reduced Chi-2
Character(Len=*), Dimension(:), Optional	Intent(out)	Scroll_Lines	If present, part of the output is stored in this text for treatment in the calling program

and

Subroutine Model_Funct (Iv, Xv, Ycalc, VSA, CalDer)

Integer	Intent(in)	Iv	Number of the component "i"
Real(Kind=CP)	Intent(in)	Xv	Value of X(i)
Real(Kind=CP)	Intent(out)	Ycalc	Value of yc at point x(i)
Type (LSQ_State_Vector_Type)	Intent(in out)	VSA	LSQ State vector type
Logical, Optional	Intent(in)	CalDer	If present, derivatives, stored in Vsa%dpv, are calculated

End Subroutine Model_Funct

Procedure for applying the Levenberg-Marquardt method for Least-Squares.

The user must provide a model function according to the interface above. The model function should use at least some of the public variables of the present module in order to set the derivatives with respect to the model parameters.

For using this routine, the user must provide:

Parameter	Description
C%lcyc	Number of cycles
C%lw	type of weighting scheme
C%Constr	constraint conditions
C%Percent	constraint conditions

Parameter	Description
VS%NP	number of model parameters
VS%NamPar	name for all possible parameters of the model
VS%PV	A set of flags values to refine or fix the parameters

The values of all possible refinable parameters are stored in the array VS%PV.

The derivatives must be calculated within Model_Funct, by using the array VS%Der. The actual refined parameters Aa are selected from the larger VS%PV array by the Integer array of flags: VS%Code. A value VS%CODE(j)=1 means that the j-th parameter is to be varied. A value VS%CODE(k)=0 means that the k-th parameter is kept fixed through the refinement cycles.

It is recommended that Model_Funct be stored in a Module. The Integer IV (counter of the loop for all observations for which the subroutine is invoked) is passed because in many cases part of the calculations and derivatives may be calculated only for iv=1 and stored in local variables that should have the SAVE attribute or be private module variables accessible by host association. The only output values of the subroutine are ycalc and Vsa%dpv(:) that vary for each "iv" point. In this version of the algorithm the derivatives with respect to the free parameters in Model_Funct should be calculated before exiting by using the following loop:

```
vs%dpv=0.0                                !Should be always be initialized for
each point
  if (present(calder)) then
    do i=1,vs%np
      if (vs%code(i) == 0) cycle
      vs%dpv(i) = derivative_wrt(i) !symbolic form for calculating the
derivative w.r.t. parameter "i"
    end do                                     !at the current point xv (observation
"iv")
  end if
```

CFML_Scattering_Chemical_Tables

Tabulated information about atomic chemical and scattering data.

Parameters

- [Num_Chem_Info](#)
- [Num_Delta_Fp](#)
- [Num_Mag_Form](#)
- [Num_Mag_J2](#)
- [Num_Mag_J4](#)
- [Num_Mag_J6](#)
- [Num_Xray_Form](#)

Variables

- [Anomalous_SC_Type](#)
- [Chem_Info_Type](#)
- [Magnetic_Form_Type](#)
- [Xray_Form_Type](#)
- [Xray_Wavelength_Type](#)

- [Anomalous_SCFac](#)
- [Chem_Info](#)
- [Magnetic_Form](#)
- [Magnetic_J2](#)
- [Magnetic_J4](#)
- [Magnetic_J6](#)
- [Xray_Form](#)
- [Xray_Wavelengths](#)

Subroutines

- [Get_Atomic_Mass](#)
- [Get_ChemSymb](#)
- [Get_Covalent_Radius](#)
- [Get_Fermi_Length](#)
- [Get_Ionic_Radius](#)
- [Remove_Chem_Info](#)
- [Remove_Delta_Fp_Fpp](#)
- [Remove_Magnetic_Form](#)
- [Remove_Xray_Form](#)
- [Set_Chem_Info](#)
- [Set_Delta_Fp_Fpp](#)
- [Set_Magnetic_Form](#)
- [Set_Xray_Form](#)

Fortran Filename

CFML_Chem_Scatt.f90

Parameters

- [Num_Chem_Info](#)
- [Num_Delta_Fp](#)
- [Num_Mag_Form](#)
- [Num_Mag_J2](#)
- [Num_Mag_J4](#)
- [Num_Mag_J6](#)
- [Num_Xray_Form](#)

Num_Chem_Info

Integer, Parameter :: Num_Chem_Info=108

Number of total [Chem_Info](#) Data

Num_Delta_Fp

Integer, Parameter :: Num_Delta_Fp=98

Number of total $\Delta F'$, $\Delta F''$ Data defined in [Anomalous_SCFac](#).

Num_Mag_Form

Integer, Parameter :: Num_Mag_Form=117

Number of total Magnetic_Form Data

Num_Mag_J2

Integer, Parameter :: Num_Mag_J2=96

Number of <j2> Magnetic_Form Data defined in [Magnetic_J2](#)

Num_Mag_J4

Integer, Parameter :: Num_Mag_J4=96

Number of <j4> Magnetic_Form Data defined in [Magnetic_J4](#)

Num_Mag_J6

Integer, Parameter :: Num_Mag_J6=38

Number of <j6> Magnetic_Form Data defined in [Magnetic_J6](#)

Num_Xray_Form

Integer, Parameter :: Num_Xray_Form=214

Number of total Xray_Form Data defined in [Xray_Form](#)

Variables

- [Anomalous_SC_Type](#)
- [Chem_Info_Type](#)
- [Magnetic_Form_Type](#)
- [Xray_Form_Type](#)
- [Xray_Wavelength_Type](#)

- [Anomalous_SCFac](#)
- [Chem_Info](#)
- [Magnetic_Form](#)
- [Magnetic_J2](#)
- [Magnetic_J4](#)
- [Magnetic_J6](#)
- [Xray_Form](#)
- [Xray_Wavelengths](#)

Anomalous_SC_Type

	Variable	Definition
Type :: Anomalous_SC_Type		
Character(Len=2)	Symb	Symbol of the Chemical species
Real(Kind=CP), Dimension (5)	Fp	Delta Fp
Real(Kind=CP), Dimension (5)	Fpp	Delta Fpp
End Type Anomalous_SC_Type		

Chem_Info_Type

	Variable	Definition
Type :: Chem_Info_Type		
Character(Len=2)	Symb	Symbol of the Chemical species
Character(Len=12)	Name	Name of the Element
Integer	Z	Atomic Number
Real(Kind=CP)	Atwe	Atomic weight
Real(Kind=CP)	RConv	Covalent Radio
Real(Kind=CP)	RWaals	Van der Waals Radio
Real(Kind=CP)	VAtm	Atomic volumen
Integer, Dimension (5)	Oxid	Oxidation State
Real(Kind=CP), Dimension (5)	Rlon	Ionic Radio (depending of the oxidation)
Real(Kind=CP)	SCTF	Scattering length Fermi
Real(Kind=CP)	SEDInc	incoherent Scattering Neutron cross-section (barns -> [10 ⁻²⁴ cm ²])
Real(Kind=CP)	SEA	Neutron Absorption cross-section (barns, for v= 2200m/s, I(A)=3.95/v (km/s))
End Type Chem_Info_Type		

Magnetic_Form_Type

	Variable	Definition
Type :: Magnetic_Form_Type		
Character(Len=4)	Symb	Symbol of the Chemical species
Real(Kind=CP), Dimension (7)	SCTM	Scattering Factors
End Type Magnetic_Form_Type		

Xray_Form_Type

	Variable	Definition
Type :: Xray_Form_Type		
Character(Len=4)	Symb	Symbol of the Chemical species
Integer	Z	Atomic Number
Real(Kind=CP), Dimension (4)	A	Coefficients for calculating the X-ray scattering factors
Real(Kind=CP), Dimension (4)	B	f(s) = Sum_{i=1,4} { a(i) exp(-b(i)*s^2) } + c, where s=sinθ/λ
Real(Kind=CP)	C	

End Type Xray_Form_Type

Xray_Wavelength_Type

	Variable	Definition
Type :: Xray_Wavelength_Type		
Character(Len=2)	Symb	Symbol of the Chemical species
Real(Kind=CP), Dimension (2)	KAlfa	K-Serie for X-ray
End Type Xray_Wavelength_Type		

Anomalous_SCFac

Type(Anomalous_SC_Type), Dimension(:), Allocatable :: Anomalous_SCFac

Table of $\Delta F'$ and $\Delta F''$ for 5 common radiations according to the items specified in the definition of [Anomalous_SC_Type](#).

The order is the following: 1=Cr, 2=Fe, 3=Cu, 4=Mo, 5=Ag

The actual dimension is defined on [Num_Delta_Fp](#)

Chem_Info

Type(Chem_Info_Type), Dimension(:), Allocatable :: Chem_Info

Tabulated chemical data according to the items specified in the definition of [Chem_Info_Type](#).

The total elements are define in [Num_Chem_Info](#)

Magnetic_Form

Type(Magnetic_Form_Type), Dimension(:), Allocatable :: Magnetic_Form

Tabulated magnetic form factor data according to the items specified in the definition of [Magnetic_Form_Type](#).

The number of total elements is defined in [Num_Mag_Form](#)

Magnetic_J2

Type(Magnetic_Form_Type), Dimension(:), Allocatable :: MAGNETIC_J2

Tabulated magnetic form factor J2 data according to the items specified in the definition of [Magnetic_Form_Type](#).

The number of total elements is defined in [Num_Mag_J2](#)

Magnetic_J4

Type(Magnetic_Form_Type), Dimension(:), Allocatable :: MAGNETIC_J4

Tabulated magnetic form factor J4 data according to the items specified in the definition of [Magnetic_Form_Type](#).

The number of total elements is defined in [Num_Mag_J4](#)

Magnetic_J6

Type(Magnetic_Form_Type), Dimension(:), Allocatable :: MAGNETIC_J6

Tabulated magnetic form factor J6 data according to the items specified in the definition of [Magnetic_Form_Type](#).

The number of total elements is defined in [Num_Mag_J6](#)

Xray_Form

Type(Xray_Form_Type), Dimension(:), Allocatable :: Xray_Form

Tabulated Xray scattering factor coefficients according to the items specified in the definition of [Xray_Form_Type](#).

The number of total elements is defined in [Num_Xray_Form](#).

Xray_Wavelengths

Type(Xray_Wavelength_Type), Dimension(7) :: Xray_Wavelengths

Tabulated K-Series for Xray according to the items specified in the definition of [Xray_Wavelength_Type](#)

Symbol	K α_1	K α_2
Cr	2.28988	2.29428
Fe	1.93631	1.94043
Cu	1.54059	1.54431
Mo	0.70932	0.71360
Ag	0.55942	0.56380
Co	1.78919	1.79321
Ni	1.65805	1.66199

Subroutines

- [Get_Atomic_Mass](#)
- [Get_ChemSymb](#)
- [Get_Covalent_Radius](#)
- [Get_Fermi_Length](#)
- [Get_Ionic_Radius](#)
- [Remove_Chem_Info](#)
- [Remove_Delta_Fp_Fpp](#)
- [Remove_Magnetic_Form](#)
- [Remove_Xray_Form](#)
- [Set_Chem_Info](#)

- [Set_Delta_Fp_Fpp](#)
- [Set_Magnetic_Form](#)
- [Set_Xray_Form](#)

[Get_Atomic_Mass](#)

Subroutine Get_Atomic_Mass (Atm, Mass)

Character(Len=2)	Intent(in)	Atm	Chemical symbol
Real(Kind=CP)	Intent(out)	Mass	Atomic mass

Provides the atomic mass given the chemical symbol of the element. in case of problems the returned mass is 0.0

[Get_ChemSymb](#)

Subroutine Get_ChemSymb (Label, ChemSymb, Z)

Character(Len=*)	Intent(in)	Label	Atom label
Character(Len=*)	Intent(out)	ChemSymb	Chemical Symbol
Integer, Optional	Intent(out)	Z	Atomic number

Subroutine to get the chemical symbol from label and optionally the atomic number

[Get_Covalent_Radius](#)

Subroutine Get_Covalent_Radius (Nam, Rad)

Character(Len=*)	Intent(in)	Nam	Chemical Symbol
Real(Kind=CP)	Intent(out)	Rad	Covalent radius

Provides the covalent radius given the chemical symbol of the element. in case of problems the returned radius is 1.4 angstroms.

[Get_Fermi_Length](#)

Subroutine Get_Fermi_Length (Nam, B)

Character(Len=*)	Intent(in)	Nam	Chemical Symbol
Real(Kind=CP)	Intent(out)	B	Fermi length

Provides the Fermi length (in 10^{-12} cm) given the chemical symbol of the element. in case of problems the returned Fermi length is 0.0

[Get_Ionic_Radius](#)

Subroutine Get_Ionic_Radius (Nam, Valence, Rad)

Character(Len=*)	Intent(in)	Nam	Chemical symbol
Integer	Intent(in)	Valence	Valence value

Real(Kind=CP)	Intent(out)	Rad	Ionic radius
----------------------	--------------------	-----	--------------

Provides the ionic radius given the chemical symbol of the element and the valence as an Integer. in case of problems the returned radius is 0.0

Remove_Chem_Info

Subroutine Remove_Chem_Info ()

Deallocate [Chem_Info](#) variable

Remove_Delta_Fp_Fpp

Subroutine Remove_Delta_Fp_Fpp ()

Deallocate [Anomalous_SCFac](#) variable

Remove_Magnetic_Form

Subroutine Remove_Magnetic_Form ()

Deallocate [Magnetic_Form](#) variable

Remove_Xray_Form

Subroutine Remove_Xray_Form ()

Deallocate [Xray_Form](#) variable

Set_Chem_Info

Subroutine Set_Chem_Info ()

Allocates and loads the [Chem_Info](#) variable according to [Chem_Info_Type](#)

Set_Delta_Fp_Fpp

Subroutine Set_Delta_Fp_Fpp ()

Allocates and loads the [Anomalous_SCFac](#) variable according to [Anomalous_SC_Type](#)

Set_Magnetic_Form

Subroutine Set_Magnetic_Form ()

Allocates and loads the [Magnetic_Form](#) variable according to [Magnetic_Form_Type](#)

Set_Xray_Form

Subroutine Set_Xray_Form ()

Allocates and loads the [Xray_Form](#) variable according to [Xray_Form_Type](#)

CFML_Symmetry_Tables

Tabulated information on Crystallographic Symmetry

Parameters

- [BC_D6H](#)
- [BC_OH](#)
- [DepMat](#)
- [IntSymD6H](#)
- [IntSymOH](#)
- [Kov_D6H](#)
- [Kov_OH](#)
- [Latt](#)
- [Laue_Class](#)
- [Ltr_A](#)
- [Ltr_B](#)
- [Ltr_C](#)
- [Ltr_F](#)
- [Ltr_I](#)
- [Ltr_R](#)
- [MagMat](#)
- [ML_D6H](#)
- [ML_OH](#)
- [Mod6](#)
- [Point_Group](#)
- [Sys_Cry](#)
- [X_D6H](#)
- [X_OH](#)
- [Zak_D6H](#)
- [Zak_OH](#)

Variables

- [Spgr_Info_Type](#)
- [Table_Equiv_Type](#)
- [Wyck_Info_Type](#)
- [Err_SymTab](#)
- [Err_SymTab_Mess](#)
- [Spgr_Info](#)
- [System_Equiv](#)
- [Wyckoff_Info](#)

Subroutines

- [Get_Generators](#)
- [Remove_Spgr_Info](#)
- [Remove_System_Equiv](#)
- [Remove_Wyckoff_Info](#)
- [Set_Spgr_Info](#)

- [Set_System_Equiv](#)
- [Set_Wyckoff_Info](#)

Fortran Filename

CFML_Sym_Table.f90

Parameters

- [BC_D6H](#)
- [BC_OH](#)
- [DepMat](#)
- [IntSymD6H](#)
- [IntSymOH](#)
- [Kov_D6H](#)
- [Kov_OH](#)
- [Latt](#)
- [Laue_Class](#)
- [Ltr_A](#)
- [Ltr_B](#)
- [Ltr_C](#)
- [Ltr_F](#)
- [Ltr_I](#)
- [Ltr_R](#)
- [MagMat](#)
- [ML_D6H](#)
- [ML_OH](#)
- [Mod6](#)
- [Point_Group](#)
- [Sys_Cry](#)
- [X_D6H](#)
- [X_OH](#)
- [Zak_D6H](#)
- [Zak_OH](#)

BC_D6H

Character(Len=*), **Dimension(24)**, **Parameter :: BC_D6H**

Bradley & Cracknell Notation for Point Group elements of 6/mmm (D6h)

Order Value

1	E
2	C+_3
3	C_-3
4	C_2

Order Value

13	I
14	S_-6
15	S+_6
16	s_h

5	C_-6	17	S+_3
6	C+_-6	18	S_-3
7	C'_23	19	s_v3
8	C'_21	20	s_v1
9	C'_22	21	s_v2
10	C'_23	22	s_d3
11	C`_21	23	s_d1
12	C`_22	24	s_d2

BC_OH

Character(Len=*), Dimension(48), Parameter :: BC_OH

Bradley & Cracknell Notation for Point Group elements of m3m (Oh)

Order	Value	Order	Value	Order	Value	Order	Value
1	E	13	C_2a	25	I	37	s_da
2	C_2z	14	C_2b	26	s_z	38	s_db
3	C_2y	15	C_-4z	27	s_y	39	S+_4z
4	C_2x	16	C+_-4z	28	s_x	40	S_-4z
5	C+_-31	17	C_-4x	29	S_-61	41	S+_4x
6	C+_-34	18	C_2d	30	S_-64	42	s_dd
7	C+_-33	19	C_2f	31	S_-63	43	d_df
8	C+_-32	20	C+_-4x	32	S_-62	44	S_-4x
9	C_-31	21	C+_-4y	33	S+_-61	45	S_-4y
10	C_-33	22	C_2c	34	S+_-63	46	s_dc
11	C_-32	23	C_-4y	35	S+_-62	47	S+_-4y
12	C_-34	24	C_2e	36	S+_-64	48	s_de

DepMat

Character(Len=*), Dimension(72), Parameter :: DepMat

Magnetic array

Order	Value	Order	Value	Order	Value
1	(Dx, Dy, Dz)	25	(-Dx,-Dy,-Dz)	49	(Dx , Dy, Dz)
2	(-Dx,-Dy, Dz)	26	(Dx, Dy,-Dz)	50	(-Dy, Dx-Dy , Dz)
3	(-Dx, Dy,-Dz)	27	(Dx,-Dy, Dz)	51	(-Dx+Dy,-Dx , Dz)
4	(Dx,-Dy,-Dz)	28	(-Dx, Dy, Dz)	52	(-Dx , -Dy, Dz)
5	(Dz, Dx, Dy)	29	(-Dz,-Dx,-Dy)	53	(Dy,-Dx+Dy, Dz)
6	(Dz,-Dx,-Dy)	30	(-Dz, Dx, Dy)	54	(Dx-Dy, Dx , Dz)
7	(-Dz,-Dx, Dy)	31	(Dz, Dx,-Dy)	55	(Dy, Dx , -Dz)
8	(-Dz, Dx,-Dy)	32	(Dz,-Dx, Dy)	56	(Dx-Dy, -Dy,-Dz)
9	(Dy, Dz, Dx)	33	(-Dy,-Dz,-Dx)	57	(-Dx , -Dx+Dy,-Dz)
10	(-Dy, Dz,-Dx)	34	(Dy,-Dz, Dx)	58	(-Dy,-Dx , -Dz)
11	(Dy,-Dz,-Dx)	35	(-Dy, Dz, Dx)	59	(-Dx+Dy, Dy,-Dz)
12	(-Dy,-Dz, Dx)	36	(Dy, Dz,-Dx)	60	(Dx , Dx-Dy,-Dz)
13	(Dy, Dx,-Dz)	37	(-Dy,-Dx, Dz)	61	(-Dx , -Dy,-Dz)

14	(-Dy,-Dx,-Dz)	38	(Dy, Dx, Dz)	62	(Dy,-Dx+Dy,-Dz)
15	(Dy,-Dx, Dz)	39	(-Dy, Dx, -Dz)	63	(Dx-Dy,Dx , -Dz)
16	(-Dy, Dx, Dz)	40	(Dy,-Dx,-Dz)	64	(Dx , Dy,-Dz)
17	(Dx, Dz,-Dy)	41	(-Dx,-Dz, Dy)	65	(-Dy, Dx-Dy,-Dz)
18	(-Dx, Dz, Dy)	42	(Dx,-Dz,-Dy)	66	(-Dx+Dy,-Dx , -Dz)
19	(-Dx,-Dz,-Dy)	43	(Dx, Dz, Dy)	67	(-Dy,-Dx , Dz)
20	(Dx,-Dz, Dy)	44	(-Dx, Dz,-Dy)	68	(-Dx+Dy, Dy, Dz)
21	(Dz, Dy,-Dx)	45	(-Dz,-Dy, Dx)	69	(Dx , Dx-Dy, Dz)
22	(Dz,-Dy, Dx)	46	(-Dz, Dy,-Dx)	70	(Dy, Dx , Dz)
23	(-Dz, Dy, Dx)	47	(Dz,-Dy,-Dx)	71	(Dx-Dy, -Dy, Dz)
24	(-Dz,-Dy,-Dx)	48	(Dz, Dy, Dx)	72	(-Dx , -Dx+Dy, Dz)

IntSymD6H

Character(Len=*), **Dimension(24)**, **Parameter** :: IntSymD6H

international Symbols for Point Group elements of 6/mmm (D6h)

Order Value

1	1
2	3+ (0, 0, z)
3	3- (0, 0, z)
4	2 (0, 0, z)
5	6- (0, 0, z)
6	6+ (0, 0, z)
7	2 (x, x, 0)
8	2 (x, 0, 0)
9	2 (0, y, 0)
10	2 (x,-x, 0)
11	2 (x,2x, 0)
12	2 (2x, x, 0)

Order Value

13	-1
14	-3+ (0, 0, z)
15	-3- (0, 0, z)
16	m (x, y, 0)
17	-6- (0, 0, z)
18	-6+ (0, 0, z)
19	m (x,-x, z)
20	m (x,2x, z)
21	m (2x, x, z)
22	m (x, x, z)
23	m (x, 0, z)
24	m (0, y, z)

IntSymOH

Character(Len=*), **Dimension(48)**, **Parameter** :: IntSymOH

international Symbols for Point Group elements of m3m (Oh)

Order Value

1	1
2	2 (0, 0, z)
3	2 (0, y, 0)
4	2 (x, 0, 0)
5	3+ (x, x, x)
6	3+ (-x, x,-x)
7	3+ (x,-x,-x)
8	3+ (-x,-x, x)
9	3- (x, x, x)
10	3- (x,-x,-x)

Order Value

13	2 (x, x, 0)
14	2 (x,-x, 0)
15	4- (0, 0, z)
16	4+ (0, 0, z)
17	4- (x, 0, 0)
18	2 (0, y, y)
19	2 (0, y,-y)
20	4+ (x, 0, 0)
21	4+ (0, y, 0)
22	2 (x, 0, x)

Order Value

25	-1
26	m (x, y, 0)
27	m (x, 0, z)
28	m (0, y, z)
29	-3+ (x, x, x)
30	-3+ (-x, x,-x)
31	-3+ (x,-x,-x)
32	-3+ (-x,-x, x)
33	-3- (x, x, x)
34	-3- (x,-x,-x)

Order Value

37	m (x,-x, z)
38	m (x, x, z)
39	-4- (0, 0, z)
40	-4+ (0, 0, z)
41	-4- (x, 0, 0)
42	m (x, y,-y)
43	m (x, y, y)
44	-4+ (x, 0, 0)
45	-4+ (0, y, 0)
46	m (-x, y, x)

11	3- (-x, -x, x)	23	4- (0, y, 0)	35	-3- (-x, -x, x)	47	-4- (0, y, 0)
12	3- (-x, x, -x)	24	2 (-x, 0, x)	36	-3- (-x, x, -x)	48	m (x, y, x)

Kov_D6H

Character(Len=*), **Dimension(24)**, **Parameter** :: Kov_D6H

Kovalev Notation for Point Group elements of 6/mmm (D6h)

Order	Value	Order	Value
1	h1	13	h13
2	h3	14	h15
3	h5	15	h17
4	h4	16	h16
5	h6	17	h18
6	h2	18	h14
7	h11	19	h23
8	h9	20	h21
9	h7	21	h19
10	h8	22	h20
11	h12	23	h24
12	h10	24	h22

Kov_OH

Character(Len=*), **Dimension(48)**, **Parameter** :: Kov_OH

Kovalev Notation for Point Group elements of m3m (Oh)

Order	Value	Order	Value	Order	Value	Order	Value
1	h1	13	h16	25	h25	37	h40
2	h4	14	h13	26	h28	38	h37
3	h3	15	h15	27	h27	39	h39
4	h2	16	h14	28	h26	40	h38
5	h9	17	h20	29	h33	41	h44
6	h10	18	h18	30	h34	42	h42
7	h12	19	h17	31	h36	43	h41
8	h11	20	h19	32	h35	44	h43
9	h5	21	h24	33	h29	45	h48
10	h7	22	h23	34	h31	46	h47
11	h6	23	h22	35	h30	47	h46
12	h8	24	h21	36	h32	48	h45

Latt

Character(Len=*), **Dimension(8)**, **Parameter** :: Latt

Lattice Traslations

Order Value

- 1** P: { 000 }
- 2** A: { 000; 0 1/2 1/2 }+
- 3** B: { 000; 1/2 0 1/2 }+
- 4** C: { 000; 1/2 1/2 0 }+
- 5** I: { 000; 1/2 1/2 1/2 }+
- 6** R: { 000; 2/3 1/3 1/3; 1/3 2/3 2/3 }+
- 7** F: { 000; 0 1/2 1/2; 1/2 0 1/2; 1/2 1/2 0 }+
- 8** Z: { 000; Unconventional Z-centering vectors }+

Laue_Class

Character(Len=*), **Dimension(16)**, **Parameter :: Laue_Class**

Laue symbols

Order Value

- 1** -1
- 2** 2/m
- 3** mmm
- 4** 4/m
- 5** 4/mmm
- 6** -3 R
- 7** -3m R
- 8** -3

Order Value

- 9** -3m1
- 10** -31m
- 11** 6/m
- 12** 6/mmm
- 13** m-3
- 14** m-3m
- 15** m3
- 16** m3m

Ltr_A

Real(Kind=CP), **Dimension(3,2)**, **Parameter :: Ltr_A**

Lattice translations of type A

$$LTR_A = \begin{pmatrix} 0 & 0 \\ 0 & \frac{1}{2} \\ 0 & \frac{1}{2} \end{pmatrix}$$

Ltr_B

Real(Kind=CP), **Dimension(3,2)**, **Parameter :: Ltr_B**

Lattice translations of type B

$$LTR_B = \begin{pmatrix} 0 & \frac{1}{2} \\ 0 & 0 \\ 0 & \frac{1}{2} \end{pmatrix}$$

Ltr_C

Real(Kind=CP), Dimension(3,2), Parameter :: Ltr_C

Lattice translations of type C

$$LTR_C = \begin{pmatrix} 0 & \frac{1}{2} \\ 0 & \frac{1}{2} \\ 0 & 0 \end{pmatrix}$$

Ltr_F

Real(Kind=CP), Dimension(3,4), Parameter :: Ltr_F

Lattice translations of type F

$$LTR_F = \begin{pmatrix} 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix}$$

Ltr_I

Real(Kind=CP), Dimension(3,2), Parameter :: Ltr_I

Lattice translations of type I

$$LTR_I = \begin{pmatrix} 0 & \frac{1}{2} \\ 0 & \frac{1}{2} \\ 0 & \frac{1}{2} \end{pmatrix}$$

Ltr_R

Real(Kind=CP), Dimension(3,3), Parameter :: Ltr_R

Lattice translations of type R

$$LTR_R = \begin{pmatrix} 0 & \frac{2}{3} & \frac{1}{3} \\ 0 & \frac{1}{3} & \frac{2}{3} \\ 0 & \frac{1}{3} & \frac{2}{3} \end{pmatrix}$$

MagMat

Character(Len=*), **Dimension(72)**, **Parameter :: MagMat**

Magnetic array

Order	Value	Order	Value	Order	Value
1	(Mx, My, Mz)	25	(-Mx,-My,-Mz)	49	(Mx , My, Mz)
2	(-Mx,-My, Mz)	26	(Mx, My,-Mz)	50	(-My, Mx-My, Mz)
3	(-Mx, My,-Mz)	27	(Mx,-My, Mz)	51	(-Mx+My,-Mx , Mz)
4	(Mx,-My,-Mz)	28	(-Mx, My, Mz)	52	(-Mx , -My, Mz)
5	(Mz, Mx, My)	29	(-Mz,-Mx,-My)	53	(My,-Mx+My, Mz)
6	(Mz,-Mx,-My)	30	(-Mz, Mx, My)	54	(Mx-My, Mx , Mz)
7	(-Mz,-Mx, My)	31	(Mz, Mx,-My)	55	(My, Mx ,-Mz)
8	(-Mz, Mx,-My)	32	(Mz,-Mx, My)	56	(Mx-My, -My,-Mz)
9	(My, Mz, Mx)	33	(-My,-Mz,-Mx)	57	(-Mx ,-Mx+My,-Mz)
10	(-My, Mz,-Mx)	34	(My,-Mz, Mx)	58	(-My,-Mx ,-Mz)
11	(My,-Mz,-Mx)	35	(-My, Mz, Mx)	59	(-Mx+My, My,-Mz)
12	(-My,-Mz, Mx)	36	(My, Mz,-Mx)	60	(Mx , Mx-My,-Mz)
13	(My, Mx,-Mz)	37	(-My,-Mx, Mz)	61	(-Mx , -My,-Mz)
14	(-My,-Mx,-Mz)	38	(My, Mx, Mz)	62	(My,-Mx+My,-Mz)
15	(My,-Mx, Mz)	39	(-My, Mx,-Mz)	63	(Mx-My, Mx ,-Mz)
16	(-My, Mx, Mz)	40	(My,-Mx,-Mz)	64	(Mx , My,-Mz)
17	(Mx, Mz,-My)	41	(-Mx,-Mz, My)	65	(-My, Mx-My,-Mz)
18	(-Mx, Mz, My)	42	(Mx,-Mz,-My)	66	(-Mx+My,-Mx ,-Mz)
19	(-Mx,-Mz,-My)	43	(Mx, Mz, My)	67	(-My,-Mx , Mz)
20	(Mx,-Mz, My)	44	(-Mx, Mz,-My)	68	(-Mx+My, My, Mz)
21	(Mz, My,-Mx)	45	(-Mz,-My, Mx)	69	(Mx , Mx-My, Mz)
22	(Mz,-My, Mx)	46	(-Mz, My,-Mx)	70	(My, Mx , Mz)
23	(-Mz, My, Mx)	47	(Mz,-My,-Mx)	71	(Mx-My, -My, Mz)
24	(-Mz,-My,-Mx)	48	(Mz, My, Mx)	72	(-Mx ,-Mx+My, Mz)

ML_D6H

Character(Len=*), **Dimension(24)**, **Parameter :: ML_D6H**

Miller & Love Notation for Point Group elements of 6/mmm (D6h)

Order	Value	Order	Value
1	1	13	13
2	3	14	15
3	5	15	17

4	4	16	16
5	6	17	18
6	2	18	14
7	9	19	21
8	7	20	19
9	11	21	23
10	12	22	24
11	10	23	22
12	8	24	20

ML_OH

Character(Len=*), Dimension(48), Parameter :: ML_OH

Miller & Love Notation for Point Group elements of m3m (Oh)

Order	Value	Order	Value	Order	Value	Order	Value
1	1	13	16	25	25	37	40
2	4	14	13	26	28	38	37
3	3	15	15	27	27	39	39
4	2	16	14	28	26	40	38
5	9	17	20	29	33	41	44
6	10	18	18	30	34	42	42
7	12	19	17	31	36	43	41
8	11	20	19	32	35	44	43
9	5	21	24	33	29	45	48
10	7	22	23	34	31	46	47
11	6	23	22	35	30	47	46
12	8	24	21	36	32	48	45

Mod6

Integer, Dimension(36,3,3), Parameter :: Mod6

Matrix types for Rotational Operators in conventional basis.

- From 1 to 24 for Oh
- From 25 to 36 for D6h

Point_Group

Character(Len=*), Dimension(39), Parameter :: Point_Group

Point Group Symbols

Order	Value	Order	Value	Order	Value	Order	Value
1	1	13	4/m	25	31m	37	432
2	-1	14	422	26	-31m	38	-43m
3	2	15	4mm	27	6	39	m-3m

4	m	16	-42m	28	-6
5	2/m	17	-4m2	29	6/m
6	222	18	4/mmm	30	622
7	mm2	19	3	31	6mm
8	m2m	20	-3	32	-62m
9	2mm	21	32	33	-6m2
10	mmm	22	3m	34	6/mmm
11	4	23	-3m	35	23
12	-4	24	312	36	m-3

Sys_Cry

Character(Len=*), **Dimension(7)**, **Parameter** :: **Sys_Cry**

System Type

Order Value

- 1** Triclinic
- 2** Monoclinic
- 3** Orthorhombic
- 4** Tetragonal
- 5** Rhombohedral
- 6** Hexagonal
- 7** Cubic

X_D6H

Character(Len=*), **Dimension(24)**, **Parameter** :: **X_D6H**

Notation for Point Group elements of 6/mmm (D6h)

Order Value

- 1** (x , y, z)
- 2** (-y, x-y, z)
- 3** (-x+y,-x , z)
- 4** (-x , -y, z)
- 5** (y,-x+y, z)
- 6** (x-y, x , z)
- 7** (y, x , -z)
- 8** (x-y, -y,-z)
- 9** (-x , -x+y,-z)
- 10** (-y,-x , -z)
- 11** (-x+y, y,-z)
- 12** (x , x-y,-z)

Order Value

- 13** (-x , -y,-z)
- 14** (y,-x+y,-z)
- 15** (x-y, x , -z)
- 16** (x , y,-z)
- 17** (-y, x-y,-z)
- 18** (-x+y,-x , -z)
- 19** (-y,-x , z)
- 20** (-x+y, y, z)
- 21** (x , x-y, z)
- 22** (y, x , z)
- 23** (x-y, -y, z)
- 24** (-x , -x+y, z)

X_OH

Character(Len=*), **Dimension(48)**, **Parameter** :: **X_OH**

Notation for Point Group elements of m3m (Oh)

Order	Value	Order	Value	Order	Value	Order	Value
1	(x, y, z)	13	(y, x, -z)	25	(-x, -y, -z)	37	(-y, -x, z)
2	(-x, -y, z)	14	(-y, -x, -z)	26	(x, y, -z)	38	(y, x, z)
3	(-x, y, -z)	15	(y, -x, z)	27	(x, -y, z)	39	(-y, x, -z)
4	(x, -y, -z)	16	(-y, x, z)	28	(-x, y, z)	40	(y, -x, -z)
5	(z, x, y)	17	(x, z, -y)	29	(-z, -x, -y)	41	(-x, -z, y)
6	(z, -x, -y)	18	(-x, z, y)	30	(-z, x, y)	42	(x, -z, -y)
7	(-z, -x, y)	19	(-x, -z, -y)	31	(z, x, -y)	43	(x, z, y)
8	(-z, x, -y)	20	(x, -z, y)	32	(z, -x, y)	44	(-x, z, -y)
9	(y, z, x)	21	(z, y, -x)	33	(-y, -z, -x)	45	(-z, -y, x)
10	(-y, z, -x)	22	(z, -y, x)	34	(y, -z, x)	46	(-z, y, -x)
11	(y, -z, x)	23	(-z, y, x)	35	(-y, z, x)	47	(z, -y, -x)
12	(-y, -z, x)	24	(-z, -y, -x)	36	(y, z, -x)	48	(z, y, x)

Zak_D6H

Character(Len=*), **Dimension(24)**, **Parameter** :: Zak_D6H

Zak Notation for Point Group elements of 6/mmm (D6h)

Order	Value	Order	Value
1	E	13	I
2	C(z)_3	14	S(5z)_6
3	C(2z)_3	15	S(z)_6
4	C_2	16	s(z)
5	C(5z)_6	17	S(z)_3
6	C(z)_6	18	S(2z)_3
7	U(xy)	19	s(xy)
8	U(x)	20	s(x)
9	U(y)	21	s(y)
10	U(3)	22	s(3)
11	U(2)	23	s(2)
12	U(1)	24	s(1)

Zak_OH

Character(Len=*), **Dimension(48)**, **Parameter** :: Zak_OH

Zak Notation for Point Group elements of m3m (Oh)

Order	Value	Order	Value	Order	Value	Order	Value
1	E	13	U(xy)	25	I	37	s(xy)
2	U(z)	14	U(-xy)	26	s(z)	38	s(-xy)
3	U(y)	15	C(3z)_4	27	s(y)	39	S(z)_4

4	U(x)	16	C(z)_4	28	s(x)	40	S(3z)_4
5	C(xyz)_3	17	C(3x)_4	29	S(5xyz)_6	41	S(x)_4
6	C(-xy-z)_3	18	U(yz)	30	S(-5xy-z)_6	42	s(yz)
7	C(x-y-z)_3	19	U(y-z)	31	S(5x-y-z)_6	43	s(y-z)
8	C(-x-yz)_3	20	C(x)_4	32	S(-5x-yz)_6	44	S(3x)_4
9	C(2xyz)_3	21	C(y)_4	33	S(xyz)_6	45	S(3y)_4
10	C(2x-y-z)_3	22	U(xz)	34	S(x-y-z)_6	46	s(xz)
11	C(2x-yz)_3	23	C(3y)_4	35	S(-x-yz)_6	47	S(y)_4
12	C(-2xy-z)_3	24	U(x-z)	36	S(-xy-z)_6	48	s(x-z)

Variables

- [Spgr_Info_Type](#)
- [Table_Equiv_Type](#)
- [Wyck_Info_Type](#)
- [Err_SymTab](#)
- [Err_SymTab_Mess](#)
- [Spgr_Info](#)
- [System_Equiv](#)
- [Wyckoff_Info](#)

Spgr_Info_Type

	Variable	Definition
Type :: Spgr_Info_Type		
Integer	N	Number of the Space group according to I.T.
Character (Len=12)	HM	Hermann-Mauguin symbol
Character (Len=16)	Hall	Hall symbol
Integer	Laue	Laue group
Integer	PG	Point group
Integer, Dimension(6)	Asu	Asymmetric unit * 24
Character (Len=5)	Inf_Extra	Extra information
End Type Spgr_Info_Type		

Table_Equiv_Type

	Variable	Definition
Type :: Table_Equiv_Type		

Character (Len=6)	SC	Schoenflies
Character (Len=17)	ML	Miller & Love
Character (Len=18)	KO	Kovalev
Character (Len=32)	BC	Bradley & Cracknell
Character (Len=18)	ZA	Zak
End Type Table_Equiv_Type		

Definition for Equivalences on a Table

Wyck_Info_Type

	Variable	Definition
Type :: Wyck_Info_Type		
Character (Len=12)	HM	Hermann-Mauguin symbol
Integer	NOrbit	Number of orbits
Character (Len=15), Dimension(24)	COrbit	Generator of the orbit
End Type Wyck_Info_Type		

Definition for Wyckoff Positions according to I.T.

Err_SymTab

Logical :: Err_SymTab

This variable is set to .TRUE. if an error occurs in procedures belonging to this module

Err_SymTab_Mess

Character (Len=150) :: Err_SymTab_Mess

This variable contains information about the last error occurred in the procedures belonging to this module

Spgr_Info

Type(Spgr_Info_Type), Dimension(:) :: Spgr_Info

General information about Space Groups

System_Equiv

Type(Table_Equiv_Type), Dimension(:), Allocatable :: System_Equiv

General information about equivalence between Notations

Wyckoff_Info

Type(Wyck_Info_Type), Dimension(:), Allocatable :: Wyckoff_Info

General info about Wyckoff Positions on IT

Subroutines

- [Get_Generators](#)
- [Remove_Spgr_Info](#)
- [Remove_System_Equiv](#)
- [Remove_Wyckoff_Info](#)
- [Set_Spgr_Info](#)
- [Set_System_Equiv](#)
- [Set_Wyckoff_Info](#)

[Get_Generators](#)

Subroutine Get_Generators (Spg, Gener)

Character (Len=*)	Intent(in)	Spg	Hermann_Mauguin symbol or number of S.Group
Character (Len=*)	Intent(out)	Gener	String with all generators

Provides the string **GENER** containing the list of the generators (as given in the IT Crystallography) corresponding to the space group of symbol **Spg**.

In **Spg** the Hermann-Mauguin symbol or the number of the space group should be given. The calling program is responsible of decoding the string **Gener**. Generator are given in the Jone's Faithful notation and the separator is the symbol ";".

Example:

Space group: R 3 c

GENER= " x+1/3,y+2/3,z+2/3; -y,x-y,z; -y,-x,z+1/2"

[Remove_Spgr_Info](#)

Subroutine Remove_Spgr_Info()

Deallocating [Spgr_Info](#) Data

[Remove_System_Equiv](#)

Subroutine Remove_System_Equiv ()

Deallocating [System_Equiv](#) variable

[Remove_Wyckoff_Info](#)

Subroutine Remove_Wyckoff_Info()

Deallocating [Wyckoff_Info](#) variable

[Set_Spgr_Info](#)

Subroutine Set_Spgr_Info ()

Set information on [Spgr_Info](#) variable

[Set_System_Equiv](#)

Subroutine Set_System_Equiv()

Define the conversion table between IT - ML - Kov - BC - Zak

The information given in this file corresponds to that of Table 6 of "Isotropy Subgroups of the 230 Crystallographic Space Groups", by Harold T Stokes and Dorian M Hatch, World Scientific, Singapore (1988).

The transformation operators that take space group elements in the international setting (international Tables of Crystallography, Hahn 1983) to space-groups elements in the Miller and Love (ML, 1967), Kovalev (Kov, 1986) Bradley and Cracknell (BC, 1972) and Zak (Zak, 1969) settings.

- In the international setting the basis vectors are always those of the conventional unit cell. in rhombohedral system the primitive basis vectors are in an obverse relationship given by (2/3 1/3 1/3), (-1/3 1/3 1/3) and (-1/3, -2/3 1/3).
- In ML the same basis vectors are chosen except that for rhombohedral system the reverse setting is adopted, so the primitive basis vectors are: t1=(1/3 -1/3 1/3), t2=(1/3, 2/3 1/3) and t3=(2/3 1/3 1/3).
- In Kovalev the a,b,c axes of the coordinate system are along the conventional basis vectors of the lattice, however in the rhombohedral system an hexagonal system is chosen so that the primitive basis vectors are a1=(-1 -1 1/3), a2=(1 0 1/3) and a3=(0 1 1/3).
- In the setting of BC the axes a,b,c of the coordinate system are chosen to be the primitive basis vectors t1,t2,t3 as defined in their book.
- The setting of Zak the basis vectors are as in the international setting, but for rhombohedral system the primitive basis vectors w.r.t. the selected hexagonal coordinate system are given by: (1/3 2/3 1) (1/3 -1/3 1) (-2/3 -1/3 1)

Symmetry and transformation operators of Space Groups can be given as 4 x 4 Seitz matrices or as a character string called Jones Faithful representation. This last representation is that used in this file.

To transform a symmetry operator "gl" in the international setting into a symmetry element "g" in one of the other settings, we simply perform the following operation: $g = gT \text{ gl } gT(-1)$, where gT is the transformation given tabulated below.

[Set_Wyckoff_Info](#)

Subroutine Set_Wyckoff_Info ()

Set information on [Wyckoff_Info](#) variable

Level 3

Concept	Module Name	Purpose
Bonds Tables...	CFML_Bond_Tables	Contain a simple subroutine providing the list of the usual bonds between atoms
Crystal Metrics...	CFML_Crystal_Metrics	Define

		crystallographic types and to provide automatic crystallographic metrics operations
<i>instrumentation on ILL...</i>	CFML_ILL_Instrm_Data	Procedures to access the (single crystals) instrument output data base at ILL
<i>Symmetry information...</i>	CFML_Crystallographic_Symmetry	Contain nearly everything needed for handling symmetry in Crystallography.

CFML_Bond_Tables

This module provide the list of the usual bonds between atoms. There are three possible values: simple, double and triple bond

Variables

- [Bond_Length_Table](#)
- [Err_Bond](#)
- [Err_Bond_Mess](#)

Subroutines

- [Get_Bonds_Table](#)
- [Init_Err_Bond](#)
- [Remove_Bonds_Table](#)
- [Set_Bonds_Table](#)

Fortran Filename

CFML_Bonds_Table.f90

Variables

- [Bond_Length_Table](#)
- [Err_Bond](#)
- [Err_Bond_Mess](#)

Bond_Length_Table

Real, Dimension(: , : , :), Allocatable :: Bond_Length_Table

Global variable holding the bond lengths between different type of atoms. Ordered by Z

`Bond_Length_Table(1,:,:)` represent the simple bond, while `Bond_Length_Table(2,:,:)` represent a double bound distances or shorter distances and finally `Bond_Length_Table(3,:,:)` represents the triple bond distance or the shortest distance.

Err_Bond

Logical :: Err_Bond

This variable is set to `.TRUE.` if an error occurs in procedures belonging to this module.

Err_Bond_Mess

Character (Len=150) :: Err_Bond_Mess

This variable contains information about the last error occurred in the procedures belonging to this module

Subroutines

- [Get_Bonds_Table](#)
- [Init_Err_Bond](#)
- [Remove_Bonds_Table](#)
- [Set_Bonds_Table](#)

Get_Bonds_Table

Subroutine Get_Bonds_Table (Symbol1, Symbol2, Bonds)

Character (Len=*)	Intent(in)	Symbol1	Atomic symbol
Character (Len=*)	Intent(in)	Symbol2	Atomic symbol
Real(Kind=CP), Dimension(3)	Intent(out)	Bonds	Bonds between Specie1 and Specie 2

or

Subroutine Get_Bonds_Table (Z1, Z2, Bonds)

Integer	Intent(in)	Z1	Atomic number for Specie 1
Integer	Intent(in)	Z2	Atomic number for Specie 2
Real(Kind=CP), Dimension(3)	Intent(out)	Bonds	Bonds between Specie1 and Specie 2

Obtain the typical distances between species of atoms

Init_Err_Bond

Subroutine Init_Err_Bond ()

Subroutine that initializes errors flags in `CFML_Bond_Tables` module.

Remove_Bonds_Table

Subroutine Remove_Bonds_Table ()

Deallocating `Bond_Length_Table` variable

Set_Bonds_Table

Subroutine Set_Bonds_Table ()

Fills the components of the [Bond_Length_Table](#) variable.

CFML_Crystal_Metrics

Module to define Crystallographic types and to provide automatic crystallographic procedures.

Variables

- [Crystal_Cell_Type](#)
- [Twofold_Axes_Type](#)
- [Err_Crys](#)
- [Err_Crys_Mess](#)

Functions

- [Cart_U_Vector](#)
- [Cart_Vector](#)
- [Convert_B_Betas](#)
- [Convert_B_U](#)
- [Convert_Betas_B](#)
- [Convert_Betas_U](#)
- [Convert_U_B](#)
- [Convert_U_Betas](#)
- [Get_Betas_from_Biso](#)
- [Rot_Matrix](#)
- [U_Equiv](#)

Subroutines

- [Change_Setting_Cell](#)
- [Get_Conventional_Cell](#)
- [Get_Cryst_Family](#)
- [Get_Deriv_Orth_Cell](#)
- [Get_Primitive_Cell](#)
- [Get_Transform_Matrix](#)
- [Get_TwoFold_Axes](#)
- [Init_Err_Crys](#)
- [Niggli_Cell](#)
- [Set_Crystal_Cell](#)
- [Write_Crystal_Cell](#)

Fortran Filename

CFML_Cryst_Types.f90

Variables

- [Crystal_Cell_Type](#)
- [Twofold_Axes_Type](#)
- [Err_Crys](#)
- [Err_Crys_Mess](#)

Crystal_Cell_Type

	Variable	Definition
Type :: Crystal_Cell_Type		
Real (Kind=CP), Dimension(3)	Cell	Lengths of the cell parameters in angstroms
Real (Kind=CP), Dimension(3)	Ang	Angles of the cell parameters in degrees
Real (Kind=CP), Dimension(3)	Cell_STD	Standar deviations of cell parameters
Real (Kind=CP), Dimension(3)	Ang_STD	
Real (Kind=CP), Dimension(3)	RCell	Reciprocal cell parameters
Real (Kind=CP), Dimension(3)	Rang	
Real (Kind=CP), Dimension(3,3)	GD	Direct Metric Tensors
Real (Kind=CP), Dimension(3,3)	GR	Reciprocal Metric Tensors
Real (Kind=CP), Dimension(3,3)	CR_Orth_Cel	P-Matrix transforming Orthonormal basis to direct Crystal cell (as I.T.) (or crystallographic components to cartesian components)
Real (Kind=CP), Dimension(3,3)	Orth_CR_Cel	Cartesian to crystallographic components
Real (Kind=CP), Dimension(3,3)	BL_M	Busing-Lewy B-matrix
Real (Kind=CP), Dimension(3,3)	BL_MInv	Inverse Busing-Levy B-matrix
Real (Kind=CP)	CellVol	Direct cell volumes
Real (Kind=CP)	RCellVol	Reciprocal cell volumes
Character(Len=1)	CartType	Cartesian Frame type: 'A' Cartesian Frame has x // a. Other Cartesian Frame has z // c
End Type Crystal_Cell_Type		

Twofold_Axes_Type

	Variable	Definition
Type :: Twofold_Axes_Type		
Integer	NTwo	Number of two-fold axes
Real (Kind=CP)	Tol	Angular tolerance (ca 3 degrees)
Real (Kind=CP), Dimension(3,12)	CAxes	Cartesian components of two-fold axes
Integer, Dimension(3,12)	DTwofold	Direct indices of two-fold axes
Integer, Dimension(3,12)	RTwofold	Reciprocal indices of two-fold axes
Integer, Dimension(12)	Dot	Scalar product of reciprocal and direct indices
Real (Kind=CP), Dimension(12)	Cross	Angle between direct and reciprocal axes (< tol)
Real (Kind=CP), Dimension(12)	Maxes	Modulus of the zone axes (two-fold axes) vectors
Real (Kind=CP), Dimension(3)	A	Cartesian components of direct cell parameters
Real (Kind=CP), Dimension(3)	B	
Real (Kind=CP), Dimension(3)	C	
End Type		
Twofold_Axes_Type		

Err_Crys

Logical :: Err_Crys

This variable is set to .TRUE. if an error occurs in procedures belonging to this module.

Err_Crys_Mess

Character (Len=150) :: Err_Crys_Mess

This variable contains information about the last error occurred in the procedures belonging to this module

Functions

- [Cart_U_Vector](#)
- [Cart_Vector](#)
- [Convert_B_Betas](#)
- [Convert_B_U](#)
- [Convert_Betas_B](#)
- [Convert_Betas_U](#)
- [Convert_U_B](#)
- [Convert_U_Betas](#)

- [Get_Betas_from_Biso](#)
- [Rot_Matrix](#)
- [U_Equiv](#)

Cart_U_Vector

Real Function Cart_U_Vector(Code, V, Celda)

Character (Len=*)	Intent(in)	Code	D: Direct R: Reciprocal
Real(Kind=CP), Dimension (3)	Intent(in)	V	Vector
Type(Crystal_Cell_Type)	Intent(in)	Celda	Cell parameters

Convert a vector in crystal space to unitary cartesian components. Return a real vector of Dimension(3)

Cart_Vector

Real Function Cart_Vector (Code, V, Celda)

Character (Len=*)	Intent(in)	Code	D: Direct R: Reciprocal
Real(Kind=CP), Dimension (3)	Intent(in)	V	Vector
Type(Crystal_Cell_Type)	Intent(in)	Celda	Cell parameters

Convert a vector in crystal space to cartesian components. Return a real vector of Dimension(3)

Convert_B_Betas

Real Function Convert_B_Betas(B, Cell)

Real(Kind=CP), Dimension (6)	Intent(in)	B	B vector: B ₁₁ B ₂₂ B ₃₃ B ₁₂ B ₁₃ B ₂₃
Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters

Convert Thermal factors from B to β . Return a vector of Dimension(6)

Convert_B_U

Real Function Convert_B_U(B)

Real(Kind=CP), Dimension (6)	Intent(in)	B	B vector: B ₁₁ B ₂₂ B ₃₃ B ₁₂ B ₁₃ B ₂₃
Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters

Convert Thermal factors from B to U. Return a vector of Dimension(6)

Convert_Betas_B

Real Function Convert_Betas_B(Beta, Cell)

Real(Kind=CP), Dimension (6)	Intent(in)	Beta	BETA vector: $\beta_{11} \beta_{22} \beta_{33} \beta_{12} \beta_{13} \beta_{23}$
Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters

Convert Thermal factors from β to B. Return a vector of Dimension(6)

Convert_Betas_U

Real Function Convert_Betas_U (Beta, Cell)

Real(Kind=CP), Dimension (6)	Intent(in)	Beta	BETA vector: $\beta_{11} \beta_{22} \beta_{33} \beta_{12} \beta_{13} \beta_{23}$
Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters

Convert Thermal factors from β to U. Return a vector of Dimension(6)

Convert_U_B

Real Function Convert_U_B (U)

Real(Kind=CP), Dimension (6)	Intent(in)	U	U vector: $U_{11} U_{22} U_{33} U_{12} U_{13} U_{23}$
------------------------------	------------	---	---

Convert Thermal factors from U to B. Return a vector of Dimension(6)

Convert_U_Betas

Real Function Convert_U_Betas (U, Cell)

Real(Kind=CP), Dimension (6)	Intent(in)	U	U vector: $U_{11} U_{22} U_{33} U_{12} U_{13} U_{23}$
Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters

Convert Thermal factors from U to β . Return a vector of Dimension(6)

Get_Betas_from_Biso

Real Function Get_Betas_from_Biso (Biso, Cell)

Real(Kind=CP)	Intent(in)	Biso	Isotropic Thermal
Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters

Expand Biso to β Thermal factors.

Rot_Matrix

Real Function Rot_Matrix (U, Phi, Celda)

Real(Kind=CP), Dimension (3)	Intent(in)	U	U vector
Real(Kind=CP)	Intent(in)	Phi	
Type(Crystal_Cell_Type)	Intent(in)	Celda	Cell parameters

Returns the matrix (Gibbs matrix (3,3)) of the active rotation of **Phi** degrees along the **U** direction: $R v = v$, the vector v is tranformed to vector v' keeping the reference frame unchanged.

If one wants to calculate the components of the vector "v" in a rotated reference frame it suffices to invoke the function using "-phi". If **Celda** is present, **U** is in **Celda** coordinates, if not **U** is in cartesian coordinates.

U_Equiv

Real Function U_Equiv (Cell, TH_U)

Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters
Real(Kind=CP), Dimension (6)	Intent(in)	TH_U	U vector: U ₁₁ U ₂₂ U ₃₃ U ₁₂ U ₁₃ U ₂₃

Subroutine to obtain the U_{eq} from U's

Subroutines

- [Change_Setting_Cell](#)
- [Get_Conventional_Cell](#)
- [Get_Cryst_Family](#)
- [Get_Deriv_Orth_Cell](#)
- [Get_Primitive_Cell](#)
- [Get_Transform_Matrix](#)
- [Get_TwoFold_Axes](#)
- [Init_Err_Crys](#)
- [Niggli_Cell](#)
- [Set_Crystal_Cell](#)
- [Write_Crystal_Cell](#)

Change_Setting_Cell

Subroutine Change_Setting_Cell (Cell, Mat, CellN, MatKind)

Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell components
Real(Kind=CP), Dimension (3,3)	Intent(in)	Mat	Transformation array
Type(Crystal_Cell_Type)	Intent(out)	CellN	New Cell components
Character (Len=*)	Intent(in)	MatKind	If present and it is IT

Transform the **CELL** object in **CELLN** using the transformation matrix **MAT**

Get_Conventional_Cell

Subroutine Get_Conventional_Cell(Twofold, Cell, Tr, Message, Ok)

Type(Twofold_Axes_Type)	Intent(in)	Twofold	
Type(Crystal_Cell_Type)	Intent(out)	Cell	Cell components
Integer, Dimension (3,3)	Intent(out)	Tr	
Character (Len=*)	Intent(out)	Message	
Logical	Intent(out)	Ok	

This subroutine provides the "conventional" (or quasi! being still tested) from the supplied object **Twofold** that has been obtained from a previous call to [Get_Twofold_Axes](#). The conventional unit cell can be deduced from the distribution of two-fold axes in the lattice. The cell produced in this procedure applies some rules for obtaining the conventional cell, for instance in monoclinic lattices (a single two-fold axis) the two-fold axis is along b and the final cell is right handed with a <= c and beta >= 90. It may be A,C or I centred. The conversion to the C-centred setting in the A and I centring, is not attempted. The angular tolerance for accepting a two-fold axis, or higher order axes, as such has been previously set into **Twofold%TOL** component.

[Get_Cryst_Family](#)

Subroutine Get_Cryst_Family (Cell, Car_Family, Car_Symbol, Car_System)

Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell components
Character (Len=*)	Intent(out)	Car_Famil y	
Character (Len=*)	Intent(out)	Car_Symb ol	
Character (Len=*)	Intent(out)	Car_Syste m	

Obtain the Crystal Family, Symbol and System from cell parameters

[Get_Deriv_Orth_Cell](#)

Subroutine Get_Deriv_Orth_Cell (CellP, De_OrthCell, CarType)

Type(Crystal_Cell_Type)	Intent(in)	CellP	Cell components
Real(Kind=CP), Dimension (3,3,6)	Intent(out)	De_OrthCell	
Character (Len=*), Optional	Intent(in)	CarType	A:

Subroutine to get derivative matrix of the transformation matrix to orthogonal frame.

Useful for calculations of standard deviations of distances and angles. The specialized subroutine calculating sigmas of distances Distance_And_Sigma is in CFML_Atom_TypeDef.

The output matrices De_OrthCell are the derivatives of, with respect to a(1),b(2),c(3),alpha(4),beta(5) and gamma(6) of the matrix CellP%CR_Orth_CEL.

[Get_Primitive_Cell](#)

Subroutine Get_Primitive_Cell (Lat_Type, Centered_Cell, Primitive_Cell, Transfm)

Character (Len=*)	Intent(in)	Lat_Type	Lattice type: P,A,B,C,I,R or F
Type(Crystal_Cell_Type)	Intent(in)	Centered_Cell	Cell components
Type(Crystal_Cell_Type)	Intent(out)	Primitive_Cell	
Real(Kind=CP), Dimension (3,3)	Intent(out)	Transfm	

Subroutine for getting the primitive cell from a centred cell

[Get_Transform_Matrix](#)

Subroutine Get_Transform_Matrix (CellA, CellB, Trm, Ok, Tol)

Type(Crystal_Cell_Type)	Intent(in)	CellA	Input Cell
Type(Crystal_Cell_Type)	Intent(in)	CellB	Input Cell
Real(Kind=CP), Dimension (3,3)	Intent(out)	Trm	Transformation array
Logical	Intent(out)	Ok	Flag
Real(Kind=CP), Optional	Intent(in)	Tol	Tolerance value

Subroutine for getting the transformation matrix between two primitive unit cells (the range of indices is fixed to -2 to 2)

Get_Twofold_Axes

Subroutine Get_Twofold_Axes(CellIN, Tol, Twofold)

Type(Crystal_Cell_Type)	Intent(in)	CellIN	Cell components
Real(Kind=CP)	Intent(in)	Tol	Angular tolerance in degrees
Type(Twofold_Axes_Type)	Intent(out)	Twofold	

Subroutine for getting the possible two-fold axes (within an angular tolerance **TOL**) existing in the lattice generated by the unit cell **CellIN**.

Strictly independent two-fold axes are stored in the variable **Twofold** that is of type [Twofold_Axes_Type](#). The output order of the two-fold axes is ascending in their modulus. Shorter vectors appear before longer ones. The conditions for a reciprocal or direct row to be a two-fold axis are discussed by Y. Le Page in J.Appl.Cryst. 15, 255 (1982).

Init_Err_Crys

Subroutine Init_Err_Crys ()

Subroutine that initializes errors flags in **CFML_Crystal_Metrics** module.

Niggli_Cell

Subroutine Niggli_Cell(AD, Niggli_Point, CellIN, Trans)

Real(Kind=CP), Dimension (6)	Intent(in out)	AD	Cell parameters as a vector
Real(Kind=CP), Dimension (5), Optional	Intent(out)	Niggli_Point	
Type(Crystal_Cell_Type), Optional	Intent(out)	CellIN	Cell components
Real(Kind=CP), Dimension (3,3), Optional	Intent(out)	Trans	

or

Subroutine Niggli_Cell(N_Mat, Niggli_Point, CellIN, Trans)

Real(Kind=CP), Dimension (2,3)	Intent(in out)	N_Mat	Niggli Matrix
Real(Kind=CP), Dimension (5), Optional	Intent(out)	Niggli_Point	
Type(Crystal_Cell_Type), Optional	Intent(out)	CellIN	Cell components

Real(Kind=CP), Dimension (3,3), Optional	Intent(out)	Trans	
---	--------------------	-------	--

or

Subroutine Niggli_Cell(A, B, C, AL, BE, GA, Niggli_Point, CellN, Trans)

Real(Kind=CP)	Intent(in out)	A	Cell Parameters
Real(Kind=CP)	Intent(in out)	B	
Real(Kind=CP)	Intent(in out)	C	
Real(Kind=CP)	Intent(in out)	AL	
Real(Kind=CP)	Intent(in out)	BE	
Real(Kind=CP)	Intent(in out)	GA	
Real(Kind=CP), Dimension (5), Optional	Intent(out)	Niggli_Point	
Type(Crystal_Cell_Type), Optional	Intent(out)	CellN	Cell components
Real(Kind=CP), Dimension (3,3), Optional	Intent(out)	Trans	

or

Subroutine Niggli_Cell(Cell, Niggli_Point, CellN, Ttrans)

Type(Crystal_Cell_Type)	Intent(in out)	Cell	Cell parameters
Real(Kind=CP), Dimension (5), Optional	Intent(out)	Niggli_Point	
Type(Crystal_Cell_Type), Optional	Intent(out)	CellN	Cell components
Real(Kind=CP), Dimension (3,3), Optional	Intent(out)	Trans	

or

Subroutine Niggli_Cell(A, B, C, Niggli_Point, CellN, Trans)

Real(Kind=CP), Dimension (3)	Intent(in out)	A	Vector in Cartesian components
Real(Kind=CP), Dimension (3)	Intent(in out)	B	Vector in Cartesian components
Real(Kind=CP), Dimension (3)	Intent(in out)	C	Vector in Cartesian components
Real(Kind=CP), Dimension (5), Optional	Intent(out)	Niggli_Point	

Type(Crystal_Cell_Type), Optional	Intent(out)	CellN	Cell components
Real(Kind=CP), Dimension (3,3), Optional	Intent(out)	Trans	

Calculates the Niggli cell according to information passed in the arguments of the subroutine.

Set_Crystal_Cell

Subroutine Set_Crystal_Cell (CellV, Angl, Celda, CarType, SCell, SAngl)

Real(Kind=CP), Dimension (3)	Intent(in)	CellV	a,b,c parameters
Real(Kind=CP), Dimension (3)	Intent(in)	Angl	Angles for cell
Type(Crystal_Cell_Type)	Intent(out)	Celda	Cell components
Character (Len=1), Optional	Intent(in)	CarType	Type of Cartesian Frame
Real(Kind=CP), Dimension (3), Optional	Intent(in)	SCell	Sigmas of a,b,c parameters
Real(Kind=CP), Dimension (3), Optional	Intent(in)	Sangl	Sigmas for angles

Constructs the object **Celda** of type [Crystal_Cell_Type](#)

Write_Crystal_Cell

Subroutine Write_Crystal_Cell (Celda, Lun)

Type(Crystal_Cell_Type)	Intent(in)	Celda	Cell parameters
Integer, Optional	Intent(in)	Lun	Unit to write the Cell information

Writes the cell characteristics in a file associated to the logical unit **Lun**

CFML_Crystallographic_Symmetry

This module contains everything needed for handling symmetry in Crystallography.

Part of the information is obtained from tabulated items in the module [CFML_Symmetry_Tables](#). In particular the correspondence of non standard settings Hermann-Mauguin symbols and Hall symbols for space groups. The construction of variables of the public type [Space_Group_Type](#) is done by using a variety of algorithms and methods.

Many procedures for handling symmetry (symbolic and algebraic) are provided in this module.

Parameters

- [Cubic](#)
- [HexaG](#)
- [Monoc](#)
- [Num_Spgr_Info](#)
- [Orthor](#)

- [Tetra](#)
- [Trigo](#)

Variables

- [Space Group Type](#)
- [Sym Oper Type](#)
- [Wyck Pos Type](#)
- [Wyckoff Type](#)
- [Lat Ch](#)
- [Err_Symm](#)
- [Err_Symm_Mess](#)
- [Hexa](#)
- [InLat](#)
- [Ltr](#)
- [NLat](#)
- [SpaceG](#)

Functions

- [ApplySO](#)
- [Axes_Rotation](#)
- [Get_Laue_Num](#)
- [Get_Multip_Pos](#)
- [Get_Occ_Site](#)
- [Get_PointGroup_Num](#)
- [Is_New_OP](#)
- [Lattice_Trans](#)
- [Spgr_Equal](#)
- [Sym_Prod](#)

Subroutines

- [DecodMatMag](#)
- [Get_Centring_Vectors](#)
- [Get_Crystal_System](#)
- [Get_HallSymb_From_Gener](#)
- [Get_Lattice_Type](#)
- [Get_Laue_PG](#)
- [Get_Laue_Str](#)
- [Get_Orbit](#)
- [Get_PointGroup_Str](#)
- [Get_SO_From_Fix](#)
- [Get_SO_From_Gener](#)
- [Get_SO_From_Hall](#)

- [Get_SO_From_HMS](#)
- [Get_Stabilizer](#)
- [Get_String_Resolv](#)
- [Get_SubOrbits](#)
- [Get_SymEl](#)
- [Get_SymKov](#)
- [Get_SymSymb](#)
- [Get_T_SubGroups](#)
- [Init_Err_Symm](#)
- [Inverse_Symm](#)
- [LatSym](#)
- [Read_MSymm](#)
- [Read_SymTrans_Code](#)
- [Read_XSym](#)
- [SearchOP](#)
- [Set_SpaceGroup](#)
- [Set_Spg_Mult_Table](#)
- [Setting_Change](#)
- [Similar_Transf_SG](#)
- [Sym_B_Relations](#)
- [Sym_Prod_ST](#)
- [Symmetry_Symbol](#)
- [Write_SpaceGroup](#)
- [Write_Sym](#)
- [Write_SymTrans_Code](#)
- [Write_Wyckoff](#)
- [Wyckoff_Orbit](#)

Fortran Filename

CFML_Symmetry.f90

Parameters

- [Cubic](#)
- [HexaG](#)
- [Monoc](#)
- [Num_Spgr_Info](#)
- [Orthor](#)
- [Tetra](#)
- [Trigo](#)

Cubic

Integer, Parameter :: Cubic=554

Index parameter for Cubic Groups

HexaG

Integer, Parameter :: HexaG=527

Index parameter for Hexagonal Groups

Monoc

Integer, Parameter :: Monoc=15

Index parameter for Monoclinic Groups

Num_Spgr_Info

Integer, Parameter :: Num_Spgr_Info=612

Total number (dimension) of space groups information pieces in [Spgr_Info](#) variable

Orthor

Integer, Parameter :: Orthor=163

Index parameter for Orthorhombic Groups

Tetra

Integer, Parameter :: Tetra=410

Index parameter for Tetragonal Groups

Trigo

Integer, Parameter :: Trigo=495

Index parameter for Trigonal Groups

Variables

- [Space_Group_Type](#)
- [Sym_Oper_Type](#)
- [Wyck_Pos_Type](#)
- [Wyckoff_Type](#)

- [Lat_Ch](#)
- [Err_Symm](#)
- [Err_Symm_Mess](#)
- [Hexa](#)
- [InLat](#)
- [Ltr](#)
- [NLat](#)
- [SpaceG](#)

Space_Group_Type

	Variable	Definition
Type :: Space_Group_Type		
Integer	NumSpg	Number of the Space Group
Character (Len=20)	Spg_Symb	Hermann-Mauguin Symbol
Character (Len=16)	Hall	Hall symbol
Character (Len=12)	CrystalSys	Crystal System
Character (Len=5)	Laue	Laue Class
Character (Len=5)	PG	Point group
Character (Len=5)	Info	Extra information
Character (Len=80)	SG_Setting	information about the SG setting: IT KO ML ZA Standard UnConventional
Logical	Hexa	
Character (Len=1)	Spg_Lat	Lattice type
Character (Len=2)	Spg_LatSy	Lattice type Symbol
Integer	NumLat	Number of lattice points in a cell
Real (Kind=CP), Dimension(3,12)	Latt_Trans	Lattice translations
Character (Len=51)	Bravais	String with Bravais symbol + translations
Character (Len=80)	Centre	information about Centric or Acentric
Integer	Centered	=0 Centric (-1 no at origin) =1 Acentric =2 Centric (-1 at origin)
Real (Kind=CP), Dimension(3)	Centre_Coord	Fractional coordinates of the inversion centre
Integer	NumOPS	Number of reduced set of S.O.
Integer	Multip	Multiplicity of the general position
Integer	Num_Gen	Minimum number of operators to generate the Group
Type (Sym_Oper_Type), Dimension(192)	SymOP	Symmetry operators
Character (Len=40), Dimension(192)	SymOPsymb	Strings form of symmetry operators
Type (Wyckoff_Type)	Wyckoff	Wyckoff information
Real (Kind=CP), Dimension(3,2)	R_ASym_Unit	Asymmetric unit in real space
End Type Space_Group_Type		

Sym_Oper_Type

	Variable	Definition
Type :: Sym_Oper_Type		
Integer, Dimension(3,3)	Rot	Rotational part of Symmetry Operator
Real (Kind=CP), Dimension(3)	Tr	Translational part of Symmetry Operator
End Type Sym_Oper_Type		

Wyck_Pos_Type

	Variable	Definition
Type :: Wyck_Pos_Type		
Integer	Mult	Multiplicity
Character (Len=6)	Site	Site Symmetry
Integer	NOrb	Number of elements in the orbit
Character (Len=40)	Orig	Original string
Character (Len=40), Dimension(48)	Str_Orbit	Orbit information
Character (Len=40), Dimension(192)	Extra_Orbit	
End Type Wyck_Pos_Type		

Wyckoff_Type

	Variable	Definition
Type :: Wyckoff_Type		
Integer	Num_Orbit	Number of Orbits
Type (Wyck_Pos_Type), Dimension(26)	Orbit	Orbit information
End Type Wyckoff_Type		

Lat_Ch

Character (Len=1) :: Lat_Ch

First character of the space group symbol

Err_Symm

Logical :: Err_Symm

This variable is set to .TRUE. if an error in procedures belonging to this module.

`Err_Symm_Mess`

Character (Len=150) :: Err_Symm_Mess

This variable contains information about the last error occurred in the procedures belonging to this module

`Hexa`

Logical :: Hexa

- `.FALSE.` Rotational part of symmetry operators belongs to m3m
- `.TRUE.` Rotational part of symmetry operators belongs to 6/mmm

`InLat`

Integer :: InLat

Ordinal index of the lattice

`Ltr`

Real, Dimension(3,10) :: Ltr

Centering Lattice Translations.

Up to 10 lattice centring vectors are allowed. Conventional lattice centring need only 4 vectors

`NLat`

Integer :: NLat

Multiplicity of the lattice

`SpaceG`

Character (Len=20) :: SpaceG

Space group symbol

`Functions`

- [ApplySO](#)
- [Axes_Rotation](#)
- [Get_Laue_Num](#)
- [Get_Multip_Pos](#)
- [Get_Occ_Site](#)
- [Get_PointGroup_Num](#)
- [Is_New_OP](#)
- [Lattice_Trans](#)
- [Spgr_Equal](#)
- [Sym_Prod](#)

ApplySO**Real Function ApplySO(OP, V)**

Type(Sym_Oper_Type)	Intent(in)	OP	Symmetry Operator Type
Real(Kind=CP), Dimension(3)	Intent(in)	V	Point vector

Return a vector of dimension 3. Apply a symmetry operator to a vector

Axes_Rotation**Integer Function Axes_Rotation (R)**

Integer, Dimension(3,3)	Intent(in)	R	Rotation part of Symmetry Operator
-------------------------	------------	---	------------------------------------

Determine the orden of rotation (valid for all bases). Return a zero if any error occurs.

Get_Laue_Num**Integer Function Get_Laue_Num (LaueClass)**

Character(Len=*)	Intent(in)	LaueClas	Laue Class string
		s	

Obtain the ordinal number correspoding to the Laue class symbol according to [Laue_Class](#) array. Zero if error is present

Get_Multip_Pos**Integer Function Get_Multip_Pos (X, Spg)**

Real(Kind=CP), Dimension(3)	Intent(in)	X	Position vector
Type(Space_Group_Type)	Intent(in)	Spg	Space Group

Obtain the multiplicity of a real space point given the space group

Get_Occ_Site**Real Function Get_Occ_Site (Pto, Spg)**

Real(Kind=CP), Dimension(3)	Intent(in)	Pto	Position vector
Type(Space_Group_Type)	Intent(in)	Spg	Space Group

Obtain the occupancy factor (site multiplicity/multiplicity) for **Pto**

Get_PointGroup_Num**Integer Function Get_PointGroup_Num(PgName)**

Character(Len=*)	Intent(in)	PgName	String for PointGroup
------------------	------------	--------	-----------------------

Obtain the ordinal number correspoding to the Point Group symbol according to [Point_Group](#) array. Zero if

Error is present

[Is_New_OP](#)

Logical Function Is_New_OP(OP, N, List_OP)

Type(Sym_Oper_Type)	Intent(in)	OP	Symmetry operator
Integer	Intent(in)	N	Number of Op in the LIST_OP
Type(Sym_Oper_Type), Dimension(:)	Intent(in)	List_O P	List of N symmetry operators

Determine if a symmetry operator is or not in a given list

[Lattice_Trans](#)

Logical Function Lattice_Trans (V, Lat)

Real(Kind=CP), Dimension(3)	Intent(in)	V	Vector
Character(Len=*)	Intent(in)	Lat	Lattice Character

Determine whether a vector is a lattice vector depending on the Bravais lattice.

[Spgr_Equal](#)

Logical Function Spgr_Equal (SpaceGroup1, SpaceGroup2)

Type(Space_Group_Type)	Intent(in)	SpaceGrou p1	Space group
Type(Space_Group_Type)	Intent(in)	SpaceGrou p2	Space group

Determine if two SpaceGroups are equal

[Sym_Prod](#)

Function Sym_Prod (SymA, SymB, ModLat)

Type(Sym_Oper_Type)	Intent(in)	SymA	Space group
Type(Sym_Oper_Type)	Intent(in)	SymB	Space group
LogicalL, Optional	Intent(in)	ModLat	

Obtain the symmetry operation corresponding to the product of two operators. The return is a variable of type [Sym_Oper_Type](#)

If **ModLat**=.true. or it is not present, the translation part of the resulting operator is reduced to have components < 1.0

[Subroutines](#)

- [DecodMatMag](#)
- [Get_Centring_Vectors](#)
- [Get_Crystal_System](#)

- [Get_HallSymb_From_Gener](#)
- [Get_Lattice_Type](#)
- [Get_Laue_PG](#)
- [Get_Laue_Str](#)
- [Get_Orbit](#)
- [Get_PointGroup_Str](#)
- [Get_SO_From_Fix](#)
- [Get_SO_From_Gener](#)
- [Get_SO_From_Hall](#)
- [Get_SO_From_HMS](#)
- [Get_Stabilizer](#)
- [Get_String_Resolv](#)
- [Get_SubOrbits](#)
- [Get_SymEl](#)
- [Get_SymKov](#)
- [Get_SymSymb](#)
- [Get_T_SubGroups](#)
- [Init_Err_Symm](#)
- [Inverse_Symm](#)
- [LatSym](#)
- [Read_MSymm](#)
- [Read_SymTrans_Code](#)
- [Read_XSym](#)
- [SearchOP](#)
- [Set_SpaceGroup](#)
- [Set_Spg_Mult_Table](#)
- [Setting_Change](#)
- [Similar_Transf_SG](#)
- [Sym_B_Relations](#)
- [Sym_Prod_ST](#)
- [Symmetry_Symbol](#)
- [Write_SpaceGroup](#)
- [Write_Sym](#)
- [Write_SymTrans_Code](#)
- [Write_Wyckoff](#)
- [Wyckoff_Orbit](#)

DecodMatMag

Subroutine DecodMatMag (Sim, XYZString)

Integer, Dimension(3,3)	Intent(in)	Sim	Rotation matrix
Character(Len=*)	Intent(out)	XYZString	String (Mx,My,Mz)

Supplies a string of the form (Mx,My,Mz) for the rotation matrix SIM.

Note: Logical [Hexa](#) must be defined.

Get_Centring_Vectors

Subroutine Get_Centring_Vectors (L, LatC)

Integer	Intent(in out)	L	Number of centring vectors
Real(Kind=CP), Dimension(:, :, :)	Intent(in out)	LatC	Centering vectors. Array (3,L)

Subroutine to complete the centring vectors of a centered lattice. It is useful when non-conventional lattices are used.

Get_Crystal_System

Subroutine Get_Crystal_System (NG, Ss, ISystm, Crys)

Integer	Intent(in)	NG	Number of Operators (not related by inversion and lattice translations)
Integer, Dimension(:, :, :)	Intent(in)	Ss	Rotation Part (3,3,48)
Integer	Intent(out)	ISystm	Number for Crystal System according to Sys_Cry
Character(Len=1)	Intent(out)	Crys	Symbol of Crystal family

or

Subroutine Get_Crystal_System (NG, Gen, ISsystm, Crys)

Integer	Intent(in)	NG	Number of Operators (not related by inversion and lattice translations)
Character(Len=*) , Dimension(:)	Intent(in)	Gen	Jones Faithful form of symmetry operators
Integer	Intent(out)	ISystm	Number for Crystal System according to SYS_CRY
Character(Len=1)	Intent(out)	Crys	Symbol of Crystal family

Obtain the number and string of the Crystal System from a set of operators

Get_HallSymb_From_Gener

Subroutine Get_HallSymb_From_Gener (SpaceGroup, SpaceH)

Type(Space_Group_Type)	Intent(in out)	SpaceGrou p	SpaceGroup
Character(Len=*)	Intent(out)	SpaceH	Hall Symbol

Determines the Hall symbol.

In general this routine try to obtain the Hall symbol from generators so you need call [Get_SO_From_Gener](#) before and call [Set_Spgr_Info](#).

Get_Lattice_Type

Subroutine Get_Lattice_Type (L, LatC, LatTyp)

Integer	Intent(in)	L	Number of centring vectors
Real(Kind=CP), Dimension(:,:)	Intent(in)	LatC	Centring vectors. Array (3,11)
Character(Len=*)	Intent(out)	LatTyp	Lattice symbol

Subroutine to get the lattice symbol from a set of centring vectors.

Get_Laue_PG

Subroutine Get_Laue_PG (SpaceGroup, Laue_Car, Point_Car)

Type(Space_Group_Type)	Intent (in)	SpaceGrou p	Space Group
Character(Len=*)	Intent (out)	Laue_Car	String with Laue symbol
Character(Len=*)	Intent (out)	Point_Car	String with Point Group symbol

Subroutine to get the information of Laue and Point Group.

Note: Point group determination is only valid for conventional bases

Get_Laue_Str

Subroutine Get_Laue_Str (lLaue, Laue_Str)

Integer	Intent (in)	lLaue	Ordinal number in Laue_Class
Character(Len=*)	Intent (out)	Laue_Str	String with the Laue class

Obtain the string for the Laue class. Control of error is present

Get_Orbit

Subroutine Get_Orbit (X, Spg, Mult, Orb, Ptr, Str, Prim)

Real(Kind=CP), Dimension(3)	Intent(in)	X	Position vector
Type(Space_Group_Type)	Intent(in)	Spg	Space Group
Integer	Intent(out)	Mult	Multiplicity
Real(Kind=CP), Dimension(:,:)	Intent(out)	Orb	List of equivalent positions
Integer, Dimension(:), Optional	Intent(out)	Ptr	Pointer to effective symops
Integer, Dimension(:), Optional	Intent(out)	Str	Pointer to stabilizer
Character(Len=*), Optional	Intent(in)	Prim	If given, only the primitive cell is considered

Obtain the multiplicity and list of equivalent positions (including centring!) modulo Integer lattice translations.

It provides also pointers to the stabilizer and to the symmetry operators changing effectively the position.

Get_PointGroup_Str

Subroutine Get_PointGroup_Str (IPG, Str)

Integer	Intent(in)	IPG	Ordinal number according to Point_Group
Character(Len=*)	Intent(out)	Str	String for Point Group

Obtain the string for the Point Group. Error control is present

[Get_SO_From_Fix](#)**Subroutine Get_SO_From_Fix (ISystm, ISymCe, IBravl, NG, SS, TS, LatSy, CO, SpaceGen)**

Integer	Intent(out)	ISystm	Number of the crystalline system 1:T, 2:M, 3:O, 4:T, 5:R-Trg, 6:H, 7:C
Integer	Intent(out)	ISymCe	0: Centric (-1 not at origin) 1: Acentric 2: Centric (-1 at origin)
Integer	Intent(out)	IBravl	1:P, 2:A, 3:B, 4:C, 5:I, 6:R, 7:F, 8:Z
Integer	Intent(in)	NG	Number of symmetry operators
Integer, Dimension(:,:,:)	Intent(in)	SS	Rotation parts of the symmetry operators (3,3,48)
Real(Kind=CP), Dimension(:,:)	Intent(in)	TS	Translation parts of the symmetry operators (3,48)
Character(Len=2)	Intent(out)	LatSy	Bravais Lattice symbol
Real(Kind=CP), Dimension(3)	Intent(out)	CO	Coordinates of origin
Character(Len=1)	Intent(out)	SpaceGen	Type of Cell

Determines some of items of the object [Space_Group_Type](#) from Fixed symmetry operators given by user.

[Get_SO_From_Gener](#)**Subroutine GET_SO_FROM_GENER (ISystm, ISymCe, IBravl, NG, SS, TS, LatSy, CO, Num_G, SpaceGen)**

Integer	Intent(out)	ISystm	Number of the crystalline system 1:T, 2:M, 3:O, 4:T, 5:R-Trg, 6:H, 7:C
Integer	Intent(out)	ISymCe	0: Centric (-1 not at origin) 1: Acentric 2: Centric (-1 at origin)
Integer	Intent(out)	IBravl	1:P, 2:A, 3:B, 4:C, 5:I, 6:R, 7:F, 8:Z
Integer	Intent(in out)	NG	Number of defined generators in: out:
Integer, Dimension(:,:,:)	Intent(in out)	SS	Number of symmetry operators in: out:
Real(Kind=CP), Dimension(:,:)	Intent(in out)	TS	Rotation parts of the given generators (3,3,48) in: out:
Character(Len=2)	Intent(out)	LatSy	Rotation parts of the symmetry operators in: out:
Real(Kind=CP),	Intent	CO	Translation parts of the given generators (3,48) in: out:
			Translation parts of the symmetry operators
			Coordinates of origin

Dimension(3)	(out)			
Integer	Intent (out)	Num_G		Minimum number of generators
Character(Len=1)	Intent (out)	SpaceGen		Type of Cell

Calculates the whole set of symmetry operators from a set of given generators.

Get_SO_From_Hall

Subroutine Get_SO_From_Hall (ISystm, ISymCe, IBravl, NG, SS, TS, LatSy, CO, Num_G, Hall)

Integer	Intent (out)	ISystm		Number of the crystalline system 1:T, 2:M, 3:O, 4:T, 5:R-Trg, 6:H, 7:C)
Integer	Intent (out)	ISymCe		0: Centric (-1 not at origin) 1: Acentric 2: Centric (-1 at origin)
Integer	Intent (out)	IBravl		1:P, 2:A, 3:B, 4:C, 5:I, 6:R, 7:F, 8:Z
Integer	Intent (out)	NG		Number of symmetry operators
Integer, Dimension(:, :, :)	Intent (out)	SS		Rotation parts of the symmetry operators (3,3,48)
Real(Kind=CP), Dimension(:, :)	Intent (out)	TS		Translation parts of the symmetry operators (3,48)
Character(Len=2)	Intent (out)	LatSy		Bravais Lattice symbol
Real(Kind=CP), Dimension(3)	Intent (out)	CO		Coordinates of origin
Integer	Intent (out)	Num_G		Number of generators
Character(Len=20)	Intent(in)	Hall		Hall Space group symbol

Subroutine to get all the information contained in the Hall symbol. This routine to interpret the Hall symbol for a space group.

Get_SO_From_HMS

Subroutine Get_SO_From_HMS (ISystm, ISymCe, IBravl, NG, SS, TS, LatSy, SpaceH)

Integer	Intent (out)	ISystm		Number of the crystalline system 1:T, 2:M, 3:O, 4:T, 5:R-Trg, 6:H, 7:C)
Integer	Intent (out)	ISymCe		0: Centric (-1 not at origin) 1: Acentric 2: Centric (-1 at origin)
Integer	Intent (out)	IBravl		1:P, 2:A, 3:B, 4:C, 5:I, 6:R, 7:F, 8:Z
Integer	Intent (out)	NG		Number of symmetry operators

Integer, Dimension(:,:)	Intent(out)	SS		Rotation parts of the symmetry operators (3,3,48)
Real(Kind=CP), Dimension(:,:)	Intent(out)	TS		Translation parts of the symmetry operators (3,48)
Character(Len=2)	Intent(out)	LatSy		Bravais Lattice symbol
Character(Len=20)	Intent(in)	SpaceH		H-M Spacegroup symbol

Subroutine to get all the information contained in the H-M symbol. Routine to interpret Hermann-Mauguin symbol for space group

Get_Stabilizer

Subroutine Get_Stabilizer (X, Spg, Order, Ptr)

Real(Kind=CP), Dimension(3)	Intent(in)	X		Position vector
Type (Space_Group_Type)	Intent(in)	Spg		Space group
Integer	Intent(out)	Order		Number of sym.op. keeping invariant the position x
Integer, Dimension(:)	Intent(out)	Ptr		Array pointing to the symmetry operators numbers of the stabilizer (point group) of x

Subroutine to obtain the list of symmetry operator of a space group that leaves invariant an atomic position. This subroutine provides a pointer to the symmetry operators of the site point group.

Get_String_Resolv

Subroutine Get_String_Resolv (T, X, IX, Symb)

Real(Kind=CP), Dimension(3)	Intent(in)	T		Traslation part
Real(Kind=CP), Dimension(3)	Intent(in)	X		real part of Variable
Integer, Dimension(3)	Intent(in)	IX		1:X, 2:Y, 3:Z
Character(Len=*)	Intent(out)	Symb		String

Returning a string for point, axes or plane give as written in fractional form from [Resolv_Sist](#) procedures in CFML_Math_3D.

Get_SubOrbits

Subroutine Get_SubOrbits (X, Spg, Ptr, Mult, Orb, Ind, Conv)

Real(Kind=CP), Dimension(3)	Intent(in)	X		Position vector
Type (Space_Group_Type)	Intent(in)	Spg		Space Group
Integer, Dimension(:)	Intent(in)	Ptr		Pointer to symops of a subgroup
Integer	Intent	Mult		Multiplicity

	(out)			
Real(Kind=CP), Dimension(:,:)	Intent(out)	Orb		List of equivalent positions
Integer, Dimension(:)	Intent(out)	Ind		Number of the suborbits
Character(Len=*) , Optional	Intent(in)	Conv		If present centring transl. are considered

Obtain the multiplicity and list of equivalent positions modulo lattice translations (including centring!) of a position. When symmetry operators of a subgroup of **SPG** is given an index vector **inD** gives the division in subOrbits.

The pointer **PTR** indicates the symmetry operators of **SPG** belonging to the subgroup. The first zero value of **PTR** terminates the search.

If the optional argument **CONV** is given the centring translations are considered. The orbits are formed by all atoms within a conventional unit cell. Otherwise the orbit is formed only with the content of a primitive cell.

Get_SymEl

Subroutine Get_SymEl (Sim, XYZString)

Integer, Dimension(3,3)	Intent(in)	Sim	Rotation matrix
Character(Len=*)	Intent(out)	XYZStrin g	String (Mx,My,Mz)

Supplies a string with the "symmetry element" (I.T.) for the rotation matrix **SIM**. They correspond to the symbols given in I.T. for space groups Pm3m and P6/mmm.

Note: Logical [Hexa](#) must be defined.

Get_SymKov

Subroutine Get_SymKov (Sim, XYZString)

Integer, Dimension(3,3)	Intent(in)	Sim	Rotation matrix
Character(Len=*)	Intent(out)	XYZStrin g	String (Mx,My,Mz)

Supplies a string with the "symmetry element" (I.T.) for the rotation matrix **SIM**. They correspond to the symbols Kovalev.

Note: Logical [Hexa](#) must be defined.

Get_SymSymb

Subroutine Get_SymSymb (Sim, Tt, StrSym)

Integer, Dimension(3,3) or Real(Kind=CP), Dimension(3,3)	Intent(in)	Sim	Rotational part of the S.O.
Real(Kind=CP), Dimension(3)	Intent(in)	Tt	Translational part of the S.O.
Character(Len=*)	Intent(out)	StrSym	String in th form X,Y,-Z, ...

Obtain the Jones Faithful representation of a symmetry operator

Get_T_SubGroups

Subroutine Get_T_SubGroups (Spg, SubG, NSG)

Type(Space_Group_Type)	Intent(in)	Spg	SpaceGroup
Type(Space_Group_Type), Dimension(:)	Intent(out)	SubG	SubGroups
Integer	Intent(out)	NSG	Number of SubGroups

Subroutine to obtain the list of all non-trivial *translationengleiche* subgroups (t-subgroups) of a given space group.

The unit cell setting is supposed to be the same as that of the input space group **SPG**. The search of space sub-groups is performed using a systematic combination of the symmetry operators of the group.

Init_Err_Symm

Subroutine Init_Err_Symm ()

Subroutine that initializes errors flags in **CFML_Crystallographic_Symmetry** module.

Inverse_Symm

Subroutine Inverse_Symm (R, T, S, U)

Integer, Dimension(3,3)	Intent(in)	R	Rotational Part
Real(Kind=CP), Dimension(3)	Intent(in)	T	Traslational part
Integer, Dimension(3,3)	Intent(out)	S	New Rotational part
Real(Kind=CP), Dimension(3)	Intent(out)	U	New traslational part

Calculates the inverse of the symmetry operator (R,t)

LatSym

Subroutine LatSym (Symb, NumL, LatC)

Character(Len=*)	Intent(in)	Symb	Space Group H-M/Hall symbol
Integer, Optional	Intent(in)	NumL	Number of centring vectors
Real(Kind=CP), Dimension (3,11), Optional	Intent(in)	LatC	Centering vectors

Provides the Lattice type of the space group of **SYMB**.

Also gives the index [InLat](#) of the lattice, the multiplicity [NLat](#) and the fractional lattice translations [Ltr](#) and [Lat_Ch](#).

Read_MSymm

Subroutine Read_MSymm (Info, Sim, P_Mag)

Character(Len=*)	Intent(in)	Info	input string with S.Op. in the form: MSYM
------------------	------------	------	---

			u,w,w,p_mag
Integer, Dimension(3,3)	Intent(out)	Sim	Rotation matrix
Real(Kind=CP)	Intent(out)	P_Mag	Magnetic Phase

Read magnetic symmetry operators in the form U,V,W, etc...

Provides the magnetic rotational matrix and phase associated to a MSYM symbol

Read_SymTrans_Code

Subroutine Read_SymTrans_Code (Code, N, Tr)

Character(Len=*)	Intent(in)	Code	String to read
Integer	Intent(out)	N	Number of Op. S.
Real(Kind=CP), Dimension(3)	Intent(out)	Tr	Traslation applied

Read a Code string for reference the symmetry operator and the Traslation applied.

Example: _2.555 : N= 2; TR=(0.0, 0.0, 0.0)
 _3.456 : N= 3, TR=(-1.0, 0.0, 1.0)

Read_XSym

Subroutine Read_XSym (Info, IStart, Sim, Tt)

Character(Len=*)	Intent(in)	Info	String with the symmetry symbol in the form: SYMM x,-y+1/2,z
Integer	Intent(in)	IStart	Starting index of info to read in
Integer, Dimension(3,3)	Intent(out)	Sim	Rotational part of the S.O.
Real(Kind=CP), Dimension(3), Optional	Intent(out)	Tt	Traslational part of S.O.

Read symmetry or transformation operators in the form X,Y,Z, etc...

Provides the rotational matrix and translation associated a to SYMM symbol in the Jones Faithful representation.

SearchOP

Subroutine SearchOP (Sim, I1, I2, ISL)

Integer, Dimension(3,3)	Intent(in)	Sim	Rotation matrix
Integer	Intent(in)	I1	index for search
Integer	Intent(in)	I2	index for search
Integer	Intent(out)	ISL	index of the matrix Mod6 (ISL,:,:)=SIM

Search the index on MOD6 variable

- Matrices of m3m (not hexagonal): I1=1 I2=24

- Matrices of 6/mmm (hexagonal): I1=25 I2=36

Set_SpaceGroup

Subroutine Set_SpaceGroup (SpaceGen, SpaceGroup, Gen, NGen, Mode, Force_Hall)

Character(Len=*)	Intent(in)	SpaceGen	String with Number, Hall or Hermann-Mauguin
Type(Space_Group_Type)	Intent(out)	SpaceGroup	Space Group
Character(Len=*), Dimension (:), Optional	Intent(in)	Gen	String Generators
Integer, Optional	Intent(in)	NGen	Number of Generators
Character(Len=*), Optional	Intent(in)	Mode	Value must be: HMS, ITC, HALL, GEN, FIX
Character(Len=*), Optional	Intent(in)	Force_Hall	If present force generation from Hall

Subroutine that construct the object SpaceGroup from the H-M or Hall symbol.

Expand the set of operators including centre of symmetry and non Integer translations for centred cells. If the optional argument **GEN** is given, then **NGEN** and **MODE="GEN"** should be given.

If the optional argument **MODE="ITC"**, the space group will be generated using the generators given in the international Tables for the standard setting. in this case the string in SPACEGEN should correspond to the Hermann-Mauguin symbol.

If the optional argument **MODE="HMS","HALL"** is given the string in SPACEGEN should correspond to the desired symbol.

If GEN, NGEN and MODE are not given but FORCE_HALL="F_HALL" is given, the generation of the symmetry operators from the symbol of the space group is according to the Hall symbol even if the provided symbol is of Hermann-Mauguin type.

The use of the different options give rise to different ordering of the symmetry operators or different origins and settings for the same space group.

Set_Spg_Mult_Table

Subroutine Set_Spg_Mult_Table (Spg, Tab, Complete)

Type(Space_Group_Type)	Intent(in)	Spg	String with Number, Hall or Hermann-Mauguin
Integer, Dimension(:,:)	Intent(out)	Tab	Table
Logical, Optional	Intent(in)	Complete	

Subroutine to construct the multiplication table of the factor group of a space group. Two operators are equal if they differ only in a lattice translation. The multiplication table is a square matrix with Integer numbers corresponding to the ordering of operators in the space group.

If **Complete** is not present, or if **Complete=.FALSE.**, we consider only the symmetry operators corresponding to the "primitive" content of the unit cell, so a maximum 48x48 matrix is needed to hold the table in this case. If **Complete** is present and **.TRUE.**, the full table is constructed.

Setting_Change

Subroutine Setting_Change (From_Syst, To_Syst, SpaceGroup, CarR_Sym, ICar_Sym)

Character(Len=2)	Intent(in)	From_Syst	Values: IT, ML, KO, BC, ZA
Character(Len=2)	Intent(in)	To_Syst	Values: IT, ML, KO, BC, ZA
Type(Space_Group_Type)	Intent(in out)	SpaceGro up	Space Group
Character(Len=35)	Intent(out)	Car_Sym	
Character(Len=35)	Intent(out)	ICar_Sym	

Translate From From_Syst to To_syst the set of symmetry operators

Similar_Transf_SG

Subroutine Similar_Transf_SG (Mat, Orig, Spg, SpgN, MatKind, Fix_Lat)

Real(Kind=CP), Dimension (3,3)	Intent(in)	Mat	Matrix transforming the basis
Real(Kind=CP), Dimension(3)	Intent(in)	Orig	Coordinates of the new origin
Type(Space_Group_Type)	Intent(in)	Spg	Space Group
Type(Space_Group_Type)	Intent(out)	SpgN	Maximum subgroup of SPG
Character(Len=*), Optional	Intent(in)	MatKind	Type of the input matrix
Character(Len=*), Optional	Intent(in)	Fix_Lat	Fixing Lattice type

Subroutine to construct a space group **SPGN** that is a maximal subgroup of the input space group **SPG** compatible with the transformation of the basis corresponding to the matrix **MAT** and the new origin **ORIG**.

The transformed **SPGN** will have (if it is the case) conventional centring vectors.

If **MATKind** is given and matkind="it"/"IT", the input matrix is given as in international Tables:

$$(a' b' c') = (a b c) \text{ Mat}$$

If **MATKind** is not given or if it is not equal to "it"/"IT" the input matrix is the transpose of the international convention (column matrices for basis vectors).

The new space group is obtained using the properties of conventional Bravais lattices and symmetry operators. Only the symmetry operators of the conventional form are retained to construct the new space group.

If the Hermann-Mauguin symbol is not given, that means it correspond to a special setting. The Hall symbol is always given.

The coordinates of the origin is always given with respect to the (a b c) basis.

If **FIX_LAT** is given a conventional lattice centring, this is fixed irrespective of the centring obtained by applying the similarity transformation. For instance is **FIX_LAT**="P" and the transformation implies new centring vectors or the input group is centred, the generators with fractional translations are removed from the group. If **FIX_LAT**="A" (or whatever) the program will add the corresponding generators irrespective that the generator is in the original/transformed group.

Sym_B_Relations

Subroutine Sym_B_Relations (OP / Symb, B_Ind, B_Fac)

Integer, Dimension(3,3) or Character(Len=*)	Intent(in)	OP	Rotation Matrix
		Symb	Symmetry string
Integer, Dimension(6)	Intent(out)	B_Ind	B index
Real(Kind=CP), Dimension(6)	Intent(out)	B_Fac	B Factor

Symmetry relations among coefficients of the anisotropic temperature factor.

Order for B is: B₁₁ B₂₂ B₃₃ B₁₂ B₁₃ B₂₃

Sym_Prod_St

Subroutine Sym_Prod_St (SymA, SymB, SymAB, ModLat)

Character(Len=*)	Intent(in)	SymA	
Character(Len=*)	Intent(in)	SymB	
Character(Len=*)	Intent(out)	SymAB	
Logical, Optional	Intent(in)	ModLat	

Obtain the symbol/Op/Matrix+trans of the symmetry operation corresponding to the product of two operators given in the Jone's Faithful(symbol) representation or in Symmetry Operator type.

If **ModLat=.TRUE.** or it is not present, the traslation part of the resulting operator is reduced to have components < 1.0

Symmetry_Symbol

Subroutine Symmetry_Symbol (OP, Symb)

Type(Sym_Oper_Type)	Intent(in)	OP	Symmetry Operator
Character(Len=*)	Intent(out)	Symb	String for the symbol of the symmetry element

or

Subroutine Symmetry_Symbol (S, T, Symb)

Integer, Dimension(3,3)	Intent(in)	S	Rotational part
Real(Kind=CP), Dimension(3)	Intent(in)	T	Traslational part
Character(Len=*)	Intent(out)	Symb	String for the symbol of the symmetry element

or

Subroutine Symmetry_Symbol (Symm, Symb)

Character(Len=*)	Intent(in)	Symm	String Symmetry Operator
Character(Len=*)	Intent(out)	Symb	String for the symbol of the symmetry element

Obtain the symbol of the symmetry element of the operator Op

Write_SpaceGroup

Subroutine Write_SpaceGroup (SpaceGroup, Iunit, Full)

Type(Space_Group_Type)	Intent(in)	SpaceGroup	Space Group
Integer, Optional	Intent(in)	Iunit	Write information on IUNIT
Logical, Optional	Intent(in)	Full	Full operator or not

Writing in file of logical unit **IUNIT** the characteristics of the space group **SpaceGroup**. Part of the information contained in **SpaceGroup** may be undefined, depending on the tabulated nature of the item.

If **FULL=.TRUE.** is present the whole group is output including the symmetry symbol associated to each operator.

Write_Sym

Subroutine Write_Sym (Lun, Indx, Sim, Tt, P_Mag, Mag)

Integer	Intent(in)	Lun	Logical unit of the file to write
Integer	Intent(in)	Indx	Ordinal of the current Symm.Operator
Integer, Dimension(3,3)	Intent(in)	Sim	Rotational part of the S.O.
Real(Kind=CP), Dimension(3)	Intent(in)	Tt	Translation part of the S.O.
Real(Kind=CP)	Intent(in)	P_Mag	Magnetic phase of the magnetic S.O.
Logical	Intent(in)	Mag	.true. if it is a magnetic S.O.

Writing the reduced set of symmetry operators.

Note: Logical [Hexa](#) must be defined (valid for conventional bases)

Write_SymTrans_Code

Subroutine Write_SymTrans_Code (N, Tr, Code)

Integer	Intent(in)	N	Number of the Symmetry Operator
Real(Kind=CP), Dimension(3)	Intent(in)	Tr	Traslational part
Character(Len=*)	Intent(out)	Code	String

Write the code string for reference the symmetry operator and the Traslation applied.

Example: N=2; TR=(0.0, 0.0, 0.0) -> CODE=_2.555
 N=3; TR=(-1.0, 0.0, 1.0) -> CODE=_3.456

Write_Wyckoff

Subroutine Write_Wyckoff (Wyckoff, Spg_Name, Lun, Sorting)

Type(Wyckoff_Type)	Intent(in)	Wyckoff	Wyckoff Type variable
Character(Len=*)	Intent(in)	Spg_Name	Space Group
Integer, Optional	Intent(in)	Lun	Unit to write the information

Logical, Optional	Intent(in)	Sorting	.true. for sorting list
--------------------------	-------------------	---------	-------------------------

Print/Write the Wyckoff positions in LUN unit

Wyckoff_Orbit

Subroutine Wyckoff_Orbit (SpaceGroup, WyckoffStr, N_Orbit, OrbitStr)

Type(Space_Group_Type)	Intent(in)	SpaceGroup	Space Group
Character(Len=*)	Intent(in)	WyckoffStr	Representative of the Orbit
Integer	Intent(out)	N_Orbit	Unit to write the information
Character(Len=*), Dimension(:)	Intent(out)	OrbitStr	.true. for sorting list

Calculation of the Wyckoff positions from the representative element

CFML_ILL_Instrm_Data

Subroutines related to instrument information from ILL

Variables

- [Basic_NumC_Type](#)
- [Basic_NumI_Type](#)
- [Basic_NumR_Type](#)
- [Calibration_Detector_Type](#)
- [Diffractometer_Type](#)
- [Generic_Numor_Type](#)
- [ILL_Data_Record_Type](#)
- [Powder_Numor_Type](#)
- [SXTAL_Numor_Type](#)
- [SXTAL_Orient_Type](#)

- [Current_Instrm](#)
- [Current_Orient](#)
- [Cycle_Number](#)
- [Err_ILLData](#)
- [Err_ILLData_Mess](#)
- [ILL_Data_Directory](#)
- [Instrm_Directory](#)
- [Machine_Name](#)
- [Year_ILLData](#)

Subroutines

- [Allocate_Numors](#)
- [Define_Uncompress_Program](#)
- [Get_Absolute_Data_Path](#)

- [Get_Next_YearCycle](#)
- [Get_Single_Frame_2D](#)
- [Init_Err_ILLData](#)
- [Initialize_Data_Directory](#)
- [Initialize_Numor](#)
- [PowderNumors_To_DiffPattern](#)
- [Read_Calibration_File](#)
- [Read_Current_Instrm](#)
- [Read_Numor](#)
- [Set_Current_Orient](#)
- [Set_Default_Instrument](#)
- [Set_ILL_Data_Directory](#)
- [Set_Instrm_Directory](#)
- [Update_Current_Instrm_UB](#)
- [Write_Current_Instrm_Data](#)
- [Write_HeaderInfo_Numor](#)
- [Write_Numor_Info](#)

Fortran Filename

CFML_ILL_instrm_Data.f90

Variables

- [Basic_NumC_Type](#)
- [Basic_NumI_Type](#)
- [Basic_NumR_Type](#)
- [Calibration_Detector_Type](#)
- [Diffractometer_Type](#)
- [Generic_Numor_Type](#)
- [ILL_Data_Record_Type](#)
- [Powder_Numor_Type](#)
- [SXTAL_Numor_Type](#)
- [SXTAL_Orient_Type](#)

- [Current_Instrm](#)
- [Current_Orient](#)
- [Cycle_Number](#)
- [Err_ILLData](#)
- [Err_ILLData_Mess](#)
- [ILL_Data_Directory](#)
- [Instrm_Directory](#)
- [Machine_Name](#)
- [Year_ILLData](#)

[Basic_NumC_Type](#)

	Variable	Definition
Type :: Basic_NumC_Type		
Integer	N	Number of elements
Character(Len=40), Dimension(:), Allocatable	NameVar	Name of the different fields
Character(Len=80), Dimension(:), Allocatable	CValues	String Values
End Type Basic_NumC_Type		

Basic_NumI_Type

	Variable	Definition
Type :: Basic_NumI_Type		
Integer	N	Number of elements
Character(Len=40), Dimension(:), Allocatable	NameVar	Name of fields
Integer, Dimension(:), Allocatable	IValues	Integer values
End Type Basic_NumI_Type		

Basic_NumR_Type

	Variable	Definition
Type :: Basic_NumR_Type		
Integer	N	Number of elements
Character(Len=40), Dimension(:), Allocatable	NameVar	Name of fields
Real(Kind=CP), Dimension(:), Allocatable	RValues	Real values
End Type Basic_NumR_Type		

Calibration_Detector_Type

	Variable	Definition
Type :: Calibration_Detector_Type		
Character(Len=4)	Name_Instr	Instrument Name

	m	
Integer	NDet	Number of Detectors in the instrument
Integer	NPointsDet	Number of Points belong to Detector
Logical, Allocatable, Dimension (:)	Active	Flag to set the respective detector in use (Dim=NDet)
Real(Kind=CP), Allocatable, Dimension(:)	PosX	Position of Detector in X (2Theta) (Dim=NDet)
Real(Kind=CP), Allocatable, Dimension(:, :)	Effic	Efficiency of each detector point (array NPointsDet x NDet)
End Type		
Calibration_Detector_Type		

Diffractometer_Type

	Variable	Definition
Type :: Diffractometer_Type		
Character(Len=80)	Info	information about the instrument
Character(Len=10)	Name_Inst	Short name of the instrument
Character(Len=15)	Geom	Eulerian_4C, Kappa_4C, Lifting_arm, Powder, Laue,...
Character(Len=6)	BL_Frame	Kind of BL-frame: z-up or z-down
Character(Len=4)	Dist_Units	distance units: mm, cm, inch
Character(Len=4)	Angl_Units	angle units: rad, deg
Character(Len=30)	Detector_Type	Point, Flat_rect, Cylin_ImPlate, Tube_PSD, ...
Real(Kind=CP)	Dist_Samp_Detector	Dist. to centre for: Point, Flat_rect, Tube_PSD Radius for: Cylin_ImPlate
Real(Kind=CP)	Wave_Min	Minimum wavelength (Laue diffractometers)
Real(Kind=CP)	Wave_Max	Maximum wavelength (Laue diffractometers)
Real(Kind=CP)	Vert	Vertical dimension
Real(Kind=CP)	Horiz	Horizontal dimension
Real(Kind=CP)	AGap	Gap between anodes
Real(Kind=CP)	CGap	Gap between cathodes
Integer	NP_Vert	Number of pixels in vertical direction
Integer	NP_Horizz	Number of pixels in horizontal direction
Integer	IGeom	1: Bissectrice (PSI=0) 2: Bissecting - HiCHI 3: Normal beam 4: Parallel (PSI=90)
Integer	IPsd	1: Flat 2: Vertically Curved detector (used in D19amd)
Real(Kind=CP), Dimension(3)	E1	Components of e1 in {i,j,k}
Real(Kind=CP), Dimension(3)	E2	Components of e2 in {i,j,k}
Real(Kind=CP), Dimension(3)	E3	Components of e3 in {i,j,k}

Integer	Num_Ang	Number of angular motors
Character(Len=12), Dimension(15)	Ang_Names	Name of angular motors
Real(Kind=CP), Dimension(15,2)	Ang_Limits	Angular limits (up to 15 angular motors)
Real(Kind=CP), Dimension(15)	Ang_Offsets	Angular offsets
Integer	Num_Displ	Number of displacement motors
Character(Len=12), Dimension(10)	Disp_Names	Name of displacement motors
Real(Kind=CP), Dimension(10,2)	Disp_Limits	Displacement limits (up to 15 displacement motors)
Real(Kind=CP), Dimension(10)	Disp_Offsets	Displacement offsets
Real(Kind=CP), Dimension(3)	Det_Offsets	Offsets X,Y,Z of the detector centre
Real(Kind=CP), Allocatable, Dimension(:,::)	Alphas	Efficiency corrections for each pixel
End Type Diffractometer_Type		

Generic_Numor_Type

	Variable	Definition
Type :: Generic_Numor_Type		
Integer	Numor	Numor
Character(Len=4)	Instr	instrument on ILL
Character(Len=10)	ExpName	Experimental Name
Character(Len=20)	Date	Date
Character(Len=80)	Title	Title
Type(Basic_NumC_Type)	SampleID	Sample Identification
Type(Basic_NumR_Type)	DiffOpt	Diffractometer Optics and Reactor Parameters
Type(Basic_NumR_Type)	MonMPar	Monochromator Motor Parameters
Type(Basic_NumR_Type)	DiffMPar	Diffractometer Motor Parameters
Type(Basic_NumR_Type)	DetPar	Detector Parameters
Type(Basic_NumI_Type)	DAcFlags	Data Acquisition Control
Type(Basic_NumR_Type)	DAcParam	Data Acquisition Parameters
Type(Basic_NumR_Type)	SampleST	Sample status
Type(Basic_NumI_Type)	ICounts	Counts as Integers
Type(Basic_NumR_Type)	RCounts	Counts as Reals
End Type		
Generic_Numor_Type		

ILL_Data_Record_Type

	Variable	Definition
--	-----------------	-------------------

Type :: ILL_Data_Record_Type		
Integer	Numor	Data set numor
Integer	NSet_Prime	Set number (groups of 100000 numor)
Integer	NTran	(key2) 0 or numcomp => data transferred?
Character(Len=4)	Inst_CH	instrument name
Character(Len=22)	Date_CH	Measurement date
Character(Len=2)	Fill_CH	(key3) leader
Character(Len=6)	User_CH	User name
Character(Len=6)	LC_CH	Local contact name
Character(Len=72)	TextT_CH	Comentary
Character(Len=8)	Scan_Motor	Principal scan motor name
Integer	NVers	Data version number
Integer	NType	Data type - single/multi/powder
Integer	KCTRL	Data function type
Integer	Manip	Principle scan angle
Integer	NBang	Number of data saved
Integer	NKMes	Pre-calculated number of points
Integer	NPDone	Actual number of points
Integer	JCode	Count on monitor/time
Integer	ICalc	Angle calculation type
Integer	IAnal	Analyser present (D10)
Integer	IMode	2th motor sense (D10)
Integer	ITGV	D19/D9 fast measurement
Integer	IRegul	Temperature monitor function
Integer	IVolt	Voltmeter function
Integer	NAxe	D10 (number of axes)
Integer	NPStart	Point starting no frag. numor (D19/16)
Integer	ILasti	Elastic measurement (D10)
Integer	ISA	Analyser motor sense (D10)
Integer	FLGKIF	Constant ki or kf (D10)
Integer	IH_SQS	D10 sqs variation on h
Integer	IK_SQS	D10 sqs variation on k
Integer	NBSQS	D10 sqs slice number
Integer	NB_Cells	Multi/powder data - number of detectors
Integer	NFree1	Data control (free)
Integer, Dimension(7)	ICDESC	
Real(Kind=CP), Dimension(35)	ValCO	RVAL(1:35)
Real(Kind=CP), Dimension(10)	ValDEF	RVAL(36:45)
Real(Kind=CP), Dimension(5)	ValENV	RVAL(46:50)
End Type		

ILL_Data_Record_Type

Definition for Data Record type

Powder_Numor_Type

	Variable	Definition
Type :: Powder_Numor_Type		
Integer	Numor	Numor
Integer	Manip	Principle scan angle
Integer	ICalc	Angle calculation type
Character(Len=32)	Header	User, local contact, date
Character(Len=4)	Instrm	Instrument Name
Character(Len=32)	Title	Title
Character(Len=8)	ScanType	Omega, Phi, etc...
Real(Kind=CP), Dimension(5)	Angles	Angles: phi, chi, omega, 2theta(gamma), psi
Real(Kind=CP), Dimension(3)	Scans	Scan start, scan step, scan width
Real(Kind=CP)	Monitor	
Real(Kind=CP)	Time	
Real(Kind=CP)	Wave	wavelength
Real(Kind=CP), Dimension(5)	Conditions	Temp-s.pt,Temp-Regul,Temp-sample,Voltmeter,Mag.field
Integer	NBData	Total number of pixels nx*ny = np_vert*np_horiz
Integer	NFrames	Total number of frames
Integer	NBang	Total number of angles moved during scan
Integer, Dimension(7)	ICDesc	Integer values
Real(Kind=CP), Dimension(:,:,), Allocatable	TMC_Ang	Time,monitor,total counts, angles*1000: To be allocated as Tmc_ang(nbang,nframes)
Real(Kind=CP), Dimension(:,:,), Allocatable	Counts	Counts array to be reshaped (np_vert,np_horiz,nframes) in case of 2D detectors. To be allocated as Counts (nbdata,nframes)
End Type		
Powder_Numor_Type		

SXTAL_Numor_Type

	Variable	Definition
Type :: SXTAL_Numor_Type		
Integer	Numor	Numor

Integer	Manip	Principle scan angle
Integer	ICalc	Angle calculation type
Character(Len=32)	Header	User, local contact, date
Character(Len=4)	Instrm	Instrument Name
Character(Len=32)	Title	Title
Character(Len=8)	ScanType	Omega, Phi, etc
Real(Kind=CP), Dimension(3)	HMin	h,k,l for Omega scans
Real(Kind=CP), Dimension(3)	HMax	
Real(Kind=CP), Dimension(5)	Angles	Phi, Chi, Omega, 2θ (Gamma), Psi
Real(Kind=CP), Dimension(3,3)	UB	UB-matrix
Real(Kind=CP), Dimension(3)	DH	delta_h, delta_k, delta_l
Real(Kind=CP), Dimension(3)	Scans	scan start, scan step, scan width
Real(Kind=CP)	Preset	
Real(Kind=CP)	Wave	Wavelength
Real(Kind=CP)	CPL_Fact	Coupling Factor
Real(Kind=CP), Dimension(5)	Conditions	Temp-s.pt,Temp-Regul,Temp-sample,Voltmeter,Mag.field
Integer	NBData	Total number of pixels nx*ny = np_vert*np_horiz
Integer	NFrames	Total number of frames
Integer	NBAng	Total number of angles moved during scan
Integer, Dimension(7)	ICDesc	Integer values
Real(Kind=CP), Allocatable, Dimension(:, :)	TMC_Ang	Array (NBANG,NFRAMES) . Time,monitor,total counts, angles*1000
Real(Kind=CP), Allocatable, Dimension(:, :)	Counts	Array ((NBDATA,NFRAMES) . To be reshaped (NP_VERT x NP_HORIZ, NFRAMES) in case of 2D detectors
End Type SXTAL_Numor_Type		

Definition for XTAL Numor type

SXTAL_Orient_Type

	Variable	Definition
Type :: SXTAL_Orient_Type		
Real(Kind=CP)	Wave	Wavelength
Real(Kind=CP), Dimension(3,3)	UB	UB matrix in Busing-Levy setting
Real(Kind=CP), Dimension(3,3)	UBInv	inverse of UB-matrix
Real(Kind=CP), Dimension(3,3)	Conv	Conversion matrix to the local setting
End Type SXTAL_Orient_Type		

Current_Instrm

Type(Diffractometer_Type) :: Current_Instrm

Define a **Current_Instrm** variable according to [Diffractometer_Type](#)

Current_Orient

Type(SXTAL_Orient_Type) :: Current_Orient

Define a **Current_Orient** variable according to [SXTAL_Orient_Type](#)

Cycle_Number

Integer :: Cycle_Number

Value to give the cycle number of Reactor at ILL

Err_ILLData

Logical :: Err_ILLData

This variable is set to **.TRUE.** if an error occurs in procedures belonging to this module.

Err_ILLData_Mess

Character (Len=150) :: Err_ILLData_Mess

This variable contains information about the last error occurred in the procedures belonging to this module

ILL_Data_Directory

Character(Len=512) :: ILL_Data_Directory

String containing information about the path for data directory for ILL

ILL_Temp_Directory

Character(Len=512) :: ILL_Temp_Directory

String containing information about the path for Temporal directory. It is used for uncompress the .Z files

Instrm_Directory

Character(Len=512) :: Instrm_Directory

String containing information about the data directory for specific instrument

Machine_Name

Character(Len=8) :: Machine_Name

String containing information about the instrument name

Year_ILLData

Integer :: Year_ILLData

Variable indicating the year for ILL data

Subroutines

- [Allocate_Numors](#)
- [Define_Uncompress_Program](#)
- [Get_Absolute_Data_Path](#)
- [Get_Next_YearCycle](#)
- [Get_Single_Frame_2D](#)
- [Init_Err_ILLData](#)
- [Initialize_Data_Directory](#)
- [Initialize_Numor](#)
- [PowderNumors_To_DiffPattern](#)
- [Read_Calibration_File](#)
- [Read_Current_Instrm](#)
- [Read_Numor](#)
- [Set_Current_Orient](#)
- [Set_Default_Instrument](#)
- [Set_ILL_Data_Directory](#)
- [Set_Instrm_Directory](#)
- [Update_Current_Instrm_UB](#)
- [Write_Current_Instrm_Data](#)
- [Write_HeaderInfo_Numor](#)
- [Write_Numor_Info](#)

Allocate_Numors

Subroutine Allocate_Numors (Num_Max, NData, Num_Ang, NFrames, Num)

Integer	Intent(in)	Num_Max	Number of components of the array (dimension of Num)
Integer	Intent(in)	NData	Number of pixels of a single frame
Integer	Intent(in)	Num_Ang	Number of angles moved simultaneously during the scan
Integer	Intent(in)	NFrames	Number of frames (number of scan points)
Type(Powder_Numor_Type), Allocatable, Dimension(:) or Type(SXTAL_Numor_Type), Allocatable, Dimension(:)	Intent(in out)	Num	Numor

Subroutine allocating and initializing the array **Num** of type [Powder_Numor_Type](#) or [SXTAL_Numor_Type](#)

Define_Uncompress_Program

Subroutine Define_Uncompress_Program (ProgName)

Character(Len=*)	Intent(in)	ProgName	Name of the uncompress program
-------------------------	-------------------	----------	--------------------------------

Routine that define the ProgName program that you wants to use

Get_Absolute_Data_Path

Subroutine Get_Absolute_Data_Path (Numor, Instrm, Path, IYear, ICycle)

Integer	Intent(in)	Numor	Numor
Character (Len=*)	Intent(in)	Instrm	instrument Name
Character (Len=*)	Intent(out)	Path	Absolute Path
Integer, Optional	Intent(in)	IYear	Year
Integer, Optional	Intent(in)	ICycle	Cycle number

Finds the absolute path to any Numor.

The base directory is set by a call to Initialize_Data_Directory. The procedure then searches for the Numor in the following order:

1. In the subdirectory defined by the year and cycle if passed as arguments to the subroutine (i.e args iyear, icycle).
2. In the subdirectory defined by the year and cycle of the previous call to Get_Absolute_Data_Path (since Numors are likely to be adjacent).
3. In the DATA subdirectory (since likely to process recent data).
4. In the DATA-1 subdirectory (same logic as above)
5. Working from the current year and most recent cycle and working back through cycles and year until the stopping at the first cycle of 1973, when the first data was archived.

Tries to find an uncompress Numor first and then tries to find a compressed Numor (.Z extension). If found the Numor is uncompressed in the a temporary directory if defined (see subroutine Initialize_Data_Directory) or else into the current directory.

Get_Next_YearCycle

Subroutine Get_Next_YearCycle (YearCycle, Reset_To_Most_Recent)

Character (Len=*)	Intent (out)	YearCycle	
Logical, Optional	Intent(in)	Reset_To_Most_Recent	

Works back through the cycles and years, returning the previous YearCycle combination as a 3 character string i.e.

if Year_ILLData = 6 and cycle_number = 5, returns '064'
 if Year_ILLData = 6 and cycle_number = 1, returns '057'

The Reset_To_Most_Recent flag allows the Year_ILLData and Cycle_Number to be set to the most recent possible.

If asked for a YearCycle before '731' (first cycle of 1973) then returns " ", since no data was archived before this date.

Get_Single_Frame_2D

Subroutine Get_Single_Frame_2D (NFr, IOrd, SNum, Dat_2D, Appl_Alphas)

Integer	Intent(in)	NFr	Frame number
Integer	Intent(in)	IOrd	Type of order: 1: D19 Banana

			2: D9/D10 3: D19 Flat 4: D20
Type(SXTAL_Numor_Type)	Intent(in)	SNum	Numor Object
Real(Kind=CP), Dimension(:,:)	Intent(out)	Dat_2D	
Logical, Optional	Intent(in)	Appl_Alph as	Efficiency corrections flag

Extracts into the real two-dimensional array **Dat_2D** the counts of the frame number NFr of the Numor object SNum, applying the efficiency corrections depending of the optional argument Appl_Alphas

Init_Err_ILLData

Subroutine Init_Err_ILLData ()

Subroutine that initializes errors flags in **CFML_ILL_Instrm_Data** module.

Initialize_Numor

Subroutine Initialize_Numor (Numor, NBAng, NBData, NFrames)

Type(Powder_Numor_Type) or Type(SXTAL_Numor_Type)	Intent(in)	Numor	Numor Object
Integer, Optional	Intent(in)	NBAng	Number of Angles moved during measurement
Integer, Optional	Intent(in)	NBData	Number of Points for each Frame
Integer, Optional	Intent(in)	NFrames	Number of Frames

Initialize the Object Numor

Initialize_Data_Directory

Subroutine Initialize_Data_Directory ()

Initialize the Data directory where data are saved at ILL.

PowderNumors_To_DiffPattern

Subroutine PowderNumors_To_DiffPattern (PNumors, N, ActList, Pat, VNorm, Detect, Cal)

Type(Powder_Numor_Type), Dimension(:)	Intent(in)	PNumors	Numor Object
Integer	Intent(in)	N	Pattern Object
Logical, Dimension(:)	Intent(in)	ActList	Active or not a particular Numor from the list
Type(Diffraction_Pattern_Type)	Intent(out)	Pat	Pattern Object
Real, optional	Intent(in)	VNorm	Normalization value (Monitor)
Integer, optional	Intent(in)	Detect	Select a particular Detector
Type(Calibration_Detector_Type), optional	Intent(in)	Cal	Calibration object for Detectors in a particular Instrument

Translate The information from Numors (adding in general) to a Diffraction Object

Read_Calibration_File

Subroutine Read_Calibration_File (FileCal, Instrm, Cal)

Character (Len=*)	Intent(in)	FileCal	Path+Filename of the Calibration file
Character (Len=*)	Intent(in)	Instrm	Instrument name
Type(Calibration_Detector_Type)	Intent(out)	Cal	Calibration object

Procedure for read a calibration file for different instruments at ILL

Actually, the routine support D1A and D2B instrument.

Read_Current_Instrm

Subroutine Read_Current_Instrm (Filenam)

Character (Len=*)	Intent(in)	Filenam	String
--------------------------	-------------------	---------	--------

Subroutine reading the file Filenam where the characteristics of the current instrument are written. The global Current_Instrm variable is filled after returning from this subroutine.

Read_Numor

Subroutine Read_Numor (Fileinfo, Instrument, N)

Character (Len=*)	Intent(in)	Fileinfo	Filename
Character (Len=*)	Intent(in)	Instrument	Instrument name
Type(Powder_Numor_Type) or Type(SXTAL_Numor_Type)	Intent(out)	N	Powder/Single-Crystal Numor

Subroutine to read a Numor of any instrument at ILL. Actually the Instrument name implemented are:

Powder Instruments:

D1A, D1B, D2B, D4, D20

Single-Crystal Instruments:

D9, D10, D19

Set_Current_Orient

Subroutine Set_Current_Orient (Wave, UB, Setting)

Real(Kind=CP)	Intent(in)	Wave	Wavelength
Real(Kind=CP), Dimension(3,3)	Intent(in)	UB	UB Matrix
Real(Kind=CP), Dimension(3,3), Optional	Intent(in)	Setting	Object

Subroutine setting the [Current_Orient](#) global variable.

If the final UB matrix is singular an error is raised by putting Err_ILLData=.true. and filling the error message

variable Err_ILLData_Mess.

Set_Default_Instrument

Subroutine Set_Default_Instrument ()

Construct the [Current_Instrm](#) as a default 4C diffractometer. The UB matrix is set to a real matrix corresponding to a measurement done on D9.

The characteristics of the diffractometer correspond to those of D9.

Set_ILL_Data_Directory

Subroutine Set_ILL_Data_Directory (FileDir)

Character (Len=*)	Intent(in)	FileDir	Proposed location of ILL data
--------------------------	-------------------	---------	-------------------------------

Assign a new directory to the global public variable [ILL_Data_Directory](#). If the directory doesn't exist the subroutine rises an error condition by putting Err_ILLData=.TRUE. and filling the error message variable Err_ILLData_Mess. If **FileDir** is blank then data are in the current directory.

Set_Instrm_Directory

Subroutine Set_Instrm_Directory (Working_Dir, Instrm, IYear, ICycle)

Character (Len=*) , Optional	Intent(in)	Working_Dir	Directory for Numors
Character (Len=*) , Optional	Intent(in)	Instrm	Name of the instrument
Character (Len=*) , Optional	Intent(in)	IYear	Year of the Numor
Character (Len=*) , Optional	Intent(in)	ICycle	Cycle of the Numor

Assign the global public variable: [Instrm_Directory](#).

If Working_Dir was defined, then it is the directory defined for Instrm_Directory. If not, then **Instrm** is used. When the routine is called using **Instrm** argument, we have two options:

- Using IYear and ICycle arguments: Instrm_Directory= ILL_Data_Directory/YYC/**Instrm**/
- Only with Instrm argument: Instrm_Directory= ILL_Data_Directory/data/**Instrm**/

It is assumed that the subroutine [Set_ILL_Data_Directory](#) has already been called.

Update_Current_Instrm_UB

Subroutine Update_Current_Instrm_UB (Filenam, UB, Wave)

Character (Len=*)	Intent(in)	Filenam	Filename
Read(Kind=CP), Dimension(3,3)	Intent(in)	UB	UB Matrix
Read(Kind=CP)	Intent(in)	Wave	Wavelength

Subroutine updating the file **Filenam** where the characteristics of the current instrument are written. The global Current_Instrm variable is re-filled with

new values of wavelength and UB-matrix. The file **Filenam** is re-written and the old version is saved with appended extension '.bak'. The Current_Orient global variable is also updated.

[Write_Current_Instrm_Data](#)**Subroutine Write_Current_Instrm_Data (Lun)**

Integer, Optional	Intent(in)	Lun	Unit to write
-----------------------------------	----------------------------	-----	---------------

Writes the characteristics of the Current_Instrm instrument in the file of logical unit Lun. If the subroutine is invoked without argument the subroutine outputs the information on the standard output (screen)

[Write_HeaderInfo_Numor](#)**Subroutine Write_HeaderInfo_Numor (Num, Lun)**

Type(Powder_Numor_Type) or Type(SXTAL_Numor_Type)	Intent(in)	Num	Numor object
Integer, Optional	Intent(in)	Lun	Unit to write

Writes the Header information of the numor objet **Num** of type [Powder_Numor_Type](#) or [SXTAL_Numor_Type](#) in the file of logical unit **Lun**.

If the subroutine is invoked without the **Lun** argument the subroutine outputs the information on the standard output (screen)

[Write_Numor_Info](#)**Subroutine Write_Powder_Numor (Num, Lun)**

Type(Powder_Numor_Type) or Type(SXTAL_Numor_Type)	Intent(in)	Num	Numor object
Integer, Optional	Intent(in)	Lun	Unit to write

Writes the characteristics of the numor objet **Num** of type [Powder_Numor_Type](#) or [SXTAL_Numor_Type](#) in the file of logical unit **Lun**.

If the subroutine is invoked without the **Lun** argument the subroutine outputs the information on the standard output (screen)

Level 4

Concept	Module Name	Purpose
Atoms...	CFML_Atom_TypeDef	Module defining different data structures concerned with atoms
Geometry...	CFML_Geometric_SXTAL	Module for geometrical calculations in single crystal instruments
Reflections...	CFML_Reflections_Utils	Procedures handling operation with Bragg reflections

CFML_Atom_TypeDef

Module for Atom Types definitions

Variables

- [Atom_Type](#)
- [Atom_List_Type](#)
- [Atoms_Cell_Type](#)
- [MAtom_Type](#)
- [MAtom_List_Type](#)

- [Err_ATMD](#)
- [Err_ATMD_Mess](#)

Functions

- [Equiv_ATOM](#)
- [WRT_Lab](#)

Subroutines

- [Allocate_Atom_List](#)
- [Allocate_Atoms_Cell](#)
- [Allocate_MAtom_List](#)
- [ATList1_ExtenCell_ATList2](#)
- [Atom_List_To_Cell](#)
- [Atom_Uequi_List](#)
- [Atoms_Cell_To_List](#)
- [Copy_Atom_List](#)
- [Deallocate_Atom_List](#)
- [Deallocate_Atoms_Cell](#)
- [Deallocate_MAtom_List](#)
- [Init_Atom_Type](#)
- [Init_Err_ATMD](#)
- [Init_MAtom_Type](#)
- [Merge_Atoms_Peaks](#)
- [Multi](#)
- [Write_Atom_List](#)
- [Write_Atoms_CFL](#)
- [Write_CFL](#)

Fortran Filename

CFML_Atom_Mod.f90

Variables

- [Atom_Type](#)
- [Atom_List_Type](#)
- [Atoms_Cell_Type](#)
- [MAtom_Type](#)
- [MAtom_List_Type](#)

- [Err_ATMD](#)
- [Err_ATMD_Mess](#)

Atom_Type

	Variable	Definition
Type :: Atom_Type		
Character (Len=10)	Lab	Label
Character (Len=2)	ChemSymb	Chemical symbol
Character (Len=4)	SFacSymb	Chemical symbol for SF
Logical	Active	Control flag
Integer	Z	Atomic number
Integer	Mult	Multiplicity site
Real (Kind=CP), Dimension(3)	X	Fractional coordinates
Real (Kind=CP), Dimension(3)	X_STD	Standard deviations of X
Real (Kind=CP), Dimension(3)	MX	Multipliers for coordinates (applied to shifts in non-linear LSQ)
Integer, Dimension(3)	LX	Ordinal numbers of LSQ parameters for atomic position
Real (Kind=CP)	Occ	Occupation factor
Real (Kind=CP)	Occ_STD	Standard deviation of OCC
Real (Kind=CP)	MOcc	Multiplier
Integer	LOcc	Ordinal number of LSQ parameter for OCC
Real (Kind=CP)	Biso	Isotropic B-Factor
Real (Kind=CP)	Biso_STD	Standard deviation of BISO
Real (Kind=CP)	MBiso	Multiplier
Integer	LBiso	Ordinal number of LSQ parameter for BISO
Character (Len=4)	UType	Values are: u_ij -> U's b_ij -> B's beta -> β's none
Character (Len=5)	THType	Values are: Isotr -> Isotropic

		Aniso -> Anisotropic Other
Real (Kind=CP), Dimension(6)	U	Coeff U11,U22,U33,U12,U13,U23
Real (Kind=CP), Dimension(6)	U_STD	Standard deviations of U's
Real (Kind=CP)	Ueq	U equivalent
Real (Kind=CP), Dimension(6)	MU	Multipliers
Integer, Dimension(6)	LU	Ordinal numbers of LSQ parameters
Real (Kind=CP)	Charge	
Real (Kind=CP)	Moment	
Integer, Dimension(5)	Ind	index vector for different purposes
Integer	NVar	Number of additional free variables (used for different purposes)
Real (Kind=CP), Dimension(10)	VarF	Free variables
End Type Atom_Type		

Atom_List_Type

	Variable	Definition
Type :: Atom_List_Type		
Integer	NAtoms	Number of Atoms in the List
Type (Atom_Type), Dimension(:,), Allocatable	Atom	Atoms information
End Type Atom_List_Type		

Atoms_Cell_Type

	Variable	Definition
Type :: Atoms_Cell_Type		
Integer	Nat	Number of Atoms
Character (Len=10), Dimension(:,), Allocatable	Noms	Name of Atoms
Real (Kind=CP), Dimension(:, :,), Allocatable	XYZ	Fractional coordinates (3, NAT)
Real (Kind=CP), Dimension(:,), Allocatable	Charge	
Real (Kind=CP), Dimension(:,), Allocatable	Moment	
Real (Kind=CP), Dimension(:, :,), Allocatable	VarF	Free Parameters (10, NAT)
Integer, Dimension(:,), Allocatable	Neighb	Number of neighbours (NAT)

Integer, Dimension(:, :, :), Allocatable	Neigh_AtM	Ptr.->neighbour (# in list)(NAT, IDP)
Real (Kind=CP), Dimension(:, :, :), Allocatable	Distance	Distances (NAT, IDP)
Real (Kind=CP), Dimension(:, :, :, :), Allocatable	Trans	Lattice translations (3, NAT, IDP)
Integer	NDist	Number of distinct distances
Real (Kind=CP), Dimension(:, :, :), Allocatable	DDist	List of distinct distances(NAT*IDP)
Character (Len=8), Dimension(:, :, :), Allocatable	DDLAb	Labels of atoms at ddist (NAT*IDP)
End Type Atoms_Cell_Type		

MAtom_Type

	Variable	Definition
Type :: MAtom_Type		
Character (Len=10)	Lab	Label
Character (Len=2)	ChemSymb	Chemical symbol
Character (Len=4)	SFacSymb	Chemical symbol for SF
Logical	Active	Control flag
Integer	Z	Atomic number
Integer	Mult	Multiplicity site
Real (Kind=CP), Dimension(3)	X	Fractional coordinates
Real (Kind=CP), Dimension(3)	X_STD	Standard deviations of X
Real (Kind=CP), Dimension(3)	MX	Multiplier parameters of coordinates
Integer, Dimension(3)	LX	Numbers of LSQ parameters for X
Real (Kind=CP)	Occ	Occupation factor
Real (Kind=CP)	Occ_STD	Standard deviation of OCC
Real (Kind=CP)	MOcc	
Integer	LOcc	
Real (Kind=CP)	Biso	Isotropic B-Factor
Real (Kind=CP)	Biso_STD	Standard deviation of BISO
Real (Kind=CP)	MBiso	
Integer	LBiso	
Character (Len=4)	UType	Values are: u_ij -> U's b_ij -> B's beta -> β's none
Character (Len=5)	THType	Values are: Isot -> Isotropic Aniso -> Anisotropic Other
Real (Kind=CP), Dimension(6)	U	Coeff U11, U22, U33, U12, U13, U23

Real (Kind=CP), Dimension(6)	U_STD	
Real (Kind=CP)	Ueq	U equivalent
Real (Kind=CP), Dimension(6)	MU	
Integer, Dimension(6)	LU	
Real (Kind=CP)	Charge	
Real (Kind=CP)	Moment	
Integer, Dimension(5)	Ind	index vector for different purposes
Integer	NVar	Number of Free parameters
Real (Kind=CP), Dimension(10)	VarF	Free parameters
Integer	NVK	Num. of propag. vectors (excl. -k)
Integer, Dimension(12)	IMat	Num. of the magnetic matrices/irrep set to be applied
Real (Kind=CP), Dimension(3,12)	SKR	Real part of Fourier Coefficient
Real (Kind=CP), Dimension(3,12)	MSKR	Multipliers for the real part of Fourier coefficients
Integer, Dimension(3,12)	LSKR	Numbers in the list of LSQ parameters
Real (Kind=CP), Dimension(3,12)	SKI	Imaginary part of Fourier Coefficient
Real (Kind=CP), Dimension(3,12)	KSKI	Multipliers for the imaginary part of Fourier coefficients
Integer, Dimension(3,12)	LSKI	Numbers in the list of LSQ parameters
Real (Kind=CP), Dimension(12)	MPhas	Magnetic Phase in fractions of 2π
Real (Kind=CP), Dimension(12)	MMPhas	Multiplier for the magnetic phase
Integer, Dimension(12)	LMPhas	Numbers in the list of LSQ parameters
Real (Kind=CP), Dimension(12,12)	CBas	Coeff. of the basis functions of IRreps, the second index is 1:nv
Real (Kind=CP), Dimension(12,12)	MBas	Multiplier for the coeff. of the basis functions of IRreps
Integer, Dimension(12,12)	LBas	Numbers in the list of LSQ parameters
End Type MAtom_Type		

MAtom_List_Type

	Variable	Definition
Type :: MAtom_List_Type		
Integer	NAtoms	Number of Atoms in the List
Type (MAtom_Type), Dimension(:), Allocatable	Atom	Atoms information
End Type MAtom_List_Type		

Err_ATMD**Logical :: Err_ATMD**

This variable is set to .TRUE. if an error occurs in procedures belonging to this module.

Err_ATMD_Mess

Character (Len=150) :: Err_ATMD_Mess

This variable contains information about the last error occurred in the procedures belonging to this module.

Functions

- [Equiv_ATM](#)
- [WRT_Lab](#)

Equiv_ATM

WRT_Lab

Character Function WRT_Lab (Nam1, Nam2)

Character (Len=*)	Intent(in)	Nam1	Atom name
Character (Len=*)	Intent(in)	Nam2	Atom name

Return a string of length 8 where merging the main part of the labels (before underscore "_") of the atoms Nam1 and Nam2.

Subroutines

- [Allocate_Atom_List](#)
- [Allocate_Atoms_Cell](#)
- [Allocate_MAtom_List](#)
- [ATList1_ExtenCell_ATList2](#)
- [Atom_List_To_Cell](#)
- [Atom_Uequi_List](#)
- [Atoms_Cell_To_List](#)
- [Copy_Atom_List](#)
- [Deallocate_Atom_List](#)
- [Deallocate_Atoms_Cell](#)
- [Deallocate_MAtom_List](#)
- [Init_Atom_Type](#)
- [Init_Err_ATMD](#)
- [Init_MAtom_Type](#)
- [Merge_Atoms_Peaks](#)
- [Multi](#)
- [Write_Atom_List](#)
- [Write_Atoms_CFL](#)
- [Write_CFL](#)

Allocate_Atom_List

Subroutine Allocate_Atom_List (N, A, Fail)

Integer	Intent(in)	N	Number of elements of A
Type(Atom_List_Type)	Intent(in out)	A	Objet to be allocated
Logical, Optional	Intent(out)	Fail	String containing atom names

Allocation of objet A of type [Atom_List_Type](#).

This subroutine should be called before using an object of type [Atom_List_Type](#).

Allocate_Atoms_Cell

Subroutine Allocate_Atoms_Cell (NAsu, Mul, DMax, Ac)

Integer	Intent(in)	NAsu	Number of atoms in asymmetric unit
Integer	Intent(in)	Mul	General multiplicity of the Space Group
Real(Kind=CP)	Intent(in)	DMax	Maximun distance to be calculated
Type(Atoms_Cell_Type)	Intent(in out)	Ac	Object

Allocation of objet AC of type [Atoms_Cell_Type](#).

Ac contains components with allocatable attribute with dimension depending on the input arguments NAsu, Mul and DMax. The variables used for calculating the dimensions are:

$$\text{NATCEL} = \text{NASU} * \text{MUL}$$

$$\text{IDP} = \text{NinT}(0.74048 * (\text{DMAX}/1.1)^3)$$

Note: This subroutine should be called before using the subroutines of this module with dummy arguments of type [Atoms_Cell_Type](#).

Allocate_MAtom_List

Subroutine Allocate_MAtom_List (N, A)

Integer	Intent(in)	N	Number of elements of A
Type(MAtom_List_Type)	Intent(in out)	A	Objet to be allocated

Allocation of objet A of type [MAtom_List_Type](#)

Note: This subroutine should be called before using an object of type [MAtom_List_Type](#).

ATList1_ExtenCell_ATList2

Subroutine ATList1_ExtenCell_ATList2 (Spg, A, B, Conven)

Type(Space_Group_Type)	Intent(in)	Spg	Space Group information
Type(Atom_List_Type)	Intent(in)	A	Atom List (asymmetric unit)
Type(Atom_List_Type)	Intent(out)	B	Atom List in unit cell
Logical	Intent(in)	Conven	.TRUE. for using the whole conventional unit cell

Subroutine to generate atoms in the primitive (Conven=.FALSE.) or the conventional unit cell (Conven=.TRUE.)

Atom_List_To_Cell

Subroutine Atom_List_To_Cell (A, AC)

Type(Atom_List_Type)	Intent(in)	A	Atom List
Type(Atoms_Cell_Type)	Intent(in out)	Ac	Atoms in CELL

Subroutine to construct an [Atoms_Cell_Type](#) object **Ac** from an [Atom_List_Type](#) object **A**.

It is supposed that both objects have been previously allocated using the appropriate procedures.

Atom_Uequi_List

Subroutine Atom_Uequi_List (Cell, A)

Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell Variable
Type(Atom_List_Type)	Intent(in out)	A	Atom list

Subroutine to obtain the U_{eq} from $U_{11} U_{22} U_{33} U_{12} U_{13} U_{23}$ for **A** object

Atoms_Cell_To_List

Subroutine Atoms_Cell_To_List (Ac, A)

Type(Atoms_Cell_Type)	Intent(in)	Ac	Atoms in CELL
Type(Atom_List_Type)	Intent(in out)	A	Atom List

Subroutine to construct an Atom List object **A** from an [Atoms_Cell_Type](#) object **Ac**.

Note: It is supposed that both objects have been previously allocated using the appropriate procedures: direct allocation for **A** and call to [Allocate_Atoms_Cell](#) for **Ac**.

Copy_Atom_List

Subroutine Copy_Atom_List (A, Ac)

Type(Atom_List_Type)	Intent(in)	A	Atom List
Type(Atom_List_Type)	Intent(out)	Ac	Atom List

Subroutine to copy an atom list to another one

Deallocate_Atom_List

Subroutine Deallocate_Atom_List (A)

Type(Atom_List_Type)	Intent(in out)	A	Objet to be allocated
----------------------	-------------------	---	-----------------------

Deallocation of objet **A** of type [Atom_List_Type](#).

Note: This subroutine should be after using an object of type [Atom_List_Type](#) that is no more needed.

[Desallocate_Atoms_Cell](#)

Subroutine **Deallocate_Atoms_Cell (Ac)**

Type(Atoms_Cell_Type)	Intent(in out)	Ac	Objet to be allocated
-----------------------	-------------------	----	-----------------------

Deallocation of objet AC of type [Atoms_Cell_Type](#).

[Deallocate_MAtom_List](#)

Subroutine **Deallocate_MAtom_List (A)**

Type(MAtom_List_Type)	Intent(in out)	A	Objet to be allocated
-----------------------	-------------------	---	-----------------------

Deallocation of objet A of type [MAtom_List_Type](#).

Note: This subroutine should be invoked after using an object of type [MAtom_List_Type](#) that is no more needed.

[Init_Atom_Type](#)

Subroutine **Init_Atom_Type (A)**

Type(Atom_Type)	Intent(in out)	A	Atom Type
-----------------	-------------------	---	-----------

initialize the variable A which is the type [Atom_Type](#)

[Init_Err_ATMD](#)

Subroutine **Init_Err_ATMD ()**

Subroutine that initializes errors flags in **CFML_Atom_TypeDef** module.

[Init_MAtom_Type](#)

Subroutine **Init_MAtom_Type (A)**

Type(MAtom_Type)	Intent(in out)	A	Atom Type
------------------	-------------------	---	-----------

Initialize the variable A which is the type [MAtom_Type](#)

[Merge_Atoms_Peaks](#)

Subroutine Merge_Atoms_Peaks (Cell, Atm, NPks, Pks, Grp, NAtm)

Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters
Type(Atom_List_Type)	Intent(in)	Atm	Atom list
Integer	Intent(in)	NPks	Number of Peaks on PKS
Real(Kind=CP), Dimension(:, :)	Intent(in)	Pks	List of Peaks
Type(Space_Group_Type)	Intent(in)	Grp	Space Group information
Type(Atom_List_Type)	Intent(out)	NAtm	New Atoms+Peaks List

This routine merge atoms and peaks on a new List.

Multi

Subroutine Multi (Lun, IPrin, Conven, Spg, A, Ac)

Integer	Intent(in)	Lun	Logical Unit for writing
Logical	Intent(in)	IPrin	.true. for writing in Lun
Logical	Intent(in)	Conven	.true. for using the whole conventional unit cell
Type(Space_Group_Type)	Intent(in)	Spg	Space Group information
Type(Atom_List_Type)	Intent(in, out)	A	Atom List
Type(Atoms_Cell_Type)	Intent(out)	Ac	Atoms in unit cell

Subroutine to obtain multiplicities and coordinates of all atoms in the conventional unit cell.

Calculates A%AT(k)%Mult and constructs, partially, the object Ac of type Atoms_Cell_Type. The generated atoms constitute the content of the primitive (Conven=.FALSE.) or the conventional unit cell (Conven=.TRUE.).

[Write_Atom_List](#)

Subroutine Write_Atom_List (Ats, Level, Lun, Mult, Cell)

Type(Atom_List_Type), Dimension(:)	Intent(in)	Ats	Atom list vector
Integer, Optional	Intent(in)	Level	Level of printed information
Integer, Optional	Intent(in)	Lun	Unit to write
Integer, Optional	Intent(in)	Mult	Multiplicity of the general position
Type(Crystal_Cell_Type), Optional	Intent(in)	Cell	Transform to thermal parameters

Write the atoms in the asymmetric unit

[Write_Atoms_CFL](#)

Subroutine Write_Atoms_CFL (Ats, Lun, Cell)

Type(Atom_List_Type), Dimension(:)	Intent(in)	Ats	Atom list vector
------------------------------------	------------	-----	------------------

Integer, Optional	Intent(in)	Lun	Unit to write
Type(Crystal_Cell_Type), Optional	Intent(in)	Cell	Transform to thermal parameters

Write the atoms in the asymmetric unit for a CFL file

[Write_CFL](#)

Subroutine Write_CFL (Lun, Cell, Spg, Atm)

Integer	Intent(in)	Lun	Unit to write
Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters
Type(Space_Group_Type)	Intent(in)	Spg	Space group information
Type(Atom_List_Type)	Intent(in)	Atm	Atom List

Write a CFL file

[CFML_EoS](#)

Routines for EoS (Equations of State) Calculations

Parameters

- [NCol_Data_Max](#)
- [N_EoSPar](#)

Variables

- [EoS_Data_Type](#)
- [EoS_Data_List_Type](#)
- [EoS_Type](#)
- [EoS_List_Type](#)
- [Err_EoS](#)
- [Err_EoS_Mess](#)

Functions

- [Alpha_Cal](#)
- [dKdT_Cal](#)
- [Get_Pressure](#)
- [Get_Pressure_Esd](#)
- [Get_Temperature](#)
- [Get_Volume](#)
- [Get_Volume_S](#)
- [K_Cal](#)
- [Kp_Cal](#)
- [Kpp_Cal](#)
- [Pressure_F](#)
- [Strain](#)

- [Strain_EoS](#)

Subroutines

- [Allocate_EoS_Data_List](#)
- [Allocate_EoS_List](#)
- [Deallocate_EoS_Data_List](#)
- [Deallocate_EoS_List](#)
- [Deriv_Partial_P](#)
- [EoS_Cal](#)
- [EoS_Cal_Esd](#)
- [FfCal_Dat](#)
- [FfCal_Dat_Esd](#)
- [FFcal_EoS](#)
- [Init_EoS_Data_Type](#)
- [Init_EoS_Thermal](#)
- [Init_EoS_Type](#)
- [Init_EoS](#)
- [Read_EoS_DataFile](#)
- [Set_EoS_Names](#)
- [Set_EoS_Use](#)
- [Set_Kp_Kpp_Cond](#)
- [Write_Info_EoS](#)

Fortran Filename

CFML_EoS_Mod.f90

Parameters

- [NCol_Data_Max](#)
- [N_EoSPar](#)

NCol_Data_Max

Integer, Parameter :: NCol_Data_Max=22

Defines the maximum number of columns allowed in input data files

N_EoSPar

Integer, Parameter :: N_EoSPar=13

Specifies the maximum number of Eos parameters allowed in Eos_type data type

Variables

- [EoS Data Type](#)
- [EoS Data List Type](#)
- [EoS Type](#)
- [EoS List Type](#)
- [Err EoS](#)
- [Err EoS Mess](#)

EoS_Data_Type

	Variable	Definition
Type :: EoS_Data_Type		
Integer	IUse	0: No active, 1: active
Integer	IGrp	Group
Real(Kind=CP)	T	Temperature
Real(Kind=CP)	P	Pressure
Real(Kind=CP)	V	Volume
Real(Kind=CP), Dimension(3)	Cell	a,b,c parameters
Real(Kind=CP), Dimension(3)	Ang	alpha, beta, gamma parameters
Real(Kind=CP)	SigT	Sigma Temperature
Real(Kind=CP)	SigP	Sigma Pressure
Real(Kind=CP)	SigV	Sigma Volume
Real(Kind=CP), Dimension(3)	SigC	Sigma for a, b and c parameters
Real(Kind=CP), Dimension(3)	SigA	Sigma for alpha, beta and gamma angles
End Type EoS_Data_Type		

This variable holds all of the crystal unit-cell data for one pressure, temperature data point.

EoS_Data_List_Type

	Variable	Definition
Type :: EoS_Data_List_Type		
Character (Len=80)	Title	Title of dataset
Character (Len=40)	System	Crystal System
Integer	N	Number of EoS Data List
Integer, Dimension (Ncol_Data_Max)	IC_DAT	Which values are input:1 if present The index is: 4 for T 6 for P 8 for V 10,12,14 for a,b,c 16,18,20 for angles above indices +1 for the corresponding esd's
Type (EoS_Data_Type), Dimension (:), allocatable	EoSD	EoS Data Parameters

End Type**EoS_Data_List_Type**

A list of Eos_Data_Type : a list of data points

EoS_Type

	Variable	Definition
Type :: EoS_Type		
Character (Len=15)	Model	Name of Eos model: None, Murnaghan, Birch-Murnaghan, Vinet, Natural-Strain
Character (Len=15)	TModel	Name for thermal model
Character (Len=5), Dimension (n_eospar)	ParName	Names of the Eos variables
Character (Len=50), Dimension (n_eospar)	Comment	Description of the Eos variables including units
Integer	IModel	Index for Model 0= None 1=Murnaghan 2=Birch-Murnaghan 3=Vinet 4=Natural Strain 5=Tait
Integer	IOrder	Order for the Model
Logical	Linear	.True. if linear EoS not volume
Integer	ITherm	Index for thermal expansion model, 0 = none 1 = Berman 2 = Fei 3 = modified HP 1998 4 = Kroll 5 = Salje lowT 6 = Holland-Powell thermal pressure
Integer, Dimension(n_eospar)	IUse	Flags for parameters allowed for a given EoS
Real(Kind=CP)	PRef	Pressure of Reference
Real(Kind=CP)	TRef	Temperature of Reference
Logical	TRef_fixed	If .True., then Tref cannot be changed by user
Real(Kind=CP), Dimension (n_eospar)	Params	Eos Parameters
Real(Kind=CP), Dimension (n_eospar)	Esd	Sigma Eos Parameters
Real(Kind=CP)	X	Spare intensive variable, after P,T
Real(Kind=CP)	WChi2	Weighted chi-squared for the refined parameters
Real(Kind=CP)	DelPMax	Maximum misfit in P between obs and calc P
Integer, Dimension(4)	IWt	Choice for weighting in LS: order is P,T,V,X, 1 = use sigmas for weights

Integer, Dimension(n_eospar)	IRef	Refinement switches
Real(Kind=CP), Dimension (n_eospar)	Factor	Scale factor for variables
Real(Kind=CP), Dimension (n_eospar)	Lastshift	Shift applied in last LS cycle to parameters
Real(Kind=CP), Dimension (n_eospar, n_eospar)	VCV	Var-Covar matrix from refinement
End Type EoS_Type		

Variables of Eos_type hold all of the parameters and settings for one EoS

EoS_List_Type

	Variable	Definition
Type :: EoS_List_Type		
Integer	N	Number of EoS List
Integer, Dimension (Ncol_Data_Max)	IC_DAT	Which values are input
Type (EoS_Type), Dimension (:), allocatable	EoS	EoS Parameters
End Type EoS_List_Type		

A list of eos_type

Err_EoS

Logical :: Err_EoS

This variable is set to .TRUE. if an error occurs in procedures belonging to this module.

Err_EoS_Mess

Character (Len=150) :: Err_EoS_Mess

This variable contains information about the last error occurred in the procedures belonging to this module

Functions

- [Alpha_Cal](#)
- [dKdT_Cal](#)
- [Get_Pressure](#)
- [Get_Pressure_Esd](#)
- [Get_Temperature](#)
- [Get_Volume](#)
- [Get_Volume_S](#)
- [K_Cal](#)
- [Kp_Cal](#)
- [Kpp_Cal](#)
- [Pressure_F](#)

- [Strain](#)
- [Strain_EoS](#)

[Alpha_Cal](#)

Real Function Alpha_Cal (P, T, EoSPar, DeltaT)

Real (Kind=CP)	Intent(in)	P	Pressure
Real (Kind=CP)	Intent(in)	T	Temperature
Type(EoS_Type)	Intent(in)	EoSPar	EoS Variable
Real (Kind=CP), Optional	Intent(in)	DeltaT	Delta T value

Calculates the thermal expansion at P and T from the Eos parameters. For volume, it is $1/V \cdot dV/dT$. For linear case alpha is correct as $1/a \cdot da/dT$

[dKdT_Cal](#)

Real Function dKdT_Cal (P, T, EoSPar, DeltaT)

Real (Kind=CP)	Intent(in)	P	Pressure
Real (Kind=CP)	Intent(in)	T	Temperature
Type(EoS_Type)	Intent(in)	EoSPar	EoS Variable
Real (Kind=CP), Optional	Intent(in)	DeltaT	Delta T value

Calculate the derivative dK/dt (or dM/dt in the linear case) at P and T

[Get_Pressure](#)

Real Function Get_Pressure (V, T, EoSPar)

Real (Kind=CP)	Intent(in)	V	Volume
Real (Kind=CP)	Intent(in)	T	Temperature
Type(EoS_Type)	Intent(in)	EoSPar	EoS Variable

Returns value of pressure at (V,T) for EoS Model defined in EoSPar variable. If any error occurs, then the final value is set to 0.0

[Get_Pressure_Esd](#)

Real Function Get_Pressure_Esd (V, T, EoSPar)

Real (Kind=CP)	Intent(in)	V	Volume
Real (Kind=CP)	Intent(in)	T	Temperature
Type(EoS_Type)	Intent(in)	EoSPar	EoS Variable

Returns the uncertainty in P at a given (V,T) point from the uncertainties and covariances in the eos parameters (in vcv matrix)

[Get_Temperature](#)

Real Function Get_Temperature (Vt, EoSPar)

Real (Kind=CP)	Intent(in)	Vt	Volume at temperature T
Type(EoS_Type)	Intent(in)	EoSPar	EoS Variable

Returns Temperature at P=0 for given V(T)

[Get_Volume](#)

Real Function Get_Volume (P, T, EoSPar)

Real (Kind=CP)	Intent(in)	P	Pressure
Real (Kind=CP)	Intent(in)	T	Temperature
Type(EoS_Type)	Intent(in)	EoSPar	EoS Variable

Find volume from EoS at given P and T

[Get_Volume_S](#)

Real Function Get_Volume (S, T, EoSPar)

Real (Kind=CP)	Intent(in)	S	Strain
Real (Kind=CP)	Intent(in)	T	Temperature
Type(EoS_Type)	Intent(in)	EoSPar	EoS Variable

Returns the value of Volume corresponding to the input finite strain (S). Only uses V0 and the type of EoS model from EoSPar.

[K_Cal](#)

Real Function K_Cal (V, T, EoSPar)

Real (Kind=CP)	Intent(in)	V	Volume
Real (Kind=CP)	Intent(in)	T	Temperature
Type(EoS_Type)	Intent(in)	EoSPar	EoS Variable

Returns value of K at this volume and temperature for the EoS

[Kp_Cal](#)

Real Function Kp_Cal (V, T, EoSPar)

Real (Kind=CP)	Intent(in)	V	Volume
Real (Kind=CP)	Intent(in)	T	Temperature
Type(EoS_Type)	Intent(in)	EoSPar	EoS Variable

Returns value of K' at this volume and temperature for the EoS

[Kpp_Cal](#)

Real Function Kpp_Cal (V, T, EoSPar)

Real (Kind=CP)	Intent(in)	V	Volume
Real (Kind=CP)	Intent(in)	T	Temperature
Type(EoS_Type)	Intent(in)	EoSPar	EoS Variable

Returns value of K'' at this volume and temperature for the EoS

[Pressure_F](#)

Real Function Pressure_F (F, S, EoSPar)

Real (Kind=CP)	Intent(in)	F	Normalized pressure
Real (Kind=CP)	Intent(in)	S	Strain
Type(EoS_Type)	Intent(in)	EoSPar	EoS Variable

Returns the value of Pressure (P) from the input Normalized Pressure(F) and Strain (S) according to the EoS model in EoSPar.

Strain

Real Function Strain (VV0, EoSPar)

Real (Kind=CP)	Intent(in)	VV0	Volume divided by V0 at this temperature
Type(EoS_Type)	Intent(in)	EoSPar	EoS Variable

Returns the value of Strain (S) at this V according with the EoS model

Note:

The values of EoSPar are NOT used. Only the type of Eos function is needed!

For linear eos, the strain is that of a^3

If $V/V0 = 0$. then an error is set

Strain_EoS

Real Function Strain_EoS (V, T, EoSPar)

Real (Kind=CP)	Intent(in)	V	Volume
Real (Kind=CP)	Intent(in)	T	Temperature
Type(EoS_Type)	Intent(in)	EoSPar	EoS Variable

Returns the value of Strain (S) at this V and T according with the EoS parameters

Subroutines

- [Allocate_EoS_Data_List](#)
- [Allocate_EoS_List](#)
- [Deallocate_EoS_Data_List](#)
- [Deallocate_EoS_List](#)
- [Deriv_Partial_P](#)
- [EoS_Cal](#)
- [EoS_Cal_Esd](#)
- [FfCal_Dat](#)
- [FfCal_Dat_Esd](#)
- [FFcal_EoS](#)
- [Init_EoS_Data_Type](#)
- [Init_EoS_Thermal](#)
- [Init_EoS_Type](#)
- [Init_Er_EoS](#)
- [Read_EoS_DataFile](#)
- [Set_EoS_Names](#)
- [Set_EoS_Use](#)

- [Set_Kp_Kpp_Cond](#)
- [Write_Info_EoS](#)

[Allocate_EoS_Data_list](#)

Subroutine Allocate_EoS_Data_List (N, E)

Integer	Intent(in)	N	Number of elements of E
Type(EoS_Data_List_Type)	Intent(in out)	E	Objet to be allocated

Allocation of objet E of eos_list_data_type.

Note:

Should be called before using an object of type eos_data_list.

[Allocate_EoS_List](#)

Subroutine Allocate_EoS_List (N, E)

Integer	Intent(in)	N	Number of elements of E
Type(EoS_List_Type)	Intent(in out)	E	Objet to be allocated

Allocation of objet E of eos_list_type.

Note:

Should be called before using an object of type eos_list.

[Deallocate_EoS_Data_List](#)

Subroutine Deallocate_EoS_Data_List (E)

Type(EoS_Data_List_Type)	Intent(in out)	E	Objet to be allocated
---------------------------------	---------------------------	---	-----------------------

De-allocation of objet E of type eos_data_list. This subroutine should be after using an object of type eos_data_list when it is no longer needed.

[Deallocate_EoS_List](#)

Subroutine Deallocate_EoS_List (E)

Type(EoS_List_Type)	Intent(in out)	E	Objet to be allocated
----------------------------	---------------------------	---	-----------------------

De-allocation of objet E of type eos_list. This subroutine should be after using an object of type eos_list when it is no longer needed.

[Deriv_Partial_P](#)

Subroutine Deriv_Partial_P (V, T, EoSPar, Td)

Real (Kind=CP)	Intent(in)	V	Volume
Real (Kind=CP)	Intent(in)	T	Temperature
Type(EoS_Type)	Intent(in)	EoSPar	EoS Variable

Real (Kind=CP), Dimension (N_EoSPar)	Intent(out)	Td	Derivatives dP/d(param)
---	--------------------	----	-------------------------

Calculates partial derivatives of pressure respect to the EoS parameters. Can be used in least-squares fitting of EoS if the quantity minimised is Pobs-Pcalc

EoS_Cal

Subroutine EoS_Cal (P, T, EoSPar, Parvals)

Real (Kind=CP)	Intent(in)	P	Pressure
Real (Kind=CP)	Intent(in)	T	Temperature
Type(EoS_Type)	Intent(in)	EoSPar	EoS Variable
Real (Kind=CP), Dimension (N_EoSPar)	Intent(out)	Parvals	Output parameter values: 1 is volume 2 is K (bulk modulus) 3 is Kprime (dK/dP) 4 is Kpp (d2K/dP2) 5 is dK/dT 6 is alpha or their linear equivalents

Returns elastic properties (not the parameter values) at this (P,T) for EoS

EoS_Cal_Esd

Subroutine EoS_Cal_Esd (P, T, EoSPar, Esd)

Real (Kind=CP)	Intent(in)	P	Pressure
Real (Kind=CP)	Intent(in)	T	Temperature
Type(EoS_Type)	Intent(in)	EoSPar	EoS Variable
Real (Kind=CP), Dimension (N_EoSPar)	Intent(out)	Esd	Output esd values: 1 is volume 2 is K (bulk modulus) 3 is Kprime (dK/dP) 4 is Kpp (d2K/dP2) 5 is dK/dT 6 is alpha or their linear equivalents

Returns esd's of the elastic properties (not the parameter values) at this P and T for EoS

FfCal_Dat

Subroutine FfCal_Dat (V, V0, P, EoSPar, F, S)

Real (Kind=CP)	Intent(in)	V	Volume
Real (Kind=CP)	Intent(in)	V0	Volume at zero pressure
Real (Kind=CP)	Intent(in)	P	Pressure
Type(EoS_Type)	Intent(in)	EoSPar	EoS Variable
Real (Kind=CP)	Intent(out)	F	Normalized pressure
Real (Kind=CP)	Intent(out)	S	Strain

Returns normalized pressure and Strain at V and P, calculated using the EoS model and V0

Note:

EoS parameters are not used in this calculation!

The only element of EoSpar that is used is the eos model type and the linear flag when the strain function is called

FfCal_Dat_Esd**Subroutine FfCal_Dat_Esd (V, SigV, V0, SigV0, P, SigP, EoSPar, F, SigF, S, SigS)**

Real (Kind=CP)	Intent(in)	V	Volume
Real (Kind=CP)	Intent(in)	SigV	Sigma of V
Real (Kind=CP)	Intent(in)	V0	Volume at zero pressure
Real (Kind=CP)	Intent(in)	SigV0	Sigma of V0
Real (Kind=CP)	Intent(in)	P	Pressure
Real (Kind=CP)	Intent(in)	SigP	Sigma of P
Type(EoS_Type)	Intent(in)	EoSPar	EoS Variable
Real (Kind=CP)	Intent(out)	F	Normalized pressure
Real (Kind=CP)	Intent(out)	SigF	Sigma of F
Real (Kind=CP)	Intent(out)	S	Strain
Real (Kind=CP)	Intent(out)	SigS	Sigma of S

Calculate the Normalised Pressure (F) and Strain (S) value at Volume (V) and Pressure (P) and also their ESD

Note:

EoS parameters are not used in this calculation!

The only element of EoSpar that is used is the eos model type and the linear flag when the strain function is called

FfCal_EoS**Subroutine FfCal_EoS (P, T, EoSPar, F, S)**

Real (Kind=CP)	Intent(in)	P	Pressure
Real (Kind=CP)	Intent(in)	T	Temperature
Type(EoS_Type)	Intent(in)	EoSPar	EoS Variable
Real (Kind=CP)	Intent(out)	F	Normalized pressure
Real (Kind=CP)	Intent(out)	S	Strain

Returns normalised pressure and Strain at this (P,T) calculated from the EoS parameters

Init_EoS_Data_Type**Subroutine Init_EoS_Data_Type (E)**

Type(EoS_Data_Type)	Intent(in out)	E	Objet to be allocated
----------------------------	---------------------------	---	-----------------------

Initialize EoS_Data_Type.

Init_EoS_Thermal**Subroutine Init_EoS_Thermal (EoSPar)**

Type(EoS_Data_Type)	Intent(in out)	EoSPar	Objet to be allocated
---------------------	-------------------	--------	-----------------------

Initialize the EoS Type for Thermal case.

[Init_EoS_Type](#)

Subroutine Init_EoS_Type (EoSPar, CLin, IThermal)

Type(EoS_Data_Type)	Intent(in out)	EoSPar	EoS Type
Character(Len=*), Optional	Intent(in)	CLin	Character variable to indicate linear EoS or not
Integer, Optional	Intent(in)	IThermal	Indicate the Thermal type

Initialize EoS_Type setting all parameters to sensible values.

[Init_EoS](#)

Subroutine Init_EoS ()

Subroutine that initializes errors flags in **CFML_EoS** module.

[Read_EoS_DataFile](#)

Subroutine Read_EoS_Datafile (FName, Dat)

Character(Len=*)	Intent(in)	FName	Name of the file
Type(Eos_Data_List_Type)	Intent (out)	Dat	data structure

General routine to read data for EoS from the 'standard format' data files

[Set_EoS_Names](#)

Subroutine Set_EoS_Names (EoSPar)

Type(EoS_Type)	Intent(in out)	EoSPar	EoS variable
----------------	-------------------	--------	--------------

Set the character variables in eos_type data structures to match the flags already set

[Set_EoS_Use](#)

Subroutine Set_EoS_Use (EoSPar)

Type(EoS_Type)	Intent(in out)	EoSPar	EoS variable
----------------	-------------------	--------	--------------

Sets the 'use' flags for Eos type based on all current settings

IUse Comments

- 0** parameter not used
- 1** parameter is used, settable, refineable
- 2** parameter is used and/or should be reported, settable, but cannot be refined
- 3** parameter is used and/or should be reported, not settable, cannot be refined (includes implied

values)

Set_Kp_Kpp_Cond

Subroutine Set_Kp_Kpp_Cond (EoSPar)

Type(EoS_Type)	Intent(in out)	EoSPar	EoS variable
----------------	-------------------	--------	--------------

Set the implied values of Kp and Kpp depending on the EoS Model and its order (both in EoSPar)

Write_Info_EoS

Subroutine Write_Info_EoS (EoSPar,iout)

Type(EoS_Type)	Intent(in)	EoSPar	EoS variable
Integer, Optional	Intent(in)	IOut	Unit

Subroutine that prints information about all of the EoS parameters on the screen or in the unit iout if it is present when call of procedure is done.

CFML_Geometry_SXTAL

Module for making geometrical calculations in Single crystal instruments.

Variables

- [PSD_Val_Type](#)
- [SXD_Val_Type](#)
- [Err_SXTGeom](#)
- [Err_SXTGeom_Mess](#)
- [PSD](#)
- [SXD](#)

Subroutines

- [Angs_4C_Bisecting](#)
- [CalAng](#)
- [Calc_Om_Ch_i_Phi](#)
- [Calc_Psi](#)
- [Cell_Fr_Ub](#)
- [Chi_Mat](#)
- [D19Psd](#)
- [Dspace](#)
- [Equatorial_Ch_i_Phi](#)
- [FixDnu](#)
- [Flat_Cone_VertDet](#)
- [GenB](#)
- [GenUB](#)
- [Get_Angs_NB](#)

- [Get_DSpacing_Theta](#)
- [Get_GaOmNu_FrChiPhi](#)
- [Get_WaveGaNu_FrZ4](#)
- [Get_Z1_D9Angls](#)
- [Get_Z1_From_Pixel](#)
- [Normal](#)
- [Normal_Beam_Angles](#)
- [Phi_Mat](#)
- [PSD_Convert](#)
- [Psi_Mat](#)
- [RefVec](#)
- [S4CNB](#)
- [Set_PSD](#)
- [SNB4C](#)
- [SXDPSPD](#)
- [Triple](#)
- [Z1FrFC](#)
- [Z1FrMD](#)
- [Z1FrNB](#)
- [Z1FrZ2](#)
- [Z1FrZ3](#)
- [Z1FrZ4](#)
- [Z2FrZ1](#)
- [Z3FrZ1](#)
- [Z4FrGN](#)
- [Z4FrZ1](#)

Fortran Filename

CFML_SXTAL_Geom.f90

Under construction!!!

Variables

- [PSD_Val_Type](#)
- [SXD_Val_Type](#)

- [Err_SXTGeom](#)
- [Err_SXTGeom_Mess](#)
- [PSD](#)
- [SXD](#)

PSD_Val_Type

	<i>Variable</i>	<i>Definition</i>
Type :: PSD_Val_Type		
Real (Kind=CP)	XOff	
Real (Kind=CP)	ZOff	
Real (Kind=CP)	Radius	
Real (Kind=CP)	YOff	
Real (Kind=CP)	CGap	
Real (Kind=CP)	AGap	
Integer	NCat	
Integer	Nano	
Integer	IPSD	
End Type PSD_Val_Type		

SXD_Val_Type

	<i>Variable</i>	<i>Definition</i>
Type :: SXD_Val_Type		
Real (Kind=CP)	DistMs	
Real (Kind=CP)	DistSd	
Real (Kind=CP)	DimX	
Real (Kind=CP)	DimZ	
Real (Kind=CP)	Xoff	
Real (Kind=CP)	YOff	
Real (Kind=CP)	ZOff	
Real (Kind=CP)	TOff	
Real (Kind=CP)	Velcon	
Integer	NXCel	
Integer	NZCel	
End Type SXD_Val_Type		

Err_SXTGeom

Logical :: Err_SXTGeom

This variable is set to .TRUE. if an error occurs in procedures belonging to this module.

Err_SXTGeom_Mess

Character (Len=150) :: Err_SXTGeom_Mess

This variable contains information about the last error occurred in the procedures belonging to this module.

PSD

Type (PSD_Val_Type) :: PSD

SXD

Type (SXD_Val_Type) :: SXD

Subroutines

- [Angs_4C_Bisecting](#)
- [CalAng](#)
- [Calc_Om_Ch_i_Phi](#)
- [Calc_Psi](#)
- [Cell_Fr_Ub](#)
- [Chi_Mat](#)
- [D19Psd](#)
- [Dspace](#)
- [Equatorial_Ch_i_Phi](#)
- [FixDnu](#)
- [Flat_Cone_VertDet](#)
- [GenB](#)
- [GenUB](#)
- [Get_Angs_NB](#)
- [Get_DSpacing_Theta](#)
- [Get_GaOmNu_FrChiPhi](#)
- [Get_WaveGaNu_FrZ4](#)
- [Get_Z1_D9Angls](#)
- [Get_Z1_From_Pixel](#)
- [Normal](#)
- [Normal_Beam_Angles](#)
- [Phi_Mat](#)
- [PSD_Convert](#)
- [Psi_Mat](#)
- [RefVec](#)
- [S4CNB](#)
- [Set_PSD](#)
- [SNB4C](#)
- [SXDPsd](#)
- [Triple](#)
- [Z1FrFC](#)
- [Z1FrMD](#)
- [Z1FrNB](#)
- [Z1FrZ2](#)
- [Z1FrZ3](#)
- [Z1FrZ4](#)
- [Z2FrZ1](#)

- [Z3FrZ1](#)
- [Z4FrGN](#)
- [Z4FrZ1](#)

Angs_4C_Bisecting

Subroutine Angs_4C_Bisecting (Wave, Z1, Tth, Om, Ch, Ph, ierr)

Real(Kind=CP)	Intent(in)	Wave	Wavelength
Real(Kind=CP), Dimension(3)	Intent(in)	Z1	
Real(Kind=CP)	Intent(out)	Tth	2Theta
Real(Kind=CP)	Intent(out)	Om	Omega
Real(Kind=CP)	Intent(out)	Ch	Chi
Real(Kind=CP)	Intent(out)	Ph	Phi
Integer	Intent(out)	ierr	Flag for error

Calculate 2-Theta, Omega (=Theta), Chi, Phi to put the vector Z1 in the bisecting diffraction condition. The reciprocal vector Z1 is given in Cartesian components with respect to the laboratory system. This geometry corresponds to the bisecting Psi=0.

CalAng

Subroutine CalAng (H, TTheta, Om, Ch, Ph, ierr, Wav, UBM, Geom)

Real(Kind=CP), Dimension(3)	Intent(in)	H	Reflection indices
Real(Kind=CP)	Intent(out)	TTheta	2Theta
Real(Kind=CP)	Intent(out)	Om	Omega
Real(Kind=CP)	Intent(out)	Ch	Chi
Real(Kind=CP)	Intent(out)	Ph	Phi
Integer	Intent(out)	ierr	Flag for error
Real(Kind=CP), Optional	Intent(in)	Wav	Wavelength
Real(Kind=CP), Dimension (3,3), Optional	Intent(in)	UBM	UB Matrix
Integer, Optional	Intent(in)	Geom	Geometric definition

This subroutine is a more general variant of [Angs_4C_Bisecting](#).

If the optional arguments are given, the corresponding values are adopted instead of those of the current instrument.

Calc_Om_Chi_Phi

Subroutine Calc_Om_Chi_Phi (Vhkl, VLab1, Psi, UB, Om, Ch, Ph, ierr)

Real(Kind=CP), Dimension(3)	Intent(in)	Vhkl	Vector Indices hkl
Real(Kind=CP), Dimension(3)	Intent(in)	VLab1	
Real(Kind=CP)	Intent(in)	Psi	Psi
Real(Kind=CP), Dimension (3,3)	Intent(in)	UB	UB Matrix
Real(Kind=CP)	Intent(in out)	Om	Omega

Real(Kind=CP)	Intent(in out)	Ch	Chi
Real(Kind=CP)	Intent(in out)	Ph	Phi
Integer	Intent(out)	Ierr	Flag for error

Calculate Om, Ch, Ph for diffraction vector at azimuthal angle Psi from the diffraction condition expressed as :

$$[TZ] = [OM].[CH].[PH].[TS].[PSI]-1 \quad \text{if } [R]=[OM].[CH].[PH]$$

then $[R]'=[TZ].[PSI].[TS]-1$

The Om, Ch, Ph angles are provided, on input, to calculate the components of the vector VLab1 in the Theta-system for Psi=0

Used only in the procedure [Flat_Cone_VertDet](#).

Calc_Psi

Subroutine Calc_Psi (Vhkl, VLab1, Om, Ch, Ph, UB, Psi, Ierr)

Real(Kind=CP), Dimension(3)	Intent(in)	Vhkl	Vector Indices hkl
Real(Kind=CP), Dimension(3)	Intent(in)	VLab1	
Real(Kind=CP)	Intent(in)	Om	Omega
Real(Kind=CP)	Intent(in)	Ch	Chi
Real(Kind=CP)	Intent(in)	Ph	Phi
Real(Kind=CP), Dimension (3,3)	Intent(in)	UB	UB Matrix
Real(Kind=CP)	Intent(out)	Psi	Psi
Integer	Intent(out)	Ierr	Flag for error

Calculate Psi for the reflection Vhkl positioned at Om, Ch and Ph.

The value of Psi is taken to be zero when (the values of Om, Ch, Ph are such that) the reflection H0 lies in the plane of Vhkl and VZ, on the same side of Vhkl as VZ. The reference vectors H0 and VZ are defined in subroutine RefVec. There, the vector VZ is the z-axis of the fixed laboratory system (Busing and Levy Convention, Y along beam, X in positive 2-THETA direction). H0 is (0,0,1) for all VHKL except when VHKL is parallel to (0,0,1), in which case (0,1,0) is chosen.

Cell_Fr_UB

Subroutine Calc_Fr_UB (UB, Ipr, DCell, RCell)

Real(Kind=CP), Dimension (3,3)	Intent(in)	UB	UB Matrix
Integer	Intent(in)	Ipr	Flag to print information
Real(Kind=CP), Dimension(6), Optional	Intent(out)	DCell	Direct cell parameters
Real(Kind=CP), Dimension(6), Optional	Intent(out)	RCell	Reciprocal cell parameters

Calculate and print cell parameters from UB-matrix

Chi_Mat

Subroutine Chi_Mat (Chi, Dum)

Real(Kind=CP)	Intent(in)	Chi	Chi
Real(Kind=CP), Dimension (3,3)	Intent(out)	Dum	

Calculate the Busing and Levy conventional rotation matrix for Chi (in degrees).

D19PSD**Subroutine D19PSD (MPSD, Ga, Nu, Cath, Anod, ierr)**

Integer	Intent(in)	MPSD	
Real(Kind=CP)	Intent(in out)	Ga	
Real(Kind=CP)	Intent(in out)	Nu	
Real(Kind=CP)	Intent(in out)	Cath	
Real(Kind=CP)	Intent(in out)	Anod	
Integer	Intent(in out)	ierr	

Specifically for D19A bannana detector, 4 X 64 degrees - 16 X 512 cells and vertically curved.

MPSD + VE - Calculate delta GAMMA and NU from the cathode, anode co-ordinates

MPSD - VE - Calculate the anode co-ordinate from NU

Some of the variables making reference to the characteristics of the detector are provisionally stored in a public type(Psd_Val_Type):: PSD

DSpace**Subroutine DSpace (Wave, Vhkl, Cell, Ds, Th, ierr)**

Real(Kind=CP)	Intent(in)	Wave	
Real(Kind=CP), Dimension(3)	Intent(in)	Vhkl	
Real(Kind=CP), Dimension(6)	Intent(in)	Cell	
Real(Kind=CP)	Intent(out)	Ds	
Real(Kind=CP)	Intent(out)	Th	
Integer	Intent(out)	ierr	

Calculate d-spacing and theta from cell parameters and wavelength assume triclinic symmetry. The reflection vector Vhkl is provided in reciprocal lattice components

Equatorial_Chi_Phi**Subroutine Equatorial_Chi_Phi (Z1, Ch, Ph)**

Real(Kind=CP), Dimension(3)	Intent(in)	Z1	
Real(Kind=CP)	Intent(out)	Ch	
Real(Kind=CP)	Intent(out)	Ph	

Calculate Chi, Phi to put the vector Z1 in the equatorial plane. The reciprocal vector Z1 is given in Cartesian components with respect to the laboratory system

FixDNu

Subroutine FixDNu (Wave, Z1, Nu, Ch, Ph, Ga, Om, ierr)

Real(Kind=CP)	Intent(in)	Wave	Wavelength
Real(Kind=CP), Dimension(3)	Intent(in)	Z1	
Real(Kind=CP)	Intent(in)	Nu	Angle
Real(Kind=CP)	Intent(out)	Ch	Angle
Real(Kind=CP)	Intent(out)	Ph	Angle
Real(Kind=CP)	Intent(out)	Ga	Angle
Real(Kind=CP)	Intent(out)	Om	Angle
Integer	Intent(out)	ierr	Flag for control error

Calculate a setting Ch, Ph,Ga,Om to put the diffracted beam at Nu.

Ph puts the diffraction vector Z1 into the Chi circle (as for bisecting geometry), Ch brings the vector to the appropriate Nu and Om then positions the beam at Ga.

Flat_Cone_VertDet**Subroutine Flat_Cone_VertDet (Wave, Z1, UB, VRho, Rho, Ch, Ph, Ga, Om, Nu, ierr)**

Real(Kind=CP)	Intent(in)	Wave	Wavelength
Real(Kind=CP), Dimension(3)	Intent(in)	Z1	
Real(Kind=CP), Dimension (3,3)	Intent(in)	UB	UB Matrix
Real(Kind=CP), Dimension(3)	Intent(in out)	VRho	
Real(Kind=CP)	Intent(out)	Rho	Angle
Real(Kind=CP)	Intent(out)	Ch	Angle
Real(Kind=CP)	Intent(out)	Ph	Angle
Real(Kind=CP)	Intent(out)	Ga	Angle
Real(Kind=CP)	Intent(out)	Om	Angle
Real(Kind=CP)	Intent(out)	Nu	Angle
Integer	Intent(out)	ierr	Flag for control error

Calculate Rho (=Psi) about given rotation vector VRho (and the corresponding angles Om, Ch, Ph) to put the vector Z1 into the flat-cone diffracting position.

GenB**Subroutine GenB (C, B)**

Type(Crystal_Cell_Type)	Intent(in)	C	Crystal Cell object
Real(Kind=CP), Dimension (3,3)	Intent(out)	B	

Calculation of B Matrix.

Note: Acta Cryst., 22, (1967), 457-464 (Eq. 3)

GenUB**Subroutine GenUB (B, H1, H2, H1O, H2O, UB, ierr)**

Real(Kind=CP), Dimension (3,3)	Intent(in)	B	Busing-Levy B-matrix
Real(Kind=CP), Dimension(3)	Intent(in)	H1	Miller indices
Real(Kind=CP), Dimension(3)	Intent(in)	H2	Miller indices
Real(Kind=CP), Dimension(3)	Intent(in)	H1O	Components in Lab system
Real(Kind=CP), Dimension(3)	Intent(in)	H2O	Components in Lab system
Real(Kind=CP), Dimension (3,3)	Intent(out)	UB	UB Matrix
Integer	Intent(out)	ierr	Flag for control error

Given the B matrix, the Miller indices of two reflections, H1 & H2, and the components of these two reflections, H1O & H2O, in the laboratory system, this subroutine provides the matrix UB. Only the direction in the laboratory system of reflections are needed, e.g. H1O and H2O may be unitary vectors or whatever other vector along these directions.

Get_Angs_NB

Subroutine Get_Angs_NB (Wave, Z1, Ga, Om, Nu, ierr)

Real(Kind=CP)	Intent(in)	Wave	Wavelength
Real(Kind=CP), Dimension(3)	Intent(in)	Z1	
Real(Kind=CP)	Intent(out)	Ga	Angle
Real(Kind=CP)	Intent(out)	Om	Angle
Real(Kind=CP)	Intent(out)	Nu	Angle
Integer	Intent(out)	ierr	Flag for control error

Calculate normal-beam angles Gamma, Omega, Nu to put the vector Z1 into the diffracting condition.

Get_DSpacing_Theta

Subroutine Get_DSpacing_Theta (Wave, Z1, Ds, Th, ierr)

Real(Kind=CP)	Intent(in)	Wave	Wavelength
Real(Kind=CP), Dimension(3)	Intent(in)	Z1	
Real(Kind=CP)	Intent(out)	Ds	Angle
Real(Kind=CP)	Intent(out)	Th	Angle
Integer	Intent(out)	ierr	Flag for control error

Calculate D-spacing (real space) and Theta from the length of Z1. The reciprocal vector Z1 is given in Cartesian components with respect to the laboratory system.

If ierr=1 the calculated d-spacing is fixed to 0.0 as well as theta. This error condition appears when the length of the reciprocal vector Z1 is lower or equal to 0.0001
If ierr=2 the reflection is outside the resolution sphere.

Get_GaOmNu_FrChiPhi

Subroutine Get_GaOmNu_FrChiPhi (Wave, Z1, Ch, Ph, Ga, Om, Nu, ierr)

Real(Kind=CP)	Intent(in)	Wave	Wavelength
Real(Kind=CP), Dimension(3)	Intent(in)	Z1	

Real(Kind=CP)	Intent(in)	Ch	Angle
Real(Kind=CP)	Intent(in)	Ph	Angle
Real(Kind=CP)	Intent(out)	Ga	Angle
Real(Kind=CP)	Intent(out)	Om	Angle
Real(Kind=CP)	Intent(out)	Nu	Angle
Integer	Intent(out)	Ierr	Flag for control error

Given Chi & Phi, calculate normal-beam angles Gamma, Omega, Nu to put the vector Z1 into the diffraction condition. The reciprocal vector Z1 is given in Cartesian components with respect to the laboratory system.

Get_WaveGaNu_FrZ4

Subroutine Get_WaveGaNu_FrZ4 (Z4, Wave, Ga, Nu, Ierr)

Real(Kind=CP), Dimension(3)	Intent(in)	Z4	
Real(Kind=CP)	Intent(out)	Wave	Wavelength
Real(Kind=CP)	Intent(out)	Ga	Angle
Real(Kind=CP)	Intent(out)	Nu	Angle
Integer	Intent(out)	Ierr	Flag for control error

Calculate Ga, Nu and wavelength for diffraction vector in Laboratory system

Get_Z1_D9Angls

Subroutine Get_Z1_D9Angls (Wave, TTheta, Om, Ch, Ph, Z1)

Real(Kind=CP)	Intent(in)	Wave	Wavelength
Real(Kind=CP)	Intent(in)	TTheta	Wavelength
Real(Kind=CP)	Intent(in)	Om	Angle
Real(Kind=CP)	Intent(in)	Ch	Angle
Real(Kind=CP)	Intent(in)	Ph	Angle
Real(Kind=CP), Dimension(3)	Intent(out)	Z1	Flag for control error

Calculate Z1 from angles for D9 Instrument

Get_Z1_From_Pixel

Subroutine Get_Z1_From_Pixel (NPx, NPy, SNum, Z1)

Integer	Intent(in)	NPx	Wavelength
Integer	Intent(in)	NPy	Wavelength
Type(SXTAL_Numor_Type)	Intent(in)	SNum	Numor
Real(Kind=CP), Dimension(3)	Intent(out)	Z1	

Calculate Z1 from Numor

Normal

Subroutine Normal (V1, Ierr)

Real(Kind=CP), Dimension(3)	Intent(in out)	V1	
Integer	Intent(out)	lerr	Error control

Normalise vector V (in Cartesian components)

Normal_Beam_Angles

Subroutine Normal_Beam_Angles (Wav, UB, H, Sig, AnBCal, ler, Zer)

Real(Kind=CP)	Intent(in)	Wav	Wavelength
Real(Kind=CP), Dimension (3,3)	Intent(in)	UB	UB Matrix
Real(Kind=CP), Dimension(3)	Intent(in)	H	Miller indices of reflection
Integer	Intent(in out)	Sig	-1 for negative Gammas
Real(Kind=CP), Dimension (:)	Intent(out)	AnBCal	Normal beam angles
Integer	Intent(in out)	ler	Zero correction on input, Error flag on output
Real(Kind=CP), Dimension(3), Optional	Intent(in)	Zer	Zero corrections of NB angles in degrees

Calculation of the normal-beam diffraction angles for reflection H from the UB matrix of the crystal.

On input:

Sig : 1/-1 defines the sign for Gamma

If Zer is provided, the calculated angles are corrected for zero shifts

On output:

Angles(calculated) = Angles(theoretical) + zero shifts

AnBCal(1:4) -> Gamma, Omega(NB), Nu, Theta (in degrees)

Error Flag (ler)

0 -> all angles are calculable

1 -> reflection hors sphere d'Ewald

2 -> reflection dans zone aveugle ==> angle "NU"

3 -> reflection dans zone aveugle ==> angle "GAMMA"

4 -> H,K,L tous les trois nuls

Phi_Mat

Subroutine Phi_Mat (Phi, Dum)

Real(Kind=CP)	Intent(in)	Phi	Phi
Real(Kind=CP), Dimension (3,3)	Intent(out)	Dum	

Calculate the Busing and Levy conventional rotation matrix for Phi or Omega. The Phi/Omega angle must be provided in degrees.

PSD_Convert

Subroutine PSD_Convert (MPSD, Gamm, GamP, Nup, XObs, ZObs, Cath, Anod, lerr)

Integer	Intent(in)	MPSD	
Real(Kind=CP)	Intent(in)	Gamm	

Real(Kind=CP)	Intent(in out)	GamP	
Real(Kind=CP)	Intent(in out)	Nup	
Real(Kind=CP)	Intent(out)	XObs	
Real(Kind=CP)	Intent(out)	ZObs	
Real(Kind=CP)	Intent(in out)	Cath	
Real(Kind=CP)	Intent(in out)	Anod	
Integer	Intent(out)	Ierr	Flag for Control error

Subroutine for getting Gamma and Nu of a reflections spot (GamP,NuP), given the gamma angle of the detector (GamM) and the pixel values (cath,anod). This is calculated when MPSD > 0, otherwise the inverse calculation is done. In both cases the detector coordinates (xobs,zobs) in mm are also calculated. The characteristics of the detector are accessed via de global variable PSD of Type(Psd_Val_Type), that should be set by the calling program.

Psi_Mat

Subroutine Psi_Mat (Psi, Dum)

Real(Kind=CP)	Intent(in)	Psi	Psi
Real(Kind=CP), Dimension (3,3)	Intent(out)	Dum	

Calculate the Busing and Levy conventional rotation matrix for Psi (in degrees).

RefVec

Subroutine RefVec (Vhkl, UB, Vs, Vz, Ierr)

Real(Kind=CP), Dimension(3)	Intent(in)	Vhkl	Vector
Real(Kind=CP), Dimension (3,3)	Intent(in)	UB	UB Matrix
Real(Kind=CP), Dimension(3)	Intent(out)	Vs	
Real(Kind=CP), Dimension(3)	Intent(out)	Vz	
Integer	Intent(out)	Ierr	Error Control flag

Calculate Vs and Vz as reference vectors for defining Psi=0. The B-L convention is that Psi=0 when the reflection hkl is in diffraction position and the c* is in the plane defined by Vhkl and Vz (z-axis of the laboratory system) for all reflections except when Vhkl is parallel to c* in which case the vector b* plays the role of c* in the above prescription. The vector Vhkl is provided with components in the reciprocal lattice.

S4CNB

Subroutine S4CNB (Angl_4C, Angl_NB, Ierr)

Real(Kind=CP), Dimension(4)	Intent(in)	Angl_4C	(/2Theta, Omega, chi, Phi/)
Real(Kind=CP), Dimension(3)	Intent(out)	Angl_NB	(/Gamma, Omega_NB, Nu/)
Integer	Intent(out)	Ierr	Error Control flag 0 -> OK 1 -> calculation of Nu impossible 2 -> calculation of Gamma impossible 3 -> calculation of phi impossible

Conversion of diffraction angles from the geometry 4-Circles to Normal Beam.

Set_PSD

Subroutine Set_PSD ()

Initialize the PSD Object

SNB4C

Subroutine SNB4C (Angl_NB, Angl_4C)

Real(Kind=CP), Dimension(4)	Intent(in)	Angl_NB	(/Gamma, Omega_NB, Nu/)
Real(Kind=CP), Dimension(3)	Intent(out)	Angl_4C	(/2Theta, Omega, chi, Phi/)

Conversion of diffraction angles from the geometry Normal Beam to 4-Circles.

SXDPSD

Subroutine SXDPSD (MPSD, Gamm, Wave, Nup, GamP, XObs, ZObs, XCel, Time, ZCel, ierr)

Integer	Intent(in)	MPSD	
Real(Kind=CP)	Intent(in)	Gamm	
Real(Kind=CP)	Intent(in out)	Nup	
Real(Kind=CP)	Intent(in out)	GamP	
Real(Kind=CP)	Intent(out)	XObs	
Real(Kind=CP)	Intent(out)	ZObs	
Real(Kind=CP)	Intent(out)	XCel	
Real(Kind=CP)	Intent(out)	Time	
Real(Kind=CP)	Intent(out)	ZCel	
Integer	Intent(out)	ierr	Flag for Control error

The coordinate system adopted, whether the origin is at the moderator, at the sample (called the fixed laboratory system), or at the surface of the PSD when positioned at 0 degrees; is Y parallel to the beam, X in the horizontal plane on the diffraction side, and Z vertical. Hence if neutrons are diffracted to the left, Z is vertically down. The PSD is driven to an angle GamM. When GamM=0, the direct beam strikes a perfectly aligned PSD at its centre C.

Call this point in space A. A is where we define NuP=0, GamP=0. The coordinates of A with respect to the sample are (0, Distsd, 0); and with respect to the moderator are (0, Distms+Distsd, 0). For a mis-aligned detector, the coordinates of A with respect to C are the translational offsets (Xoff, Yoff, Zoff) in mm.

With the PSD at a general position GamM, the point where the direct beam struck it is rotated to O, where now NuP=0, GamP=GamM. For convenience, define a new cartesian system by rotating the axes of the fixed laboratory system about the vertical, such that X is now along the line joining the sample and O on the PSD. In this system, the coordinates of a Bragg peak P with respect to C are (Xobs, 0, Zobs) in mm, or (Xcel, 0, Zcel) in pixels.

Hence the coordinates of P with respect to O, in this system, are:

$$(x) \quad (Xobs + Xoff)$$

$$(y) = (\text{Distsd} + \text{Yoff}) \text{ and: } \tan(\text{GamP} - \text{GamM}) = x/y$$

$$(z) = (\text{Zobs} + \text{Zoff}) \quad \tan(\text{NuP}) = z/\sqrt{x^*x + y^*y}$$

The PSD front surface measures Dimx by Dimy mm, and is divided into Nxcel by Nzcel pixels.

Time is the time coordinate (bin) relative to an elapsed time Toff after the emission of a pulse at the moderator. The effect of moderator thickness on Time is NOT included yet. Distot is the total distance travelled from the moderator to a particular pixel on the PSD surface, in a total time Timtot.

Note: Routine not tested, probably obsolete for present SXD!!!

Triple

Subroutine Triple (V1, V2, Tv, ierr)

Real(Kind=CP), Dimension(3)	Intent(in out)	V1	
Real(Kind=CP), Dimension(3)	Intent(in out)	V2	
Real(Kind=CP), Dimension (3,3)	Intent(out)	Tv	(/ V1, (V1 x V2) x V1, V1 x V2/)
Integer	Intent(out)	ierr	

Construct orthonormal triplet matrix TV

Z1FrFC

Subroutine Z1FrFC (Wave, TTh, Om, Ch, Ph, Z1)

Real(Kind=CP)	Intent(in)	Wave	Wavelength
Real(Kind=CP)	Intent(in)	TTh	2 Theta
Real(Kind=CP)	Intent(in)	Om	Angle
Real(Kind=CP)	Intent(in)	Ch	Angle
Real(Kind=CP)	Intent(in)	Ph	Angle
Real(Kind=CP), Dimension(3)	Intent(out)	Z1	

Calculate diffraction vector Z1 from TTh, Om, Ch, Ph (Need not be bisecting, but Z4 is assumed to be in the equatorial plane)

Z1FrMD

Subroutine Z1FrMD (Wave, Ch, Ph, Ga, Om, Nu, Z1)

Real(Kind=CP)	Intent(in)	Wave	Wavelength
Real(Kind=CP)	Intent(in)	Ch	Angle
Real(Kind=CP)	Intent(in)	Ph	Angle
Real(Kind=CP)	Intent(in)	Ga	Angle
Real(Kind=CP)	Intent(in)	Om	Angle
Real(Kind=CP)	Intent(in)	Nu	Angle
Real(Kind=CP), Dimension(3)	Intent(out)	Z1	

Calculate diffraction vector Z1 from Ch, Ph, Ga, Om, Nu for a multi-detector. The angles Chi, Phi, Gamma, Omega and Nu for the equatorial plane are Chi, Phi, 2Theta and Omega (Nu=0).

Z1FrNB

Subroutine Z1FrNB (Wave, Ga, Om, Nu, Z1)

Real(Kind=CP)	Intent(in)	Wave	Wavelength
Real(Kind=CP)	Intent(in)	Ga	Angle
Real(Kind=CP)	Intent(in)	Om	Angle
Real(Kind=CP)	Intent(in)	Nu	Angle
Real(Kind=CP), Dimension(3)	Intent(out)	Z1	

Calculate diffraction vector Z1 from Ga, Om, Nu, assuming CH=PH=0. This is the normal beam geometry for a Lifting arm detector or for a PSD with a single Omega axis for the sample.

Z1FrZ2**Subroutine Z1FrZ2 (Z2, Ph, Z1)**

Real(Kind=CP), Dimension(3)	Intent(in)	Z2	
Real(Kind=CP)	Intent(in)	Ph	Angle
Real(Kind=CP), Dimension(3)	Intent(out)	Z1	

Calculate

$$Z1 = Ph^T Z2$$

Z1FrZ3**Subroutine Z1FrZ3 (Z3, Ch, Ph, Z1)**

Real(Kind=CP), Dimension(3)	Intent(in)	Z3	
Real(Kind=CP)	Intent(in)	Ch	
Real(Kind=CP)	Intent(in)	Ph	
Real(Kind=CP), Dimension(3)	Intent(out)	Z1	

Calculate

$$Z1 = Ph^T Ch^T Z3$$

Z1FrZ4**Subroutine Z1FrZ4 (Z4, Om, Ch, Ph, Z1)**

Real(Kind=CP), Dimension(3)	Intent(in)	Z4	
Real(Kind=CP)	Intent(in)	Om	
Real(Kind=CP)	Intent(in)	Ch	
Real(Kind=CP)	Intent(in)	Ph	
Real(Kind=CP), Dimension(3)	Intent(out)	Z1	

Calculate

$$Z1 = Ph^T Ch^T Om^T Z3$$

Z2FrZ1

Subroutine Z2FrZ1 (Z1, Ph, Z2)

Real(Kind=CP), Dimension(3)	Intent(in)	Z1	
Real(Kind=CP)	Intent(in)	Ph	Angle
Real(Kind=CP), Dimension(3)	Intent(out)	Z2	

Calculate

$$Z2 = Ph \cdot Z1$$

Note: The reciprocal vector Z1 is given in Cartesian components with respect to the laboratory system.

Z3FrZ1

Subroutine Z3FrZ1 (Z1, Ch, Ph, Z3)

Real(Kind=CP), Dimension(3)	Intent(in)	Z1	
Real(Kind=CP)	Intent(in)	Ch	
Real(Kind=CP)	Intent(in)	Ph	
Real(Kind=CP), Dimension(3)	Intent(out)	Z3	

Calculate

$$Z3 = Ch \cdot Ph \cdot Z1$$

Note: The reciprocal vector Z1 is given in Cartesian components with respect to the laboratory system.

Z4FrGN

Subroutine Z4FrGN (Wave, Ga, Nu, Z4)

Real(Kind=CP)	Intent(in)	Wave	Wavelength
Real(Kind=CP)	Intent(in)	Ga	Angle
Real(Kind=CP)	Intent(in)	Nu	Angle
Real(Kind=CP), Dimension(3)	Intent(out)	Z4	

Calculates diffraction vector of a reflection in the Laboratory system from the angles Ga and Nu.

Z4FrZ1

Subroutine Z4FrZ1 (Z1, Om, Ch, Ph, Z4)

Real(Kind=CP), Dimension(3)	Intent(in)	Z1	
Real(Kind=CP)	Intent(in)	Om	
Real(Kind=CP)	Intent(in)	Ch	
Real(Kind=CP)	Intent(in)	Ph	
Real(Kind=CP), Dimension(3)	Intent(out)	Z4	

Calculate

$$Z4 = Om \cdot Ch \cdot Ph \cdot Z1$$

CFML_Reflections_Utils

Module containing a series of procedures handling operation with Bragg reflections

Variables

- [Reflect_Type](#)
- [Reflection_Type](#)
- [Reflection_List_Type](#)

- [Err_Ref](#)
- [Err_Ref_Mess](#)
- [HKL_Ref_Conditions](#)

Functions

- [AsU_HKL](#)
- [Get_HEquiv_AsU](#)
- [Get_MaxNumRef](#)
- [HKL_Absent](#)
- [HKL_Equal](#)
- [HKL_Equiv](#)
- [HKL_Mult](#)
- [HKL_R](#)
- [HKL_S](#)
- [Unit_Cart_HKL](#)

Subroutines

- [HKL_Equiv_List](#)
- [HKL_Gen](#)
- [HKL_Gen_SXTAL](#)
- [HKL_RP](#)
- [HKL_Uni](#)
- [Init_Err_Ref](#)
- [Init_RefList](#)
- [Search_Exinctions](#)
- [Write_AsU](#)
- [Write_RefList_Info](#)

Fortran Filename

CFML_Refct_Util.f90

Variables

- [Reflect_Type](#)
- [Reflection_Type](#)
- [Reflection_List_Type](#)

- [Err_Ref](#)
- [Err_Ref_Mess](#)
- [HKL_Ref_Conditions](#)

Reflect_Type

	Variable	Definition
Type :: Reflect_Type		
Integer, Dimension(3)	H	Indices for reflection (hkl)
Integer	Mult	Multiplicity
Real(Kind=CP)	S	$\sin\theta/\lambda$
End Type Reflect_Type		

Reflection_Type

	Variable	Definition
Type :: Reflection_Type		
Integer, Dimension(3)	H	index of reflection (hkl)
Integer	Mult	Multiplicity
Real(Kind=CP)	Fo	Observed Structure Factor
Real(Kind=CP)	Fc	Calculated Structure Factor
Real(Kind=CP)	SFo	Sigma of Fo
Real(Kind=CP)	S	$\sin\theta/\lambda$
Real(Kind=CP)	W	Weight
Real(Kind=CP)	Phase	Phase in degrees
Real(Kind=CP)	A	Real part of the Structure Factor
Real(Kind=CP)	B	Imaginary part of the Structure Factor
Real(Kind=CP)	AA	Free parameter
Real(Kind=CP)	BB	Free parameter
End Type Reflection_Type		

Reflection_List_Type

	Variable	Definition
Type :: Reflection_List_Type		
Integer	NRef	Number of Reflections
Type(Reflection_Type), Dimension(:), Allocatable	Ref	Reflection List
End Type Reflection_List_Type		

Err_RefI

Logical :: Err_RefI

This variable is set to .TRUE. if an error occurs in procedures belonging to this module

Err_RefI_Mess

Character (Len=150) :: Err_RefI_Mess

This variable contains information about the last error occurred in the procedures belonging to this module

HKL_Ref_Conditions

Character(Len=80), Dimension(58) :: HKL_Ref_Conditions

Reflection conditions for Lattices, glide planes, screw axes

Functions

- [AsU_HKL](#)
- [Get_HEquiv_AsU](#)
- [Get_MaxNumRef](#)
- [HKL_Absent](#)
- [HKL_Equal](#)
- [HKL_Equiv](#)
- [HKL_Mult](#)
- [HKL_R](#)
- [HKL_S](#)
- [Unit_Cart_HKL](#)

AsU_HKL

Integer Function AsU_HKL (H, SpaceGroup)

Integer, Dimension(3)	Intent (in)	H	
Type(Space_Group_Type)	Intent (in)	SpaceGroup	

Obtain an equivalent reflection in asymmetric unit using simple transformation rules for each crystal system.

When these rules are not satisfied the output is the (0,0,0) reflection. For obtaining a reflection within the asymmetric unit given an input reflection the best is to use the function: [Get_HEquiv_AsU](#)

Note: in default , we assumed that $F(hkl)=F(-h -k -l)$.

Get_HEquiv_AsU

Integer Function Get_HEquiv_AsU (H, SpaceGroup)

Integer, Dimension(3)	Intent (in)	H	
Type(Space_Group_Type)	Intent (in)	SpaceGroup	

Provides a reflection equivalent to the input one but within the asymmetric unit

Get_MaxNumRef

Integer Function Get_MaxNumRef (SinTlMax, VolCell, SinTlMin, Mult)

Real(Kind=CP)	Intent (in)	SinTlMax	Maximum sinθ/λ
Real(Kind=CP)	Intent (in)	VolCell	Direct Cell Volume
Real(Kind=CP), Optional	Intent (in)	SinTlMin	Minimum sinθ/λ
Integer, Optional	Intent (in)	Mult	General Multiplicity

Provides an upper limit of the expected maximum number of reflections up to **SinTlMax** for a volume **VolCell** of the primitive cell. If the optional argument **SinTlMin** is given, the result is the number of reflections in the interval (**SinTlMin**,**SinTlMax**).

If **Mult** is provided the result is divided by this multiplicity so we obtain the expected number of unique reflections.

HKL_Absent

Logical Function HKL_Absent (H, SpaceGroup)

Integer / Real(Kind=CP), Dimension(3)	Intent (in)	H	
Type(Space_Group_Type)	Intent (in)	SpaceGroup	

Returns the value .TRUE. if the reflection is absent.

HKL_Equal

Logical Function HKL_Equal (H, K)

Integer / Real(Kind=CP),	Intent	H	Reflection vector
--------------------------	--------	---	-------------------

Dimension(3)	(in)		
Integer / Real(Kind=CP), Dimension(3)	Intent (in)	K	Reflection vector

Returns the value .TRUE. if two reflections are equal.

HKL_Equiv

LOGICAL Function HKL_EQUIV (H, K, SPACEGROUP, FRIEDEL)

Integer / Real(Kind=CP), Dimension(3)	Intent(in)	H	Reflection vector
Integer / Real(Kind=CP), Dimension(3)	Intent(in)	K	Reflection vector
TYPE(SPACE_GROUP_TYPE)	Intent(in)	SPACEGRO	Space group information UP
LOGICAL, Optional	Intent(in)	FRIEDEL	

Calculate if two reflections are equivalent

HKL_Mult

Integer Function HKL_Mult (H, SpaceGroup, Friedel)

Integer / Real(Kind=CP), Dimension(3)	Intent (in)	H	Reflection vector
Type(Space_Group_Type)	Intent (in)	SpaceGroup	Space group information
Logical, Optional	Intent (in)	Friedel	

Calculate the multiplicity of the reflection H

HKL_R

Integer / Real Function HKL_R (H, OP)

Integer / Real(Kind=CP), Dimension(3)	Intent (in)	H	Reflection vector
Type(Sym_Oper_Type)	Intent (in)	OP	Symmetry operator

Calculate the equivalent reflection

HKL_S

Real Function HKL_S (H, CrystalCell)

Integer / Real(Kind=CP), Dimension(3)	Intent (in)	H	Reflection vector
Type(Crystal_cell_Type)	Intent (in)	CrystalCell	Cell Parameters

Calculate: $\sin\theta/\lambda = 1/(2d)$

Unit_Cart_HKL

Real Function UNIT_CART_HKL (H, CRYSTALCELL)

Integer / Real(Kind=CP), Dimension(3)	Intent (in)	H	Reflection vector
Type(Crystal_Cell_Type)	Intent (in)	CrystalCell	Cell Parameters

Calculate a unitary vector in the cartesian crystal frame along a reciprocal vector hkl (reciprocal lattice)

Subroutines

- [HKL_Equiv_List](#)
- [HKL_Gen](#)
- [HKL_Gen_SXTAL](#)
- [HKL_RP](#)
- [HKL_Uni](#)
- [Init_Err_Ref](#)
- [Init_RefList](#)
- [Search_Exinctions](#)
- [Write_AsU](#)
- [Write_RefList_Info](#)

HKL_Equiv_List

Subroutine HKL_Equiv_List (H, SpaceGroup, Friedel, Mul, HList)

Integer / Real(Kind=CP), Dimension(3)	Intent (in)	H	Reflection
Type(Space_Group_Type)	Intent (in)	SpaceGroup	Space group
Logical	Intent (in)	Friedel	
Integer	Intent (out)	Mul	Multiplicity
Integer / Real(Kind=CP), Dimension(3, SpaceGroup%NumOps*2)	Intent (out)	HList	

Calculate the multiplicity of the reflection and the list of all equivalent reflections. Friedel law assumed if **Friedel=.true.**.

HKL_Gen

Subroutine HKL_Gen (CrystalCell, SpaceGroup, Friedel, Value1, Value2, Num_Ref, Reflex)

Type(Crystal_Cell_Type)	Intent (in)	CrystalCell	Cell Parameters
Type(Space_Group_Type)	Intent (in)	SpaceGroup	Space group
Logical	Intent (in)	Friedel	If TRUE, Friedel law applied
Real(Kind=CP)	Intent (in)	Value1	Range in Sinθ/λ
Real(Kind=CP)	Intent (in)	Value2	
Integer	Intent (out)	Num_Ref	Number of generated reflections
Type(Reflect_Type)	Intent (out)	Reflex	List of generated hkl,mult, s

Calculate unique reflections between two values of Sinθ/λ. The output is not ordered.

HKL_Gen_SXTAL

Subroutine HKL_Gen_SXTAL (CrystalCell, SpaceGroup, STIMax, Num_Ref, Reflex, Ord)

Type(Crystal_Cell_Type)	Intent (in)	CrystalCell	Cell Parameters
Type(Space_Group_Type)	Intent (in)	SpaceGroup	Space group
Real(Kind=CP)	Intent (in)	STIMax	Maximum Sinθ/λ
Integer	Intent (out)	Num_Ref	Number of generated reflections
Type(Reflect_Type) or Type(Reflection_List_Type)	Intent (out)	Reflex	List of generated hkl,mult, s
Integer, Dimension(3), Optional	Intent (out)	Ord	Order for loop of hkl-indices

Calculate all allowed reflections up to a maximum value of Sinθ/λ.

The output is not ordered but the user can obtain the reflections generated in a particular way by providing the Integer vector **Ord**, containing a permutation of the three numbers 1,2,3. By default the loop generating the hkl-indices uses the vector **Ord**=(/3,2,1/), this means that the inner loop (more rapidly changing index) is the l-index, then the k-index and finally the h-index.

HKL_RP

Subroutine HKL_RP (H, Phase, OP, K, PhaseN)

Integer / Real(Kind=CP), Dimension(3)	Intent (in)	H	Reflection vector
Real(Kind=CP)	Intent (in)	Phase	Phase in Degrees
Type(Sym_Oper_Type)	Intent (in)	OP	Symmetry operator

Integer / Real(Kind=CP), Dimension(3)	Intent(out)	K	Equivalent reflection vector
Real(Kind=CP)	Intent(out)	PhaseN	Phase in Degrees of the equivalent reflection

Calculate the equivalent reflection and Phase

HKL_Uni

Subroutine HKL_Uni (CrystalCell, SpaceGroup, Friedel, Value1, Value2, Code, Num_Ref, Reflex)

Type(Crystal_Cell_Type)	Intent(in)	CrystalCell	Cell Parameters
Type(Space_Group_Type)	Intent(in)	SpaceGroup	Space group
Logical	Intent(in)	Friedel	If TRUE, Friedel law applied
Real(Kind=CP)	Intent(in)	Value1	Range in $\sin\theta/\lambda$
Real(Kind=CP)	Intent(in)	Value2	
Character(Len=1)	Intent(in)	Code	Value: R : d-spacing are input
Integer	Intent(out)	Num_Ref	Number of generated reflections
Type(reflect_Type), Dimension(:) or Type(Reflection_Type), Dimension(:) or Type(Reflection_List_Type), Dimension(:)	Intent(out)	Reflex	Ordered set of reflections

Calculate unique reflections between two values (Value1,Value2) of $\sin\theta/\lambda$

Init_Err_Refl

Subroutine Init_Err_Refl ()

Subroutine that initializes errors flags in **CFML_Reflections_Utils** module.

Init_RefList

Subroutine Init_RefList (Reflex, N)

Type(Reflection_List_Type)	Intent(in)	Reflex	
Integer, Optional	Intent(in)	N	Number of reflections on the List

initialize the Reflection List Variable Reflex

Search_Extinctions

Subroutine Search_Extintions (SpaceGroup, Iunit)

Type(Space_Group_Type)	Intent (in)	SpaceGroup	Space group
Integer, Optional	Intent (in)	lunit	Unit to write

Write information about the Reflections extintion for SpaceGroup

Write_AsU

Subroutine Write_AsU (SpaceGroup, lunit)

Type(Space_Group_Type)	Intent (in)	SpaceGroup	Space group
Integer, Optional	Intent (in)	lunit	Unit to write

Write information about the asymmetric unit for reciprocal space.

Write_RefList_Info

Subroutine Write_RefList_Info (Reflex, lunit, Mode)

Type(Reflection_List_Type)	Intent (in)	Reflex	Reflection list
Integer, Optional	Intent (in)	lunit	Unit to write
Character(Len=*), Optional	Intent (in)	Mode	Value: NUC : For Nuclear reflections Rest: X-Ray reflections

Write information about the Reflection List

Level 5

Concept	Module Name	Purpose
Geometry...	CFML_Geometry_Calc	Geometry Calculations
Propagation vectors...	CFML_Propagation_Vectors	Procedures handling operations with propagation/modulation vectors
Structure Factors...	CFML_Structure_Factors	Structure Factors Calculations

[CFML_Geometry_Calc](#)

Routines for Geometry Calculations

Variables

- [Coordination_Type](#)
- [Point_List_Type](#)

- [Coord_Info](#)
- [Err_Geom](#)
- [Err_Geom_Mess](#)

Functions

- [Angle_Dihedral](#)
- [Angle_Mod](#)
- [Angle_UV](#)
- [Coord_Mod](#)
- [Distance](#)
- [Matrix_PhiTheChi](#)
- [Matrix_RX](#)
- [Matrix_RY](#)
- [Matrix_RZ](#)

Subroutines

- [Allocate_Coordination_Type](#)
- [Allocate_Point_List](#)
- [Calc_Dist_Angle](#)
- [Calc_Dist_Angle_Sigma](#)
- [Deallocate_Coordination_Type](#)
- [Deallocate_Point_List](#)
- [Distance_And_Sigma](#)
- [Get_Euler_From_Fract](#)
- [Get_PhiTheChi](#)
- [Get_Transf_List](#)
- [Init_Err_Geom](#)
- [P1_Dist](#)
- [Print_Distances](#)
- [Set_Orbits_InList](#)
- [Set_TDist_Coordination](#)
- [Set_TDist_Partial_Coordination](#)

Fortran Filename

CFML_Geom_Calc.f90

Variables

- [Coordination_Type](#)
- [Point_List_Type](#)
- [Coord_Info](#)
- [Err_Geom](#)
- [Err_Geom_Mess](#)

Coordination_Type

	Variable	Definition
Type :: Coordination_Type		
Integer	NAtoms	Number of atoms
Integer	Max_Coor	Maximum number of connected atoms to a given one
Integer, Dimension(:), Allocatable	Coord_Num	Counter of distances connected to the current atom
Integer, Dimension(:, :,), Allocatable	N_CooAtm	Pointer to the ordinal number in the list of the attached atom to the atom given by the first index
Integer, Dimension(:, :,), Allocatable	N_Sym	Number of symmetry operator to apply to N_COOATM
Real (Kind=CP), Dimension(:, :,), Allocatable	Dist	List of distances related to an atom
Real (Kind=CP), Dimension(:, :,), Allocatable	S_Dist	List of Sigma(distances)
Real (Kind=CP), Dimension(:, :, :,), Allocatable	Tr_Coo	
End Type Coordination_Type		

Point_List_Type

	Variable	Definition
Type :: Point_List_Type		
Integer	NP	Number of points in list
Character (Len=12), Dimension(:), Allocatable	Nam	Name/label associated to each point
Integer, Dimension(:), Allocatable	P	Integer pointer for various purposes
Real, Dimension(:, :,), Allocatable	X	Fractional coordinates of points
End Type Point_List_Type		

Coord_Info

Type (Coordination_Type) :: Coord_Info

Coordination information

Err_Geom

Logical :: Err_Geom

This variable is set to .TRUE. if an error occurs in procedures belonging to this module.

Err_Geom_Mess

Character (Len=150) :: Err_Geom_Mess

This variable contains information about the last error occurred in the procedures belonging to this module

Functions

- [Angle_Dihedral](#)
- [Angle_Mod](#)
- [Angle_UV](#)
- [Coord_Mod](#)
- [Distance](#)
- [Matrix_PhiTheChi](#)
- [Matrix_RX](#)
- [Matrix_RY](#)
- [Matrix_RZ](#)

Angle_Dihedral

Real Function Angle_Dihedral (U, V, W)

Real(Kind=CP), Dimension(3)	Intent(in)	U	Vector
Real(Kind=CP), Dimension(3)	Intent(in)	V	Vector
Real(Kind=CP), Dimension(3)	Intent(in)	W	Vector

or

Real Function Angle_Dihedral (RI, RJ, RK, RN)

Real(Kind=CP), Dimension(3)	Intent(in)	RI	Vector
Real(Kind=CP), Dimension(3)	Intent(in)	RJ	Vector
Real(Kind=CP), Dimension(3)	Intent(in)	RK	Vector
Real(Kind=CP), Dimension(3)	Intent(in)	RN	Vector

Calculates the dihedral angle between planes U-V and V-W, where vectors U,V,W are given in cartesian components.

Calculates the dihedral angle corresponding to the four points (RI,RJ,RK,RN) given in cartesian components.

The definition used for the dihedral angle is the following:

$$\phi(i, j, k, n) = \alpha \cos \left\{ \frac{(\mathbf{r}_{ij} \times \mathbf{r}_{jk})(\mathbf{r}_{jk} \times \mathbf{r}_{kn})}{|\mathbf{r}_{ij} \times \mathbf{r}_{jk}| |\mathbf{r}_{jk} \times \mathbf{r}_{kn}|} \right\}$$

with this definition the sign of PHI is positive if the vector product

$$(\mathbf{r}_{ij} \times \mathbf{r}_{jk}) \times (\mathbf{r}_{jk} \times \mathbf{r}_{kn})$$

is in the same direction as \mathbf{r}_{jk} , and negative if the direction is opposite.

Angle_Mod

Real Function Angle_Mod (X)

<code>Real(Kind=CP)</code>	<code>Intent(in)</code>	X	Value
----------------------------	-------------------------	---	-------

or

<code>Real(Kind=CP), Dimension(:)</code>	<code>Intent(in)</code>	X	Value
--	-------------------------	---	-------

Calculates the angle $[-\pi, \pi]$

Angle_UV

Real Function Angle_UV (U,V,G)

<code>Integer, Dimension(:)</code>	<code>Intent(in)</code>	U	Vector
<code>Integer, Dimension(:)</code>	<code>Intent(in)</code>	V	Vector
<code>Real(Kind=CP), Dimension(:, :,)</code> , <code>Optional</code>	<code>Intent(in)</code>	G	Metric tensor

or

<code>Real(Kind=CP), Dimension(:)</code>	<code>Intent(in)</code>	U	Vector
<code>Real(Kind=CP), Dimension(:)</code>	<code>Intent(in)</code>	V	Vector
<code>Real(Kind=CP), Dimension(:, :,)</code> , <code>Optional</code>	<code>Intent(in)</code>	G	Vector

Calculates the angle between vectors **U** and **V** given in cartesian components. If **G** is not given cartesian components are assumed.

Coord_Mod

Real Function Coord_Mod (X)

<code>Real(Kind=CP)</code>	<code>Intent(in)</code>	X	Value
----------------------------	-------------------------	---	-------

or

Real(Kind=CP), Dimension(:)	Intent(in)	X	Value
------------------------------------	-------------------	---	-------

Calculates the coordinates between [0,1)

Distance

Real Function Distance (X0, X1, Cell)

Real(Kind=CP), Dimension(3)	Intent(in)	X0	Point
Real(Kind=CP), Dimension(3)	Intent(in)	X1	Point
Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters

or

Real Function Distance (X0, X1, Code)

Real(Kind=CP), Dimension(3)	Intent(in)	X0	Point
Real(Kind=CP), Dimension(3)	Intent(in)	X1	Point
Character(Len=*) , Optional	Intent(in)	Code	Values: C : Cartesian (Default) S : Spherical

Calculate distance between two points.

Matrix_PhiTheChi

Real Function Matrix_PhiTheChi (Phi, Theta, Chi, Code)

Real(Kind=CP)	Intent(in)	Phi	Phi
Real(Kind=CP)	Intent(in)	Theta	Theta
Real(Kind=CP)	Intent(in)	Chi	Chi
Character(Len=*) , Optional	Intent(in)	Code	Values: R : Values are in radians (Default) D : Values are in degrees

Calculate the active rotation matrix (3,3) corresponding to the composition of a rotation around Z of angle Chi, followed by a rotation of angle Theta around the Y-axis and a subsequent rotation of angle Phi around z.

The matrix is $M = Rz(\Phi) \cdot Ry(\Theta) \cdot Rz(\Chi)$

The columns represent the components of the unitary vectors {u,v,w} that may be considered as an alternative orthonormal frame to the canonical {i,j,k}. Applying the matrix M to a point in {i,j,k} gives another point in {i,j,k} obtained by the successive application of the three rotations given above. The transpose (inverse) of the M-matrix, when applied to a point in {i,j,k}, gives the coordinates of the same point referred to the frame {u,v,w}.

Matrix_RX

Real Function Matrix_RX (Ang, Code)

Real(Kind=CP)	Intent(in)	Ang	Angle
----------------------	-------------------	-----	-------

Character(Len=*) , Optional	Intent(in)	CODE	Values: R : Values are in radians (Default) D : Values are in degrees
------------------------------------	-------------------	------	---

Calculate the active rotation matrix (3,3) corresponding to the positive rotation of an angle around the x-axis.

Matrix_RY

Real Function Matrix_RY (Ang, Code)

Real(Kind=CP)	Intent(in)	Ang	Angle
Character(Len=*) , Optional	Intent(in)	Code	Values: R : Values are in radians (Default) D : Values are in degrees

Calculate the active rotation matrix (3,3) corresponding to the positive rotation of an angle around the y-axis.

Matrix_RZ

Real Function Matrix_RZ (Ang, Code)

Real(Kind=CP)	Intent(in)	Ang	Angle
Character(Len=*) , Optional	Intent(in)	Code	Values: R : Values are in radians (Default) D : Values are in degrees

Calculate the active rotation matrix (3,3) corresponding to the positive rotation of an angle around the z-axis.

Subroutines

- [Allocate_Coordination_Type](#)
- [Allocate_Point_List](#)
- [Calc_Dist_Angle](#)
- [Calc_Dist_Angle_Sigma](#)
- [Deallocate_Coordination_Type](#)
- [Deallocate_Point_List](#)
- [Distance_And_Sigma](#)
- [Get_Euler_From_Fract](#)
- [Get_PhiTheChi](#)
- [Get_Transf_List](#)
- [Init_Err_Geom](#)
- [P1_Dist](#)
- [Print_Distances](#)
- [Set_Orbits_InList](#)
- [Set_TDist_Coordination](#)
- [Set_TDist_Partial_Coordination](#)

Allocate_Coordination_Type

Subroutine Allocate_Coordination_Type (NAsu, NumOPs, DMax, Max_Coor)

Integer	Intent(in)	NAsu	Number of atoms in asymmetric unit
Integer	Intent(in)	NumOPs	Number of S.O. excluding lattice centerings
Real(Kind=CP)	Intent(in)	DMax	Maximum distance to be calculated
Integer	Intent(out)	Max_Coor	Maximum coordination allowed

Allocation of variable [Coord_Info](#).**Note:** Should be called before using this module.

Allocate_Point_List

Subroutine Allocate_Point_List (N, PI, Ier)

Integer	Intent(in)	N	Dimension for allocating components
Type(Point_List_Type)	Intent(in,out)	PI	Type with allocatable components
Integer	Intent(out)	Ier	If /= 0 an error occurred

Allocation of an objet of type [Point_List_Type](#)

Calc_Dist_Angle

Subroutine Calc_Dist_Angle (DMax, DAngl, Cell, Spg, A, Lun)

Real(Kind=CP)	Intent(in)	DMax	Max. Distance to calculate
Real(Kind=CP)	Intent(in)	DAngl	Max. distance for angle calculations
Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters
Type(Space_Group_Type)	Intent(in)	Spg	Space group information
Type(Atom_List_Type)	Intent(in)	A	Atoms information
Integer, Optional	Intent(in)	Lun	Logical Unit for writing

Subroutine to calculate distances and angles, below the prescribed distances DMax and DAngl (angles of triplets at distance below DAngl to an atom), without standard deviations. If DAngl=0.0, no angle calculations are done. Writes results in file (unit=Lun) if Lun is present. Control for error is present.

Calc_Dist_Angle_Sigma

Subroutine Calc_Dist_Angle_Sigma (DMax, DAngl, Cell, Spg, A, Lun, Lun_Cons, Lun_CIF)

Real(Kind=CP)	Intent(in)	DMax	Max. Distance to calculate
Real(Kind=CP)	Intent(in)	DAngl	Max. distance for angle calculations
Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters
Type(Space_Group_Type)	Intent(in)	Spg	Space group information
Type(Atom_List_Type)	Intent(in)	A	Atoms information

Integer, Optional	Intent(in)	Lun	Logical Unit for writing
Integer, Optional	Intent(in)	Lun_Con s	Logical unit for writing restraints
Integer, Optional	Intent(in)	Lun_CIF	Logical unit for writing CIF file with distances and angles

Subroutine to calculate distances and angles, below the prescribed distances DMax and DAngl (angles of triplets at distance below DAngl to an atom), without standard deviations. If DAngl=0.0, no angle calculations are done. Writes results in file (unit=Lun) if Lun is present. Control for error is present.

Deallocate_Coordination_Type

Subroutine Deallocate_Coordination_Type ()

Deallocation of variable [Coord_Info](#)

Deallocate_Point_List

Subroutine Deallocate_Point_List (PI)

Type(Point_List_Type)	Intent(in out)	PI	Type with allocatable components
------------------------------	---------------------------	----	----------------------------------

Deallocation of an objet of type [Point_List_Type](#)

Distance_And_Sigma

Subroutine Distance_And_Sigma (CellP, Derm, X0, X1, S0, S1, Dis, S)

Type(Crystal_Cell_Type)	Intent(in)	CellP	Cell parameters
Real(Kind=CP), Dimension(3,3,6)	Intent(in)	Derm	Matrix of derivatives of CELLP% CR_ORTH_CEL
Real(Kind=CP), Dimension(3)	Intent(in)	X0	Point vector
Real(Kind=CP), Dimension(3)	Intent(in)	X1	Point vector
Real(Kind=CP), Dimension(3)	Intent(in)	S0	Sigma of Point Vector
Real(Kind=CP), Dimension(3)	Intent(in)	S1	Sigma of Point Vector
Real(Kind=CP)	Intent(out)	Dis	Distance
Real(Kind=CP)	Intent(out)	S	Sigma of Distance

Calculate de Distance and sigma between two points in fractional coordinates

Get_Euler_From_Fract

Subroutine Get_Euler_From_Fract (X1, X2, X3, MT, Phi, Theta, Chi, EuM, Code)

Real(Kind=CP), Dimension(3)	Intent(in)	X1	Point vector
Real(Kind=CP), Dimension(3)	Intent(in)	x2	Point vector
Real(Kind=CP), Dimension(3)	Intent(in)	X3	Point vector
Real(Kind=CP), Dimension(3,3)	Intent(in)	MT	Matrix transforming to Cartesian coordinates
Real(Kind=CP)	Intent(out)	Phi	Angle PHI

Real(Kind=CP)	Intent(out)	Theta	Angle Theta
Real(Kind=CP)	Intent(out)	Chi	Angle CHI
Real(Kind=CP), Dimension(3,3), Optional	Intent(out)	EuM	
Character(Len=*) , Optional	Intent(in)	Code	

Subroutine to obtain the Euler angles (2nd setting) of a Cartesian frame having as origin the point X3, the z-axis along X1-X3 and the XZ plane coincident with the plane generated by the two vectors (X2-X3,X1-X3).

Get_PhiTheChi

Subroutine Get_PhiTheChi (MT, Phi, Theta, Chi, Code)

Real(Kind=CP), Dimension(3,3)	Intent(in)	MT	Matrix transforming to Cartesian coordinates
Real(Kind=CP)	Intent(out)	Phi	Angle PHI
Real(Kind=CP)	Intent(out)	Theta	Angle Theta
Real(Kind=CP)	Intent(out)	Chi	Angle CHI
Character(Len=*) , Optional	Intent(in)	Code	Values: R : Radians (Default) D : Degrees

Calculate the Euler Angles corresponding to an orthogonal matrix. The definition of the Euler angles in this case correspond to the active rotation matrix obtained from the composition of a rotation around Z of angle Chi, followed by a rotation of angle Theta around the Y-axis and a subsequent rotation of angle Phi around Z.

The matrix is supposed to be of the form: $M = Rz(\Phi).Ry(\Theta).Rz(\chi)$

A checking of the input matrix is given before calculating the angles.

The user must check the logical variable Err_Geom after calling this subroutine. If Err_Geom=.TRUE. it means that the input matrix is not orthogonal.

Get_Transf_List

Subroutine Get_Transf_List (Trans, OX, PL, NPL, IFail)

Real(Kind=CP), Dimension(3,3)	Intent(in)	Trans	Matrix transforming the basis
Real(Kind=CP), Dimension(3)	Intent(in)	OX	Coordinates of origin of the new basis
Type(Point_List_Type)	Intent(in)	PL	Point list
Type(Point_List_Type)	Intent(in out)	NPL	List of transformed points
Integer	Intent(out)	IFail	If /=0 matrix inversion failed

Subroutine to get the fractional coordinates of the points of the input list PL in the new transformed cell (a'= trans a) displaced to the new origin OX. The coordinates are generated using only lattice translations. All coordinates are reduced to be between 0.0 and 1.0, so that $0.0 \leq x,y,z < 1.0$

Init_Err_Geom

Subroutine Init_Err_Geom ()

Subroutine that initializes errors flags in **CFML_Geometry_Calc** module.

P1_Dist**Subroutine P1_Dist (DMax, Cell, Spg, Ac, Lun)**

Real(Kind=CP)	Intent(in)	DMax	Max. Distance to calculate
Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters
Type(Space_Group_Type)	Intent(in)	Spg	Space group information
Type(Atoms_Cell_Type)	Intent(in out)	Ac	Atoms information
Integer, Optional	Intent(in)	Lun	Logical Unit for writing

Subroutine calculate distances, below the prescribed distances DMAX, without standard deviations. No symmetry is applied: only lattice translations.

Print_Distances**Subroutine Print_Distances (Lun, DMax, Cell, Spg, A)**

Integer	Intent(in)	Lun	Logical Unit for writing
Real(Kind=CP)	Intent(in)	DMax	Max. Distance to calculate
Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters
Type(Space_Group_Type)	Intent(in)	Spg	Space group information
Type(Atom_List_Type)	Intent(in)	A	Atoms information

Subroutine to print distances, below the prescribed distances DMax, without standard deviations.

Set_Orbits_InList**Subroutine Set_Orbits_InList (Spg, PL)**

Type(Space_Group_Type)	Intent(in)	Spg	Space group
Type(Point_List_Type)	Intent(in out)	PL	Point list

Set up of the Integer pointer PL%P in the object **PL** of type [Point_List_Type](#). Each point is associated with the number of an orbit. This pointer is useful to get the asymmetric unit with respect to the input space group of an arbitrary list of points (atom coordinates).

Set_TDist_Coordination**Subroutine Set_TDist_Coordination (Max_Coor, DMax, Cell, Spg, A)**

Integer	Intent(in)	Max_Co or	Maximum expected coordination
Real(Kind=CP)	Intent(in)	DMax	Max. Distance to calculate
Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters
Type(Space_Group_Type)	Intent(in)	Spg	Space group information
Type(Atom_List_Type)	Intent(in)	A	Atoms information

Subroutine to calculate distances, below the prescribed distance **DMax**. Sets up the coordination type:

[Coord_Info](#) for each atom in the asymmetric unit

The input argument **Max_Coor** is obtained, before calling the present procedure, by a call to [Allocate_Coordination_Type](#) with arguments:(A%NAtoms,SPG%Multip,DMax,Max_Coor)

Further calls to this routine do not need a previous call to [Allocate_Coordination_Type](#).

Set_TDist_Partial_Coordination

Subroutine Set_TDist_Partial_Coordination (List, Max_Coor, DMax, Cell, Spg, A)

Integer	Intent(in)	List	Modified atom
Integer	Intent(in)	Max_Coor	Maximum expected coordination
Real(Kind=CP)	Intent(in)	DMax	Max. Distance to calculate
Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters
Type(Space_Group_Type)	Intent(in)	Spg	Space group information
Type(Atom_List_Type)	Intent(in)	A	Atoms information

Modify the coordination type: Coord_Info for the atoms affected by the change of atom "List"

This routine is a modification of Set_TDist_Coordination to avoid superfluous calculations in global optimization methods. It assumes that Set_TDist_Coordination has previously been called and the object Coord_Info has already been set.

CFML_Propagation_Vectors

Series of procedures handling operation with Propagation vectors

Variables

- [Group_K_Type](#)

Functions

- [HK_Equiv](#)
- [K_Equiv](#)
- [K_Equiv_Minus_K](#)

Subroutines

- [K_Star](#)
- [Write_Group_K](#)

Fortran Filename

CFML_Propagk.f90

Variables

- [Group_K_Type](#)

Group_K_Type

	Variable	Definition
--	-----------------	-------------------

Type :: Group_K_Type

Type(Space_Group_Type)	G0	initial Space group
Integer	NGK	Number of elements of G_k
Logical	K_Equiv_Minus_K	TRUE if k equiv -k
Integer, Dimension(192)	P	Pointer to operations of G0 that changes/fix k
Integer, Dimension(48,48)	CO	
Integer	NK	Number of star arms
Real(Kind=CP), Dimension(3,24)	StartK	Star of the wave vector k

End Type Group_K_Type

The Integer pointer P is used as follows:

If we defined the object G as -> Group_K_Type

- G%P(1:NGK) gives the numeral of the symmetry operators of G%G0 belonging to G_k.
- G%P(192:193-NK) gives the numeral of the the symmetry operators of G%G0 that transform the initial k-vector to the other arms of the star.
- G%CO(:,KK) gives also the numerals of the the symmetry operators of G%G0 that transform the initial k-vector to the arm kk of the star to the representative of the coset decomposition of G%G0 with respect to G_k.

Functions

- [HK_Equiv](#)
- [K_Equiv](#)
- [K_Equiv_Minus_K](#)

HK_Equiv

Logical Function HK_Equiv (H, K, SpaceGK, Friedel)

Real(Kind=CP), Dimension(3)	Intent (in)	H	
Real(Kind=CP), Dimension(3)	Intent (in)	K	
Type(Group_K_Type)	Intent (in)	SpaceGK	
Logical, Optional	Intent (in)	Friedel	

Calculate if two real reflections are equivalent

K_Equiv

Logical Function K_Equiv (H, K, LatTyp)

Real(Kind=CP), Dimension(3)	Intent (in)	H	
Real(Kind=CP), Dimension(3)	Intent (in)	K	
Character (Len=*)	Intent (in)	LatTyp	

Calculate if two k-vectors are equivalent in the sense that **H** is equivalent to **K** if **H-K** belongs to the reciprocal lattice.

Only lattice type is needed.

K_Equiv_Minus_K

Logical Function K_Equiv_Minus_K (Vec, Lat)

Real(Kind=CP), Dimension(3)	Intent (in)	Vec	
Character (Len=*)	Intent (in)	Lat	

Determine whether a k-vector is equivalent to -k

Subroutines

- [K_Star](#)
- [Write_Group_K](#)

K_Star

Subroutine K_Star (K, SpaceGroup, GK)

Integer, Dimension(3)	Intent (in)	K	
Type(Space_Group_Type)	Intent (in)	SpaceGroup	
Type(Group_K_Type)	Intent (in)	GK	

Calculate the star of the propagation vector and the group of the vector k.

Write_Group_K

Subroutine Write_Group_K (GK, Lun)

Type(Group_K_Type)	Intent	GK	
--------------------	--------	----	--

Integer, Optional	(in) Intent (in)	Lun	Logical unit write
--------------------------	-------------------------------	-----	--------------------

Subroutine to write the operators of the propagation vector group and the list of all vectors {k}, belonging to the star of k.

CFML_Structure_Factors

Main module for Structure Factors Calculations

Variables

- [Err_SFac](#)
- [Err_SFac_Mess](#)

Subroutines

- [Calc_HKL_StrFactor](#)
- [Calc_StrFactor](#)
- [Init_Calc_StrFactors](#)
- [Init_Calc_HKL_StrFactors](#)
- [Init_Structure_Factors](#)
- [Modify_SF](#)
- [Structure_Factors](#)
- [Write_Structure_Factors](#)

Fortran Filename

CFML_Sfac.f90

Variables

- [Err_SFac](#)
- [Err_SFac_Mess](#)

Err_SFac

Logical :: Err_SFac

This variable is set to .TRUE. if an error occurs in procedures belonging to this module

Err_SFac_Mess

Character (Len=150) :: Err_SFac_Mess

This variable contains information about the last error occurred in the procedures belonging to this module

Subroutines

- [Calc_HKL_StrFactor](#)
- [Calc_StrFactor](#)
- [Init_Calc_StrFactors](#)
- [Init_Calc_HKL_StrFactors](#)
- [Init_Structure_Factors](#)
- [Modify_SF](#)
- [Structure_Factors](#)
- [Write_Structure_Factors](#)

Calc_HKL_StrFactor

Subroutine Calc_HKL_StrFactor (Mode, Rad, HN, SN, Atm, Grp, SF2, Deriv, Fc)

Character(Len=*)	Intent(in)	Mode	Values: S : SXTAL P : Powder
Character(Len=*)	Intent(in)	Rad	Radiation: X-rays, Neutrons
Integer	Intent(in)	HN	Reflection H
Real(Kind=CP)	Intent(in)	SN	$(\sin\theta/\lambda)^2$
Type(Atom_List_Type)	Intent(in)	Atm	Atoms information
Type(Space_Group_Type)	Intent(in)	Grp	Space group information
Real(Kind=CP)	Intent(out)	SF2	
Real(Kind=CP), Dimension(:), Optional	Intent(out)	Deriv	
Complex, Optional	Intent(out)	Fc	

Calculate Structure Factor for reflection HN=(hkl) not related with previous lists and derivatives with respect to refined parameters.

This subroutine calculates the form-factors internally without using global tables. The purpose of this procedure is to avoid the use of too much memory in tables.

Calc_StrFactor

Subroutine Calc_StrFactor (Mode, Rad, NN, SN, Atm, Grp, SF2, Deriv, Fc)

Character(Len=*)	Intent(in)	Mode	Values: S : SXTAL P : Powder
Character(Len=*)	Intent(in)	Rad	Radiation: X-rays, Neutrons
Integer	Intent(in)	NN	
Real(Kind=CP)	Intent(in)	SN	$(\sin\theta/\lambda)^2$
Type(Atom_List_Type)	Intent(in)	Atm	Atoms information
Type(Space_Group_Type)	Intent(in)	Grp	Space group information
Real(Kind=CP)	Intent(out)	SF2	
Real(Kind=CP), Dimension(:), Optional	Intent(out)	Deriv	

Complex, Optional	Intent(out)	Fc	
--------------------------	--------------------	----	--

Calculate Structure Factor for reflection NN in the list and derivatives with respect to refined parameters

Init_Calc_StrFactors

Subroutine Init_Calc_StrFactors (Reflex, Atm, Grp, Mode, Lambda, Lun)

Type(Reflection_List_Type)	Intent(in)	Reflex	Reflection information
Type(Atom_List_Type)	Intent(in)	Atm	Atoms information
Type(Space_Group_Type)	Intent(in)	Grp	Space group information
Character(Len=*), Optional	Intent(in)	Mode	Value: NUC : For Nuclear reflections Rest: X-Ray reflections
Real(Kind=CP), Optional	Intent(in)	Lambda	Wavelength
Integer, Optional	Intent(in)	Lun	Logical unit for writing scatt-factors

Allocates and initializes arrays for [Calc_StrFactor](#) calculations.

Calculations of fixed tables are performed. Should be called before using the subroutine [Calc_StrFactor](#)

Init_Calc_HKL_StrFactors

Subroutine Init_Calc_HKL_StrFactors (Atm, Mode, Lambda, Lun)

Type(Atom_List_Type)	Intent(in)	Atm	Atoms information
Character(Len=*), Optional	Intent(in)	Mode	Value: NUC : For Nuclear reflections Rest: X-Ray reflections
Real(Kind=CP), Optional	Intent(in)	Lambda	Wavelength
Integer, Optional	Intent(in)	Lun	Logical unit for writing scatt-factors

Allocates and initializes arrays for hkl - Structure Factors calculations.

No calculation of fixed tables is performed. Should be called before using the subroutine [Calc_HKL_StrFactor](#)

Init_Structure_Factors

Subroutine Init_Structure_Factors (Reflex, Atm, Grp, Mode, Lambda, Lun)

Type(Reflection_List_Type)	Intent(in out)	Reflex	Reflection information
Type(Atom_List_Type)	Intent(in)	Atm	Atoms information
Type(Space_Group_Type)	Intent(in)	Grp	Space group information
Character(Len=*), Optional	Intent(in)	Mode	Value: NUC : For Nuclear reflections Rest: X-Ray reflections
Real(Kind=CP), Optional	Intent(in)	Lambda	Wavelength
Integer, Optional	Intent(in)	Lun	Logical unit write

Allocates and initializes arrays for Structure Factors calculations. A calculation of fixed tables is also performed.

Modify_SF

Subroutine Modify_SF (Reflex, Atm, Grp, List, NList, Mode)

Type(Reflection_List_Type)	Intent(in out)	Reflex	Reflection information
Type(Atom_List_Type)	Intent(in)	Atm	Atoms information
Type(Space_Group_Type)	Intent(in)	Grp	Space group information
Integer, Dimension(:)	Intent(in)	List	
Integer	Intent(in)	NList	
Character(Len=*), Optional	Intent(in)	Mode	Value: NUC : For Nuclear reflections Rest: X-Ray reflections

Recalculation of Structure Factors because a list of Atoms parameters were modified.

List variable contains the number of atoms to be changed.

Structure_Factors

Subroutine Structure_Factors (Atm, Grp, Reflex, Mode, Lambda)

Type(Atom_List_Type)	Intent(in)	Atm	Atoms information
Type(Space_Group_Type)	Intent(in)	Grp	Space group information
Type(REFLECTION_LIST_TYPE)	Intent(in out)	Reflex	Reflection information
Character(Len=*), Optional	Intent(in)	Mode	Value: NUC : For Nuclear reflections Rest: X-Ray reflections
Real(Kind=CP), Optional	Intent(in)	Lambda	Wavelength

Calculate the Structure Factors from a list of Atoms and a set of reflections.

A call to [Init_Structure_Factors](#) is a pre-requisite for using this subroutine.

Write_Structure_Factors

Subroutine Write_Structure_Factors (Lun, Reflex, Mode)

Integer	Intent (in)	Lun	Logical unit write
Type(Reflection_List_Type)	Intent (in)	Reflex	Reflection list
Character(Len=*), Optional	Intent (in)	Mode	Value: NUC : For Nuclear reflections Rest: X-Ray reflections

Writes in logical unit=Lun the list of structure factors

Level 6

Concept	Module Name	Purpose
Configurations...	CFML_BVS_Energy_Calc	Procedures related to calculations of energy or configuration properties depending on the crystal structure: BVS, Energy,....
Maps...	CFML_Maps_Calculations	Procedures related to operations on arrays describing maps
Molecular...	CFML_Molecular_Crystals	Types and procedures related to molecules in crystals

[CFML_BVS_Energy_Calc](#)

Module containing procedures related to calculations of Energy or Configuration properties depending on the crystal structure: BVS, Energy,....

Parameters

- [BVS_Anions](#)
- [BVS_Anions_N](#)
- [BVS_Anions_Rlon](#)
- [BVS_Species_N](#)

Variables

- [Atoms_Conf_List_Type](#)
- [BVS_Par_Type](#)
- [BVS_Table](#)
- [Err_Conf](#)
- [Err_Conf_Mess](#)

Subroutines

- [Allocate_Atoms_Conf_List](#)
- [Calc_BVS](#)
- [Calc_Map_BVS](#)
- [Cost_BVS](#)
- [Cost_BVS_CoulombRep](#)

- [Deallocate Atoms Conf List](#)
- [Deallocate BVS Table](#)
- [Init Err Conf](#)
- [Set BVS Table](#)
- [Set Table D0 B](#)
- [Species on List](#)

Fortran Filename

CFML_Conf_Calc.f90

Parameters

- [BVS_Anions](#)
- [BVS_Anions_N](#)
- [BVS_Anions_RIon](#)
- [BVS_Species_N](#)

BVS_Anions

Character (Len=*), **Dimension(BVS_Anions_N) :: BVS_Anions**

Anions tabulated in Bond Valence parameters from O'Keefe, Bresse, Brown

Values are:

Order	Anion
1	O-2
2	F-1
3	CL-1
4	BR-1
5	I-1
6	S-2
7	SE-2
8	TE-2
9	N-3
10	P-3
11	AS-3
12	H-1
13	O-1
14	SE-1

BVS_Anions_N

Integer :: BVS_Anions_N=14

Number of anions tabulated in BV Tables by O'Keefe, Breese, Brown

BVS_Anions_RIon

Real, Dimension(BVS_Anions_N) :: BVS_Anions_Rlon

Ionic radii for anions in Bond Valence parameters table

Values are:

Order	Value
1	1.40
2	1.19
3	1.67
4	1.95
5	2.16
6	1.84
7	1.98
8	2.21
9	1.71
10	2.12
11	2.22
12	2.08
13	1.35
14	1.80

BVS_Species_N

Integer :: BVS_Species_N=247

Maximum number of species in BVS_Table

Variables

- Atoms_Conf_List_Type
- BVS_Par_Type
- BVS_Table
- Err_Conf
- Err_Conf_Mess

Atoms_Conf_List_Type

	Variable	Definition
Type :: Atoms_Conf_List_Type		
Integer	NAtoms	Total number of atoms in the list
Integer	N_Spec	Number of different species in the list
Integer	N_Anions	Number of anions in the list
Integer	N_Cations	Number of cations in the list

Real (Kind=CP)	Tol	Tolerance(%) for sum of radii conditions
Real (Kind=CP)	TotAtoms	Total number of atoms in the unit cell
Character (Len=4), Dimension(:), Allocatable	Species	Symbol + valence
Real (Kind=CP), Dimension(:), Allocatable	Radius	ionic/atomic radius of species
Type (Atom_Type), Dimension(:), Allocatable	Atom	Atom information
End Type		
Atoms_Conf_List_Type		

BVS_Par_Type

	Variable	Definition
Type :: BVS_Par_Type		
Character (Len=4)	Symb	Chemical symbol
Real (Kind=CP), Dimension (BVS_Anions_N)	D0	D0 Parameter
Real (Kind=CP), Dimension (BVS_Anions_N)	B_Par	B Parameter
Integer, Dimension (BVS_Anions_N)	RefNum	Integer pointing to the reference paper
End Type BVS_Par_Type		

BVS_Table

Type (BVS_Par_Type), Dimension(:), Allocatable :: BVS_Table

Global variable containing BVS parameters for calculations. The dimension is defined for the parameter [BVS_Species_N](#)

Err_Conf

Logical :: Err_Conf

This variable is set to .TRUE. if an error occurs in procedures belonging to this module.

Err_Conf_Mess

Character (Len=150) :: Err_Conf_Mess

This variable contains information about the last error occurred in the procedures belonging to this module

Subroutines

- [Allocate_Atoms_Conf_List](#)
- [Calc_BVS](#)
- [Calc_Map_BVS](#)
- [Cost_BVS](#)
- [Cost_BVS_CoulombRep](#)
- [Deallocate_Atoms_Conf_List](#)
- [Deallocate_BVS_Table](#)
- [Init_Err_Conf](#)
- [Set_BVS_Table](#)
- [Set_Table_D0_B](#)
- [Species_on_List](#)

[Allocate_Atoms_Conf_List](#)

Subroutine Allocate_Atoms_Conf_List (N, A)

Integer	Intent(in)	N	Atoms in asymmetric unit
Type(Atoms_Conf_List_Type)	Intent(in out)	A	Objet to be allocated

Allocation of objet A of type [Atoms_Conf_List_Type](#). This subroutine should be called before using an object of type [Atoms_Conf_List_Type](#).

[Calc_BVS](#)

Subroutine Calc_BVS (A, IPr, N_BVSM, BVS_M, FileCod)

Type(Atoms_Conf_List_Type)	Intent(in)	A	Atoms information
Integer, Optional	Intent(in)	IPr	Logical unit write
Integer, Optional	Intent(in)	N_BVSM	Number of modifications
Character (Len=*) , Dimension(:) , Optional	Intent(in)	BVS_M	Text with BVS parameters
Character (Len=*) , Optional	Intent(in)	FileCod	

Subroutine to calculate Bond-Valence sums.

Before calling this subroutine it is the responsibility of the calling program to make a previous call to [Calc_Dist_Angles_Sigma](#) in order to update the internal private variables related to distance/angle calculations.

[Calc_Map_BVS](#)

Subroutine Calc_Map_BVS (A, Spg, Cell, FileCod, NDimX, NDimY, NDimZ, AtName, DRMax)

Type(Atoms_Conf_List_Type)	Intent(in)	A	Atoms information
Type(Space_Group_Type)	Intent(in)	Spg	Space group
Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters
Character (Len=*)	Intent(in)	FileCod	
Integer	Intent(in)	NDimX	Dimension in x-axis for the BVS map
Integer	Intent(in)	NDimY	Dimension in y-axis for the BVS map

Integer	Intent(in)	NDimZ	Dimension in z-axis for the BVS map
Character (Len=*)	Intent(in)	AtName	
Real(Kind=CP)	Intent(in)	DRMax	

Calculate a map of BVS values where each point of the grid is determined by a specie representative defined in **AtName**.

The BVS value is evaluated into of **DRMax** value. The BVS map is saved in the file called given as FileCod.

Cost_BVS

Subroutine Cost_BVS (A, GII, GIC)

Type(Atoms_Conf_List_Type)	Intent(in)	A	Atoms information
Real(Kind=CP)	Intent(out)	GII	Global instability index
Character (Len=*) , Optional	Intent(in)	GIC	If present GII_c is put in GII

Subroutine to calculate the Global instability index.

Before calling this subroutine it is the responsibility of the calling program to make a previous call to [Set_TDist_Coordination](#) in order to update the internal private variables related to distance/angle calculations.

All items corresponding to the bond-valence parameters contained in A have to be properly set before calling this procedure.

Cost_BVS_CoulombRep

Subroutine Cost_BVS_CoulombRep (A, GII, ERep)

Type(Atoms_Conf_List_Type)	Intent(in)	A	Atoms information
Real(Kind=CP)	Intent(out)	GII	Global instability index
Real(Kind=CP)	Intent(out)	ERep	Pseudo Repulsion Coulomb "energy"

Subroutine to calculate the Global instability index GII_a and a pseudo Coulomb repulsion energy useful to avoid cation-cation and anion-anion overlap when using this cost function for predicting or solving a ionic crystal structure. It was used in the old program PiXSA, by J. Pannetier, J. Bassas-Alsina, J.Rodriguez-Carvajal and V. Caaignaert, in "Prediction of Crystal Structures from Crystal Chemistry Rules by Simulated Annealing", Nature 346, 343-345 (1990).

Before calling this subroutine it is the responsibility of the calling program to make a previous call to [Set_TDist_Coordination](#) in order to update the internal [Coord_Info](#) variable related to distance and angle calculations.

Deallocate_Atoms_Conf_List

Subroutine Deallocate_Atoms_Conf_List (A)

Type(Atoms_Conf_List_Type)	Intent(in out)	A	Objet to be allocated
-----------------------------------	---------------------------	---	-----------------------

De-allocation of objet A of type [Atoms_Conf_List_Type](#). This subroutine should be after using an object of type [Atoms_Conf_List_Type](#) that is no more needed.

Deallocate_BVS_Table

Subroutine Deallocate_BVS_Table ()

Deallocating [BVS_Table](#)

[Init_Err_Conf](#)

Subroutine Init_Err_Conf ()

Subroutine that initializes errors flags in **CFML_BVS_Energy_Calc** module.

[Set_BVS_Table](#)

Subroutine Set_BVS_Table ()

Fills the parameters for BVS from O'Keefe, Bresse, Brown in the [BVS_Table](#) variable

[Set_Table_D0_B](#)

Subroutine Set_Table_D0_B (A, N_BVSM, BVS_M)

Type(Atoms_Conf_List_Type)	Intent(in)	A	Atoms information
Integer, Optional	Intent(in)	N_BVSM	Number of bvs strings with externally provided values
Character (Len=*), Dimension(:), Optional	Intent(in)	BVS_M	Text with BVS parameters

Set external values for D0 and B in BVS calculations

[Species_on_List](#)

Subroutine Species_on_List (A, Mulg, Tol)

Type(Atoms_Conf_List_Type)	Intent(in)	A	Atoms information
Integer, Optional	Intent(in)	Mulg	
Real(Kind=CP), Optional	Intent(in)	Tol	

Determines the different species in the List and, optionally, sets the tolerance factor for ionic radii conditions and provides "corrected" occupation factors (mult/Mulg) when the user is using a multiplier. The general multiplicity of the space group MulG must be provided in such a case. This first free variable of the Atom-type A%ATOMVFREE(1) is set to the corrected occupation. The first atom in the list must completely occupy its site.

CFML_Maps_Calculations

Subroutines related to operations on the array's map

Parameters

- [Max_Points](#)

Variables

- [Cube_Info_Type](#)
- [Cube_Info](#)
- [Err_Maps](#)
- [Err_Maps_Mess](#)

Functions

- [Index_Cube](#)
- [Vertice_Point](#)
- [Vertices_Cube](#)
- [VPoint_in_Cube](#)
- [VPoint_in_Line](#)
- [VPoint_in_Square](#)

Subroutines

- [Calculate_Contour2D](#)
- [Calculate_Mesh](#)
- [Init_Err_Maps](#)
- [Load_ExtendedMap](#)
- [Load_Section](#)
- [Search_Peaks](#)
- [Set_Cube_Info](#)
- [Statistic_Map](#)

Fortran Filename

CFML_Maps.f90

Parameters

- [Max_Points](#)

Max_Points

Integer, Parameter :: Max_Points = 150000

Number of maximum points permitted

Variables

- [Cube_Info_Type](#)
- [Cube_Info](#)

- [Err_Maps](#)
- [Err_Maps_Mess](#)

Cube_Info_Type

	Variable	Definition
Type :: Cube_Info_Type		
Integer	NElem	Number of Elements
Integer	Code	Code of Elements
Integer, Dimension(12)	EdgesS	Code for Edge connections
End Type Cube_Info_Type		

Cube_Info

Type(Cube_Info_Type), Dimension(0:255) :: Cube_Info

Information of Mesh in a cube

Err_Maps

Logical :: Err_Maps

This variable is set to .TRUE. if an error occurs in procedures belonging to this module.

Err_Maps_Mess

Character (Len=150) :: Err_Maps_Mess

This variable contains information about the last error occurred in the procedures belonging to this module.

Functions

- [Index_Cube](#)
- [Vertice_Point](#)
- [Vertices_Cube](#)
- [VPoint_in_Cube](#)
- [VPoint_in_Line](#)
- [VPoint_in_Square](#)

Index_Cube

Integer Function Index_Cube (IV, MC)

Integer, Dimension(8)	Intent(in)	IV	Vertices state On/Off
Logical	Intent(in)	MC	If .TRUE. Code for Triangles (128-255), if

			not give code from 0-127
--	--	--	--------------------------

Return the index for Marching cubes algorithm

Vertice_Point

Real Function Vertice_Point (Code_Edge)

Integer	Intent(in)	Code_Edg e	
----------------	-------------------	---------------	--

or

Real Function Vertice_Point (Code_Edge, D0, D1, D2, D3, D4, D5, D6, D7, D8, D9)

Integer	Intent(in)	Code_Edg e	
Real(Kind=CP)	Intent(in)	D0	
Real(Kind=CP)	Intent(in)	D1	
Real(Kind=CP)	Intent(in)	D2	
Real(Kind=CP)	Intent(in)	D3	
Real(Kind=CP)	Intent(in)	D4	
Real(Kind=CP)	Intent(in)	D5	
Real(Kind=CP)	Intent(in)	D6	
Real(Kind=CP)	Intent(in)	D7	
Real(Kind=CP)	Intent(in)	D8	
Real(Kind=CP)	Intent(in)	D9	

Return the relative position point from (i,j,k) of V1

Given a binary dataset, linear interpolation is not needed to extract isosurfaces. When a cell edge in a binary dataset has both on and off corners, the midpoint of the edge is the intersection being looked for.

Vertices_Cube

Integer Function Vertices_Cube (Index_Cube)

Integer	Intent(in)	Index_Cub e	index
----------------	-------------------	----------------	-------

Return the state of the 8 vertices of the cube in Marching cubes algorithm

VPoint_in_Cube

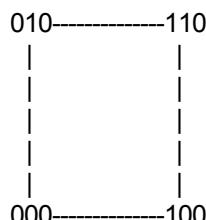
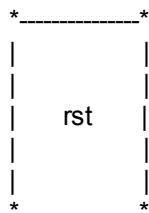
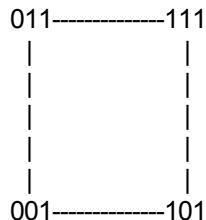
Real Function VPoint_in_Cube (R, S, T, X000, X001, X010, X011, X100, X101, X110, X111)

Real(Kind=CP)	Intent(in)	R	
Real(Kind=CP)	Intent(in)	S	
Real(Kind=CP)	Intent(in)	T	
Real(Kind=CP)	Intent(in)	X000	Value of the Point 000
Real(Kind=CP)	Intent(in)	X001	Value of the Point 001

Real(Kind=CP)	Intent(in)	X010	Value of the Point 010
Real(Kind=CP)	Intent(in)	X011	Value of the Point 011
Real(Kind=CP)	Intent(in)	X100	Value of the Point 100
Real(Kind=CP)	Intent(in)	X101	Value of the Point 101
Real(Kind=CP)	Intent(in)	X110	Value of the Point 110
Real(Kind=CP)	Intent(in)	X111	Value of the Point 111

Function that interpolate the value into a cube

Diagram:



VPoint_in_Line

Real Function VPoint_in_Line (R,X0,X1)

Real(Kind=CP)	Intent(in)	R	R is distance between the ends points
Real(Kind=CP)	Intent(in)	X0	Value of the Point 0
Real(Kind=CP)	Intent(in)	X1	Value of the Point 1

Function that interpolate the value

Diagram: 0---r---1

VPoint_in_Square

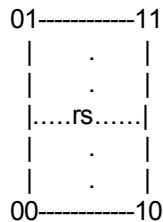
Real Function VPoint_in_Square (R, S, X00, X01, X10, X11)

Real(Kind=CP)	Intent(in)	R	R is distance between the ends points
----------------------	-------------------	---	---------------------------------------

Real(Kind=CP)	Intent(in)	S	
Real(Kind=CP)	Intent(in)	X00	Value of the Point 00
Real(Kind=CP)	Intent(in)	X01	Value of the Point 01
Real(Kind=CP)	Intent(in)	X10	Value of the Point 10
Real(Kind=CP)	Intent(in)	X11	Value of the Point 11

Function that interpolate the value on square

Diagram:



Subroutines

- [Calculate Contour2D](#)
 - [Calculate Mesh](#)
 - [Init Err Maps](#)
 - [Load ExtendedMap](#)
 - [Load Section](#)
 - [Search Peaks](#)
 - [Set Cube Info](#)
 - [Statistic Map](#)

Calculate_Condition2D

Subroutine Calculate_Condition2D(D, ILB, IUB, JLB, JUB, X, Y, Z, NLV, NTP, XYZ)

Real(Kind=CP), Dimension (ILB:IUB, JLB:JUB)	Intent(in)	D	Section 2D
Integer	Intent(in)	ILB	Lower limit on the first dimension
Integer	Intent(in)	IUB	Upper limit on the first dimension
Integer	Intent(in)	JLB	Lower limit on the second dimension
Integer	Intent(in)	JUB	Upper limit on the second dimension
Real(Kind=CP), Dimension (ILB:IUB)	Intent(in)	X	Limits values on X
Real(Kind=CP), Dimension (JLB:JUB)	Intent(in)	Y	Limits values on Y
Real(Kind=CP), Dimension(:)	Intent(in)	Z	Levels values
Integer	Intent(in)	NLV	Number of levels
Integer	Intent(in out)	NTP	Number of points
Real(Kind=CP), Dimension(:,:,)	Intent(out)	XYZ	XY Points

Calculate the Contour 2D of a section

Calculate_Mesh

Subroutine Calculate_Mesh (Rho, NGrid, NLevel, Levels, MC_Method, NPoints, XYZ, Limits, Step)

Real(Kind=CP), Dimension(:,:,:)	Intent(in)	Rho	Array
Integer, Dimension(3)	Intent(in)	NGrid	Grid dimensions od RHO
Integer	Intent(in)	NLevel	Number of levels
Real(Kind=CP), Dimension(NLevel)	Intent(in)	Levels	Levels values
Character(Len=*)	Intent(in)	MC_Method	Values: TR : Mesh using Triangles Other : Rectangle and triangles
Integer, Dimension(NLevel)	Intent(out)	NPoints	Number of points
Real(Kind=CP), Dimension(:, :, :)	Intent(out)	XYZ	Points
Real(Kind=CP), Dimension(2,3), Optional	Intent(in)	Limits	Limits
Integer, Dimension(3), Optional	Intent(in)	Step	Step to do calculations

Calculate the 3D Contour

Init_Err_Maps

Subroutine Init_Err_Maps ()

Subroutine that initializes errors flags in **CFML_Maps_Calculations** module.

Load_ExtendedMap

Subroutine Load_ExtendedMap (Rho, NGrid, Limits, RhoNew)

Real(Kind=CP), Dimension(:,:,:)	Intent(in)	Rho	Array
Integer, Dimension(3)	Intent(in)	NGrid	Grid dimensions of RHO
Real(Kind=CP), Dimension(2,3)	Intent(in)	Limits	Limits
Real(Kind=CP), Dimension(:, :, :)	Intent(out)	RhoNew	RHO Extended

RhoNew has one dimension more in each dimension than Rho. This routine is useful for 2D representation.

RHO(NX,NY,NZ) -> RHONEW(NX+1,NY+1,NZ+1)

Load_Section

Subroutine Load_Section (Rho, NGrid, IMap, Section, Limits, NGrid2, DMap)

Real(Kind=CP), Dimension(:,:,:)	Intent(in)	Rho	Array
Integer, Dimension(3)	Intent(in)	NGrid	Grid dimensions of RHO
Integer	Intent(in)	IMap	
Integer	Intent(in)	Section	
Real(Kind=CP), Dimension(2,2)	Intent(in)	Limits	Limits
Real(Kind=CP), Dimension(2)	Intent(in)	NGrid2	
Real(Kind=CP), Dimension(:, :, :)	Intent(out)	DMap	Section 2D

Load a particular section of a map according to the new limits. This routine only works with fractional coordinates

Search_Peaks

Subroutine Search_Peaks (Rho, Grp, Cell, NPFound, Peaks, ABS_Code)

Real(Kind=CP), Dimension(:,:,:)	Intent(in)	Rho	Array
Type(Space_Group_Type)	Intent(in)	Grp	SpaceGroup
Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters
Integer	Intent(in out)	NPFound	Number of Peaks to found
Real(Kind=CP), Dimension (4,NPFound)	Intent(out)	Peaks	Peak List
Logical, Optional	Intent(in)	ABS_Code	logical to use absolute value on RHO

General procedure to search peaks on Rho

Set_Cube_Info

Subroutine Set_Cube_Info ()

Set values for [Cube_Info](#) Variable.

- From 0 to 127 the code is defined according the next table.

Code	Figure	Process
1	Triangle	Pto1 -> Pto2 -> Pto3 -> Pto1
2	Trapezoide	Pto1 -> Pto2 -> Pto3 -> Pto4 -> Pto1
3	Triangle + Trapezoide	Pto1 -> Pto2 -> Pto3 -> Pto1 Pto4 -> Pto5 -> Pto6 -> Pto7 -> Pto4
4	Triangle + Triangle + Trapezoide	Pto1 -> Pto2 -> Pto3 -> Pto1 Pto4 -> Pto5 -> Pto6 -> Pto4 Pto7 -> Pto8 -> Pto9 -> Pto10 -> Pto7
5	Triangle + Line	Pto1 -> Pto2 -> Pto3 -> Pto1 Pto1 -> Pto4
6	Triangle + Triangle + Line	Pto1 -> Pto2 -> Pto3 -> Pto1 Pto4 -> Pto5 -> Pto6 -> Pto4 Pto4 -> Pto7

- From 128 to 255 all is defined using triangles.

Statistic_Map

Subroutine Statistic_Map (Rho, MaxV, MinV, AveV, SigmaV)

Real(Kind=CP), Dimension(:,:,:)	Intent(in)	Rho	Array
Real(Kind=CP)	Intent(out)	MaxV	Maximum value of Rho
Real(Kind=CP)	Intent(out)	MinV	Minimum value of Rho
Real(Kind=CP)	Intent(out)	AveV	Average value of Rho
Real(Kind=CP)	Intent(out)	SigmaV	Sigma value of Rho

Some statistic parameters of the map

CFML_Molecular_Crystals

Module to define molecules on Crystals

Variables

- [Molecule_Type](#)
- [Molecular_Crystal_Type](#)
- [Err_Molec](#)
- [Err_Molec_Mess](#)

Subroutines

- [Cartesian_To_Fractional](#)
- [Cartesian_To_Spherical](#)
- [Cartesian_To_ZMatrix](#)
- [Empiric_Formula](#)
- [Fix_Orient_Cartesian](#)
- [Fix_Reference](#)
- [Fractional_To_Cartesian](#)
- [Fractional_To_Spherical](#)
- [Fractional_To_ZMatrix](#)
- [Init_Err_Molec](#)
- [Init_Molecule](#)
- [MolCrys_To_AtomList](#)
- [Molec_To_AtomList](#)
- [Read_Free_Atoms](#)
- [Read_Molecule](#)
- [Set_Euler_Matrix](#)
- [Spherical_To_Cartesian](#)
- [Spherical_To_Fractional](#)
- [Spherical_To_ZMatrix](#)
- [Write_Free_Atoms](#)
- [Write_Molecular_Crystal](#)
- [Write_Molecule](#)
- [ZMatrix_To_Cartesian](#)
- [ZMatrix_To_Fractional](#)
- [ZMatrix_To_Spherical](#)

Fortran Filename

Variables

- [Molecule_Type](#)
- [Molecular_Crystal_Type](#)
- [Err_Molec](#)
- [Err_Molec_Mess](#)

Molecule_Type

	Variable	Definition
Type :: Molecule_Type		
Character(Len=80)	Name_Mol	Global name for the molecule
Integer	NAtoms	Number of atoms
Logical	In_XTAL	TRUE if global coordinates xcentre, orient are defined
Logical	Is_EulerMat	TRUE if the Euler Matrix has been set
Logical	Is_Connect	TRUE if the connectivity is correct
Character(Len=1)	Rot_Type	Type of rotational angles E : Conventional Euler angles (alpha,beta,gamma) P : Second variant of Euler angles (default) Polar:(theta,phi,chi)
Character(Len=1)	Coor_Type	Type of internal coordinates C : Cartesian F : Fractional (only if in_XTAL=.TRUE.) S : Spherical Z : Z-Matrix
Character(Len=3)	Therm_Type	Type of thermal factor ISO : No collective motion T : Translational TL : Translational + Vibrational TLS : Translational + Vibrational + Correlation
Real(Kind=CP), Dimension(3)	XCentre	Fractional coordinates of the centre
Real(Kind=CP), Dimension(3)	MXCentre	Refinement codes of Fractional coordinates of the centre
Integer, Dimension(3)	LXCentre	Numbers of LSQ parameters for Fractional coordinates of the centre
Real(Kind=CP), Dimension(3)	Orient	Orientation angles (Euler angles or variant ...)
Real(Kind=CP), Dimension(3)	MOrient	Refinement codes of Orientation angles (Euler angles or variant ...)
Integer, Dimension(3)	LOrient	Numbers of LSQ parameters for Orientation angles (Euler angles or variant ...)
Real(Kind=CP), Dimension(6)	T_TLS	Translational Thermal factor tensor

Real(Kind=CP), Dimension(6)	MT_TLS	Refinement codes of Translational Thermal factor tensor
Integer, Dimension(6)	IT_TLS	Numbers of LSQ parameters for Translational Thermal factor tensor
Real(Kind=CP), Dimension(6)	L_TLS	Librational Thermal factor tensor
Real(Kind=CP), Dimension(6)	ML_TLS	Refinement codes of Librational Thermal factor tensor
Integer, Dimension(6)	IL_TLS	Numbers of LSQ parameters for Librational Thermal factor tensor
Real(Kind=CP), Dimension(3,3)	S_TLS	TL-correlation Thermal factor
Real(Kind=CP), Dimension(3,3)	MS_TLS	Refinement codes of TL-correlation Thermal factor
Integer, Dimension(3,3)	IS_TLS	Numbers of LSQ parameters for TL-correlation Thermal factor
Real(Kind=CP), Dimension(3,3)	Euler	Euler matrix
Character(Len=6), Dimension(:), Allocatable	AtName	Atom Name
Character(Len=4), Dimension(:), Allocatable	AtSymb	Atom species
Integer, Dimension(:), Allocatable	ATZ	Atomic Number
Integer, Dimension(:, :,), Allocatable	Ptr	Pointer to scat.factors (first index -> pattern)
Real(Kind=CP), Dimension(:, :,), Allocatable	I_Coor	internal coordinates (d,ang,dang)
Real(Kind=CP), Dimension(:, :,), Allocatable	MI_Coor	Refinement codes of internal coordinates
Integer, Dimension(:, :,), Allocatable	LI_Coor	Numbers of LSQ parameters for internal coordinates
Real(Kind=CP), Dimension(:,), Allocatable	Biso	Isotropic temperature factor
Real(Kind=CP), Dimension(:,), Allocatable	MBiso	Refinement codes of Isotropic temperature factor
Integer, Dimension(:,), Allocatable	LBiso	Numbers of LSQ parameters for Isotropic temperature factor
Real(Kind=CP), Dimension(:,), Allocatable	Occ	Occupation factor
Real(Kind=CP), Dimension(:,), Allocatable	MOcc	Refinement codes of Occupation factor
Integer, Dimension(:,), Allocatable	LOcc	Numbers of LSQ parameters for Occupation factor
Integer, Dimension(:,), Allocatable	NB	Number of neighbours
Integer, Dimension(:, :,), Allocatable	InB	index of neighbours
Integer, Dimension(:, :,), Allocatable	TB	Type of bonds
Integer, Dimension(:, :,), Allocatable	Conn	Connectivity (N1,N2,N3)
End Type Molecule_Type		

Molecular_Crystal_Type

	Variable	Definition
Type :: Molecular_Crystal_Type		
Integer	N_Free	Number of free atoms
Integer	N_Mol	Number of Molecules
Integer	N_Species	Number of species
Integer	NPat	
Type(<i>Crystal_Cell_Type</i>)	Cell	Cell information
Type(<i>Space_Group_Type</i>)	Spg	Space Group information
Type(<i>Atom_Type</i>), Dimension(:), Allocatable	Atm	Free Atoms
Type(<i>Molecule_Type</i>), Dimension(:), Allocatable	Mol	Molecules
End Type		
Molecular_Crystal_Type		

Err_Molec

Logical :: Err_Molec

This variable is set to .TRUE. if an error occurs in procedures belonging to this module

Err_Molec_Mess

Character (Len=150) :: Err_Molec_Mess

This variable contains information about the last error occurred in the procedures belonging to this module

Subroutines

- [Cartesian_To_Fractional](#)
- [Cartesian_To_Spherical](#)
- [Cartesian_To_ZMatrix](#)
- [Empirc_Formula](#)
- [Fix_Orient_Cartesian](#)
- [Fix_Reference](#)
- [Fractional_To_Cartesian](#)
- [Fractional_To_Spherical](#)
- [Fractional_To_ZMatrix](#)
- [Init_Err_Molec](#)
- [Init_Molecule](#)
- [MolCrys_To_AtomList](#)
- [Molec_To_AtomList](#)
- [Read_Free_Atoms](#)
- [Read_Molecule](#)
- [Set_Euler_Matrix](#)
- [Spherical_To_Cartesian](#)
- [Spherical_To_Fractional](#)
- [Spherical_To_ZMatrix](#)
- [Write_Free_Atoms](#)

- [Write_Molecular_Crystal](#)
- [Write_Molecule](#)
- [ZMatrix_To_Cartesian](#)
- [ZMatrix_To_Fractional](#)
- [ZMatrix_To_Spherical](#)

Cartesian_To_Fractional

Subroutine Cartesian_To_Fractional (Molecule, Cell, NewMolecule)

Type(Molecule_Type)	Intent(in out)	Molecule	Molecule Object
Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters
Type(Molecule_Type), Optional	Intent(out)	NewMolecule	Molecule Object

Subroutine to transform the internal coordinates of a molecule from Cartesian coordinates to Fractional coordinates.

If a third argument is present the subroutine creates a new molecule (copy of the old one) with fractional coordinates, preserving the input molecule in Cartesian Coordinates. Otherwise the input molecule is changed on output.

Cartesian_To_Spherical

Subroutine Cartesian_To_Spherical (Molecule, NewMolecule)

Type(Molecule_Type)	Intent(in out)	Molecule	Molecule Object
Type(Molecule_Type), Optional	Intent(out)	NewMolecule	Molecule Object

Subroutine to transform the internal coordinates of a molecule from Cartesian coordinates to Spherical coordinates.

If a second argument is present the subroutine creates a new molecule (copy of the old one) with spherical coordinates, preserving the input molecule in Cartesian Coordinates. Otherwise the input molecule is changed on output.

Cartesian_To_ZMatrix

Subroutine Cartesian_To_ZMatrix (Molecule, NewMolecule, Cell, D_Min, D_Max)

Type(Molecule_Type)	Intent(in out)	Molecule	Molecule Object
Type(Molecule_Type), Optional	Intent(out)	NewMolecule	Molecule Object
Type(Crystal_Cell_Type), Optional	Intent(in)	CEll	Cell parameters
Real(Kind=CP), Optional	Intent(in)	D_Min	
Real(Kind=CP), Optional	Intent(in)	D_Max	

Subroutine to transform the internal coordinates of a molecule from Cartesian coordinates to Z-Matrix.

If a second argument is present the subroutine creates a new molecule (copy of the old one) with Z-matrix, preserving the input molecule in Cartesian Coordinates. Otherwise the input molecule is changed on output.

The input cartesian coordinates may be defined with respect to another internal frame. The final internal frame is that defined for Z-matrices: the x-axis is from the first to the second atom and the x-y plane is

formed by the three first atoms. The Euler matrix and the molecular centre in the crystallographic system is changed in consequence.

Empirc_Formula

Subroutine Empirc_Formula (Atm / MolCrys / Molecule, Formula, Form_Weight)

Type(Atom_List_Type) or Type(Molecular_Crystal_Type) or Type(Molecule_Type)	Intent(in)	Atm MolCrys Molecule	Atom information
Character(Len=*)	Intent(out)	Formula	Empirc Formula
Real(Kind=CP), Optional	Intent(out)	Form_Weight	

Obtain the Empirc Formula from Atm/Molcrys/Molecule variable

Fix_Orient_Cartesian

Subroutine Fix_Orient_Cartesian (Molecule, NewMolecule, NAtom_O, NAtom_X, NAtom_XY, Mat)

Type(Molecule_Type)	Intent(in out)	Molecule	Molecule Object
Type(Molecule_Type), Optional	Intent(out)	NewMolecule	Molecule Object
Integer, Optional	Intent(in)	NAtom_O	
Integer, Optional	Intent(in)	NAtom_Y	
Integer, Optional	Intent(in)	NAtom_XY	
Real(Kind=CP), Dimension(3,3), Optional	Intent(out)	Mat	

Subroutine to transform the Cartesian coordinates of the molecule choosing which atom is the origin, which define the X axis and which defines the XY Plane

If the second argument is present the subroutine creates a new molecule preserving the input molecule in Cartesian. Otherwise the input molecule is changed on output.

If Natom_O is absent, then the first atom on the molecule will be the origin.

If Natom_X is absent, then the second atom on the molecule will define the X axis.

If Natom_XY is absent, then the third atom on the molecule will define the XY Plane.

The optional output matrix Mat is the active rotation matrix passing from the old Cartesian frame to the new one. The transpose matrix has served to transform the original Cartesian coordinates.

Fix_Reference

Subroutine Fix_Reference (Molecule, NewMolecule, NAtom_O, NAtom_X, NAtom_XY)

Type(Molecule_Type)	Intent(in out)	Molecule	Molecule Object
Type(Molecule_Type), Optional	Intent(out)	NewMolecule	Molecule Object
Integer, Optional	Intent(in)	NAtom_O	
Integer, Optional	Intent(in)	NAtom_Y	

Integer, Optional	Intent(in)	NAtom_XY	
--------------------------	-------------------	----------	--

Subroutine to order the molecule choosing which atom is the origin, which define the X axis and which defines the XY Plane.

If the second argument is present the subroutine creates a new molecule preserving the input molecule in Cartesian. Otherwise the input molecule is changed on output.

If Natom_0 is absent, then the first atom on the molecule will be the origin.

If Natom_X is absent, then the second atom on the molecule will define the X axis.

If Natom_XY is absent, then the third atom on the molecule will define the XY Plane.

Fractional_To_Cartesian

Subroutine Fractional_To_Cartesian (Molecule, Cell, NewMolecule)

Type(Molecule_Type)	Intent(in out)	Molecule	Molecule Object
Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters
Type(Molecule_Type), Optional	Intent(out)	NewMolecule	Molecule Object

Subroutine to transform the Fractional coordinates to Cartesian internal coordinates of a molecule.

If NewMolecule is present the subroutine creates a new molecule (copy of the old one) with cartesian coordinates, preserving the input molecule in fractional. Otherwise the input molecule is changed on output.

Fractional_To_Spherical

Subroutine Fractional_To_Spherical (Molecule, Cell, NewMolecule)

Type(Molecule_Type)	Intent(in out)	Molecule	Molecule Object
Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters
Type(Molecule_Type), Optional	Intent(out)	NewMolecule	Molecule Object

Subroutine to transform the internal coordinates of a molecule from Fractional coordinates to Spherical coordinates.

If a third argument is present the subroutine creates a new molecule (copy of the old one) with Spherical coordinates, preserving the input molecule in Fractional Coordinates. Otherwise the input molecule is changed on output.

Fractional_To_ZMatrix

Subroutine Fractional_To_ZMatrix (Molecule, Cell, NewMolecule)

Type(Molecule_Type)	Intent(in out)	Molecule	Molecule Object
Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters
Type(Molecule_Type), Optional	Intent(out)	NewMolecule	Molecule Object

Subroutine to transform the internal coordinates of a molecule from Fractional coordinates to Zmatrix coordinates.

If a third argument is present the subroutine creates a new molecule (copy of the old one) with Zmatrix coordinates, preserving the input molecule in Fractional Coordinates. Otherwise the input molecule is changed on output.

Init_Err_Molec**Subroutine Init_Err_Molec ()**

Subroutine that initializes errors flags in **CFML_Molecular_Crystals** module.

Init_Molecule**Subroutine Init_Molecule (Molecule, NAtm)**

Type(Molecule_Type)	Intent(out)	Molecule	Molecule object
Integer, Optional	Intent(in)	NAtm	Number of Atoms

Initialize the Variable Molecule. If NAtm if given the allocate the respective fields depending of this value

MolCrys_To_AtomList**Subroutine MolCrys_To_AtomList (MolCrys, Atm)**

Type(Molecular_Crystal_Type)	Intent(in)	MolCrys	Molecule Object
Type(Atom_List_Type)	Intent(out)	Atm	Atoms information

Subroutine to pass all information from [Molecular_Crystal_Type](#) to [Atom_List_Type](#)

Molec_To_AtomList**Subroutine Molec_To_AtomList (Molec, Atm, Coor_Type, Cell)**

Type(Molecule_Type)	Intent(in)	Molec	Molecule Object
Type(Atom_List_Type)	Intent(out)	Atm	Atoms information
Character(Len=*), Optional	Intent(in)	Coor_Type	
Type(Crystal_Cell_Type), Optional	Intent(in)	Cell	Cell parameters

Subroutine to pass all information from [Molecule_Type](#) to [Atom_List_Type](#)

Coor_Type determine the type of coordinates parameter in output. In general **Cell** if necessary to obtain on output fractional coordinates or special case for Z-Matrix.

Read_Free_Atoms**Subroutine Read_Free_Atoms (Lun, AtmF, N)**

Integer	Intent(in)	Lun	Logical unit to be read
Type(Atom_Type), Dimension(:)	Intent(out)	AtmF	Free atoms
Integer	Intent(out)	N	Free atoms read

Subroutine to read a set of Free Atoms from a file.

The format is:

ATOMS N_Atoms

internal Coordinates for Atoms (N_Atoms Lines): Atom_Name(6) Atom_Specie(4) Coordinates(3) Biso

Occ [VARY]

if VARY is present as last option on the internal Coordinates line, then an extra line is read:
 Codes_Coordinates(3) Code_BIso Code_Occ

[Read_Molecule](#)

Subroutine Read_Molecule (Lun, Molecule)

Integer	Intent(in)	Lun	Logical unit to be read
Type(Molecule_Type)	Intent(out)	Molecule	Molecule Object

or

Subroutine Read_Molecule (File_Dat, N_Ini, N_End, Molecule)

Character(Len=*) , Dimension(:)	Intent(in)	File_Dat	Name of the File to read
Intent(in)	Intent(in)	N_Ini	initial line to be read
Integer	Intent(in)	N_End	Last line to be read
Type(Molecule_Type)	Intent(out)	Molecule	Molecule Object

Subroutine to read a molecule from a file.

The format of the file is:

MOLE[X] N_ATOMS MOLECULE_NAME COORDinATES_TYPE

where

Variables	Definitions
N_ATOMS	Number of atoms in the molecule definition
MOLECULE_NAME	Name for the molecule
COORDinATES_TYPE	Values are: C : Cartesian coordinates F : Fractional coordinates S : Spherical coordinates Z : Z-Matrix coordinates

If keyword **MOLEX** is present, then the next line will be read (6 reals, 2 characters):

MOLECULE_CENTRE(3) MOLECULE_ORIENT(3) ROTATIONAL_ANGLE(1)_TYPE
THERMAL_FACTOR_TYPE(3)

where

Variables	Definitions
MOLECULE_CENTRE	Coordinate of Center of Molecule
MOLECULE_ORIENT	Angles orientation
ROTATIONAL_ANGLE_TYPE	Values are: E : Conventional Euler angles (alpha, beta, gamma) P : Polar Euler angles (Phi, theta, Chi) (default)
THERMAL_FACTOR_TYPE	Values are: ISO : No collective motion TLS : Traslational + Librational + Correlation TL : Traslational + Librational T : Traslational

According to Thermal Factors, next lines will be read

**Thermal Definitions
Factors**

T 6 Thermal Factors (Line1) + 6 Codes Thermal Factors (Line2)

TL 6 Thermal Factors (Line1) + 6 Codes Thermal Factors (Line2)
6 Thermal Factors (Line3) + 6 Codes Thermal Factors (Line4)

TLS 6 Thermal Factors (Line1) + 6 Codes Thermal Factors (Line2)
6 Thermal Factors (Line3) + 6 Codes Thermal Factors (Line4)
9 Thermal Factors (Line5) + 9 Codes Thermal Factors (Line6)

internal Coordinates for Atoms (N_Atoms Lines):

ATOM_NAME(6) ATOM_SPECIES(4) COORDinATES(3) N1 N2 N3 BISO OCC [VARY]

If VARY is present as last option on the internal Coordinates line, then an extra line is read:

CODES_COORDinATES(3) CODE_BISO(1) CODE_OCC(1)

[Set_Euler_Matrix](#)

Subroutine Set_Euler_Matrix (RT, Phi, Theta, Chi, Eu)

Character(<i>Len=*</i>)	Intent(in)	RT	Values are: E : Conventional Euler angles (alpha, beta, gamma) P : Polar angles
Real(<i>Kind=CP</i>)	Intent(in)	Phi	Angle Phi
Real(<i>Kind=CP</i>)	Intent(in)	Theta	Angle Theta
Real(<i>Kind=CP</i>)	Intent(in)	Chi	Angle Chi
Real(<i>Kind=CP</i>), Dimension(3,3)	Intent(out)	Eu	Euler array

Subroutine to obtain the Euler active matrix to transform a point to another point.

For instance the internal coordinates of a molecule can be transformed to absolute positions using columns vectors.

If the Cartesian coordinates of an atom in the molecular frame is the column vector X_m , the cartesian coordinates in the crystal frame X are obtained from: $X = Eu X_m$
The internal coordinates of a point are obtained from $X_m = Eu^T X$

[Spherical_To_Cartesian](#)

Subroutine Spherical_To_Cartesian (Molecule, NewMolecule)

Type(Molecule_Type)	Intent(in out)	Molecule	Molecule Object
Type(Molecule_Type), Optional	Intent(out)	NewMolecule	Molecule Object

Subroutine to transform the internal coordinates of a molecule from Spherical coordinates to Cartesian coordinates.

If a second argument is present the subroutine creates a new molecule (copy of the old one) with spherical coordinates, preserving the input molecule in Cartesian Coordinates. Otherwise the input molecule is

changed on output.

Spherical_To_Fractional

Subroutine Spherical_To_Fractional (Molecule, Cell, NewMolecule)

Type(Molecule_Type)	Intent(in out)	Molecule	Molecule Object
Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters
Type(Molecule_Type), Optional	Intent(out)	NewMolecule	Molecule Object

Subroutine to transform the internal coordinates of a molecule from Spherical coordinates to Fractional coordinates.

If a third argument is present the subroutine creates a new molecule (copy of the old one) with Fractional coordinates, preserving the input molecule in Spherical Coordinates. Otherwise the input molecule is changed on output.

Spherical_To_ZMatrix

Subroutine Spherical_To_ZMatrix (Molecule, NewMolecule, Cell)

Type(Molecule_Type)	Intent(in out)	Molecule	Molecule Object
Type(Molecule_Type), Optional	Intent(out)	NewMolecule	Molecule Object
Type(Crystal_Cell_Type), Optional	Intent(in)	Cell	Cell parameters

Subroutine to transform the internal coordinates of a molecule from Spherical coordinates to Zmatrix coordinates.

If a second argument is present the subroutine creates a new molecule (copy of the old one) with Zmatrix coordinates, preserving the input molecule in Spherical Coordinates. Otherwise the input molecule is changed on output.

Write_Free_Atoms

Subroutine Write_Free_Atoms (AtmF, N, Lun)

Type(Atom_Type), Dimension(:)	Intent(in)	AtmF	Free atoms
Integer	Intent(in)	N	Free atoms read
Integer, Optional	Intent(in)	Lun	Logical unit to be written

Write information about Free Atoms

Write_Molecular_Crystal

Subroutine Write_Molecular_Crystal (MolCrys, Lun)

Type(Molecular_Crystal_Type)	Intent(in)	MolCrys	Molecule
Integer, Optional	Intent(in)	Lun	Logical unit to be written

Write information about Molecular Crystal

Write_Molecule**Subroutine Write_Molecule (Molecule, Lun)**

Type(Molecule_Type)	Intent(in)	Molecule	Molecule
Integer, Optional	Intent(in)	Lun	Logical unit to be written

Write information about molecule

ZMatrix_To_Cartesian**Subroutine ZMatrix_To_Cartesian (Molecule, NewMolecule)**

Type(Molecule_Type)	Intent(in out)	Molecule	Molecule Object
Type(Molecule_Type), Optional	Intent(out)	NewMolecule	Molecule Object

Subroutine to transform the internal coordinates of a molecule from Z-matrix to Cartesian coordinates.

If a second argument is present the subroutine creates a new molecule (copy of the old one) with cartesian coordinates, preserving the input molecule. Otherwise the input molecule is changed on output.

ZMatrix_To_Fractional**Subroutine ZMatrix_To_Fractional (Molecule, Cell, NewMolecule)**

Type(Molecule_Type)	Intent(in out)	Molecule	Molecule Object
Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell parameters
Type(Molecule_Type), Optional	Intent(out)	NewMolecule	Molecule Object

Subroutine to transform the internal coordinates of a molecule from Z-matrix to Fractional coordinates.

If a third argument is present the subroutine creates a new molecule (copy of the old one) with fractional coordinates, preserving the input molecule in Z-matrix. Otherwise the input molecule is changed on output.

ZMatrix_To_Spherical**Subroutine Spherical_To_ZMatrix_To_Spherical (Molecule, NewMolecule)**

Type(Molecule_Type)	Intent(in out)	Molecule	Molecule Object
Type(Molecule_Type), Optional	Intent(out)	NewMolecule	Molecule Object

Subroutine to transform the internal coordinates of a molecule from Zmatrix coordinates to Spherical coordinates.

If a second argument is present the subroutine creates a new molecule (copy of the old one) with Spherical coordinates, preserving the input molecule in Zmatrix Coordinates. Otherwise the input molecule is changed on output.

Level 7

Concept	Module Name	Purpose
---------	-------------	---------

<i>Formats...</i>	CFML_IO_Formats	Procedures for handling different formats for input/output
-------------------	--	--

[**CFML_IO_Formats**](#)

Creation/Conversion for several formats

Variables

- [File_List_Type](#)
- [Interval_Type](#)
- [Job_Info_Type](#)

- [Err_Form](#)
- [Err_Form_Mess](#)

Subroutines

- [File_To_FileList](#)
- [Get_Job_Info](#)
- [Get_Phase_File](#)
- [Init_Err_Form](#)
- [Read_Atom](#)
- [Read_Cell](#)
- [Read_CIF_Atom](#)
- [Read_CIF_Cell](#)
- [Read_CIF_ChemicalName](#)
- [Read_CIF_Cont](#)
- [Read_CIF_Hall](#)
- [Read_CIF_HM](#)
- [Read_CIF_Lambda](#)
- [Read_CIF_Symm](#)
- [Read_CIF_Title](#)
- [Read_CIF_Z](#)
- [Read_File_Atom](#)
- [Read_File_Cell](#)
- [Read_File_Lambda](#)
- [Read_File_RNGSintL](#)
- [Read_File_Spg](#)
- [Read_File_Transf](#)
- [Read_SHX_Atom](#)
- [Read_SHX_Cell](#)
- [Read_SHX_Cont](#)
- [Read_SHX_Fvar](#)
- [Read_SHX_Latt](#)
- [Read_SHX_Symm](#)

- [Read SHX Titl](#)
- [Read_Uvals](#)
- [Readn_Set_XTAL_Structure](#)
- [Write_CIF_Ppwder_Profile](#)
- [Write_CIF_Template](#)
- [Write_SHX_Template](#)

Fortran Filename

CFML_Form_CIF.f90

Variables

- [File_List_Type](#)
- [Interval_Type](#)
- [Job_Info_Type](#)
- [Err_Form](#)
- [Err_Form_Mess](#)

File_List_Type

	Variable	Definition
Type :: File_List_Type		
Integer	NLines	Number of lines
Character(Len=132), Dimension(:), Allocatable	Line	Lines
End Type File_List_Type		

Interval_Type

	Variable	Definition
Type :: Interval_Type		
Real (Kind=CP)	MinA	Low limit
Real (Kind=CP)	MaxB	High limit
End Type Interval_Type		

Job_Info_Type

	Variable	Definition

Type :: Job_Info_Type		
Character(Len=120)	Title	Title
Integer	Num_Phases	Number of phases
Integer	Num_Patterns	Number of patterns
Integer	Num_CMD	Number of command lines
Character(Len=16), Dimension(:), Allocatable	Patt_Typ	Type of Pattern
Character(Len=128), Dimension(:), Allocatable	Phas_Nam	Name of phases
Character(Len=128), Dimension(:), Allocatable	CMD	Command lines: text for actions
Type(Interval_Type), Dimension(:), Allocatable	Range_STL	Range in $\sin\theta/\lambda$
Type(Interval_Type), Dimension(:), Allocatable	Range_Q	Range in $4\pi \cdot \sin\theta/\lambda$
Type(Interval_Type), Dimension(:), Allocatable	Range_D	Range in d-spacing
Type(Interval_Type), Dimension(:), Allocatable	Range_2Theta	Range in 2θ-spacing
Type(Interval_Type), Dimension(:), Allocatable	Range_Energy	Range in Energy
Type(Interval_Type), Dimension(:), Allocatable	Range_TOF	Range in Time of Flight
Type(Interval_Type), Dimension(:), Allocatable	Lambda	Lambda
Real (Kind=CP), Dimension(:), Allocatable	Ratio	ratio λ_2/λ_1
Real (Kind=CP), Dimension(:), Allocatable	DTT1	d-to-TOF coefficients
Real (Kind=CP), Dimension(:), Allocatable	DTT2	
End Type Job_Info_Type		

Err_Form

Logical :: Err_Form

This variable is set to .TRUE. if an error occurs in procedures belonging to this module.

Err_Form_Mess

Character (Len=150) :: Err_Form_Mess

This variable contains information about the last error occurred in the procedures belonging to this module

Subroutines

- [File_To_FileList](#)
- [Get_Job_Info](#)
- [Get_Phase_File](#)

- [Init Err Form](#)
- [Read Atom](#)
- [Read Cell](#)
- [Read CIF Atom](#)
- [Read CIF Cell](#)
- [Read CIF ChemicalName](#)
- [Read CIF Cont](#)
- [Read CIF Hall](#)
- [Read CIF HM](#)
- [Read CIF Lambda](#)
- [Read CIF Symm](#)
- [Read CIF Title](#)
- [Read CIF Z](#)
- [Read File Atom](#)
- [Read File Cell](#)
- [Read File Lambda](#)
- [Read File RNGSintL](#)
- [Read File Spg](#)
- [Read File Transf](#)
- [Read SHX Atom](#)
- [Read SHX Cell](#)
- [Read SHX Cont](#)
- [Read SHX Fvar](#)
- [Read SHX Latt](#)
- [Read SHX Symm](#)
- [Read SHX Titl](#)
- [Read Uvals](#)
- [Readn Set XTAL Structure](#)
- [Write CIF Pwpder Profile](#)
- [Write CIF Template](#)
- [Write SHX Template](#)

[File_To_FileList](#)

Subroutine FILE_To_FileList (File_Dat, File_List)

Character(Len=*) , Dimension ()	Intent(in)	File_Dat	input data file
Type(File_List_Type)	Intent(out)	File_List	File list structure

Charge an external file to an object of [File_List_Type](#)

[Get_Job_Info](#)

Subroutine Get_Job_Info (File_Dat, I_Ini, I_End, Job_Info)

Character(Len=*) , Dimension ()	Intent(in)	File_Dat	input data file
Integer	Intent(in)	I_Ini	initial line to explore
Integer	Intent(in)	I_End	Final line to explore

Type(Job_Info_Type)	Intent(out)	Job_Info	Object to be constructed
---------------------	-------------	----------	--------------------------

Constructor of the object [Job_Info_Type](#).

The array of strings File_Dat have to be provided as input. It contains lines corresponding to the input control file.

Get_Phases_File

Subroutine Get_Phases_File (Filecode, NPhas, PhaseName, ILines)

Character(Len=*)	Intent(in)	Filecode	Name of the file
Integer	Intent(out)	NPhas	Number of Phase presents on the file
Character(Len=*), Dimension (:)	Intent(out)	PhaseName	Array containing the Name of each phase
Integer, Dimension (2,:)	Intent(out)	ILines	Line number to located each phase into file filecode

This routine determines how many phases are present in the **filecode** file. Nphas give the total phases present in the file and Phasename contains the name for each phase. The array ILines have the line number where each phase is located. ILine(1,i) give the first line for phase i and ILine(2,i) give the last line for phase i.

At the moment only cif and PCR files can be used.

Init_Err_Form

Subroutine Init_Err_Form ()

Subroutine that initializes errors flags in **CFML_IO_Formats** module.

Read_Atom

Subroutine Read_Atom (Line, Atomo)

Character(Len=*)	Intent(in out)	Line	input string with ATOM directive
Type(Atom_Type)	Intent(out)	Atomo	Parameters on variable

Subroutine to read the atom parameters from a given Line it construct the object Atomo of Atom_Type.

Read_Cell

Subroutine Read_Cell (Line, Celda)

Character(Len=*)	Intent(in out)	Line	input string with CELL directive
Real (Kind=CP), Dimension(6)	Intent(out)	Celda	Parameters on variable

Subroutine to read the cell parameters from a given Line.

Assumes the string Line has been read from a file and starts with the word CELL, that is removed before reading the values of the parameters.

Control of error is present

Read_CIF_Atom

Subroutine Read_CIF_Atom (Filevar, NLine_Ini, NLine_End, N_Atom, Atm_List)

Character(Len=*) , Dimension(:)	Intent(in)	Filevar		input strings information
Integer	Intent(in,out)	NLine_Ini	in: out:	Line to beginning search Current line on Filevar
Integer	Intent(in)	NLine_End		Line to the End search
Integer	Intent(out)	N_Atom		Actual number of atoms
Type(Atom_List_Type)	Intent(out)	Atm_List		Atom list

Obtaining Atoms parameters from a CIF file. A control error is present.

Read_CIF_Cell

Subroutine Read_CIF_Cell (Filevar, NLine_Ini, NLine_End, Celda, STDCelda)

Character(Len=*) , Dimension(:)	Intent(in)	Filevar		input strings information
Integer	Intent(in,out)	NLine_Ini	in: out:	Line to beginning search Current line on Filevar
Integer	Intent(in)	NLine_End		Line to the End search
Real(Kind=CP), Dimension(6)	Intent(out)	Celda		Cell parameters
Real(Kind=CP), Dimension(6)	Intent(out)	STDCelda		Standar values for cell parameters

Read Cell parameters from CIF file

Read_CIF_ChemicalName

Subroutine Read_CIF_ChemicalName(Filevar, NLine_Ini, NLine_End, ChemName)

Character(Len=*) , Dimension(:)	Intent(in)	Filevar		input strings information
Integer	Intent(in,out)	NLine_Ini	in: out:	Line to beginning search Current line on Filevar
Integer	Intent(in)	NLine_End		Line to the End search
Character(Len=*)	Intent(out)	ChemName		Chemical name information

Obtaining Chemical Name from CIF file

Read_CIF_Cont

Subroutine Read_CIF_Cont (Filevar, NLine_Ini, NLine_End, N_Elem_Type, Elec_Type, N_Elem)

Character(Len=*) , Dimension(:)	Intent(in)	Filevar		input strings information
Integer	Intent(in out)	NLine_Ini	in: out:	Line to beginning search Current line on Filevar
Integer	Intent(in)	NLine_End		Line to the End search
Integer	Intent(out)	N_Element_Type		Number of different elements
Character(Len=*) , Dimension(:)	Intent(out)	Element_Type		Element type characters
Real(Kind=CP) , Dimension(:) , Optional	Intent(out)	N_Elem		Number of elements

Obtaining the chemical contents from CIF file

Read_CIF_Hall

Subroutine Read_CIF_Hall(Filevar, NLine_Ini, NLine_End, Spgr_Ha)

Character(Len=*) , Dimension(:)	Intent(in)	Filevar		input strings information
Integer	Intent(in out)	NLine_Ini	in: out:	Line to beginning search Current line on Filevar
Integer	Intent(in)	NLine_End		Line to the End search
Character(Len=*)	Intent(out)	Spgr_Ha		Hall symbol

Obtaining the Hall symbol of the Space Group

Read_CIF_HM

Subroutine Read_CIF_HM (Filevar, NLine_Ini, NLine_End, Spgr_HM)

Character(Len=*) , Dimension(:)	Intent(in)	Filevar		input strings information
Integer	Intent(in out)	NLine_Ini	in: out:	Line to beginning search Current line on Filevar
Integer	Intent(in)	NLine_End		Line to the End search
Character(Len=*)	Intent(out)	Spgr_HM		Hermann-Mauguin symbol

Obtaining the Hermann-Mauguin symbol of the Space Group

Read_CIF_Lambda

Subroutine Read_CIF_Lambda (Filevar, NLine_Ini, NLine_End, Lambda)

Character(Len=*) , Dimension(:)	Intent(in)	Filevar		input strings information
Integer	Intent(in out)	NLine_Ini	in: out:	Line to beginning search Current line on Filevar
Integer	Intent(in)	NLine_End		Line to the End search

Real(Kind=CP)	Intent(out)	Lambda		Lambda value
----------------------	--------------------	--------	--	--------------

Obtaining the radiation length on CIF file

Read_CIF_Symm

Subroutine Read_CIF_Symm (Filevar, NLine_Ini, NLine_End, N_Oper, Oper_Symm)

Character(Len=*), Dimension(:)	Intent(in)	Filevar		input strings information
Integer	Intent(in,out)	NLine_Ini	in: out:	Line to beginning search Current line on Filevar
Integer	Intent(in)	NLine_End		Line to the End search
Integer	Intent(out)	N_Oper		Number of Operators
Character(Len=*), Dimension(:)	Intent(out)	Oper_Sym		Vector with Symmetry Operators

Obtaining Symmetry Operators from CIF file

Read_CIF_Title

Subroutine Read_CIF_Title (Filevar, NLine_Ini, NLine_End, Title)

Character(Len=*), Dimension(:)	Intent(in)	Filevar		input strings information
Integer	Intent(in,out)	NLine_Ini	in: out:	Line to beginning search Current line on Filevar
Integer	Intent(in)	NLine_End		Line to the End search
Character(Len=*)	Intent(out)	Title		Title

Obtaining Title from CIF file

Read_CIF_Z

Subroutine Read_CIF_Z (Filevar, NLine_Ini, NLine_End, Z)

Character(Len=*), Dimension(:)	Intent(in)	Filevar		input strings information
Integer	Intent(in,out)	NLine_Ini	in: out:	Line to beginning search Current line on Filevar
Integer	Intent(in)	NLine_End		Line to the End search
Integer	Intent(out)	Z		Number of molecules on Unit cell

Unit formula from CIF file

Read_File_Atom

Subroutine Read_File_Atom (Filevar, NLine_Ini, NLine_End, Atomos)

Character(Len=*) , Dimension(:)	Intent(in)	Filevar		input strings information
Integer	Intent(in,out)	NLine_Ini	in: out:	Line to beginning search Current line on Filevar
Integer	Intent(in)	NLine_End		Line to the End search
Type(Atom_List_Type) or TYPE(Point_List_Type)	Intent(out)	Atomos		Atom list / Point list

Subroutine to read an atom (or point) list from a file. **Atomos** should be previously allocated. Control of error is present.

Read_File_Cell**Subroutine Read_File_Cell (Filevar, NLine_Ini, NLine_End, Celda)**

Character(Len=*) , Dimension(:)	Intent(in)	Filevar		input strings information
Integer	Intent(in,out)	NLine_Ini	in: out:	Line to beginning search Current line on Filevar
Integer	Intent(in)	NLine_End		Line to the End search
Real(Kind=CP) , Dimension(6) or Type(Crystal_Cell_Type)	Intent(out)	Celda		Cell parameters

Read Cell Parameters from file. Control error is present

Read_File_Lambda**Subroutine Read_File_Lambda (Filevar, NLine_Ini, NLine_End, V1, V2, V3)**

Character(Len=*) , Dimension(:)	Intent(in)	Filevar		input strings information
Integer	Intent(in,out)	NLine_Ini	in: out:	Line to beginning search Current line on Filevar
Integer	Intent(in)	NLine_End		Line to the End search
Real(Kind=CP)	Intent(out)	V1		Lambda1 value
Real(Kind=CP)	Intent(out)	V2		Lambda2 value
Real(Kind=CP)	Intent(out)	V3		Ratio Lambda2/Lambda1

Read wavelengths and ratio from a file

If no value is read, Lambda1=Lambda2=1.54056 Angstroms, ratio=0.0

If only one value is read Lambda1=Lambda2=v1, ratio=0

If only two values are read Lambda1=v1, Lambda2=v2, ratio=0.5

Read_File_RNGSinTL

Subroutine Read_File_RNGSinTL (Filevar, NLine_Ini, NLine_End, V1, V2)

Character(Len=*) , Dimension(:)	Intent(in)	Filevar		input strings information
Integer	Intent(inout)	NLine_Ini	in: out:	Line to beginning search Current line on Filevar
Integer	Intent(in)	NLine_End		Line to the End search
Real(Kind=CP)	Intent(out)	V1		Lower value in sinθ/λ
Real(Kind=CP)	Intent(out)	V2		Upper value in sinθ/λ

Read range for in sinθ/λ [v1,v2]

If only one value is read v1=0 and v2= read value

If the keyword RNGSL is not given in the file, the default values are v1=0.0, v2=1.0

Read_File_Spg

Subroutine Read_File_Spg (Filevar, NLine_Ini, NLine_End, Spgr, Sub)

Character(Len=*) , Dimension(:)	Intent(in)	Filevar		input strings information
Integer	Intent(inout)	NLine_Ini	in: out:	Line to beginning search Current line on Filevar
Integer	Intent(in)	NLine_End		Line to the End search
Character(Len=*)	Intent(out)	Spgr		Space Group symbol
Character(Len=*) , Optional	Intent(in)			The space group symbol is a subgroup of an already given space group
Character(Len=*) , Optional	Intent(in)	Sub		The space group symbol is a subgroup of an already given space group

Reads the card Spgr in Filevar. Control of error is present

Read_File_Transf

Subroutine Read_File_Transf (Filevar, NLine_Ini, NLine_End, Transf, Orig)

Character(Len=*) , Dimension(:)	Intent(in)	Filevar		input strings information
Integer	Intent(inout)	NLine_Ini	in: out:	Line to beginning search Current line on Filevar
Integer	Intent(in)	NLine_End		Line to the End search
Real(Kind=CP) , Dimension(3,3)	Intent(out)	Transf		Transformation array
Real(Kind=CP) , Dimension(3)	Intent(out)	Orig		

Read transformation matrix for changing the space group or cell setting. First the matrix M is read row by row and then the origin in the old setting is finally read. A single line with 12 real numbers should be given.

Example:

```
TRANS m11 m12 m13 m21 m22 m33 m31 m32 m33 o1 o2 o3
```

That's means:

$$\begin{aligned}a' &= m_{11} a + m_{12} b + m_{13} c \\b' &= m_{21} a + m_{22} b + m_{23} c \\c' &= m_{31} a + m_{32} b + m_{33} c\end{aligned}$$

$$X = \text{inv}(M_t) (X-O)$$

[Read_SHX_Atom](#)**Subroutine Read_SHX_Atom (Filevar, NLine_Ini, NLine_End, N_FVar, FVar, Elec_Type, Celda, Atm_List)**

Character(Len=*), Dimension(:)	Intent(in)	Filevar		input strings information
Integer	Intent(inout)	NLine_Ini	in: out:	Line to beginning search Current line on Filevar
Integer	Intent(in)	NLine_End		Line to the End search
Integer	Intent(in)	N_FVar		Number of parameters on FVAR
Real (Kind=CP), Dimension(:)	Intent(in)	FVar		Values for FVAR
Character(Len=*), Dimension(:)	Intent(in)	Elec_Type		Elements type
Type(Crystal_Cell_Type)	Intent(in)	Celda		Cell Parameter
Type(Atom_List_Type)	Intent(out)	Atm_List		Atom list

Obtaining Atoms parameters from SHELX files (.ins or .res)

[Read_SHX_Cell](#)**Subroutine Read_SHX_Cell (Filevar, NLine_Ini, NLine_End, Celda, STDCelda, Lambda, Z)**

Character(Len=*), Dimension(:)	Intent(in)	Filevar		input strings information
Integer	Intent(inout)	NLine_Ini	in: out:	Line to beginning search Current line on Filevar
Integer	Intent(in)	NLine_End		Line to the End search
Real (Kind=CP), Dimension(6)	Intent(out)	Celda		Cell Parameter
Real (Kind=CP), Dimension(6)	Intent(out)	STDCeld a		Standar deviations for Cell parameters
Real (Kind=CP)	Intent(out)	Lambda		Lambda
Integer	Intent(out)	Z		Number of molecules on unit cell

Obtaining Cell Parameter from SHELX file

Read_SHX_Cont

Subroutine Read_SHX_Cont (Filevar, NLine_Ini, NLine_End, N_ELEM_TYPE, ELEM_TYPE, N_ELEM)

Character(Len=*) , Dimension(:)	Intent(in)	Filevar		input strings information
Integer	Intent(in,out)	NLine_Ini	in: out:	Line to beginning search Current line on Filevar
Integer	Intent(in)	NLine_End		Line to the End search
Integer	Intent(out)	N_Elem_Type		Number of different species
Character(Len=*) , Dimension(:)	Intent(out)	Elem_Type		Character to identify the specie
Integer, Dimension(:), Optional	Intent(out)	N_Elem		Number of elements into the same species

Obtaining Chemical contents from SHELX file (.ins or .res)

Read_SHX_FVar

Subroutine Read_SHX_FVar (Filevar, NLine_Ini, NLine_End, N_FVar, FVar)

Character(Len=*) , Dimension(:)	Intent(in)	Filevar		input strings information
Integer	Intent(in,out)	NLine_Ini	in: out:	Line to beginning search Current line on Filevar
Integer	Intent(in)	NLine_End		Line to the End search
Integer	Intent(out)	N_FVar		Number of parameters on FVAR
Real (Kind=CP) , Dimension(:)	Intent(out)	FVar		Values for FVAR

Obtaining FVAR parameters from SHELX file (.ins or .res)

Read_SHX_Latt

Subroutine Read_SHX_Latt (Filevar, NLine_Ini, NLine_End, Latt)

Character(Len=*) , Dimension(:)	Intent(in)	Filevar		input strings information
Integer	Intent(in,out)	NLine_Ini	in: out:	Line to beginning search Current line on Filevar
Integer	Intent(in)	NLine_End		Line to the End search
Integer	Intent(out)	Latt		Lattice number

Obtaining lattice from SHELX file (.ins or .res)

Read_SHX_Symm

Subroutine Read_SHX_Symm (Filevar, NLine_Ini, NLine_End, N_Oper,

Oper_Symm)

Character(Len=*) , Dimension (:)	Intent(in)	Filevar		input strings information
Integer	Intent(in out)	NLine_Ini	in: out:	Line to beginning search Current line on Filevar
Integer	Intent(in)	NLine_End		Line to the End search
Integer	Intent(out)	N_Oper		Number of Operators
Character(Len=*) , Dimension (:)	Intent(out)	Oper_Symm		String for Symmetry Operators

Obtaining Symmetry Operators from SHELX file (.ins or .res)

Read_SHX_Titl**Subroutine Read_SHX_Titl (Filevar, NLine_Ini, NLine_End, Title)**

Character(Len=*) , Dimension (:)	Intent(in)	Filevar		input strings information
Integer	Intent(in out)	NLine_Ini	in: out:	Line to beginning search Current line on Filevar
Integer	Intent(in)	NLine_End		Line to the End search
Character(Len=*)	Intent(out)	Title		Title string

Obtaining Title from SHELX file (.ins or .res)

Read_UVals**Subroutine Read_UVals (Line, Atomo, ULabel)**

Character(Len=*)	Intent(in out)	Line	input string
Type(Atom_Type)	Intent(in out)	Atomo	Parameters on variable
Character(Len=4)	Intent(in)	ULabel	U_ij; B_ij; BETA

Subroutine to read the anisotropic thermal parameters from a given Line it completes the object **Atomo** of type Atom. Assumes the string Line has been read from a file and starts with one of the words (u_ij, b_ij or beta), that is removed before reading the values of the parameters.

ReadN_Set_XTAL_Structure**Subroutine ReadN_Set_XTAL_Structure (Filenam, MolCrys, Mode, IPhase, Job_Info, File_List)**

Character(Len=*)	Intent(in)	Filenam	Name of File
Type(Molecular_Crystal_Type)	Intent(out)	MolCrys	Molecule information
Character(Len=*) , Optional	Intent(in)	Mode	
Integer , Optional	Intent(in)	IPhase	

Type(Jonb_Info_Type), Optional	Intent(out)	Job_Info	
Type(File_List_Type), Optional	Intent(out)	File_List	

or

Subroutine ReadN_Set_XTAL_Structure (Filenam, Cell, Spg, A, Mode, IPhase, Job_Info, File_List)

Character(Len=*)	Intent(in)	Filenam	Name of File
Type(Crystal_Cell_Type)	Intent(out)	Cell	Cell Parameters
Type(Space_Group_Type)	Intent(out)	Spg	Space Group
Type(Atom_List_Type)	Intent(out)	A	Atom List
Character(Len=*), Optional	Intent(in)	Mode	
Integer, Optional	Intent(in)	IPhase	
Type(Jonb_Info_Type), Optional	Intent(out)	Job_Info	
Type(File_List_Type), Optional	Intent(out)	File_List	

Subroutine to read an input file and construct the crystal structure in terms of the object MolCrys or Cell, Spg and A. The optional argument IPhase is an integer telling to the program to read the phase number IPhase in the case of the presence of more than one phase. If absent only the first phase is read.

Write_CIF_Powder_Profile

Subroutine Write_CIF_Powder_Profile (Filename, Code)

Character(Len=*)	Intent(in)	Filename	Name of File
Integer	Intent(in)	Code	Values are: 0 Shelxs-Patterson 1 Shelxs-Direct Methods 2 Shelxl-Refinement

Write a CIF Powder profile file

Write_CIF_Template

Subroutine Write_CIF_Template (Filename, Type_Data, Code)

Character(Len=*)	Intent(in)	Filename	Name of File
Integer	Intent(in)	Type_Data	0 Single Crystal 1 Powder Data
Integer	Intent(in)	Code	Values are: 0 Shelxs-Patterson 1 Shelxs-Direct Methods 2 Shelxl-Refinement

Write a CIF File

Write_SHX_Template

Subroutine Write_SHX_Template (Filename, Code, Title, Lambda, Z, Celda, Space, Atomics)

Character(Len=*)	Intent(in)	Filename	Name of File
Integer	Intent(in)	Code	Values are: 0 Shelxs-Patterson 1 Shelxs-Direct Methods 2 Shelxl-Refinement
Character(Len=*)	Intent(in)	Title	Title
Real (Kind=CP)	Intent(in)	Lambda	Lambda
Integer	Intent(in)	Z	
Type(Crystal_Cell_Type)	Intent(in)	Celda	Cell parameters
Type(Space_Group_Type)	Intent(in)	Space	Space group
Type(Atom_List_Type)	Intent(in)	Atoms	Atom List

Write a Shelx File

Level 8

Concept	Module Name	Purpose
Refinement...	<u>CFML_Keywords_Code_Parser</u>	Refinable Codes parser
Magnetic Symmetry...	<u>CFML_Magnetic_Symmetry</u>	Procedures handling operations with Magnetic Symmetry and Magnetic Structures
Simulated Annealing...	<u>CFML_Simulated_Annealing</u>	Module for Global Optimization using Simulated Annealing

[CFML_Keywords_Code_Parser](#)

Module with procedure for Refinable Codes on Parameters

Parameters

- [Code_Nam](#)
- [Key_Code](#)

Variables

- [Angle_Restraint_Type](#)
- [Distance_Restraint_Type](#)
- [Torsion_Restraint_Type](#)
- [Ang_Rest](#)

- [Dis_Rest](#)
- [Err_RefCodes](#)
- [Err_RefCodes_Mess](#)
- [NP_Cons](#)
- [NP_Max](#)
- [NP_Refi](#)
- [NP_Rest_Ang](#)
- [NP_Rest_Dis](#)
- [NP_Rest_Tor](#)
- [Tor_Rest](#)
- [V_BCon](#)
- [V_Bounds](#)
- [V_List](#)
- [V_Name](#)
- [V_Vec](#)
- [V_Shift](#)

Subroutines

- [Allocate_RestParam](#)
- [Allocate_VParam](#)
- [Get_RestAng_Line](#)
- [Get_RestDis_Line](#)
- [Get_RestTor_Line](#)
- [Init_Err_RefCodes](#)
- [Init_RefCodes](#)
- [Read_RefCodes_File](#)
- [VState_To_AtomsPar](#)
- [Write_Info_RefCodes](#)
- [Write_Info_RefParams](#)
- [Write_Restraints_ObsCalc](#)

Fortran Filename

CFML_Refcodes.f90

Parameters

- [Code_Nam](#)
- [Key_Code](#)

Code_Nam

Character (Len=*), Dimension(21), Parameter :: CODE_NAM

Variable for treatment codes

Value	CODE_NAM
1	X
2	Y
3	Z
4	B
5	OCC
6	B11
7	B22
8	B33
9	B12
10	B13
11	B23
12	Bns
13	XC
14	YC
15	ZC
16	THETA
17	PHI
18	CHI
19	TH_L
20	TH_T
21	TH_S

Key_Code

Character (Len=*), **Dimension(8)**, **Parameter :: Key_Code**

Key codes defined in the module

Value	CODE_NAM
1	XYZ
2	OCC
3	BIS
4	BAN
5	ALL
6	CEN
7	ORI
8	THE

Variables

- [Angle_Restraint_Type](#)
- [Distance_Restraint_Type](#)
- [Torsion_Restraint_Type](#)

- [Ang_Rest](#)
- [Dis_Rest](#)
- [Err_RefCodes](#)
- [Err_RefCodes_Mess](#)
- [NP_Cons](#)
- [NP_Max](#)
- [NP_Refi](#)
- [NP_Rest_Ang](#)
- [NP_Rest_Dis](#)
- [NP_Rest_Tor](#)
- [Tor_Rest](#)
- [V_BCon](#)
- [V_Bounds](#)
- [V_List](#)
- [V_Name](#)
- [V_Vec](#)
- [V_Shift](#)

Angle_Restraint_Type

	<i>Variable</i>	<i>Definition</i>
Type :: Angle_Restraint_Type		
Real (Kind=CP)	AObs	Observed angle
Real (Kind=CP)	ACalc	Calculated angle
Real (Kind=CP)	Sigma	Sigma value
Integer, Dimension(8)	P	index vector
Character (Len=8), Dimension(2)	STCode	
End Type		
Angle_Restraint_Type		

Distance_Restraint_Type

	<i>Variable</i>	<i>Definition</i>
Type :: Distance_Restraint_Type		
Real (Kind=CP)	DObs	Observed distance
Real (Kind=CP)	DCalc	Calculated distance
Real (Kind=CP)	Sigma	Sigma value
Integer, Dimension(2)	P	index vector
Character (Len=8)	STCode	
End Type		
Distance_Restraint_Type		

Torsion_Restraint_Type

	Variable	Definition
Type :: Torsion_Restraint_Type		
Real (Kind=CP)	TObs	Observed torsion angle
Real (Kind=CP)	TCalc	Calculated torsion angle
Real (Kind=CP)	Sigma	Sigma value
Integer, Dimension(4)	P	index vector
Character (Len=8), Dimension(3)	STCode	
End Type Torsion_Restraint_Type		

Ang_Rest

Type (Angle_Restraint_Type), Dimension(:), Allocatable :: Ang_Rest

Relations for Angle Restraints

Dis_Rest

Type (Distance_Restraint_Type), Dimension(:), Allocatable :: Dis_Rest

Relations for Distance Restraints

Err_RefCodes

Logical :: Err_RefCodes

This variable is set to .TRUE. if an error occurs in procedures belonging to this module.

Err_RefCodes_Mess

Character (Len=150) :: Err_RefCodes_Mess

This variable contains information about the last error occurred in the procedures belonging to this module

NP_Cons

Integer :: NP_Cons

Number of Constraints relations

NP_Max

Integer :: NP_Max

Number of Maximum Parameters to Refine

NP_Refi

Integer :: NP_Refi

Number of Refinable Parameters

NP_Rest_Ang

Integer :: NP_Rest_Ang

Number of Angle Restraints relations

NP_Rest_Dis

Integer :: NP_Rest_Dis

Number of Distance Restraints relations

NP_Rest_Tor

Integer :: NP_Rest_Tor

Number of Torsion Restraints relations

Tor_Rest

Type (Torsion_Restraint_Type), Dimension(:), Allocatable :: Tor_Rest

Relations for Torsion Angle Restraints

V_BCon

Integer, Dimension(:), Allocatable :: V_BCon

Vector of Boundary Conditions

V_Bounds

Real(Kind=CP), Dimension(:, :,), Allocatable :: V_Bounds

Vector of Lower, Upper limits and Step for Parameters

V_List

Integer, Dimension(:), Allocatable :: V_List

Vector of index point the atom order

V_Name

Character(Len=20), Dimension(:), Allocatable :: V_Name

Vector of Name of Refinable Parameters

V_Vec

Real(Kind=CP), Dimension(:), Allocatable :: V_Vec

Vector of Parameters

V_Shift**Real(Kind=CP), Dimension(:), Allocatable :: V_Shift**

Vector of holding the shift of parameters

Subroutines

- [Allocate_RestParam](#)
- [Allocate_VParam](#)
- [Get_RestAng_Line](#)
- [Get_RestDis_Line](#)
- [Get_RestTor_Line](#)
- [Init_Err_RefCodes](#)
- [Init_RefCodes](#)
- [Read_RefCodes_File](#)
- [VState_To_AtomsPar](#)
- [Write_Info_RefCodes](#)
- [Write_Info_RefParams](#)
- [Write_Restraints_ObsCalc](#)

[Allocate_RestParam](#)**Subroutine Allocate_RestParam (File_Dat)**

Type(File_List_Type)	Intent(out)	File_Dat	File list structure
----------------------	-------------	----------	---------------------

Allocate vectors Ang_Rest, Dist_Rest, Tor_Rest

[Allocate_VParam](#)**Subroutine Allocate_VParam (N)**

Integer	Intent(in)	N
---------	------------	---

Allocate vectors V_Vec, V_Bounds, V_Name, V_Bcon, V_Shift, V_list

If N is equal zero it deallocates the vectors

[Get_RestAng_Line](#)**Subroutine Get_RestAng_Line (Line, FAtom)**

Character(Len=*)	Intent(in)	Line	input data
Type(Atom_List_Type)	Intent(in out)	FAtom	Atom type structure

Get angle restraints relations for Free atoms Type

Example:

Angle [sig] At1a At1b At1c At2a At2b At2c....

[Get_RestDis_Line](#)

Subroutine Get_RestDis_Line (Line, FAtom)

Character(<i>Len</i>=*)	Intent(in)	Line	input data
Type(Atom_List_Type)	Intent(in out)	FAtom	Atom type structure

Get distance restraints relations for Free atoms type

Example:

Dist [sig] At1a At1b At2a At2b ...

[Get_RestTor_Line](#)**Subroutine Get_RestTor_Line (Line, FAtom)**

Character(<i>Len</i>=*)	Intent(in)	Line	input data
Type(Atom_List_Type)	Intent(in out)	FAtom	Atom type structure

Get torsion restraints relations for Free atoms type

Example:

Torsion [sig] At1a At1b At1c At1d At2a At2b At2c At2d....

[Init_Err_RefCodes](#)**Subroutine Init_Err_RefCodes ()**

Subroutine that initializes errors flags in **CFML_Keywords_Code_Parser** module.

[Init_RefCodes](#)**Subroutine Init_RefCodes(FAtom / MolCrys / Molec)**

Type(Atom_List_Type) or Type(Molecular_Crystal_Type) or Type(Molecule_Type)	Intent(in out)	FAtom MolCrys Molec	Atom type structure
--	-----------------------	---------------------------	---------------------

Initialize all refinement codes

[Read_RefCodes_File](#)**Subroutine Read_RefCodes_File(Filedat, N_Ini, N_End, FAtom / MolCrys / Molec, Spgr)**

Type(File_List_Type)	Intent(in)	Filedat	File list type
Integer	Intent(in)	N_Ini	initial line
Integer	Intent(in)	N_End	Final line

Type(Atom_List_Type) or Type(Molecular_Crystal_Type) or Type(Molecule_Type)	Intent(in out)	FAtom MolCrys Molec	Atom type structure Molecular crystal type structure Molecule type structure
Type(Space_Group_Type)	Intent(in)	Spgr	Space group information

Subroutine for treatment of Codes controls taken from FAtom/Molcrys/Molec

VState_To_AtomsPar

Subroutine VState_To_AtomsPar(FAtom / MolCrys / Molec, Mode)

Type(Atom_List_Type) or Type(Molecular_Crystal_Type) or Type(Molecule_Type)	Intent(in out)	FAtom MolCrys Molec	Atom type structure Molecular crystal type structure Molecule type structure
Character (Len=*) , Optional	Intent(in)	Mode	Space group information

Update the values to the variable FAtom/MolCrys/Molec from Vector

Write_Info_RefCodes

Subroutine Write_Info_RefCodes(FAtom / MolCrys / Molec, IUnit)

Type(Atom_List_Type) or Type(Molecular_Crystal_Type) or Type(Molecule_Type)	Intent(in out)	FAtom MolCrys Molec	Atom type structure Molecular crystal type structure Molecule type structure
Integer , Optional	Intent(in)	IUnit	Unit for output information

Write the information about Refinement Codes

Write_Info_RefParams

Subroutine Write_Info_RefParams(IUnit)

Integer , Optional	Intent(in)	IUnit	Unit for output information
----------------------------------	-------------------	-------	-----------------------------

Write the information about Refinement parameters in file associated with logical unit IUNIT. If no argument is passed the standard output (iunit=6) is used.

Write_Restraints_ObsCalc

Subroutine Write_Restraints_ObsCalc(A, IUnit)

Type(Atom_List_Type)	Intent(in)	A	Atom type structure
Integer , Optional	Intent(in)	IUnit	Unit for output information

Write the current values of the "observed" and calculated restraints, as well as the corresponding cost value.

CFML_Magnetic_Symmetry

Series of procedures handling operations with Magnetic Symmetry and Magnetic Structures

Variables

- [Magnetic_Domain_Type](#)
- [Magnetic_Group_Type](#)
- [MagSymm_K_Type](#)
- [MSym_Oper_Type](#)

- [Err_MagSym](#)
- [Err_MagSym_Mess](#)

Functions

- [ApplyMSO](#)

Subroutines

- [Init_Err_MagSym](#)
- [Init_MagSymm_K_Type](#)
- [ReadN_Set_Magnetic_Structure](#)
- [Set_Shubnikov_Group](#)
- [Write_Magnetic_Structure](#)
- [Write_Shubnikov_Group](#)

Fortran Filename

CFML_MagSymm.f90

Variables

- [Magnetic_Domain_Type](#)
- [Magnetic_Group_Type](#)
- [MagSymm_K_Type](#)
- [MSym_Oper_Type](#)

- [Err_MagSym](#)
- [Err_MagSym_Mess](#)

Magnetic_Domain_Type

	Variable	Definition
Type :: Magnetic_Domain_Type		
Integer	ND	Number of rotational domains (not counting chiral domains)
Logical	Chir	.TRUE. if chirality domains exist
Integer, Dimension(3,3,24)	DMat	Domain matrices to be applied to Fourier Coefficients
Real (Kind=CP), Dimension(2,24)	POP	Populations of domains (sum=1, the second value is /=0 for CHIR=.TRUE.)
Real (Kind=CP), Dimension(2,24)	LPOP	Number of the refined parameter
Real (Kind=CP), Dimension(2,24)	MPOP	Refinement codes for populations
End Type Magnetic_Domain_Type		

Magnetic S-domains corresponds to a different magnetic structure obtained from the domain 1 (actual model) by applying a rotational operator to the Fourier coefficients of magnetic moments. This rotational operator corresponds to a symmetry operator of the paramagnetic group that is lost in the ordered state. Chirality domains are simply obtained by changing the sign of the imaginary components of the Fourier coefficients. For each rotational domain two chiralities domains exist.

Magnetic_Group_Type

	Variable	Definition
Type :: Magnetic_Group_Type		
Character (Len=30)	Shubnikov	Shubnikov symbol (Hermann-Mauguin + primes)
Type(Space_Group_Type)	Spg	.TRUE. if chirality domains exist
Integer, Dimension(192)	TInv	When a component is +1 no time inversion is associated if tinv(i)=-1, the time inversion is associated to operator "i"
End Type Magnetic_Group_Type		

A magnetic group type is adequate when $k=(0,0,0)$. It contains as the second component the crystallographic space group. The first component is the Shubnikov Group symbol and the third component is an Integer vector with values -1 or 1 when time inversion is associated (-1) with the corresponding crystallographic symmetry operator o not (1).

MagSymm_K_Type

	Variable	Definition
Type :: MagSymm_K_Type		
Character (Len=31)	MagModel	Name to characterize the magnetic symmetry

Character (Len=1)	Latt	Symbol of the crystallographic lattice
Integer	Nirreps	Number of irreducible representations (max=4, if nirreps /= 0 => nmsym=0)
Integer	NMSym	Number of magnetic operators per crystallographic operator (max=8)
Integer	Centred	=0 centric centre not at origin =1 acentric =2 centric (-1 at origin)
Integer	MCentred	=1 Anti/a-centric Magnetic symmetry = 2 centric magnetic symmetry
Integer	NVK	Number of independent propagation vectors
Real (Kind=CP), Dimension(3,12)	KVec	Propagation vectors
Integer	NumLat	Number of centring lattice vectors
Real (Kind=CP), Dimension(3,4)	Ltr	Centring translations
Integer	NumOps	Reduced number of crystallographic Symm. Op.
Integer	Multip	General multiplicity of the space group
Integer, Dimension(4)	NBas	Number of basis functions per IRREP (if nbas < 0, the corresponding basis is complex).
Integer, Dimension(12,4)	IComp	indicator (0 pure real/ 1 pure imaginary) for coefficients of basis fucntions
Character (Len=40), Dimension(48)	SymOpSymb	Alphanumeric Symbols for SYMM
Type(Sym_Oper_Type), Dimension(48)	SymOp	Crystallographic symmetry operators
Character (Len=40), Dimension(48,8)	MSymOpSymb	Alphanumeric Symbols for MSYMM
Type(Msym_Oper_Type), Dimension(48,8)	MSymOp	Magnetic symmetry operators
Complex (Kind=CP), Dimension(3,12, 48,4)	BasF	Basis functions of the irreps of Gk
End Type MagSymm_K_Type		

Definition of the **MagSymm_K_Type** derived type, encapsulating the information concerning the crystallographic symmetry, propagation vectors and magnetic matrices. Needed for calculating magnetic structure factors.

MSym_Oper_Type

	Variable	Definition
Type :: MSym_Oper_Type		
Integer, Dimension(3,3)	Rot	Rotational Part of Symmetry Operator
Real (Kind=CP)	Phas	Phase in fraction of 2π
End Type MSym_Oper_Type		

Definition of Magnetic symmetry operator Type

Err_MagSym

Logical :: Err_MagSym

This variable is set to .TRUE. if an error occurs in procedures belonging to this module.

Err_MagSym_Mess

Character (Len=150) :: Err_MagSym_Mess

This variable contains information about the last error occurred in the procedures belonging to this module

Functions

- [ApplyMSO](#)

ApplyMSO

Complex Function ApplyMSO (Op, SK)

Type(MSym_Oper_Type)	Intent(in)	Op	Magnetic Symmetry Operator Type
Complex, Dimension(3)	Intent(in)	SK	Complex vector

Return a vector of dimension 3. Apply a magnetic symmetry operator to a complex vector: Skp = ApplyMSO(Op,Sk)

Subroutines

- [Init_Err_MagSym](#)
- [Init_MagSymm_K_Type](#)
- [ReadN_Set_Magnetic_Structure](#)
- [Set_Shubnikov_Group](#)
- [Write_Magnetic_Structure](#)
- [Write_Shubnikov_Group](#)

Init_Err_MagSym

Subroutine Init_Err_MagSym ()

Subroutine that initializes errors flags in **CFML_Magnetic_Symmetry** module.

Init_MagSymm_K_Type

Subroutine Init_MagSymm_K_Type (MGP)

Type(MagSymm_K_Type)	Intent(in out)	MGP	input string with CELL directive
----------------------	-------------------	-----	----------------------------------

Subroutine to initialize the [MagSymm_K_Type](#) variable **MGP**.

ReadN_Set_Magnetic_Structure

Subroutine ReadN_Set_Magnetic_Structure (File_CFL, N_Ini, N_End, MGP, AM, SGO, Mag_Dom)

Type(File_List_Type)	Intent(in)	File_CFL	input File
Integer	Intent(in out)	N_Ini	initial line
Integer	Intent(in)	N_End	Final line
Type(MagSymm_K_Type)	Intent(out)	MGP	
Type(MAtom_List_Type)	Intent(out)	AM	
Type(Magnetic_Group_Type), Optional	Intent(out)	SGO	
Type(Magnetic_Domain_Type), Optional	Intent(out)	Mag_Dom	

Subroutine for reading and construct the [MagSymm_K_Type](#) variable **MGP**. It is supposed that the CFL file is included in the File_List_Type variable File_CFL.

On output N_Ini, N_End hold the lines with the starting and ending lines with information about a magnetic phase. Optionally the Magnetic space group (Shubnikov group) may be obtained separately for further use. Magnetic S-domains are also read in case of providing the optional variable Mag_Dom.

Set_Shubnikov_Group

Subroutine Set_Shubnikov_Group (Shubk, SG, MGP)

Character(Len=*)	Intent(in)	Shubk	
Type(Magnetic_Group_Type)	Intent(out)	SG	
Type(MagSymm_K_Type)	Intent(in out)	MGP	

This subroutine is not completed ... it is still in development

Write_Magnetic_Structure

Subroutine Write_Magnetic_Structure (lpr, MGP, AM, SGO, Mag_Dom)

Integer	Intent(in)	lpr	input unit file
Type(MagSymm_K_Type)	Intent(out)	MGP	
Type(MAtom_List_Type)	Intent(out)	AM	
Type(Magnetic_Domain_Type), Optional	Intent(out)	Mag_Dom	

Subroutine to write out the information about the magnetic symmetry and magnetic structure in unit IPR.

Write_Shubnikov_Group

Subroutine Write_Shubnikov_Group (SG, lunit)

Type(Magnetic_Group_Type)	Intent(in)	SG	
---------------------------	------------	----	--

Integer, Optional**Intent(in)****lunit**

Subroutine to write out the information about the Shubnikov_Group

CFML_Simulated_Annealing

Module for Global Optimization using Simulated Annealing.

Currently there is available only a generic Simulated Annealing subroutine. That must be called with the name of a user-supplied subroutine to calculate the cost function as an argument. The calling program must define at least two variables of derived types SimAnn_Conditions_Type and State_Vector_Type respectively.

The generic simulated annealing procedure can use the constant step algorithm or the Corana algorithm depending on the values of the corresponding component of the SimAnn_Conditions_Type user-defined variable.

Parameters

- [NP_Conf](#)
- [NP_SAn](#)

Variables

- [MultiState_Vector_Type](#)
- [SimAnn_Conditions_Type](#)
- [State_Vector_Type](#)
- [Err_SAn](#)
- [Err_SAn_Mess](#)

Subroutines

- [SAnn_Opt_MultiConf](#)
- [Set_SimAnn_Cond](#)
- [Set_SimAnn_MStateV](#)
- [Set_SimAnn_StateV](#)
- [SimAnneal_Gen](#)
- [SimAnneal_MultiConf](#)
- [Write_SimAnn_Cond](#)
- [Write_SimAnn_MStateV](#)
- [Write_SimAnn_StateV](#)

Fortran Filename

CFML_Optimization_SAn.f90

Parameters

- [NP_Conf](#)
- [NP_SAn](#)

NP_CONF

Integer, Parameter :: NP_CONF = 30

Maximum number of initial configurations in parallel

NP_SAN

Integer, Parameter :: NP_SAN = 80

Maximum number of parameters in the model

Variables

- [MultiState_Vector_Type](#)
- [SimAnn_Conditions_Type](#)
- [State_Vector_Type](#)
- [Err_SAn](#)
- [Err_SAn_Mess](#)

MultiState_Vector_Type

	Variable	Definition
Type :: MultiState_Vector_Type		
Integer	NPar	Number of parameters of the model
Integer	NConf	Number of configurations
Integer, Dimension (NP_SAn, NP_Conf)	Code	=0 fixed parameter =1 variable parameter
Integer, Dimension (NP_SAn)	Bound	=0 fixed boundaries =1 periodic boundaries
Real (Kind=CP), Dimension (NP_SAn, NP_Conf)	State	Vector State with the current configuration
Real (Kind=CP), Dimension (NP_SAn, NP_Conf)	STP	Step vector (one value for each parameter)
Real (Kind=CP), Dimension (NP_CONF)	Cost	Vector with cost of the different configurations
Real (Kind=CP), Dimension (NP_SAn)	Low	Low-limit value of parameters
Real (Kind=CP), Dimension (NP_SAn)	High	High-limit value of parameters
Real (Kind=CP), Dimension (NP_SAn)	Config	Vector State with the best configuration

Character (Len=15), Dimension (NP_SAn)	NamPar	Name of parameters of the model
End Type MultiState_Vector_Type		

SimAnn_Conditions_Type

	Variable	Definition
Type :: SimAnn_Conditions_Type		
Real (Kind=CP)	T_Ini	initial temperature
Real (Kind=CP)	Anneal	Kirkpatrick factor for Annealing
Real (Kind=CP)	Accept	Minimum percentage of accepted configurations
Real (Kind=CP)	Threshold	Good solutions have cost values below this (used in Sann_Opt_MultiConf)
Integer	InitConfig	Flag determining if the first configuration is random or read
Integer	NAigor	Flag determining if the Corana algorithm is selected (0) or not (/=0)
Integer	NM_Cycl	Number of Cycles per temp in SA searches
Integer	Num_Temps	Maximum number of temperatures in SA
Integer	Num_Therm	Number of thermalization cycles in SA
Integer	Num_Conf	Number of parallel configurations in SA
Character (Len=60)	Cost_Function_Name	Name of Function
Integer	Seed	If different from zero, holds the seed for random number generator
End Type SimAnn_Conditions_Type		

State_Vector_Type

	Variable	Definition
Type :: State_Vector_Type		
Integer	NPar	Number of parameters of the model
Integer, Dimension (NP_SAn)	Code	=0 fixed parameter =1 variable parameter
Integer, Dimension (NP_SAn)	Bound	=0 fixed boundaries =1 periodic boundaries
Real (Kind=CP), Dimension	State	Vector State with the current configuration

(NP_SAn)		
Real (Kind=CP), Dimension (NP_SAn)	STP	Step vector (one value for each parameter)
Real (Kind=CP), Dimension (NP_SAn)	Low	Low-limit value of parameters
Real (Kind=CP), Dimension (NP_SAn)	High	High-limit value of parameters
Real (Kind=CP), Dimension (NP_SAn)	Config	Vector State with the best configuration
Real (Kind=CP)	Cost	Cost of the best configuration
Character (Len=15), Dimension (NP_SAn)	NamPar	Name of parameters of the model
End Type State_Vector_Type		

Err_SAn

Logical :: Err_SAn

This variable is set to .TRUE. if an error occurs in procedures belonging to this module.

Err_SAn_Mess

Character (Len=150) :: Err_SAn_Mess

This variable contains information about the last error occurred in the procedures belonging to this module

Subroutines

- [SAnn_Opt_MultiConf](#)
- [Set_SimAnn_Cond](#)
- [Set_SimAnn_MStateV](#)
- [Set_SimAnn_StateV](#)
- [SimAnneal_Gen](#)
- [SimAnneal_MultiConf](#)
- [Write_SimAnn_Cond](#)
- [Write_SimAnn_MStateV](#)
- [Write_SimAnn_StateV](#)

SAnn_Opt_MultiConf

Subroutine SAnn_Opt_MultiConf (Model_Funct, C, VS, Ipr, Filesav, FST)

Defined Subroutine Model_Funct		Model_Funct	
Type(SimAnn_Conditions_Type)	Intent(in out)	C	
Type (MultiState_Conditions_Type)	Intent(in out)	VS	
Integer	Intent(in)	Ipr	
Character (Len=*), Optional	Intent(in)	Filesav	

Character (Len=*) , Optional	Intent(in)	FST	
--	-------------------	-----	--

Subroutine Model_Funct (V, Cost)

Real(Kind=CP), Dimension (:)	Intent(in)	V	Variables
Real(Kind=CP)	Intent(out)	Cost	Value of Model

End Subroutine Model_Funct

If **FST** is present the user need define the next subroutine

Subroutine Write_FST (FST_File, V, Cost)

Character (Len=*)	Intent(in)	FST_File	
Real(Kind=CP), Dimension (:)	Intent(in)	V	Variables
Real(Kind=CP)	Intent(in)	Cost	Value of Model

End Subroutine Write_FST

Multi-Configurational Simulated Annealing with local optimization when a configuration with cost lower than a threshold value is given or when one of the Markov chains stalls

Set_SimAnn_Cond**Subroutine Set_SimAnn_Cond (File_List, C)**

Type(File_List_Type)	Intent(in)	File_List	
Type(SimAnn_Conditions_Type)	Intent(in)	C	Conditions

Subroutine for reading and set up the [SimAnn_Conditions_Type](#) variable **C**

Set_SimAnn_MStateV**Subroutine Set_SimAnn_MStateV (N, NSol, Con, Bounds, VNam, Vec, VS, Cod)**

Integer	Intent(in)	N	Number of parameters
Integer	Intent(in)	NSol	Number of Solutions
Integer, Dimension (:)	Intent(in)	Con	Number of Configurations
Real(Kind=CP), Dimension (:,:)	Intent(in)	Bounds	Boundary conditions (1,:) -> Low (2,:) -> High (3,:) -> Step
Character (Len=*) , Dimension (:)	Intent(in)	VNam	Names of parameters
Real(Kind=CP), Dimension (:)	Intent(in)	Vec	initial value of parameters
Type(State_Vector_Type)	Intent(out)	VS	initial State vector
Integer, Dimension (:) , Optional	Intent(in)	Cod	If present, cod(i)=0 fix the "i" parameter

Subroutine for setting up the [State_Vector_Type](#) variable **VS**

Set_SimAnn_StateV**Subroutine Set_SimAnn_StateV (N, Con, Bounds, VNam, Vec, VS, Cod)**

Integer	Intent(in)	N	Number of parameters
Integer, Dimension (:)	Intent(in)	Con	Number of Configurations
Real(Kind=CP), Dimension (:,:)	Intent(in)	Bounds	Boundary conditions (1,:) -> Low (2,:) -> High (3,:) -> Step
Character (Len=*), Dimension (:)	Intent(in)	VNam	Names of parameters
Real(Kind=CP), Dimension (:)	Intent(in)	Vec	initial value of parameters
Type(State_Vector_Type)	Intent(out)	VS	initial State vector
Integer, Dimension (:), Optional	Intent(in)	Cod	If present, cod(i)=0 fix the "i" parameter

Subroutine for setting up the [State_Vector_Type](#) variable **VS**

SimAnneal_Gen

Subroutine SimAnneal_Gen (Model_Funct, C, VS, lpr, Filesav, FST)

Defined Subroutine Model_Funct		Model_Funct	
Type(SimAnn_Conditions_Type)	Intent(in, out)	C	
Type(State_Vector_Type)	Intent(in, out)	VS	
Integer	Intent(in)	lpr	
Character (Len=*), Optional	Intent(in)	Filesav	
Character (Len=*), Optional	Intent(in)	FST	

Subroutine Model_Funct (V, Cost)

Real(Kind=CP), Dimension (:)	Intent(in)	V	Variables
Real(Kind=CP)	Intent(out)	Cost	Value of Model

End Subroutine Model_Funct

If FST is present the user need define the next subroutine

Subroutine Write_FST (FST_File, V, Cost)

Character (Len=*)	Intent(in)	FST_File	
Real(Kind=CP), Dimension (:)	Intent(in)	V	Variables
Real(Kind=CP)	Intent(in)	Cost	Value of Model

End Subroutine Write_FST

SimAnneal_MultiConf

Subroutine SimAnneal_MultiConf (Model_Funct, NSol, C, VS, lpr, Filesav, FST)

Defined Subroutine Model_Funct		Model_Funct		
Integer	Intent(in, out)	NSol		

Type (SimAnn_Conditions_Type)	Intent(in out)	C		
Type (MultiState_Conditions_Type)	Intent(in out)	VS		
Integer	Intent(in)	Ipr		
Character (Len=*) , Optional	Intent(in)	Filesav		
Character (Len=*) , Optional	Intent(in)	FST		

Subroutine Model_Funct (V, Cost)

Real(Kind=CP), Dimension (:)	Intent(in)	V	Variables
Real(Kind=CP)	Intent(out)	Cost	Value of Model

End Subroutine Model_Funct

If FST is present the user need define the next subroutine

Subroutine Write_FST (FST_File, V, Cost)

Character (Len=*)	Intent(in)	FST_File	
Real(Kind=CP), Dimension (:)	Intent(in)	V	Variables
Real(Kind=CP)	Intent(in)	Cost	Value of Model

End Subroutine Write_FST

Write_SimAnn_Cond**Subroutine Write_SimAnn_Cond (Ipr, C)**

Integer	Intent(in)	Ipr	input unit file
Type(SimAnn_Conditions_Type)	Intent(in)	C	SAn Conditions

Subroutine for writing in unit Ipr the [SimAnn_Conditions_Type](#) variable C

Write_SimAnn_MStateV**Subroutine Write_SimAnn_MState (Ipr, VS, Text, Cost)**

Integer	Intent(in)	Ipr	input unit file
Type(State_Vector_Type)	Intent(in)	VS	State vector
Character (Len=*)	Intent(in)	Text	
Integer, Optional	Intent(in)	Cost	

Subroutine for writing in unit Ipr the [State_Vector_Type](#) variable VS

Write_SimAnn_StateV**Subroutine Write_SimAnn_State (Ipr, VS, Text)**

Integer	Intent(in)	Ipr	input unit file
Type(State_Vector_Type)	Intent(in)	VS	State vector
Character (Len=*)	Intent(in)	Text	

Subroutine for Writing in unit lpr the [State_Vector_Type](#) VS

Level 9

Concept	Module Name	Purpose
Magnetic Structure Factors...	CFML_Magnetic_Structure_Factors	Magnetic Structure Factors Calculations
Polarimetry	CFML_Polarimetry	Procedures to calculate the polarization tensor as measured using CRYOPAD

[CFML_Magnetic_Structure_Factors](#)

Main module for Magnetic Structure Factors Calculations

Variables

- [MagH_Type](#)
- [MagH_List_Type](#)
- [MagHD_Type](#)
- [MagHD_List_Type](#)

- [Err_MSFac](#)
- [Err_MSFac_Mess](#)

Subroutines

- [Calc_Mag_Interaction_Vector](#)
- [Calc_Magnetic_StrF_MIV](#)
- [Calc_Magnetic_StrF_MIV_Dom](#)
- [Gen_Satellites](#)
- [Init_Err_MSFac](#)
- [Init_Mag_Structure_Factors](#)
- [Mag_Structure_Factors](#)
- [Modify_MSF](#)
- [Write_Mag_Structure_Factors](#)

Fortran Filename

CFML_Msfac.f90

Variables

- [MagH_Type](#)
- [MagH_List_Type](#)
- [MagHD_Type](#)
- [MagHD_List_Type](#)

- [Err_MSFac](#)
- [Err_MSFac_Mess](#)

MagH_Type

	Variable	Definition
Type :: MagH_Type		
Logical	Keqv_Minus	True if k equivalent to -k
Integer	Mult	Multiplicity of the reflection (useful for powder calculations)
Integer	Num_K	number of the propagation vector V _k
Real (Kind=CP)	SignP	+1 for -V _k and -1 for +V _k
Real (Kind=CP)	S	sinθ/λ
Real (Kind=CP)	SQMIV	Square of the Magnetic interaction vector
Real (Kind=CP), Dimension (3)	H	H +/- k
Complex (Kind=CP), Dimension (3)	MSF	magnetic structure factor
Complex (Kind=CP), Dimension (3)	MIV	magnetic interaction vector
End Type MagH_Type		

Define the scattering vector vector H+k and the sign -1 for H+k and +1 for H-k.
Includes the magnetic interaction vector MiV = Mper = M

MagH_List_Type

	Variable	Definition
Type :: MagH_List_Type		
Integer	NRef	
Type (MagH_Type), Dimension (:), Allocatable	MH	
End Type MagH_List_Type		

Define a list of magnetic reflections containing the scattering vector, the magnetic structure factor and the magnetic interaction vector.

MagHD_Type

	Variable	Definition
Type :: MagHD_Type		
Logical	Keqv_Minus	True if k equivalent to -k
Integer	Num_K	number of the propagation vector V _k
Real (Kind=CP)	SignP	+1 for -V _k and -1 for +V _k
Real (Kind=CP)	S	$\sin\theta/\lambda$
Real (Kind=CP)	SQAMIV	Square of the Average Magnetic interaction vector
Real (Kind=CP)	SQMIV	Average of the Square of Magnetic interaction vectors
Real (Kind=CP), Dimension (3)	H	H +/- k
Complex (Kind=CP), Dimension (3,2,24)	MSF	Magnetic structure factors of each domain (second dimension for chirality domains)
Complex (Kind=CP), Dimension (3,2,24)	MIV	Magnetic interaction vector of each domain
Complex (Kind=CP), Dimension (3)	AMIV	Average Magnetic interaction vector = 1/nd Sum{ pop(i) Miv(:,i)}
End Type MagHD_Type		

Define the scattering vector vector H+k and the sign -1 for H+k and +1 for H-k.
 Includes the average magnetic interaction vector AMiV(:) = 1/nd Sum[i]{ pop(i) Miv(:,i)}
 This type should be used whenever magnetic domains are present (single crystal work)

MagHD_List_Type

	Variable	Definition
Type :: MagHD_List_Type		
Integer	NRef	
Type (MagHD_Type), Dimension (:), Allocatable	MH	
End Type MagHD_List_Type		

Define a list of magnetic reflections containing the scattering vector, the magnetic structure factor and the magnetic interaction vector for each of the domains.

Err_MSFac

Logical :: Err_MSFac

This variable is set to .TRUE. if an error occurs in procedures belonging to this module.

Err_MSFac_Mess

Character (Len=150) :: Err_MSFac_Mess

This variable contains information about the last error occurred in the procedures belonging to this module

Subroutines

- [Calc_Mag_Interaction_Vector](#)
- [Calc_Magnetic_StrF_MIV](#)
- [Calc_Magnetic_StrF_MIV_Dom](#)
- [Gen_Satellites](#)
- [Init_Err_MSFac](#)
- [Init_Mag_Structure_Factors](#)
- [Mag_Structure_Factors](#)
- [Modify_MSF](#)
- [Write_Mag_Structure_Factors](#)

Calc_Mag_Interaction_Vector

Subroutine Calc_Mag_Interaction (Reflex, Cell, Mode)

Type(MagH_List_Type)	Intent(in out)	Reflex	Magnetic reflections list
Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell Parameters
Character (Len=*) , Optional	Intent(in)	Mode	

Calculate the Magnetic interaction vector from Magnetic Structure factors, reflections and cell parameters. The components are given with respect to the crystallographic unitary direct cell system: {e1,e2,e3}. If Mode is given the components are with respect to the cartesian frame defined in Cell.

Calc_Magnetic_STRF_MIV

Subroutine Calc_Magnetic_STRF_MIV (Cell, MGP, Atm, MH, Mode)

Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell Parameters
Type(MagSymm_K_Type)	Intent(in)	MGP	
Type(MAtom_List_Type)	Intent(in out)	Atm	
Type(MagH_Type)	Intent(in out)	MH	
Character (Len=*) , Optional	Intent(in)	Mode	

Calculate the Magnetic interaction vector from Magnetic Structure factors, reflections and cell parameters. The components are given with respect to the crystallographic unitary direct cell system: {e1,e2,e3}. If Mode is given the components are with respect to the cartesian frame defined in Cell.

Calc_Magnetic_STRF_MIV_Dom

Subroutine Calc_Magnetic_STRF_MIV_Dom (Cell, MGP, Atm, Mag_Dom, MH, Mode)

Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell Parameters
Type (MagSymm_K_Type)	Intent(in)	MGP	
Type (MAtom_List_Type)	Intent(in)	Atm	

Type (Magnetic_Domain_Type)	Intent(in)	Mag_Dom		
Type (MagHD_Type)	Intent(in out)	MH		
Character (Len=*) Optional	Intent(in)	Mode		

Calculate the Magnetic interaction vector from Magnetic Structure factors, reflections and cell parameters. The components are given with respect to the crystallographic unitary direct cell system: {e1,e2,e3}. If Mode is given the components are with respect to the cartesian frame defined in Cell.

Gen_Satellites

Subroutine Gen_Satellites (Cell, Grp, SMax, H, Ord, Powder)

Type(Crystal_Cell_Type)	Intent(in)	Cell		Cell Parameters
Type (MagSymm_K_Type)	Intent(in)	Grp		
Real(Kind=CP)	Intent(in)	SMax		
Type (MagH_List_Type)	Intent(in out)	H		
Logical, Optional	Intent(in)	Ord		
Logical, Optional	Intent(in)	Powder		

Generates half reciprocal sphere of Integer reflections and add satellites according to the information given in GRP.

Init_Err_MSFac

Subroutine Init_Err_MSFac ()

Initialize the errors flags in this Module

Init_Mag_Structure_Factors

Subroutine Init_Mag_Structure_Factors (Reflex, Atm, Grp, Lun)

Type(MagH_List_Type)	Intent(in)	Reflex		
Type(MAtom_List_Type)	Intent(in)	Atm		
Type(MagSymm_K_Type)	Intent(in)	Grp		
Integer, Optional	Intent(in)	Lun		output unit

Allocates and initializes arrays for Magnetic Structure Factors calculations. A calculation of fixed tables is also performed.

Mag_Structure_Factors

Subroutine Mag_Structure_Factors (Atm, Grp, Reflex)

Type(MAtom_List_Type)	Intent(in)	Atm		
Type(MagSymm_K_Type)	Intent(in)	Grp		
Type(MagH_List_Type)	Intent(in)	Reflex		

Calculate the Magnetic Structure Factors from a list of magnetic Atoms and a set of reflections.
A call to [Init_Mag_Structure_Factors](#) is a pre-requisite for using this subroutine. In any case the subroutine calls [Init_Mag_Structure_Factors](#) if SF_initialized=.false.

Modify_MSF

Subroutine Modify_MSF (Reflex, Atm, Grp, List, NList)

Type(MagH_List_Type)	Intent(in)	Reflex		
Type(MAtom_List_Type)	Intent(in)	Atm		
Type(MagSymm_K_Type)	Intent(in)	Grp		
Integer, Dimension (:)	Intent(in)	List		
Integer	Intent(in)	NList		

Recalculation of Magnetic Structure Factors because a list of Atoms parameters were modified. The List variable contains the numbers in the list of the atoms to be changed.

Write_Mag_Structure_Factors

Subroutine Write_Structure_Factors (Lun, Reflex, Grp)

Integer	Intent(in)	Lun		output unit
Type(MagH_List_Type)	Intent(in)	Reflex		
Type(MagSymm_K_Type)	Intent(in)	Grp		

Writes in logical unit Lun the list of structure factors contained in the Reflex type information

CFML_Polarimetry

Module for Polarisation calculations

the polar tensor is calculated respect to the coordinate frame defined in the Blume equations (Phys. Rev. Vol. 130 p.1670-1676, 1963, see also the definitions below in magn_inter_Vec_PF). As input the nuclear structure factor, the magnetic interaction vector with respect to the crystal frame and the matrices defined in CFML_Crystal_Metrics for the crystal frame are needed.

Variables

- [Polar_Calc_Type](#)
- [Polar_Calc_List_Type](#)
- [Polar_Info_Type](#)
- [Polar_Obs_Type](#)
- [Polar_Obs_List_Type](#)

Subroutines

- [Calc_Polar_Dom](#)
- [Set_Polar_Info](#)

- [Write_Polar_Info](#)
- [Write_Polar_Line](#)

Fortran Filename

CFML_Polar.f90

Variables

- [Polar_Calc_Type](#)
- [Polar_Calc_List_Type](#)
- [Polar_Info_Type](#)
- [Polar_Obs_Type](#)
- [Polar_Obs_List_Type](#)

Polar_Calc_Type

	Variable	Definition
Type :: Polar_Calc_Type		
Real(Kind=CP), Dimension(3)	H	Scattering vector in hkl
Real(Kind=CP), Dimension(3)	SPV	Second vector in Scattering plane apart of scattering vector to define plane
Type(Crystal_Cell_Type)	Cell	Unit Cell of Crystal
Real(Kind=CP)	P	Magnitude of initial polarisation vector
Complex, Dimension(3,2,24)	MIV	Magnetic interaction vector
Complex	NSF	Nuclear structure factor
Real(Kind=CP)	NC	Nuclear scattering contribution
Real(Kind=CP), Dimension(2,24)	MY	Magnetic contribution along y
Real(Kind=CP), Dimension(2,24)	MZ	Magnetic contribution along z
Real(Kind=CP), Dimension(2,24)	RY	Real part of nuclear magnetic interference term along y
Real(Kind=CP), Dimension(2,24)	RZ	Real part of nuclear magnetic interference term along z
Real(Kind=CP), Dimension(2,24)	IY	Imaginary part of nuclear magnetic interference term along y
Real(Kind=CP), Dimension(2,24)	IZ	Imaginary part of nuclear magnetic interference term along z
Real(Kind=CP), Dimension(2,24)	TC	Chiral contribution
Real(Kind=CP), Dimension(2,24)	MM	Magnetic-magnetic interference term
Real(Kind=CP), Dimension(3,2,24)	CS	Three different elastic cross-sections depending on the direction of the initial polar vector
Real(Kind=CP), Dimension(3,3)	PIJ	Polarisation tensor

End Type Polar_Calc_Type

Polar_Calc_List_Type

	Variable	Definition
Type :: Polar_Calc_List_Type		
Integer	Nref	Number of reflections
Type(Polar_Calc_Type), Dimension(:), Allocatable	Polari	Calculated Polarisation tensor for the Reflection List
End Type Polar_Calc_List_Type		

Polar_Info_Type

	Variable	Definition
Type :: Polar_Info_Type		
Real(Kind=CP), Dimension(3)	H	Scattering vector in hkl
Real(Kind=CP), Dimension(3)	SPV	Second vector in Scattering plane apart of scattering vector to define plane
Type(Crystal_Cell_Type)	Cell	Unit Cell of Crystal
Real(Kind=CP)	P	Magnitude of initial polarisation vector
Complex, Dimension(3)	MIV	Magnetic interaction vector
Complex	NSF	Nuclear structure factor
Real(Kind=CP)	NC	Nuclear scattering contribution
Real(Kind=CP)	MY	Magnetic contribution along y
Real(Kind=CP)	MZ	Magnetic contribution along z
Real(Kind=CP)	RY	Real part of nuclear magnetic interference term along y
Real(Kind=CP)	RZ	Real part of nuclear magnetic interference term along z
Real(Kind=CP)	IY	Imaginary part of nuclear magnetic interference term along y
Real(Kind=CP)	IZ	Imaginary part of nuclear magnetic interference term along z
Real(Kind=CP)	TC	Chiral contribution
Real(Kind=CP)	MM	Magnetic-magnetic interference term
Real(Kind=CP), Dimension(3)	CS	Three different elastic cross-sections depending on the direction of the initial polar vector
Real(Kind=CP), Dimension(3,3)	PIJ	Polarisation tensor
End Type Polar_Info_Type		

Polar_Obs_Type

	Variable	Definition
Type :: Polar_Obs_Type		
Real(Kind=CP), Dimension(3)	H	Scattering vector in hkl
Real(Kind=CP), Dimension(3,3)	OPIJ	
Real(Kind=CP), Dimension(3,3)	SOPIJ	
End Type Polar_Obs_Type		

Polar_Obs_List_Type

	Variable	Definition
Type :: Polar_Obs_List_Type		
Integer	Nref	Number of reflections
Type(Polar_Obs_Type), Dimension(:), Allocatable	Polaro	Observed Polarisation tensor for the Reflection List
End Type Polar_Obs_List_Type		

Subroutines

- [Calc_Polar_Dom](#)
- [Set_Polar_Info](#)
- [Write_Polar_Info](#)
- [Write_Polar_Line](#)

Calc_Polar_Dom

Subroutine Calc_Polar_Dom (Cell, H, SPV, Pin, NSF, Mag_Dom, MH, Polari)

Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell Parametrs
Real(Kind=CP), Dimension (3)	Intent(in)	H	Scattering vector in hkl
Real(Kind=CP), Dimension (3)	Intent(in)	SPV	Second Scattering plane vector in hkl
Real(Kind=CP)	Intent(in)	Pin	Magnitude of initial polarisation
Complex	Intent(in)	NSF	Nuclear Scattering Factor
Type	Intent(in)	Mag_Dom	

(Magnetic_Domain_Type)			
Type(MagHD_TYPE)	Intent(in out)	MH	Magnetic interaction vector
Type(Polar_Info_Type)	Intent(out)	Polari	All information about polarisation in one point hkl

Calculates Polarization matrix for domain case

Set_Polar_Info

Subroutine Set_Polar_Info (Cell, H, SPV, Pin, NSF, MIV, Polari)

Type(Crystal_Cell_Type)	Intent(in)	Cell	Cell Parametrs
Real(Kind=CP), Dimension (3)	Intent(in)	H	Scattering vector in hkl
Real(Kind=CP), Dimension (3)	Intent(in)	SPV	Second Scattering plane vector in hkl
Real(Kind=CP)	Intent(in)	Pin	Magnitude of initial polarisation
Complex	Intent(in)	NSF	Nuclear Scattering Factor
Complex, Dimension (3)	Intent(in)	MIV	Magnetic interaction vector
Type(Polar_Info_Type)	Intent(out)	Polari	All information about polarisation in one point hkl

Initializes the variable **Polari** using the type [Polar_Info_Type](#)

Write_Polar_Info

Subroutine Write_Polar_Info (Polari, Lun, Info)

Type(Polar_Info_Type)	Intent(in)	Polari	Polarisation in one point hkl
Integer, Optional	Intent(in)	Lun	Unit to write
Character (Len=*), Optional	Intent(in)	Info	Values are: P : also print information about coordinate frame C : also print information about crystal B : also print information about both

Outputs the polarisation info type in nice form

Write_Polar_Line

Subroutine Write_Polar_Line (Polari, Lun)

Type(Polar_Info_Type)	Intent(in)	Polari	Polarisation in one point hkl
Integer, Optional	Intent(in)	Lun	Unit to write

outputs the polarization info type in line form, so you can write it to a file