In[3]:=
```
<< QMRITools`
```

In[9]:=
```
Column@QMRIToolsPackages[]
```

Out[9]=
```
CardiacTools
CoilTools
DenoiseTools
DixonTools
ElastixTools
GeneralTools
GradientTools
ImportTools
IVIMTools
JcouplingTools
MaskingTools
NiftiTools
PhysiologyTools
PlottingTools
ProcessingTools
RelaxometryTools
SimulationTools
TensorTools
VisteTools
```

In[10]:=
```
QMRIToolsFunctions[60]
```

Functions

ADCCalc
AddNoise
AlignRespLog
AngleCalc
AngleMap
AnisoFilterTensor
ApplyCrop
AutoCropData
BayesianIVIMFit2
BayesianIVIMFit3
BlochSeries
Bmatrix
BmatrixCalc
BmatrixConv
BmatrixInv
BmatrixRot
BmatrixToggle
BullseyePlot
BvalRead
CalculateGfactor
CalculateMoments
CalculateWallMap
CalibrateEPGT2Fit
CardiacCoordinateSystem
CardiacSegment
CentralAxes
ClearTemporaryVariables
CoilSNRCalc
ColorFAPlot
CompilebleFunctions
CompressNiiFiles
ConcatenateDiffusionData
ConditionNumberCalc
ConvertGrads
Correct
CorrectBmatrix
CorrectGradients
CorrectJoinSetMotion
CorrectParMap
CreateDiffData
CreateHeart
CreateT2Dictionary
CropData
CutData
Data2DToVector
Data3DToVector
DataTransformation
DatRead
DatTot
DatTotXLS
DatWrite
DcmToNii
DeNoise
Deriv
DevideNoZero
DictionaryMinSearch
DixonReconstruct
DixonToPercent
DriftCorrect
DTItoolExp
DTItoolExpFile
DTItoolExpInd
DTItoolExpTens
ECalc
ECVCalc
EigensysCalc
EigenvalCalc
EigenvecCalc
EnergyCalc
EPGSignal
EPGT2Fit
ErrorPlot
ExcludeSlices
ExpNoZero
ExportBmat
ExportBval
ExportBvec
ExportNii
ExportVol
ExtractNiiFiles
FACalc
FConvert
FConverti
FiberDensityMap
FiberLengths
FileSelect
FinalGrads
FindCoilPosition
FindCrop
FindMaxDimensions
FindOrder
FindOutliers
FitData
FracCorrect
FullGrad
GenerateGradients
GenerateGradientsGUI
GetGradientScanOrder
GetMaskData
GetMaskMeans
GetPulseProfile
GetSliceData
GetSliceNormal
GetSliceNormalDir
GetSlicePositions
GetSpinSystem
GfactorSimulation
GradBmatrix
GradientPlot
GradRead
GradSeq
GridData
GridData3D
HelixAngleCalc
Hist
Hist2
HistogramPar
HomoginizeData
ImportBmat
ImportBval
ImportBvalvec
ImportBvec
ImportDTI
ImportExploreDTItens
ImportGradObj
ImportNii
ImportNiiDiff
ImportNiiDix
ImportNiiT1
ImportNiiT2
ImportPhyslog
ImportRespirect
ImportVol
InvertDataset
IVIMCalc
IVIMCorrectData
IVIMFunction
IVIMResiduals
JoinSets
LapFilter
ListSpherePlot
LoadCoilSetup
LoadCoilTarget
LoadFiberTracts
LogNoZero
MADNoZero
MakeCoilLayout
MakeECVBloodMask
MakeNoisePlots
MakeSliceImages
MakeWeightMask
Mask
MaskData
MaskHelix
MeanNoZero
MeanRange
MeanSignal
MeanStd
MedCouple
MedianNoZero
MemoryUsage
MergeSegmentations
NNLeastSquares
NoiseCorrelation
NoiseCovariance
NonLinearEPGFit
NormalizeData
NumberTableForm
OverPlusCalc
PadToDimensions
ParameterCalc
ParameterFit
ParameterFit2
PCADeNoise
PCAFitEq
PCAFitHist
PhaseAlign
PlotContour
PlotCorrection
PlotData
PlotData3D
PlotDefGrid
PlotDuty
PlotIVIM
PlotMoments
PlotPhyslog
PlotRespiract
PlotSegmentMask
PlotSegments
PlotSequence
PlotSimulation
PlotSimulationAngle
PlotSimulationAngleH
PlotSimulationHist
PlotSimulationVec
PlotSpectrum
Pulses
QMRIToolsFuncPrint
QMRIToolsFunctions
QMRIToolsPackages
RadialSample
ReadBrukerDiff
ReadBvalue
ReadDicom
ReadDicomDiff
ReadDicomDir
ReadDicomDirDiff
ReadGradients
ReadTransformParamet
ReadVoxSize
RegisterCardiacData
RegisterData
RegisterDataSplit
RegisterDataTransfor
RegisterDataTransfor
RegisterDiffusionDat
RegisterDiffusionDat
RemoveIsoImages
RemoveMaskOverlaps
RescaleData
RescaleSegmentation
ResidualCalc
ReverseCrop
RMSNoZero
ROIMask
SaveImage
SegmentMask
SequencePulseAcquire
SequenceSpinEcho
SequenceSteam
SequenceTSE
SetupDataStructure
ShiftPar
SigmaCalc
Signal
SimAddPhase
SimAngleParameters
SimEvolve
SimHamiltonian
SimParameters

Options

AffineDirections
AnisoFilterSteps
AnisoKappa
AnisoStepTime
AnisoWeightType
AxesLabel
AxesMethod
BackgroundValue
BinaryType
BloodMaskRange
BmatrixOut
BsplineDirections
BsplineSpacing
BullPlotMethod
CenterFrequency
ChainSteps
CoilArrayPlot
CoilSurfaceVoxelSize
ColorFunction
ColorFunction
ColorFunction
ColorValue
CompressNii
ConditionCalc
ContourStyle
ConvertDcm
CorrectPar
CropInit
CropOutput
CropPadding
CutOffMethod
DeleteTempDirectory
DeNoiseIterations
DeNoiseKernel
DeNoiseMonitor
DictB1Range
DictT2fRange
DictT2Range
DistanceMeasure
Distribution
DixonAmplitudes
DixonFieldStrength
DixonFilterInput
DixonFilterOutput
DixonFilterSize
DixonFrequencies
DixonIterations
DixonMaskThreshhold
DixonPrecessions
DixonTollerance
DropSamples
DropSlices
EPGCalibrate
EPGFitPoints
EPGMethod
EPGRelaxPars
EPGSmoothB1
FieldStrength
FileType
FilterMaps
FilterShape
FilterSize
FilterType
FindTransform
FitConstrains
FitFunction
FitOutput
FitSigma
FixPseudoDiff
FixPseudoDiffSD
FlipAxes
FlipBvec
FlipGrad
FullOutput
FullSphere
GetMaskOutput
GOutput
GradType
GRegularization
GridLineSpacing
HelixMethod
HistogramBins
HistogramBinsA
ImageLegend
ImageResolution
ImageSize
ImageSize
ImageSize
ImageSize
ImageSize
InterpolationOrder
InterpolationOrder
InterpolationOrderReg
InterpolationOrderRegA
Iterations
IterationsA
IVIMComponents
IVIMConstrained
IVIMConstrains
IVIMFixed
IVIMTensFit
JoinSetSplit
LineStep
LineThreshold
Linewidth
LinewidthShape
MagnetizationVector
MakeCheckPlot
MaskClosing
MaskCompartment
MaskComponents
MaskFiltKernel
MaskSmoothing
MaskWallMap
MeanMethod
MeanRes
Method
Method
Method
Method
Method
MethodReg
MethodReg
MethodRegA
MonitorCalc
MonitorEPGFit
MonitorIVIMCalc
MonitorUnwrap
MotionCorrectSets
NiiDataType
NiiMethod
NiiScaling
NoiseSize
NormalizeIVIM
NormalizeSets
NormalizeSignal
NumberSamples
NumberSamplesA
OrderSpan
OutlierIncludeZero
OutlierIterations
OutlierMethod
OutlierOutput
OutlierRange
OutputCalibration
OutputCheckImage
OutputCoilSurface
OutputImage
OutputMethod
OutputPlot
OutputSamples
OutputSNR
OutputTransformation
OutputType
OutputWeights
PadDirection
PaddOverlap
PadValue
Parallelize
Parallelize
PCAComponents
PCAFitParameters
PCAKernel
PCAOutput
PCATollerance
PCAWeighting
PeakNumber
PhaseEncoding
PlotColor
PlotLabel
PlotLabel
PlotRange
PlotRange
PlotRange
PlotRange
PlotRange
PlotSolution
PlotSpace
PlotStyle
PositiveZ
PrintTempDirectory
RadialSamples
ReadoutBandwith
ReadoutOutput
ReadoutPhase
ReadoutSamples
RegistrationTarget
Reject
Reject
RejectMap
ReportFits
Resolutions
ResolutionsA
ReverseData
ReverseSets
RobustFit
RobustFitParameters
RotateGradient
RotateGradients
RotationCorrect
RowSize
Runs
SampleStep
ScaleCorrect
Scaling
SeedDensity
ShowFit
ShowHelixPlot
SliceRange
SliceRangeSamples
SmartMaskOutput
SmartMethod
SmoothHelix
SmoothSNR
SortVecs
SpectrumColor
SphereColor
SphereSize
SplitMethod
StartPoints
StartSlices
Steps
StepSizeI
Strictness
SwitchAxes
TableAlignments
TableDepth
TableDirections
TableHeadings
TableMethod
TableSpacing
TempDirectory
TensOutput
TextOffset
TextSize
UnitMulti

In[12]:= **QMRIToolsFunctions["All", 8]**

CardiacTools

Functions

BullseyePlot
CalculateWallMap
CardiacCoordinateSystem
CardiacSegment
CentralAxes
CreateHeart
ECVCalc
ExcludeSlices
HelixAngleCalc
MakeECVBloodMask
MaskHelix
PlotSegmentMask
PlotSegments
RadialSample
TransmuralPlot

Options

AxesMethod
BackgroundValue
BloodMaskRange
BullPlotMethod
ColorFunction
CutOffMethod
DistanceMeasure
DropSamples
GridLineSpacing
HelixMethod
ImageSize
LineStep
LineThreshold
MaskWallMap
Method
OutputCheckImage
PlotLabel
PlotRange
PlotStyle
RadialSamples
RowSize
ShowFit
ShowHelixPlot
SmoothHelix
StartPoints
StartSlices
TextOffset
TextSize

CoilTools

Functions

CoilSNRCalc
FindCoilPosition
LoadCoilSetup
LoadCoilTarget
MakeCoilLayout
MakeNoisePlots
MakeWeightMask
NoiseCorrelation
NoiseCovariance

Options

CoilArrayPlot
CoilSurfaceVoxelSize
ColorFunction
ImageSize
OutputCoilSurface
PlotRange

DenoiseTools

Functions

AnisoFilterTensor
DeNoise
PCADeNoise
PCAFitEq
PCAFitHist
WeightMapCalc

Options

AnisoFilterSteps
AnisoKappa
AnisoStepTime
AnisoWeightType
DeNoiseIterations
DeNoiseKernel
DeNoiseMonitor
FitSigma
Method
PCAFitParameters
PCAKernel
PCAOutput
PCATollerance
PCAWeighting
PlotSolution

DixonTools

Functions

DixonReconstruct
DixonToPercent
SimulateDixonSignal
Unwrap
UnwrapSplit

Options

DixonAmplitudes
DixonFieldStrength
DixonFilterInput
DixonFilterOutput
DixonFilterSize
DixonFrequencies
DixonIterations
DixonMaskThreshhold
DixonPrecessions
DixonTollerance
MonitorUnwrap
UnwrapDimension

ElastixTools

Functions

ReadTransformParameters TransformData
RegisterCardiacData
RegisterData
RegisterDataSplit
RegisterDataTransform
RegisterDataTransformSplit
RegisterDiffusionData
RegisterDiffusionDataSplit

Options

AffineDirections InterpolationOrderRegA OutputTransformation UseGPU
BsplineDirections Iterations PCAComponents
BsplineSpacing IterationsA PrintTempDirectory
DeleteTempDirectory MethodReg RegistrationTarget
FindTransform MethodRegA Resolutions
HistogramBins NumberSamples ResolutionsA
HistogramBinsA NumberSamplesA SplitMethod
InterpolationOrderReg OutputImage TempDirectory

GeneralTools

Functions

ApplyCrop DevideNoZero LogNoZero QMRIToolsFunctions Sum
AutoCropData ExpNoZero MADNoZero QMRIToolsPackages Ten
ClearTemporaryVariables FileSelect MeanNoZero RescaleData Ten
CompilebleFunctions FindCrop MedianNoZero ReverseCrop Tra
CropData FindMaxDimensions MemoryUsage RMSNoZero Vec
CutData GridData NNLeastSquares SaveImage
Data2DToVector GridData3D PadToDimensions StdFilter
Data3DToVector LapFilter QMRIToolsFuncPrint StichData

Options

CropInit PadDirection
CropOutput PadValue
CropPadding WindowTitle
FileType
ImageResolution
ImageSize
InterpolationOrder
OutputWeights

GradientTools

Functions

Bmatrix
BmatrixCalc
BmatrixConv
BmatrixInv
BmatrixRot
BmatrixToggle
CalculateMoments
ConditionNumberCalc
ConvertGrads
CorrectBmatrix
CorrectGradients
EnergyCalc
FinalGrads
FindOrder
FullGrad
GenerateGradients
GenerateGradientsGUI
GetGradientScanOrder
GetSliceNormal
GetSliceNormalDir
GradBmatrix
GradSeq
ImportGradObj
OverPlusCalc
UniqueBvalPosition

Options

ConditionCalc
FlipAxes
FlipGrad
FullSphere
GradType
Method
MethodReg
OrderSpan
OutputPlot
OutputType
PhaseEncoding
Runs
Steps
StepSizeI
SwitchAxes
UnitMulti
UseGrad
VisualOpt

ImportTools

Functions

BvalRead
GradRead
ReadBrukerDiff
ReadBvalue
ReadDicom
ReadDicomDiff
ReadDicomDir
ReadDicomDirDiff
ReadGradients
ReadVoxSize
ShiftPar

Options

BmatrixOut
ConvertDcm
RotateGradient
ScaleCorrect

IVIMTools

Functions

BayesianIVIMFit2
BayesianIVIMFit3
CorrectParMap
FConvert
FConverti
FracCorrect
HistogramPar
IVIMCalc
IVIMCorrectData
IVIMFunction
IVIMResiduals
ThetaConv
ThetaConvi

Options

ChainSteps
CorrectPar
FilterMaps
FilterSize
FilterType
FitConstrains
FixPseudoDiff
FixPseudoDiffSD
IVIMComponents
IVIMConstrained
IVIMConstrains
IVIMFixed
IVIMTensFit
Method
MonitorIVIMCalc
OutputSamples
Parallelize
UpdateStep

JcouplingTools

Functions

GetSpinSystem
PhaseAlign
PlotSpectrum
SequencePulseAcquire
SequenceSpinEcho
SequenceSteam
SequenceTSE
SimAddPhase
SimEvolve
SimHamiltonian
SimReadout
SimRotate
SimSignal
SimSpoil
SysTable

Options

CenterFrequency
FieldStrength
Linewidth
LinewidthShape
PlotRange
ReadoutBandwith
ReadoutOutput
ReadoutPhase
ReadoutSamples
SpectrumColor

MaskingTools

Functions

GetMaskData
HomoginizeData
Mask
MaskData
MeanSignal
MergeSegmentations
NormalizeData
RemoveMaskOverlaps
RescaleSegmentation
ROIMask
SegmentMask
SmoothMask
SmoothSegmentation
SplitSegmentations

Options

GetMaskOutput
MaskClosing
MaskComponents
MaskFiltKernel
MaskSmoothing
UseMask

NiftiTools

Functions

CompressNiiFiles
DcmToNii
ExportBmat
ExportBval
ExportBvec
ExportNii
ExtractNiiFiles
ImportBmat
ImportBval
ImportBvalvec
ImportBvec
ImportExploreDTItens
ImportNii
ImportNiiDiff
ImportNiiDix
ImportNiiT1
ImportNiiT2

Options

CompressNii
FlipBvec
Method
NiiDataType
NiiMethod
NiiScaling
RotateGradients

PhysiologyTools

Functions

AlignRespLog
ImportPhyslog
ImportRespirect
PlotPhyslog
PlotRespiract

Options

OutputMethod
SampleStep

PlottingTools

Functions

GetSliceData
GetSlicePositions
GradientPlot
ListSpherePlot
MakeSliceImages
PlotContour
PlotCorrection
PlotData
PlotData3D
PlotDefGrid
PlotDuty
PlotIVIM
PlotMoments
PlotSequence

Options

ColorFunction
ContourStyle
DropSlices
ImageLegend
ImageSize
MakeCheckPlot
Method
NormalizeIVIM
PeakNumber
PlotColor
PlotRange
PlotSpace
PositiveZ
SphereColor
SphereSize

ProcessingTools

Functions

CorrectJoinSetMotion
DataTransformation
DatTot
DatTotXLS
ErrorPlot
FiberDensityMap
FiberLengths
FindOutliers
FitData
GetMaskMeans
Hist
Hist2
InvertDataset
JoinSets
MeanRange
MeanStd
MedCouple
NumberTableForm
ParameterFit
ParameterFit2
SetupDataStructure
SmartMask
SNRCalc
SNRMapCalc
SplitSets

Options

AxesLabel
ColorValue
FitFunction
FitOutput
ImageSize
InterpolationOrder
JoinSetSplit
MaskCompartment
MeanMethod
Method
MotionCorrectSets
NormalizeSets
OutlierIncludeZero
OutlierIterations
OutlierMethod
OutlierOutput
OutlierRange
OutputSNR
PaddOverlap
PlotLabel
ReverseData
ReverseSets
Scaling
SeedDensity
SmartMaskOutput
SmartMethod
SmoothSNR
Strictness
TableAlignments
TableDepth
TableDirections
TableHeadings
TableMethod
TableSpacing

RelaxometryTools

Functions

CalibrateEPGT2Fit
CreateT2Dictionary
DictionaryMinSearch
EPGSignal
EPGT2Fit
NonLinearEPGFit
T1Fit
T1rhoFit
T2Fit
TriExponentialT2Fit

Options

DictB1Range
DictT2fRange
DictT2Range
EPGCalibrate
EPGFitPoints
EPGMethod
EPGRelaxPars
EPGSmoothB1
Method
MonitorEPGFit
OutputCalibration

SimulationTools

Functions

AddNoise
BlochSeries
CalculateGfactor
CreateDiffData
GetPulseProfile
GfactorSimulation
PlotSimulation
PlotSimulationAngle
PlotSimulationAngleHist
PlotSimulationHist
PlotSimulationVec
Pulses
Signal
SimAngleParameters
SimParameters
SimulateSliceEPG
Tensor

Options

GOutput
GRegularization
MagnetizationVector
NoiseSize
PlotRange
Reject
ReportFits
SliceRange
SliceRangeSamples
SortVecs
TensOutput

TensorTools

Functions

ADCCalc	ECalc	SigmaCalc
AngleCalc	EigensysCalc	SortDiffusionData
AngleMap	EigenvalCalc	TensorCalc
ColorFAPlot	EigenvecCalc	TensorCorrect
ConcatenateDiffusionData	FACalc	
Correct	ParameterCalc	
Deriv	RemoveIsoImages	
DriftCorrect	ResidualCalc	

Options

Distribution	Reject
FilterShape	RejectMap
FullOutput	RobustFit
MeanRes	RobustFitParameters
Method	RotationCorrect
MonitorCalc	UseMask
NormalizeSignal	
Parallelize	

Viste Tools

Functions

DatRead	ImportVol
DatWrite	LoadFiberTracts
DTItoolExp	
DTItoolExpFile	
DTItoolExpInd	
DTItoolExpTens	
ExportVol	
ImportDTI	

Options

BinaryType

In[7]:= **QMRIToolsFuncPrint[]**

CardiacTools

Functions

BullseyePlot[data, segmask] generates a AHA–17 segement bullseye plot.

BullseyePlot[list] generates a AHA–17 segement
bullseye plot of the lists (which needs to have 17 values) provide.

data is a 3D volume used for the plot.

segmask is the AHA–17 segmentation
resulting form the CardiacSegment function when AHA17 is selected.

Output is a bullseye plot or a plotwindow, depending
on the Method which can be "Dynamic" else it will be static. ≫

```
Attributes[BullseyePlot] = {Protected, ReadProtected}

Options[BullseyePlot] = {TextOffset → 0.5, TextSize → 12, PlotRange → Automatic,
   ColorFunction → TemperatureMap, BullPlotMethod → Dynamic, ImageSize → 200}

SyntaxInformation[BullseyePlot] = {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

CalculateWallMap[mask,vox] calculates the wall distance map and the wall derivative.

Output is {wallmap, wallDerivative}. ≫

```
Attributes[CalculateWallMap] = {Protected, ReadProtected}

Options[CalculateWallMap] = {ShowFit → True, MaskWallMap → True}

SyntaxInformation[CalculateWallMap] = {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

QMRITools`CardiacTools`CardiacCoordinateSystem

```
Attributes[CardiacCoordinateSystem] = {Protected, ReadProtected}

Options[CardiacCoordinateSystem] = {ShowFit → False}

SyntaxInformation[CardiacCoordinateSystem] = {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

CardiacSegment[data, mask, off] allows to segment the heart in 1,
    4, 6 or AHA–17 segements for each slice 360 radial samples are generated.

data is a background image on which all overlays are projected.
mask is the mask of the left ventricle (same as
    used for CentralAxes) and defines the area in which the data is sampled.
off is the centerpoints generated by CentralAxes.

Output is {segmask, segang, {points, slices}}. ≫

```
Attributes[CardiacSegment] = {Protected, ReadProtected}

Options[CardiacSegment] =
 {StartPoints → Default, StartSlices → Default, LineThreshold → 0.25, LineStep → 0.5}

SyntaxInformation[CardiacSegment] = {ArgumentsPattern → {_, _, _, OptionsPattern[]}}
```

CentralAxes[mask, vox] calculates the center of the
    lumen from a mask of the left ventricle. vox is the voxels size, {slice, x, y}.
CentralAxes[mask, maskp, vox] allows for fancy visualization of the other structures using maskp.

Output is {centerpoints, normalvecs, inout} or {centerpoints, normalvecs, inout, fit}. ≫

```
Attributes[CentralAxes] = {Protected, ReadProtected}
```

```
Options[CentralAxes] = {ShowFit → True, RowSize → Automatic, AxesMethod → Cubic}
```

```
SyntaxInformation[CentralAxes] = {ArgumentsPattern → {_, _, _., OptionsPattern[]}}
```

CreateHeart[] creates a simulated left ventricle shape.
CreateHeart[pars] creates a simulated left ventricle shape with predifined parameters pars.

Output is the heart shape, the voxel size and
the parameters needed to generate the heart, {mask, vox, pars}.

```
Attributes[CreateHeart] = {Protected, ReadProtected}
```

```
SyntaxInformation[CreateHeart] = {ArgumentsPattern → {_.}}
```

ECVCalc[T1pre, T1post, hema] calculates the ECVmap using MakeECVBloodMask.
ECVCalc[T1pre, T1post, bloodMask, hema] calculates the ECVmap using bloodMask. ≫

```
Attributes[ECVCalc] = {Protected, ReadProtected}
```

ExcludeSlices[data] excludes slices that
do not look like the others based on various distance measures.

Output is an array with 1 or 0 with the dimensiosn {slices, diff dirs} ≫

```
Attributes[ExcludeSlices] = {Protected, ReadProtected}
```

```
Options[ExcludeSlices] = {CutOffMethod → Auto, DistanceMeasure → 5}
```

```
SyntaxInformation[ExcludeSlices] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

HelixAngleCalc[eigenvectors, mask, vox] calculates
the helix angle matrix of cardiac data using only a left ventricle mask.
HelixAngleCalc[eigenvectors, mask, vox] calculates the helix angle matrix of cardiac
data using only a left ventricle mask, and a maskp for visualization.
HelixAngleCalc[eigenvectors, mask, centerpoint, vec, inout, vox] calculates the
helix angle matrix of cardiac data using only a left ventricle mask.
HelixAngleCalc[eigenvectors, mask, maskp, centerpoint, vec, inout, vox] calculates the helix
angle matrix of cardiac data using a left vantricle mask and a maskp for visualization.

eigenvectors are the tensor eigenvectors calculated with EigenvecCalc.
mask is a mask of the left ventricle.
maskp is a mask used for visualization.
vox is the voxels size, {slice, x, y}.

The folowing values are calculated
automaticlay Using CentralAxes but can also be provided as an input.
centerpoint is the center of each slice calculated with CentralAxes.
inout is the inner and outer radius calcualted with CentralAxes.
vec is the vector describin the central axes of the heart, calculated with CentralAxes.

Output is het fiber angle matrix FAM = {9, slice, x, y} or {FAM, plot}. ≫

```
Attributes[HelixAngleCalc] = {Protected, ReadProtected}
```

```
Options[HelixAngleCalc] = {ShowHelixPlot → True, HelixMethod → Slow, AxesMethod → Quadratic}
```

```
SyntaxInformation[HelixAngleCalc] = {ArgumentsPattern → {_, _, _, _., OptionsPattern[]}}
```

MakeECVBloodMask[T1pre, T1post] makes a bloodpool mask based on the T1pre and
T1post images. It assumes that the hart is cropped with the blood in the center. ≫

```
Attributes[MakeECVBloodMask] = {Protected, ReadProtected}
```

```
Options[MakeECVBloodMask] = {BloodMaskRange → {1400, {0, 700}}, OutputCheckImage → True}
```

```
SyntaxInformation[MakeECVBloodMask] = {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

MaskHelix[helix, mask] masks helix angle data, sets
the background to –100 and allows for Median filter of the helix mask.
helix can be a singel map or the FAM.

Output is the masked helix angle data. ≫

```
Attributes[MaskHelix] = {Protected, ReadProtected}
```

```
Options[MaskHelix] = {BackgroundValue → -100, SmoothHelix → False}
```

```
SyntaxInformation[MaskHelix] = {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

PlotSegmentMask[mask, segmask, vox] plots the mask segements created by CardiacSegment.

mask is a mask the left ventricle that was used in the CardiacSegment.
segmask is the output of CardiacSegemnt.
vox is the voxels size, {slice, x, y}.

Output is a plot window. ≫

```
Attributes[PlotSegmentMask] = {Protected, ReadProtected}
```

```
SyntaxInformation[PlotSegmentMask] = {ArgumentsPattern → {_, _, _}}
```

PlotSegments[mask, data, segang] shows how the heart wil be sampled by RadialSample.

mask is a mask the left ventricle that was used in the CardiacSegment.
function and the segang is the output of the cardaic SegmentFunction.

Output is a plot window. ≫

```
Attributes[PlotSegments] = {Protected, ReadProtected}
```

```
Options[PlotSegments] = {RadialSamples → 10}
```

```
SyntaxInformation[PlotSegments] = {ArgumentsPattern → {_, _, _, OptionsPattern[]}}
```

RadialSample[mask, data, segang] radialy samples the provided parametermap data.

The mask should be a mask of the left ventricle that was used in the CardiacSegment.
segang is the output of the cardaic SegmentFunction.

Output is {points, vals} which are orderd as indicated by the user. ≫

```
Attributes[RadialSample] = {Protected, ReadProtected}
```

```
Options[RadialSample] = {RadialSamples → 10, DropSamples → 0}
```

```
SyntaxInformation[RadialSample] = {ArgumentsPattern → {_, _, _, OptionsPattern[]}}
```

TransmuralPlot[data] plots transmural profiles of the data which are created by RadialSample.

data can be a single profile or a list of profiles.
    In the second case the mean and standardeviations are plotted.

Output is a plot of the transmural profile. ≫

```
Attributes[TransmuralPlot] = {Protected, ReadProtected}
```

```
Options[TransmuralPlot] = {GridLineSpacing → 10, PlotStyle → RGBColor[1, 0, 0],
  PlotRange → Automatic, ImageSize → 300, Method → Median, PlotLabel → None}
```

```
SyntaxInformation[TransmuralPlot] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

Options

AxesMethod is an option for HelixAngleCalc and CentralAxes. Can be "Linear", "Quadratic", "Cubic". ≫

```
Attributes[AxesMethod] = {Protected, ReadProtected}
```

BackgroundValue is an option for MaskHelix. Sets the backgroud value (default is –100). ≫

```
Attributes[BackgroundValue] = {Protected, ReadProtected}
```

BloodMaskRange is an option for MakeECVBloodMask. ≫

```
Attributes[BloodMaskRange] = {Protected, ReadProtected}
```

BullPlotMethod is an option for BullseyePlot. Can be "Dynamic" of "Normal".
"Dynamic" allows to change plotting parameters in Manipulation window. ≫

```
Attributes[BullPlotMethod] = {Protected, ReadProtected}
```

ColorFunction is an option for graphics functions
that specifies a function to apply to determine colors of elements. ≫

```
Attributes[ColorFunction] = {Protected}
```

CutOffMethod is an option for ExcludeSlices. Default
value is "Auto" or it can be a fixed percentage (value between 0 and .5) ≫

```
Attributes[CutOffMethod] = {Protected, ReadProtected}
```

DistanceMeasure is an option for ExcludeSlices. Defaul value is 5. (1 ManhattanDistance, 2
SquaredEuclideanDistance, 3 EuclideanDistance, 4 Correlation, 5 SpearmanRho ≫

```
Attributes[DistanceMeasure] = {Protected, ReadProtected}
```

DropSamples is an option for RadialSample and PlotSegments. Defines how manny samples
are droped form star and end. Can be an number or set (strat, end) of numbers. ≫

```
Attributes[DropSamples] = {Protected, ReadProtected}
```

GridLineSpacing is an option of TransmuralPlot. It defines the spacing of the gridlines. ≫

```
Attributes[GridLineSpacing] = {Protected, ReadProtected}
```

HelixMethod is an option for HelixAngleCalc. Can be "Slow" or "Fast".
"Slow" uses wall distance interpolation and can take long for high res datasets.
"Fast" uses wall distance calculation using circular approximation of the ventricle. ≫

```
Attributes[HelixMethod] = {Protected, ReadProtected}
```

ImageSize is an option that specifies the overall size of an image to display for an object. ≫

```
Attributes[ImageSize] = {Protected}
```

LineStep is an option for CardiacSegment. ≫

```
Attributes[LineStep] = {Protected, ReadProtected}
```

LineThreshold is an option for CardiacSegment. Can be number between
0 and 1. Increasing the value will decrease the amount of wall sampled. ≫

```
Attributes[LineThreshold] = {Protected, ReadProtected}
```

MaskWallMap is an option for CalculateWallMap. if True or False. ≫

```
Attributes[MaskWallMap] = {Protected, ReadProtected}
```

Method is an option for various algorithm–intensive
functions that specifies what internal methods they should use. ≫

```
Attributes[Method] = {Protected}
```

QMRITools`CardiacTools`OutputCheckImage

```
Attributes[OutputCheckImage] = {Protected, ReadProtected}
```

PlotLabel is an option for graphics functions that specifies an overall label for a plot. ≫

```
Attributes[PlotLabel] = {Protected}
```

PlotRange is an option for graphics functions
that specifies what range of coordinates to include in a plot. ≫

```
Attributes[PlotRange] = {Protected, ReadProtected}
```

PlotStyle is an option for plotting and related
functions that specifies styles in which objects are to be drawn. ≫

```
Attributes[PlotStyle] = {Protected}
```

RadialSamples is an option for RadialSample and
PlotSegments. Defines how manny transmural samples are taken. ≫

```
Attributes[RadialSamples] = {Protected, ReadProtected}
```

RowSize is an option for CentralAxes. defines the number or images per showing the segmentation. Can be "Automatic" of an integer. ≫

```
Attributes[RowSize] = {Protected, ReadProtected}
```

ShowFit is an option for CentralAxes. True shows the fit of the central axes. ≫

```
Attributes[ShowFit] = {Protected, ReadProtected}
```

ShowHelixPlot is an option for HelixAngleCalc. If true the it
also outputs a visulization of the local myocardial coordinate system. ≫

```
Attributes[ShowHelixPlot] = {Protected, ReadProtected}
```

SmoothHelix is an option for MaskHelix, sets the kernelsize for the MedianFilter. ≫

```
Attributes[SmoothHelix] = {Protected, ReadProtected}
```

StartPoints is an option for CardiacSegment.
Value is "Default" or the point list given by CardiacSegment. ≫

```
Attributes[StartPoints] = {Protected, ReadProtected}
```

StartSlices is an option for CardiacSegment. Value is "Default" or the list given by CardiacSegment. ≫

```
Attributes[StartSlices] = {Protected, ReadProtected}
```

TextOffset is an option for BullseyePlot. Determines where the text is placed, can be 0 to 1. ≫

```
Attributes[TextOffset] = {Protected, ReadProtected}
```

TextSize is an option for BullseyePlot. Determines the text size. ≫

```
Attributes[TextSize] = {Protected, ReadProtected}
```

CoilTools

Functions

CoilSNRCalc[coils, noise] calculates the sensitivity
weighted snr of multiple coil elements using magnitude signal and noise.

Output is {data, noise, sos, snr, sigmap, weights}. ≫

```
Attributes[CoilSNRCalc] = {Protected, ReadProtected}
```

```
SyntaxInformation[CoilSNRCalc] = {ArgumentsPattern → {_, _}}
```

FindCoilPosition[weights] finds the coil posision by locating the highest intensity location in
the coil weight map, which can be obtianed by LoadCoilSetup or SumOfSquares.
Internally it uses MakeWeightMask to remove the noise of the weightmasks.
FindCoilPosition[weights, mask] limits the search region to the provided mask. ≫

```
Attributes[FindCoilPosition] = {Protected, ReadProtected}
```

```
Options[FindCoilPosition] = {OutputCoilSurface → False, CoilSurfaceVoxelSize → {1, 1, 1}}
```

```
SyntaxInformation[FindCoilPosition] = {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

LoadCoilSetup[file] load a very specific type of coil experiment, a dynmic
scan with a setup of which the second dynamic is a noise measurement.
The input file is the Nii file that conatins the individualy reconstruted coil images and the noise data.
Internaly it uses CoilSNRCalc and SumOfSquares.

Output is the coil data with coil noise data and snrmap based on
the SumOfSquares addition, the SOS reconstruction and the SOS weights.
{dataC, noiseC, sosC, snrC, sigmapC, weights, vox}. ≫

```
Attributes[LoadCoilSetup] = {Protected, ReadProtected}
```

```
SyntaxInformation[LoadCoilSetup] = {ArgumentsPattern → {_}}
```

LoadCoilTarget[file] loads a very specific typ of experiment,
a dynamic scan with with the second dynmaic is a noise measuremnt.
The input file is the Nii file that conatins the scanner reconstruction and the noise data.
Internaly it uses SNRMapCalc,

Output is the reconstructed data with noise data
and snrMap {dataC, noiseC, sosC, snrC, sigmapC, weights, vox}. ≫

```
Attributes[LoadCoilTarget] = {Protected, ReadProtected}
```

```
SyntaxInformation[LoadCoilTarget] = {ArgumentsPattern → {_}}
```

MakeCoilLayout[{name, size, number}] makes a coil
grid with label name, partioned in size rows and with label number.
MakeCoilLayout[{name, size, number}, val] makes a coil grid with label name,
partioned in size rows and with label the val at location number.
MakeCoilLayout[{coils..}] same but for multile coils grids. Each coil
grid is defined as {name, size, number}.
MakeCoilLayout[{coils..}, val] savem but for multiple coil grids. ≫

```
Attributes[MakeCoilLayout] = {Protected, ReadProtected}
```

```
Options[MakeCoilLayout] =
 {PlotRange → Automatic, ColorFunction → SunsetColors, ImageSize → 100, CoilArrayPlot → False}
```

MakeNoisePlots[noise] returns a grid of plots of the noise per channel
MakeNoisePlots[noise, {met, prt}] met can be "Grid" with
prt a number or Automatic. Else all plots will be returend as a list of plots.
MakeNoisePlots[noise, {met, prt}, sub] sub defines how much the noise is
subsampled, default is 40 (every 40th sample is used in plot). ≫

```
Attributes[MakeNoisePlots] = {Protected, ReadProtected}
```

```
SyntaxInformation[MakeNoisePlots] = {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

MakeWeightMask[weights] creates a mask
of homogeneous regions of weightmaps removing the noise. ≫

```
Attributes[MakeWeightMask] = {Protected, ReadProtected}
```

```
SyntaxInformation[MakeWeightMask] = {ArgumentsPattern → {_}}
```

NoiseCorrelation[noise] calculates the noise correlation matrix, noise is {nrCoils, noise Samples}. ≫

```
Attributes[NoiseCorrelation] = {Protected, ReadProtected}
```

```
SyntaxInformation[NoiseCorrelation] = {ArgumentsPattern → {_}}
```

NoiseCovariance[noise] calculates the noise covariance matrix, noise is {nrCoils, noise Samples}. ≫

```
Attributes[NoiseCovariance] = {Protected, ReadProtected}
```

```
SyntaxInformation[NoiseCovariance] = {ArgumentsPattern → {_}}
```

Options

CoilArrayPlot is an option for MakeCoilLayout. If True
and values are provided it makes an arrayplot of the coil layouts ≫

```
Attributes[CoilArrayPlot] = {Protected, ReadProtected}
```

CoilSurfaceVoxelSize is an option for
FindCoilPosition. Specifies the voxel size used for OutputCoilSurface. ≫

```
Attributes[CoilSurfaceVoxelSize] = {Protected, ReadProtected}
```

ColorFunction is an option for graphics functions
that specifies a function to apply to determine colors of elements. ≫

```
Attributes[ColorFunction] = {Protected}
```

ImageSize is an option that specifies the overall size of an image to display for an object. ≫

```
Attributes[ImageSize] = {Protected}
```

OutputCoilSurface is an option for FindCoilPosition. If
set true it will also output a SurfacePlot of the coil location volume. ≫

```
Attributes[OutputCoilSurface] = {Protected, ReadProtected}
```

PlotRange is an option for graphics functions
that specifies what range of coordinates to include in a plot. ≫

```
Attributes[PlotRange] = {Protected, ReadProtected}
```

DenoiseTools

Functions

AnisoFilterTensor[tens, diffdata] Filter the
tensor tens using an anisotropic diffusion filter (Perona–Malik).
It uses the diffusion weighted data diffdata to find edges that are not visible in the tensor.
Edge weights based on the diffusion data are averaged over all normalized diffusion direction.

Output is the smoothed tensor. ≫

```
Attributes[AnisoFilterTensor] = {Protected, ReadProtected}
```

```
Options[AnisoFilterTensor] =
 {AnisoWeightType → 2, AnisoKappa → 5., AnisoStepTime → 1, AnisoFilterSteps → 5}
```

```
SyntaxInformation[AnisoFilterTensor] = {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

DeNoise[data,sigma,filtersize] removes Rician noise with standard deviation "sigma"
from the given dataset using a kernel with size "filtersize" a gaussian kernel.
DeNoise[data,sigma,filtersize, Kernel–>"kerneltype"] removes Rician noise with standard deviation
"sigma" from the given dataset using a kernel with size "filtersize" and type "kerneltype".

Output is data denoised. ≫

```
Attributes[DeNoise] = {Protected, ReadProtected}
```

```
Options[DeNoise] = {DeNoiseKernel → Gaussian, DeNoiseMonitor → False, DeNoiseIterations → 1}
```

```
SyntaxInformation[DeNoise] = {ArgumentsPattern → {_, _, _, OptionsPattern[]}}
```

PCADeNoise[data] removes rician noise from the data with PCA.
PCADeNoise[data, mask] removes rician noise from the data with PCA only withing the mask.
PCADeNoise[data, mask, sig] removes rician noise from the data
with PCA only withing the mask using sig as prior knowledge or fixed value.

Output is de {data denoise, sigma map} by default if PCAOutput is Full
then fitted {data dnoise , {sigma fit, average sigma}, {number components,
number of fitted voxesl, number of max fits}, total fit –time per 500 ittt}. ≫

```
Attributes[PCADeNoise] = {Protected, ReadProtected}
```

```
Options[PCADeNoise] = {PCAKernel → 5, PCAFitParameters → {10, 6, 10}, FitSigma → False,
  PCAOutput → Full, Method → Equation, PCATollerance → 0, PCAWeighting → True}
```

```
SyntaxInformation[PCADeNoise] = {ArgumentsPattern → {_, _., _., OptionsPattern[]}}
```

PCAFitEq[data] fits the marchencopasteur distribution to the PCA of the data using grid search.
PCAFitEq[data, sig] fits the marchencopasteur distribution to
the PCA of the data using sig as start value or fixed value using grid search.

Output is {simga, number of noise comp, and denoised matrix}. ≫

```
Attributes[PCAFitEq] = {Protected, ReadProtected}
```

```
SyntaxInformation[PCAFitEq] = {ArgumentsPattern → {_, _, _., _.}}
```

PCAFitHist[data] fits the marchencopasteur distribution to the PCA of the data using hist fit.
PCAFitHist[data, sig] fits the marchencopasteur distribution
to the PCA of the data using sig as start value or fixed value using hist fit.

Output is {simga, number of noise comp, and denoised matrix, itterations}. ≫

```
Attributes[PCAFitHist] = {Protected, ReadProtected}
```

```
Options[PCAFitHist] = {PlotSolution → False, FitSigma → True, PCAFitParameters → {10, 6, 10}}
```

```
SyntaxInformation[PCAFitHist] = {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

WeightMapCalc[diffdata] calculates a weight map which is used in AnisoFilterTensor.

Output is a weight map of the diffdata which is high in isotropic regions and low at edges. ≫

```
Attributes[WeightMapCalc] = {Protected, ReadProtected}
```

```
Options[WeightMapCalc] = {AnisoWeightType → 2, AnisoKappa → 10.}
```

```
SyntaxInformation[WeightMapCalc] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

Options

AnisoFilterSteps is an option for AnisoFilterTensor and
defines the amoutn of diffusin steps taken. Higher is more smoothing ≫

```
Attributes[AnisoFilterSteps] = {Protected, ReadProtected}
```

AnisoKappa is an option for AnisoFilterTensor and WeightMapCalc and
defines the weighting strenght, all data is normalize to 100 before filetering. ≫

```
Attributes[AnisoKappa] = {Protected, ReadProtected}
```

AnisoStepTime is an option for AnisoFilterTensor
and defines the diffusion time, when small more step are needed. ≫

```
Attributes[AnisoStepTime] = {Protected, ReadProtected}
```

AnisoWeightType is an option for AnisoFilterTensor and WeightMapCalc and defines
the weighting, eigher 1, the exponent of (–g/kappa) or 2, 1/(1+g/kappa). ≫

```
Attributes[AnisoWeightType] = {Protected, ReadProtected}
```

DeNoiseIterations is and option for DeNoise. Specifies the number of the denoising iterations. ≫

```
Attributes[DeNoiseIterations] = {Protected, ReadProtected}
```

DeNoiseKernel is and option for DeNoise. Values can be "Disk", "Box" or "Gaussian". ≫

```
Attributes[DeNoiseKernel] = {Protected, ReadProtected}
```

DeNoiseMonitor is and option for DeNoise. Monitor the denoising progres. ≫

```
Attributes[DeNoiseMonitor] = {Protected, ReadProtected}
```

FitSigma is an option of PCAFitHist, PCAFitEq and
PCADeNoise, if set True sig is fitted if set False sigma is fixed to input value. ≫

```
Attributes[FitSigma] = {Protected, ReadProtected}
```

Method is an option for various algorithm–intensive
functions that specifies what internal methods they should use. ≫

```
Attributes[Method] = {Protected}
```

PCAFitParameters is an option of PCADeNoise and PCAFitHist. {nb, pi,
maxit} = bins, initial signal components, maximum number of itterations. ≫

```
Attributes[PCAFitParameters] = {Protected, ReadProtected}
```

PCAKernel is an option of PCADeNoise. It sets the kernel size. ≫

```
Attributes[PCAKernel] = {Protected, ReadProtected}
```

PCAOutput is an option of PCADeNoise. If output is full the
output is {datao, {output[[1]], sigmat}, {output[[2]], output[[3]], j}, timetot}.
Else the output is {datao, sigmat}. ≫

```
Attributes[PCAOutput] = {Protected, ReadProtected}
```

PCATollerance is an option of PCADeNoise and shuld be an integer > 0.
Default value is 0. When increased the denoise method removes less noise. ≫

```
Attributes[PCATollerance] = {Protected, ReadProtected}
```

PCAWeighting is an option of PCADeNoise and can be True
of False. Default value is False. When True the weights of the per voxel
result are calculated based on the number of non noise components. ≫

```
Attributes[PCAWeighting] = {Protected, ReadProtected}
```

PlotSolution is an option for PCAFitHist, if set true it dispays the fitting itterations. ≫

```
Attributes[PlotSolution] = {Protected, ReadProtected}
```

DixonTools

Functions

DixonReconstruct[real, imag, echo] reconstruxt Dixon data with initital guess b0 = 0 and T2star = 0.
DixonReconstruct[real, imag, echo, b0] reconstructs Dixon data with intitial guess T2star = 0.
DixonReconstruct[real, imag, echo, b0, t2] reconstructs Dixon data.

real is the real data in radials.
imag is the imaginary data in radians.
B0 can be estimated from two phase images using Unwrap.
T2 can be estimated from multiple echos using T2fit.

Output is {{watF,fatF},{watSig,fatSig},{inphase,outphase},{B0,T2star},itterations}. ≫

```
Attributes[DixonReconstruct] = {Protected, ReadProtected}
```

```
Options[DixonReconstruct] = {DixonPrecessions → -1, DixonFieldStrength → 3,
  DixonFrequencies → {{0}, {3.8, 3.4, 3.13, 2.67, 2.46, 1.92, 0.57, -0.6}},
  DixonAmplitudes → {{1}, {0.089, 0.598, 0.047, 0.077, 0.052, 0.011, 0.035, 0.066}},
  DixonIterations → 5, DixonTollerance → 0.01, DixonMaskThreshhold → 0.05,
  DixonFilterInput → True, DixonFilterOutput → True, DixonFilterSize → 1}
```

```
SyntaxInformation[DixonReconstruct] = {ArgumentsPattern → {_, _, _, _., _., OptionsPattern[]}}
```

DixonToPercent[water, fat] converts the dixon water and fat data to percent maps.

Output is {waterFraction, fatFraction}. ≫

```
Attributes[DixonToPercent] = {Protected, ReadProtected}
```

```
SyntaxInformation[DixonToPercent] = {ArgumentsPattern → {_, _}}
```

SimulateDixonSignal[echo, fr, B0, T2] simulates an Dixon gradient echo sequence with
echotimes echo in ms, fat fraction fr, field of resonance B0 in Hz and relaxation T2 in ms.

```
Attributes[SimulateDixonSignal] = {Protected, ReadProtected}

Options[SimulateDixonSignal] = {DixonPrecessions → -1, DixonFieldStrength → 3,
  DixonFrequencies → {{0}, {3.8, 3.4, 3.13, 2.67, 2.46, 1.92, 0.57, -0.6}},
  DixonAmplitudes → {{1}, {0.089, 0.598, 0.047, 0.077, 0.052, 0.011, 0.035, 0.066}}}

SyntaxInformation[SimulateDixonSignal] = {ArgumentsPattern → {_, _, _, _, OptionsPattern[]}}
```

Unwrap[data] unwraps the given dataset. ≫

```
Attributes[Unwrap] = {Protected, ReadProtected}

Options[Unwrap] = {MonitorUnwrap → True, UnwrapDimension → 2D}

SyntaxInformation[Unwrap] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

UnwrapSplit[phase, data] unwarps the give phase dataset but splits the data into left and right using SplitData based in the data and performs the unwrapping seperately. ≫

```
Attributes[UnwrapSplit] = {Protected, ReadProtected}

Options[UnwrapSplit] = {}

SyntaxInformation[UnwrapSplit] = {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

Options

DixonAmplitudes is an options for DixonReconstruct. Defines the amplitudes of the fat peaks being used. ≫

```
Attributes[DixonAmplitudes] = {Protected, ReadProtected}
```

DixonFieldStrength is an options for DixonReconstruct. Defines the fieldstrengths on which the data was acquired. ≫

```
Attributes[DixonFieldStrength] = {Protected, ReadProtected}
```

DixonFilterInput is an options for DixonReconstruct. If True the input b0 and T2star values are smoothed using a gaussian kernel. ≫

```
Attributes[DixonFilterInput] = {Protected, ReadProtected}
```

DixonFilterOutput is an options for DixonReconstruct. If True the out b0 and T2star values are smoothed Median filter and lowpassfiltering after which the water and fat maps are recomputed. ≫

```
Attributes[DixonFilterOutput] = {Protected, ReadProtected}
```

DixonFilterSize is an options for DixonReconstruct. Defines the number of voxel with which the input b0 and T2star values are smoothed. ≫

```
Attributes[DixonFilterSize] = {Protected, ReadProtected}
```

DixonFrequencies is an options for
DixonReconstruct. Defines the frequencies of the fat peaks being used. ≫

```
Attributes[DixonFrequencies] = {Protected, ReadProtected}
```

DixonIterations is an options for DixonReconstruct. Defines the maximum itterations the fit can use. ≫

```
Attributes[DixonIterations] = {Protected, ReadProtected}
```

DixonMaskThreshhold is an options for DixonReconstruct. Defines at which threshhold the dixon
reconstruction considers a voxel to be background noise. Defualt values is 0.05. ≫

```
Attributes[DixonMaskThreshhold] = {Protected, ReadProtected}
```

DixonPrecessions is an options for
DixonReconstruct. Defines the rotation of the signal {–1,1} default is –1. ≫

```
Attributes[DixonPrecessions] = {Protected, ReadProtected}
```

DixonTollerance is an options for DixonReconstruct. Defines at which change per
itteration of b0 and R2star the ittarative methods stops. Default value is 0.1. ≫

```
Attributes[DixonTollerance] = {Protected, ReadProtected}
```

MonitorUnwrap is an option for Unwrap. Monitor the unwrapping progress. ≫

```
Attributes[MonitorUnwrap] = {Protected, ReadProtected}
```

UnwrapDimension is an option for Unwrap. Can be "2D" or "3D". 2D is for unwarpping
2D images or unwrapping the individual images from a 3D dataset (does not
unwrap in the slice direction). 3D unwraps a 3D dataset in all dimensions. ≫

```
Attributes[UnwrapDimension] = {Protected, ReadProtected}
```

ElastixTools

Functions

ReadTransformParameters[directory] reads the tranfomation parameters
generated by RegisterData. The directory should be the TempDirectory
were the registration is stored. DeleteTempDirectory should be False.

Output is the affine transformation vector per volume. ≫

```
Attributes[ReadTransformParameters] = {Protected, ReadProtected}
```

```
SyntaxInformation[ReadTransformParameters] = {ArgumentsPattern → {_}}
```

RegisterCardiacData[data] registers the data using a 2D algorithm. data can be 3D or 4D.
RegisterCardiacData[{data,vox}] registers the data series using the given voxel size.
RegisterCardiacData[{data,mask}] registers the data series only using data whithin the mask.
RegisterCardiacData[{data,mask,vox}] registers the
    data series using the given voxel size only using data within the mask.

Output is the registered data. ≫

```
Attributes[RegisterCardiacData] = {Protected, ReadProtected}

Options[RegisterCardiacData] = {RegistrationTarget → Mean,
  Iterations → 250, Resolutions → 1, HistogramBins → 64, NumberSamples → 2000,
  InterpolationOrderReg → 3, BsplineSpacing → 30, BsplineDirections → {1, 1, 1},
  AffineDirections → {1, 1, 1}, MethodReg → affine, OutputImage → True,
  TempDirectory → Default, DeleteTempDirectory → True, PrintTempDirectory → True,
  OutputTransformation → False, UseGPU → {False, Automatic}, PCAComponents → 3}

SyntaxInformation[RegisterCardiacData] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

RegisterData[data] registers the data series. If data is 3D it performs multiple 2D registration, if data is 4D it performs multipe 3D registration. The target is the first image orvolume in the series.
RegisterData[{data, vox}] registers the data series using the given voxel size.
RegisterData[{data, mask}] registers the data series only using data whithin the mask.
RegisterData[{data, mask, vox}] registers the data series using the given voxel size only using data within the mask.

RegisterData[target, moving] registers the moving data to the target data. target can be 2D or 3D. moving can be the same of one dimension higher than the target.
RegisterData[{target, mask, vox},{moving, mask, vox}] registers the data using the given voxel size only using data within the mask.
RegisterData[{target, vox}, moving] registers the data using the given voxel size.
RegisterData[target, {moving, vox}] registers the data using the given voxel size.
RegisterData[{target, vox}, {moving, vox}] registers the data using the given voxel size.

RegisterData[{target, mask}, moving] registers the data series only using data whithin the mask.
RegisterData[target, {moving, mask}] registers the data series only using data whithin the mask.
RegisterData[{target, mask}, moving] registers the data series only using data whithin the mask.
RegisterData[{target, mask}, {moving, mask}] registers the data series only using data whithin the mask.

RegisterData[target, {moving, mask, vox}] registers the data series using the given voxel size only using data within the mask.
RegisterData[{target, mask}, {moving, mask, vox}] registers the data series using the given voxel size only using data within the mask.
RegisterData[{target, vox}, {moving, mask, vox}] registers the data series using the given voxel size only using data within the mask.
RegisterData[{target, mask, vox}, moving] registers the data series using the given voxel size only using data within the mask.
RegisterData[{target, mask, vox}, {moving, mask}] registers the data series using the given voxel size only using data within the mask.
RegisterData[{target, mask, vox}, {moving, vox}] registers the data series using the given voxel size only using data within the mask.
RegisterData[{target, mask}, {moving, vox}] registers the data series using the given voxel size only using data within the mask.
RegisterData[{target, vox}, {moving, mask}] registers the data series using the given voxel size only using data within the mask.

Output is the registered data with the dimensions of the moving data.
If OutputTransformation is True it also outputs the translation, rotation scale and skew of all images or volumes. ≫

```
Attributes[RegisterData] = {Protected, ReadProtected}

Options[RegisterData] =
 {Iterations → 250, Resolutions → 1, HistogramBins → 64, NumberSamples → 2000,
  InterpolationOrderReg → 3, BsplineSpacing → 30, BsplineDirections → {1, 1, 1},
  AffineDirections → {1, 1, 1}, MethodReg → affine, OutputImage → True,
  TempDirectory → Default, DeleteTempDirectory → True, PrintTempDirectory → True,
  OutputTransformation → False, UseGPU → {False, Automatic}, PCAComponents → 3}

SyntaxInformation[RegisterData] = {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

RegisterDataSplit[target, moving] is identical to RegisterData
data however left and right side of the data are registered seperately.

Splitting the data is done using the function CutData and merged wit Stich data.
Output is the registered data. ≫

```
Attributes[RegisterDataSplit] = {Protected, ReadProtected}

Options[RegisterDataSplit] =
 {Iterations → 250, Resolutions → 1, HistogramBins → 64, NumberSamples → 2000,
  InterpolationOrderReg → 3, BsplineSpacing → 30, BsplineDirections → {1, 1, 1},
  AffineDirections → {1, 1, 1}, MethodReg → affine, OutputImage → True, TempDirectory → Default,
  DeleteTempDirectory → True, PrintTempDirectory → True, OutputTransformation → False,
  UseGPU → {False, Automatic}, PCAComponents → 3, SplitMethod → Mean}

SyntaxInformation[RegisterDataSplit] = {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

RegisterDataTransform[target, moving, {moving2nd,
vox}] performs the registration exactly as RegisterData. target and
moving are the inputs for Registerdata, which can be {data,mask,vox}.
After the registeration is done the moving2nd data is deformed
acording to the output of the registrtion of moving.

moving2nd can have the same dimensions
of moving or one dimension higher (e.g. 3D and 3D or 3D and 4D).

Output is {registered moving, deformed moving2nd}. ≫

```
Attributes[RegisterDataTransform] = {Protected, ReadProtected}

Options[RegisterDataTransform] =
 {Iterations → 250, Resolutions → 1, HistogramBins → 64, NumberSamples → 2000,
  InterpolationOrderReg → 3, BsplineSpacing → 30, BsplineDirections → {1, 1, 1},
  AffineDirections → {1, 1, 1}, MethodReg → affine, OutputImage → True,
  TempDirectory → Default, DeleteTempDirectory → True, PrintTempDirectory → True,
  OutputTransformation → False, UseGPU → {False, Automatic}, PCAComponents → 3}

SyntaxInformation[RegisterDataTransform] = {ArgumentsPattern → {_, _, _, OptionsPattern[]}}
```

RegisterDataTransformSplit[target, moving, {moving2nd, vox}] is idenditcal
to RegisterDataTransform with the same functionality as RegisterDataSplit.
This means the data is split in two using the function CutData and merged wit Stich data.

Output is {registered moving, deformed moving2nd}. ≫

```
Attributes[RegisterDataTransformSplit] = {Protected, ReadProtected}
```

```
Options[RegisterDataTransformSplit] =
 {Iterations → 250, Resolutions → 1, HistogramBins → 64, NumberSamples → 2000,
  InterpolationOrderReg → 3, BsplineSpacing → 30, BsplineDirections → {1, 1, 1},
  AffineDirections → {1, 1, 1}, MethodReg → affine, OutputImage → True, TempDirectory → Default,
  DeleteTempDirectory → True, PrintTempDirectory → True, OutputTransformation → False,
  UseGPU → {False, Automatic}, PCAComponents → 3, SplitMethod → Mean}
```

```
SyntaxInformation[RegisterDataTransformSplit] =
 {ArgumentsPattern → {_, _, _, OptionsPattern[]}}
```

RegisterDiffusionData[{dtidata, vox}] registers a diffusion dataset.
dtidata should be 4D {slice, diff, x, y}. vox is the voxelsize of the data.
RegisterDiffusionData[{dtidata, dtimask, vox}] registers the data series
using the given voxel size only using data within the mask.
RegisterDiffusionData[{dtidata ,vox}, {anatdata, voxa}] registers a diffusion
dataset. The diffusion data is also registered to the anatdata.
RegisterDiffusionData[{dtidata, dtimask, vox}, {anatdata, voxa}] registers the
data series using the given voxel size only using data within the mask.
RegisterDiffusionData[{dtidata,vox}, {anatdata, anatmask, voxa}] registers the
data series using the given voxel size only using data within the mask.
RegisterDiffusionData[{dtidata, dtimask, vox}, {anatdata, anatmask, voxa}] registers
the data series using the given voxel size only using data within the mask.

Output is the registered dtidata and, if anatdata is given, the
registered dtidata in anatomical space. If OutputTransformation is True it also
outputs the translation, rotation scale and skew of all images or volumes. ≫

```
Attributes[RegisterDiffusionData] = {Protected, ReadProtected}
```

```
Options[RegisterDiffusionData] =
 {Iterations → 250, Resolutions → 1, HistogramBins → 64, NumberSamples → 2000,
  InterpolationOrderReg → 3, BsplineSpacing → 30, BsplineDirections → {0, 1, 1},
  AffineDirections → {0, 1, 1}, MethodReg → affineDTI, OutputImage → True,
  TempDirectory → Default, DeleteTempDirectory → True, PrintTempDirectory → True,
  OutputTransformation → False, UseGPU → {False, Automatic}, PCAComponents → 3,
  IterationsA → 1000, ResolutionsA → 1, HistogramBinsA → 64, NumberSamplesA → 20000,
  InterpolationOrderRegA → 1, MethodRegA → {rigidDTI, bspline}, RegistrationTarget → Fist}
```

```
SyntaxInformation[RegisterDiffusionData] = {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

RegisterDiffusionDataSplit[dtidata, vox] is identical to Register
diffusion data however left and right side of the data are registered seperately.
RegisterDiffusionDataSplit[{dtidata, vox}, {anatdata, voxa}] is identical to Register
diffusion data however left and right side of the data are registered seperately.
RegisterDiffusionDataSplit[{dtidata, dtimask, vox}, {anatdata, anatmask, voxa}] is identical to
Register diffusion data however left and right side of the data are registered seperately.

Splitting the data is done using the function CutData and merged wit Stich data.
Output is the registered data. ≫

```
Attributes[RegisterDiffusionDataSplit] = {Protected, ReadProtected}
```

```
Options[RegisterDiffusionDataSplit] := Options[RegisterDiffusionData]
```

```
SyntaxInformation[RegisterDiffusionDataSplit] = {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

TransformData[{data,vox}] deforms the data according to the last output of register data.
The directory should be the TempDirectory were
the registration is stored. DeleteTempDirectory should be False. ≫

```
Attributes[TransformData] = {Protected, ReadProtected}
```

```
Options[TransformData] = {TempDirectory → Default,
  FindTransform → Auto, DeleteTempDirectory → All, PrintTempDirectory → True}
```

```
SyntaxInformation[TransformData] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

Options

AffineDirections is an option for RegisterData ad RegisterDiffusionData.
It gives the directions in which data can
be moved when registering diffusion data to anatomical space. ≫

```
Attributes[AffineDirections] = {Protected, ReadProtected}
```

BsplineDirections is an option for RegisterData ad RegisterDiffusionData.
It gives the direction in which the bsplines are
allowed to move when registering diffusion data to anatomical space. ≫

```
Attributes[BsplineDirections] = {Protected, ReadProtected}
```

BsplineSpacing is an options for RegisterData,
RegisterDiffusionData, RegisterCardiacData and RegisterDataTransform.
It specifies the spacing of the bsplines if the method is "bspline". ≫

```
Attributes[BsplineSpacing] = {Protected, ReadProtected}
```

DeleteTempDirectory an options for RegisterData,
RegisterDiffusionData, RegisterCardiacData and RegisterDataTransform.
It specifies if the temp directory should be deleted after the registration is finisched. ≫

```
Attributes[DeleteTempDirectory] = {Protected, ReadProtected}
```

FindTransform is an option for TransformData and
RegisterTransformData. It specifies where to find the transformfile. ≫

```
Attributes[FindTransform] = {Protected, ReadProtected}
```

HistogramBins is an options for RegisterData, RegisterDiffusionData, and RegisterDataTransform.
It specifies the number of bins of the joined histogram used by the registration functions. ≫

```
Attributes[HistogramBins] = {Protected, ReadProtected}
```

HistogramBinsA is an option for RegisterDiffusionData.
It specifies the number of bins of the joined
histogram used when registering diffusion data to anatomical space. ≫

```
Attributes[HistogramBinsA] = {Protected, ReadProtected}
```

InterpolationOrderReg is an options for
RegisterData, RegisterDiffusionData, and RegisterDataTransform.
It specifies the interpolation order used in the registration functions. ≫

```
Attributes[InterpolationOrderReg] = {Protected, ReadProtected}
```

InterpolationOrderRegA is an option for RegisterDiffusionData.
It specifies the interpolation order used in the
registration functions when registering diffusion data to anatomical space. ≫

```
Attributes[InterpolationOrderRegA] = {Protected, ReadProtected}
```

Iterations is an options for RegisterData, RegisterDiffusionData, and RegisterDataTransform.
It specifies the number of iterations used by the registration functions. ≫

```
Attributes[Iterations] = {Protected, ReadProtected}
```

IterationsA is an option for RegisterDiffusionData.
It specifies the number of iterations
used when registering diffusion data to anatomical space. ≫

```
Attributes[IterationsA] = {Protected, ReadProtected}
```

MethodReg is an options for RegisterData,
RegisterDiffusionData, RegisterCardiacData and RegisterDataTransform.
It spefifies which registration method to use. Mehtods can be be
"rigid","affine", "bspline" or "cyclyc". ≫

```
Attributes[MethodReg] = {Protected, ReadProtected}
```

MethodRegA is an option for RegisterDiffusionData.
It spefifies which registration method to use when registering diffusion
data to anatomical space. Mehtods can be be "rigid","affine" or "bspline". ≫

```
Attributes[MethodRegA] = {Protected, ReadProtected}
```

NumberSamples is an options for RegisterData, RegisterDiffusionData, and RegisterDataTransform.
It specifies the number of random samples
that are taken each iteration used by the registration functions. ≫

```
Attributes[NumberSamples] = {Protected, ReadProtected}
```

NumberSamplesA is an option for RegisterDiffusionData.
It specifies the number of random samples that are
taken each iteration when registering diffusion data to anatomical space. ≫

```
Attributes[NumberSamplesA] = {Protected, ReadProtected}
```

OutputImage is an options for RegisterData, RegisterDiffusionData, and RegisterDataTransform.
It specifies if the result image should be writen in the TempDirectory as nii file. ≫

```
Attributes[OutputImage] = {Protected, ReadProtected}
```

OutputTransformation is an option for RegisterData ad RegisterDiffusionData.
It specifies if the tranformation paramters (translation, rotation,
scale and skew) should be given as output in the registration functions. ≫

```
Attributes[OutputTransformation] = {Protected, ReadProtected}
```

PCAComponents is an option for RegisterData. It
speciefies how many PCA components are used if method is set to "PCA" ≫

```
Attributes[PCAComponents] = {Protected, ReadProtected}
```

PrintTempDirectory is an options for RegisterData,
RegisterDiffusionData, RegisterCardiacData and RegisterDataTransform.
It spefifies if the location of the temp directory should be deplayed. ≫

```
Attributes[PrintTempDirectory] = {Protected, ReadProtected}
```

RegistrationTarget is an option for RegisterDiffusionData and RegisterCardiacData. Specifies which target to uses for registration if using "rigid", "affine" or "bspline" as MethodReg.
If the MethodReg is "cyclyc" or "PCA" it does not need a target and this options does nothing.
Values can be "First", "Mean" or "Median". ≫

```
Attributes[RegistrationTarget] = {Protected, ReadProtected}
```

Resolutions is an options for RegisterData, RegisterDiffusionData, and RegisterDataTransform.
It specifies the number of scale space resolutions used by the registration functions. ≫

```
Attributes[Resolutions] = {Protected, ReadProtected}
```

ResolutionsA is an option for RegisterDiffusionData.
It specifies the number of scale space resolutions used when registering diffusion data to anatomical space. ≫

```
Attributes[ResolutionsA] = {Protected, ReadProtected}
```

SplitMethod is an option for RegisterDataSplit and RegisterDataTransformSplit. values can be "mean", "moving", "target" ≫

```
Attributes[SplitMethod] = {Protected, ReadProtected}
```

TempDirectory is an options for RegisterData, RegisterDiffusionData, RegisterCardiacData and RegisterDataTransform.
It specifies the temprary directory used to perform and output the registration. ≫

```
Attributes[TempDirectory] = {Protected, ReadProtected}
```

UseGPU is an option for RegisterData. The value is {bool, gpu} where bool is True or False, and gpu is the gpu ID which is an integer or Automatic. ≫

```
Attributes[UseGPU] = {Protected, ReadProtected}
```

GeneralTools

Functions

ApplyCrop[data,crop] aplies the corpped region obtained form CropData to the data.
ApplyCrop[data,crop,{voxorig,voxnew}] aplies the corpped region obtained form CropData to the data. ≫

```
Attributes[ApplyCrop] = {Protected, ReadProtected}
```

```
SyntaxInformation[ApplyCrop] = {ArgumentsPattern → {_, _, _.}}
```

AutoCropData[data] crops the data by removing all background zeros.
AutoCropData[data,pad] crops the data by removing all background zeros with padding of pad. ≫

```
Attributes[AutoCropData] = {Protected, ReadProtected}

Options[AutoCropData] = {CropPadding → 5}

SyntaxInformation[AutoCropData] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

ClearTemporaryVariables[] Clear temporary variables. ≫

```
Attributes[ClearTemporaryVariables] = {Protected, ReadProtected}

SyntaxInformation[ClearTemporaryVariables] = {ArgumentsPattern → {_.}}
```

CompilebleFunctions[] generates a list of all compilable functions. ≫

```
Attributes[CompilebleFunctions] = {Protected, ReadProtected}

SyntaxInformation[CompilebleFunctions] = {ArgumentsPattern → {}}
```

CropData[data] creates a dialog window to crop the data (assumes voxsize (1,1,1)).
CropData[data,vox] creates a dialog window to crop the data. ≫

```
Attributes[CropData] = {Protected, ReadProtected}

Options[CropData] = {CropOutput → All, CropInit → Automatic}

SyntaxInformation[CropData] = {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

CutData[data] splits the data in two equal sets left and right. ≫

```
Attributes[CutData] = {Protected, ReadProtected}

SyntaxInformation[CutData] = {ArgumentsPattern → {_, _.}}
```

Data2DToVector[data] converst the data to vector.
Data2DToVector[data,mask] converst the data within the mask to vector.

the data can be reconstructed using VectorToData.

output is the vecotrized data and a list contining the original
data dimensions and a list with the data coordinates. {vec, {dim,pos}}. ≫

```
Attributes[Data2DToVector] = {Protected, ReadProtected}

SyntaxInformation[Data2DToVector] = {ArgumentsPattern → {_, _.}}
```

Data3DToVector[data] converst the data to vector..
Data3DToVector[data,mask] converst the data within the mask to vector.

the data can be reconstructed using VectorToData.

output is the vecotrized data and a list contining the original
data dimensions and a list with the data coordinates. {vec, {dim,pos}}. ≫

```
Attributes[Data3DToVector] = {Protected, ReadProtected}
```

```
SyntaxInformation[Data3DToVector] = {ArgumentsPattern → {_, _.}}
```

DevideNoZero[a, b] devides a/b but when b=0 the result is 0. a can be a number or vector. ≫

```
Attributes[DevideNoZero] = {Protected, ReadProtected}
```

```
SyntaxInformation[DevideNoZero] = {ArgumentsPattern → {_, _}}
```

ExpNoZero[val] return the Exp of the val which can be anny dimonsion array. if val=0 the output is 0. ≫

```
Attributes[ExpNoZero] = {Protected, ReadProtected}
```

```
SyntaxInformation[ExpNoZero] = {ArgumentsPattern → {_}}
```

FileSelect[action] creates a systemdialog wicht returns
    file/foldername action can be "FileOpen", "FileSave" or "Directory".
FileSelect[action, {type}] same but allows the definition of filetypes
    for "FileOpen" and "FileSave" e.g. "jpg" or "pdf". ≫

```
Attributes[FileSelect] = {Protected, ReadProtected}
```

```
Options[FileSelect] = {WindowTitle → Automatic}
```

```
SyntaxInformation[FileSelect] = {ArgumentsPattern → {_, _., _., OptionsPattern[]}}
```

FindCrop[data] finds the crop values of the data by removing all zeros surrounding the data. ≫

```
Attributes[FindCrop] = {Protected, ReadProtected}
```

```
Options[FindCrop] = {CropPadding → 5}
```

```
SyntaxInformation[FindCrop] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

FindMaxDimensions[{data1, data2, ..}] finds
    the maximal dimensions of all datasets. Each dataset is 3D. ≫

```
Attributes[FindMaxDimensions] = {Protected, ReadProtected}
```

```
SyntaxInformation[FindMaxDimensions] = {ArgumentsPattern → {_}}
```

GridData[{data1,data2,...}, part] makes a grid of multiple datasets with part sets on each row ≫

```
Attributes[GridData] = {Protected, ReadProtected}
```

```
SyntaxInformation[GridData] = {ArgumentsPattern → {_, _}}
```

GridData3D[{data1,data2,...}, part] same as grid data, but only
    works on 4D data where the data is gridded in axial, coronal and sagital.

```
Attributes[GridData3D] = {Protected, ReadProtected}
```

LapFilter[data] Laplacian filter of data with kernel size 0.8.
LapFilter[data, ker] Laplacian filter of data with kernel ker. ≫

```
Attributes[LapFilter] = {Protected, ReadProtected}
```

LogNoZero[val] return the log of the val which can be anny dimonsion array. if val=0 the output is 0. ≫

```
Attributes[LogNoZero] = {Protected, ReadProtected}
```

```
SyntaxInformation[LogNoZero] = {ArgumentsPattern → {_}}
```

MADNoZero[vec] return the MAD error of the vec which can
be anny dimonsion array. if vec={0...} the output is 0. Zeros are ignored ≫

```
Attributes[MADNoZero] = {Protected, ReadProtected}
```

```
SyntaxInformation[MADNoZero] = {ArgumentsPattern → {_}}
```

MeanNoZero[data] calculates the mean of the data ignoring the zeros. ≫

```
Attributes[MeanNoZero] = {Protected, ReadProtected}
```

```
SyntaxInformation[MeanNoZero] = {ArgumentsPattern → {_, _.}}
```

MedianNoZero[data] calculates the Median of the data ignoring the zeros. ≫

```
Attributes[MedianNoZero] = {Protected, ReadProtected}
```

```
SyntaxInformation[MedianNoZero] = {ArgumentsPattern → {_, _.}}
```

MemoryUsage[] gives a table of which definitions use up memory.
MemoryUsage[n] gives a table of which
definitions use up memory, where n is the amout of definitions to show. ≫

```
Attributes[MemoryUsage] = {Protected, ReadProtected}
```

```
SyntaxInformation[MemoryUsage] = {ArgumentsPattern → {_.}}
```

NNLeastSquares[A, y] performs a Non Negative Linear Least Squares fit.
finds an x that solves the linear least−squares problem for the matrix equation A.x==y.

output is the solution x. ≫

```
Attributes[NNLeastSquares] = {Protected, ReadProtected}
```

```
SyntaxInformation[NNLeastSquares] = {ArgumentsPattern → {_, _}}
```

PadToDimensions[data, dim] pads the data to dimensions dim. ≫

```
Attributes[PadToDimensions] = {Protected, ReadProtected}
```

```
Options[PadToDimensions] = {PadValue → 0., PadDirection → Center}
```

```
SyntaxInformation[PadToDimensions] = {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

QMRIToolsFuncPrint[] gives a list of all the QMRITools functions with their usage infomation. ≫

```
Attributes[QMRIToolsFuncPrint] = {Protected, ReadProtected}
```

```
SyntaxInformation[QMRIToolsFuncPrint] = {ArgumentsPattern → {_.}}
```

QMRIToolsFunctions[] give list of all the QMRITools packages, functions and options.
QMRIToolsFunctions[p] print a table with length p of all the QMRITools functions and options.
QMRIToolsFunctions["toobox"] gives a list of all the functions and options in toolbox.
QMRIToolsFunctions["toobox", p] gives a table op length p of all the
functions and options in toolbox. If toolbox is "All" it will list all toolboxes. ≫

```
Attributes[QMRIToolsFunctions] = {Protected, ReadProtected}
```

```
SyntaxInformation[QMRIToolsFunctions] = {ArgumentsPattern → {_., _.}}
```

QMRIToolsPackages[] give list of all the QMRITools pacakges. ≫

```
Attributes[QMRIToolsPackages] = {Protected, ReadProtected}
```

```
SyntaxInformation[QMRIToolsPackages] = {ArgumentsPattern → {}}
```

RescaleData[data,dim] rescales image/data to given dimensions.
RescaleData[data,{vox1, vox2}] rescales image/data from size vox1 to size vox2. ≫

```
Attributes[RescaleData] = {Protected, ReadProtected}
```

```
Options[RescaleData] = {InterpolationOrder → 3}
```

```
SyntaxInformation[RescaleData] = {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

ReverseCrop[data,dim,crop] reverses the crop
on the cropped data with crop values crop to the original size dim.
ReverseCrop[data,dim,crop,{voxorig,voxnew}] reverses the crop on the
cropped data with crop values crop to the original size dim. ≫

```
Attributes[ReverseCrop] = {Protected, ReadProtected}
```

```
SyntaxInformation[ReverseCrop] = {ArgumentsPattern → {_, _, _, _.}}
```

RMSNoZero[vec] return the RMS error of the vec which can
be anny dimonsion array. if vec={0...} the output is 0. Zeros are ignored ≫

```
Attributes[RMSNoZero] = {Protected, ReadProtected}
```

```
SyntaxInformation[RMSNoZero] = {ArgumentsPattern → {_}}
```

SaveImage[image] exports graph to image,
ImageSize, FileType and ImageResolution can be given as options.
SaveImage[image, "filename"] exports graph to image with "filname",
ImageSize, FileType and ImageResolution can be given as options. ≫

```
Attributes[SaveImage] = {Protected, ReadProtected}

Options[SaveImage] = {ImageSize → 6000, FileType → .jpg, ImageResolution → 300}

SyntaxInformation[SaveImage] = {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

StdFilter[data] StandardDeviation filter of data using gaussian kernel 2.
StdFilter[data, ker] StandardDeviation filter of data using kernel with size ker. ≫

```
Attributes[StdFilter] = {Protected, ReadProtected}
```

StichData[datal,datar] joins left and right part of the data generated by CutData. ≫

```
Attributes[StichData] = {Protected, ReadProtected}

SyntaxInformation[StichData] = {ArgumentsPattern → {_, _}}
```

SumOfSquares[{data1, data2, ... datan}] calculates the sum of squares of the datasets.
Output is the SoS and the weights, or just the SoS. ≫

```
Attributes[SumOfSquares] = {Protected, ReadProtected}

Options[SumOfSquares] = {OutputWeights → True}

SyntaxInformation[SumOfSquares] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

TensMat[tensor] transforms tensor form vector format
{xx,yy,zz,xy,xz,yz} to matrix format {{xx,xy,xz},{xy,yy,yz},{xz,yz,zz}}. ≫

```
Attributes[TensMat] = {Protected, ReadProtected}

SyntaxInformation[TensMat] = {ArgumentsPattern → {_}}
```

TensVec[tensor] transforms tensor form matrix format
{{xx,xy,xz},{xy,yy,yz},{xz,yz,zz}} to vector format {xx,yy,zz,xy,xz,yz}. ≫

```
Attributes[TensVec] = {Protected, ReadProtected}

SyntaxInformation[TensVec] = {ArgumentsPattern → {_}}
```

TransData[data,dir] Rotates the dimesions of the
data to left or rigthg. For example {z,x,y} to {x,y,z} dir is "l" or "r". ≫

```
Attributes[TransData] = {Protected, ReadProtected}

SyntaxInformation[TransData] = {ArgumentsPattern → {_, _}}
```

VectorToData[vec, {dim,pos}] converts the vectroized data, using
Data2DToVector or Data3DToVector, back to its original Dimensoins ≫

```
Attributes[VectorToData] = {Protected, ReadProtected}

SyntaxInformation[VectorToData] = {ArgumentsPattern → {_, {_, _}}}
```

Options

CropInit is an option for CropData. By default the crop is
not initialized bu can be with {{xmin,xmax},{ymin,ymax},{zmin,zmax}}. ≫

```
Attributes[CropInit] = {Protected, ReadProtected}
```

CropOutput is an option for CropData, can be "All","Data" or "Crop". ≫

```
Attributes[CropOutput] = {Protected, ReadProtected}
```

CropPadding is an option for AutoCropData or
FindCrop. It specifies how much padding to use around the data. ≫

```
Attributes[CropPadding] = {Protected, ReadProtected}
```

FileType["*file*"] gives the type of a file, typically File, Directory, or None. ≫

```
Attributes[FileType] = {Protected}
```

ImageResolution is an option for Export, Rasterize, and related functions
that specifies at what resolution bitmap images should be rendered. ≫

```
Attributes[ImageResolution] = {Protected}
```

ImageSize is an option that specifies the overall size of an image to display for an object. ≫

```
Attributes[ImageSize] = {Protected}
```

InterpolationOrder is an option for Interpolation, as well as ListLinePlot, ListPlot3D, ListContourPlot,
and related functions, that specifies what order of interpolation to use. ≫

```
Attributes[InterpolationOrder] = {Protected}
```

OutputWeights is an option for SumOfSqares. If True it also output the SoS weights. ≫

```
Attributes[OutputWeights] = {Protected, ReadProtected}
```

PadDirection is an option for PadToDimensions.
It specifies the direction of padding, "Center", "Left" or "Right". ≫

```
Attributes[PadDirection] = {Protected, ReadProtected}
```

PadValue is an option for PadToDimensions. It specifies the value of the padding. ≫

```
Attributes[PadValue] = {Protected, ReadProtected}
```

WindowTitle is an option that specifies the title to give for a window. ≫

```
Attributes[WindowTitle] = {Protected}
```

GradientTools

Functions

Bmatrix[bvec,grad] creates bmatrix form grad and bvec in form {–bxx, –byy, –bzz, –bxy, –bxz, –byz ,1}.
Bmatrix[{bvec,grad}] creates bmatrix form grad and bvec in form {bxx, byy, bzz, bxy, bxz, byz}. ≫

```
Attributes[Bmatrix] = {Protected, ReadProtected}
```

```
Options[Bmatrix] = {Method → DTI}
```

```
SyntaxInformation[Bmatrix] = {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

BmatrixCalc["folder", grads] calculates the true bmatrix from the exported sequence parameters from
the philips scanner that are stored in "folder" for each of the gradient directions grads. ≫

```
Attributes[BmatrixCalc] = {Protected, ReadProtected}
```

```
Options[BmatrixCalc] = {UseGrad → {1, 1, {1, 1}, 1, 1}, OutputType → Matrix,
  Method → Numerical, StepSizeI → 0.05, UnitMulti → 1/1000, PhaseEncoding → A,
  FlipAxes → {{1, 1, 1}, {1, 1, 1}}, SwitchAxes → {{1, 2, 3}, {1, 2, 3}}}
```

```
SyntaxInformation[BmatrixCalc] = {ArgumentsPattern → {_, _, _., OptionsPattern[]}}
```

BmatrixConv[bm] converts the bmatrix form 7 to 6 or from 6 to 7. ≫

```
Attributes[BmatrixConv] = {Protected, ReadProtected}
```

```
SyntaxInformation[BmatrixConv] = {ArgumentsPattern → {_}}
```

BmatrixInv[bm] generates a bvecotr and gradiens directions form a given bmatrx.
BmatrixInv[bm, bvi] generates a bvecotr and
gradiens directions form a given bmatrx using the given bvalues bvi. ≫

```
Attributes[BmatrixInv] = {Protected, ReadProtected}
```

```
SyntaxInformation[BmatrixInv] = {ArgumentsPattern → {_, _.}}
```

BmatrixRot[bmat, rotmat] Rotates the B–matrix. ≫

```
Attributes[BmatrixRot] = {Protected, ReadProtected}
```

```
SyntaxInformation[BmatrixRot] = {ArgumentsPattern → {_, _}}
```

BmatrixToggle[bmat, axes, flip], axes can be any order
of {"x","y","z"}. flip should be {1,1,1},{1,1,–1},{1,–1,1} or {–1,1,1}. ≫

```
Attributes[BmatrixToggle] = {Protected, ReadProtected}
```

```
SyntaxInformation[BmatrixToggle] = {ArgumentsPattern → {_, _, _}}
```

CalculateMoments[{Gt, hw, te}, t] calculates the 0th to 3th order moments
of the sequence created by GradSeq. Output is {{Gt, M0, M1, M2, M3}, vals}. ≫

```
Attributes[CalculateMoments] = {Protected, ReadProtected}
```

```
SyntaxInformation[CalculateMoments] = {ArgumentsPattern → {_, _}}
```

ConditionNumberCalc[grads] calcualtes the condition number of the gradient set. ≫

```
Attributes[ConditionNumberCalc] = {Protected, ReadProtected}
```

```
SyntaxInformation[ConditionNumberCalc] = {ArgumentsPattern → {_}}
```

ConvertGrads[grad, bv] converts the gradients to txt format, which is needed for FinalGrads. ≫

```
Attributes[ConvertGrads] = {Protected, ReadProtected}
```

```
SyntaxInformation[ConvertGrads] = {ArgumentsPattern → {_, _}}
```

CorrectBmatrix[bmat, transformation] corrects the bmatrix bmat with
the tranformation parameters from RegisterData or RegisterDiffusionData.

Output is the corrected bmatrix. ≫

```
Attributes[CorrectBmatrix] = {Protected, ReadProtected}
```

```
Options[CorrectBmatrix] = {MethodReg → Full}
```

```
SyntaxInformation[CorrectBmatrix] = {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

CorrectGradients[grad, transformation] corrects the gradient directions grad with
the tranformation parameters from RegisterData or RegisterDiffusionData.

Output is the corrected gradient vector. ≫

```
Attributes[CorrectGradients] = {Protected, ReadProtected}
```

```
Options[CorrectGradients] = {MethodReg → Rotation}
```

```
SyntaxInformation[CorrectGradients] = {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

EnergyCalc[grads] calcualtes the total Energy of the gradient set. ≫

```
Attributes[EnergyCalc] = {Protected, ReadProtected}
```

```
SyntaxInformation[EnergyCalc] = {ArgumentsPattern → {_}}
```

FinalGrads[grtxt,{int,intn},{rand,order}] finalizes the gradient txt file.
grtxt is the output from the function ConvertGrads, which convert the grad to txt format.
int is True or False, if set to True it interleaves b=0 gradients every intn directions.
rand indicates if the gradients need to be
randomized, for this it uses the order which is the output of FindOrder. ≫

```
Attributes[FinalGrads] = {Protected, ReadProtected}
```

```
SyntaxInformation[FinalGrads] = {ArgumentsPattern → {_, {_, _}, {_, _}}}
```

FindOrder[grad,bv] finds the optimal order of the gradient directions which minimizes the duty cycle.
The output is needed for FinalGrads.
grad is a list of gradient sets and bv is a
list of b–values with the same number as the list of gradient sets. ≫

```
Attributes[FindOrder] = {Protected, ReadProtected}
```

```
Options[FindOrder] = {OrderSpan → Auto}
```

```
SyntaxInformation[FindOrder] = {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

FullGrad is an option for Grad. Default is True. When true
the gradient directions wil be loaded with the first gradient {0,0,0}. ≫

```
Attributes[FullGrad] = {Protected, ReadProtected}
```

GenerateGradients[numb] optimizes a set with numb gradients, numb mus be an integer.
GenerateGradients[{numb, fixed}] optimizes a set with numb gradients, numb must ba an integer and
fixed a list of 3D coordiantes e.g. {{0,0,1},{0,1,0}}. The fixed gradients will not be moved.
GenerateGradients[{numb1, numb2 ...}, alpha] optimizes a multi shel gradient set with
numb gradients per shel. If alpha is set to 0.5 equal importance is given tothe
optimal distribution of each shell en the enitre set. if alpha is 0 only the sub
shels will be optimized, if alpha is set to 1 only the global set wil be optimized. ≫

```
Attributes[GenerateGradients] = {Protected, ReadProtected}
```

```
Options[GenerateGradients] = {Steps → 1000, Runs → 1,
  VisualOpt → False, GradType → Normal, ConditionCalc → False, FullSphere → False}
```

```
SyntaxInformation[GenerateGradients] = {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

GenerateGradientsGUI[] runs the
GenerateGradients function in GUI with output for the philips system. ≫

```
Attributes[GenerateGradientsGUI] = {Protected, ReadProtected}
```

GetGradientScanOrder[grad, bval] determines
the scanorder based on the txt file provided to the scanner as input.
GetGradientScanOrder[file, grad, bval] determines the scanorder based
on the txt file provided to the scanner as input. ≫

```
Attributes[GetGradientScanOrder] = {Protected, ReadProtected}
```

```
SyntaxInformation[GetGradientScanOrder] = {ArgumentsPattern → {_, _, _.}}
```

GetSliceNormal[file] imports the slice normal from a dicom image. ≫

```
Attributes[GetSliceNormal] = {Protected, ReadProtected}
```

```
SyntaxInformation[GetSliceNormal] = {ArgumentsPattern → {_, _.}}
```

GetSliceNormalDir[file] imports the slice normal from a enhanced dicom image. ≫

```
Attributes[GetSliceNormalDir] = {Protected, ReadProtected}
```

```
SyntaxInformation[GetSliceNormalDir] = {ArgumentsPattern → {_}}
```

GradBmatrix[Gt, hw, te, t] Calculates the true bmatrix from the sequence created by GradSeq. ≫

```
Attributes[GradBmatrix] = {Protected, ReadProtected}
```

```
Options[GradBmatrix] = {OutputPlot → False, Method → Analytical, StepSizeI → 0.025}
```

```
SyntaxInformation[GradBmatrix] = {ArgumentsPattern → {_, _, _, _, OptionsPattern[]}}
```

GradSeq[pars, t, grad] Creates a sequence from the gradient pars imported by ImportGradObj. ≫

```
Attributes[GradSeq] = {Protected, ReadProtected}
```

```
Options[GradSeq] = {UseGrad → {0, 1, {1, 0}, 1}, FlipGrad → False, UnitMulti → 1,
  PhaseEncoding → A, FlipAxes → {{1, 1, 1}, {1, 1, 1}}, SwitchAxes → {{1, 2, 3}, {1, 2, 3}}}
```

```
SyntaxInformation[GradSeq] = {ArgumentsPattern → {_, _, _, OptionsPattern[]}}
```

ImportGradObj[folder] Imports the gradient par files exported from the philips scanner. ≫

```
Attributes[ImportGradObj] = {Protected, ReadProtected}
```

```
SyntaxInformation[ImportGradObj] = {ArgumentsPattern → {_}}
```

OverPlusCalc[grads] determines the minimal overplus factor of of the gradient set. ≫

```
Attributes[OverPlusCalc] = {Protected, ReadProtected}
```

```
SyntaxInformation[OverPlusCalc] = {ArgumentsPattern → {_}}
```

UniqueBvalPosition[bval] generates a list of all the unique bvalues and their positions.
UniqueBvalPosition[bval, num] generates a list of all the unique bvalues and
their positions that are present in the dataset equal or more than num times ≫

```
Attributes[UniqueBvalPosition] = {Protected, ReadProtected}
```

```
SyntaxInformation[UniqueBvalPosition] = {ArgumentsPattern → {_, _.}}
```

Options

ConditionCalc is an option for GenerateGradients if set to true
GenerateGradients will also give the condition number evolution of the system. ≫

```
Attributes[ConditionCalc] = {Protected, ReadProtected}
```

FlipAxes is an option for GradSeq. Defaul value is {{1,1,1},{1,1,1}}. First three
values are for diffusion gradients last three are for the acquisition gradients. ≫

```
Attributes[FlipAxes] = {Protected, ReadProtected}
```

FlipGrad is an option for GradSeq. When FlipGrad is true the gr180 is fliped. ≫

```
Attributes[FlipGrad] = {Protected, ReadProtected}
```

FullSphere is an option for GenerateGradients. If set True the
gradients will be optimized on a full sphere rather than half a sphere. ≫

```
Attributes[FullSphere] = {Protected, ReadProtected}
```

GradType is what type of gradient set wil
be produced in GenerateGradients "Normal" or "OverPlus". ≫

```
Attributes[GradType] = {Protected, ReadProtected}
```

Method is an option for various algorithm–intensive
functions that specifies what internal methods they should use. ≫

```
Attributes[Method] = {Protected}
```

MethodReg is an options for RegisterData,
RegisterDiffusionData, RegisterCardiacData and RegisterDataTransform.
It spefifies which registration method to use. Mehtods can be be
"rigid","affine", "bspline" or "cyclyc". ≫

```
Attributes[MethodReg] = {Protected, ReadProtected}
```

OrderSpan is an options for FindOrder. ≫

```
Attributes[OrderSpan] = {Protected, ReadProtected}
```

OutputPlot is an option for GradBmatrix. It
specifies if the plots of the gradients should also be exported. ≫

```
Attributes[OutputPlot] = {Protected, ReadProtected}
```

OutputType is an option for BmatrixCalc. Values can be "Matrix" of "Gradients". ≫

```
Attributes[OutputType] = {Protected, ReadProtected}
```

PhaseEncoding is an options of GradSeq. Values can be "A", "P", "R" and "L". ≫

```
Attributes[PhaseEncoding] = {Protected, ReadProtected}
```

Runs is an option for GenerateGradients. Set how often the minimalization
function is run. The best solution of all runs is the output. Default value is 1. ≫

```
Attributes[Runs] = {Protected, ReadProtected}
```

Steps is the number of step that is used in Generate Grads. ≫

```
Attributes[Steps] = {Protected, ReadProtected}
```

StepSizeI is an option for GradBmatrix.
Specifies the integration stepsize is Method –> "Numerical" is used. ≫

```
Attributes[StepSizeI] = {Protected, ReadProtected}
```

SwitchAxes is an option for GradSeq. Defaul value is {{1,2,3},{1,2,3}}. First three
values are for diffusion gradients last three are for the acquisition gradients. ≫

```
Attributes[SwitchAxes] = {Protected, ReadProtected}
```

UnitMulti is an option for GradSeq. Defaul
value is 10^–3. Defines the scaling of the gradient strength. ≫

```
Attributes[UnitMulti] = {Protected, ReadProtected}
```

UseGrad is an option for GradSeq. The default value
is {0, 1, {1, 0}, 1} where {grex, gr180, {grepi1, grepi2}, grdiff, grflow}. ≫

```
Attributes[UseGrad] = {Protected, ReadProtected}
```

VisualOpt is an option for GenerateGradients. Show
the minimalization proces of eacht calculation step. Default is False. ≫

```
Attributes[VisualOpt] = {Protected, ReadProtected}
```

ImportTools

Functions

BvalRead[file] imports the bvalue from a .dcm file. file must be a string. ≫

```
Attributes[BvalRead] = {Protected, ReadProtected}
```

```
SyntaxInformation[BvalRead] = {ArgumentsPattern → {_}}
```

GradRead[filename] imports the diffusion gradient direction from a .dcm file.
filename must be a string. ≫

```
Attributes[GradRead] = {Protected, ReadProtected}
```

```
Options[GradRead] = {ConvertDcm → True}
```

```
SyntaxInformation[GradRead] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

ReadBrukerDiff[""] imports the bruker diffusion data selected by the input dialog.
ReadBrukerDiff["file"] imports the bruker diffusion data from "file", file must be location of 2dseq. ≫

```
Attributes[ReadBrukerDiff] = {Protected, ReadProtected}
```

```
Options[ReadBrukerDiff] = {BmatrixOut → True}
```

```
SyntaxInformation[ReadBrukerDiff] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

ReadBvalue[folder,nr] imports the gradient
directions from the dicom header of the first nr of files in de given folder.
folder must be a string, nr must be a int. Uses BvalRead. ≫

```
Attributes[ReadBvalue] = {Protected, ReadProtected}
```

```
SyntaxInformation[ReadBvalue] = {ArgumentsPattern → {_, _}}
```

ReadDicom[folder] imports all dicom files from the given folder.
ReadDicom[{file1, file2,...}] imports all the given filenames.
ReadDicom[folder, {file1, file2,...}] imports all the given filenames from the given folder.
ReadDicom[folder, partsize] imports all dicom
files from the given folder and partions them in given partsize.
ReadDicom[{file1, file2, ...}, partsize] imports all the given filenames
and partions them in given partsize.
ReadDicom[folder, {file1, file2, ...}, partsize] imports all the given filenames
from the given folder and partions them in given partsize. ≫

```
Attributes[ReadDicom] = {Protected, ReadProtected}
```

```
Options[ReadDicom] = {ScaleCorrect → False}
```

```
SyntaxInformation[ReadDicom] = {ArgumentsPattern → {_, _., _., OptionsPattern[]}}
```

ReadDicomDiff[folder, part] imports all dicom files
from the given folder and the corresponding diffusion parameters.

part is the number of diffusion images per slice including the unweighted images. ≫

```
Attributes[ReadDicomDiff] = {Protected, ReadProtected}
```

```
Options[ReadDicomDiff] = {ScaleCorrect → False}
```

```
SyntaxInformation[ReadDicomDiff] = {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

ReadDicomDir[file] reads the image data from a dicom directory. ≫

```
Attributes[ReadDicomDir] = {Protected, ReadProtected}
```

```
SyntaxInformation[ReadDicomDir] = {ArgumentsPattern → {_, _.}}
```

ReadDicomDirDiff[file] reads the image
data and relevant diffuison parameters from a dicom directory. ≫

```
Attributes[ReadDicomDirDiff] = {Protected, ReadProtected}
```

```
Options[ReadDicomDirDiff] = {RotateGradient → True}
```

```
SyntaxInformation[ReadDicomDirDiff] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

ReadGradients[folder, nr] imports the diffusion gradient
directions from the dicom header of the first nr of files in de given folder.

folder must be a string, nr must be a int. Uses GradRead. ≫

```
Attributes[ReadGradients] = {Protected, ReadProtected}
```

```
SyntaxInformation[ReadGradients] = {ArgumentsPattern → {_, _}}
```

ReadVoxSize[filename] imports the voxelsize from a .dcm file. filename must be a string.
Imports the pixel and slice spacing from the dicom
header. Output is a list containg the voxels size {slice thickness, x, y}. ≫

```
Attributes[ReadVoxSize] = {Protected, ReadProtected}
```

```
SyntaxInformation[ReadVoxSize] = {ArgumentsPattern → {_}}
```

ShiftPar[B0file.dcm,DTIfile.dcm] imports the parameters from the dicom
headeand and calculates the needed values to preform B0 field map correction.
Needs a B0 dicom file and a diffusion dicom file. ≫

```
Attributes[ShiftPar] = {Protected, ReadProtected}
```

```
SyntaxInformation[ShiftPar] = {ArgumentsPattern → {_, _}}
```

Options

BmatrixOut is a option for ImportBrukerData if
True the bmatrix is given, if false the gradients and bvec are given. ≫

```
Attributes[BmatrixOut] = {Protected, ReadProtected}
```

ConvertDcm is an option for GradRead. ≫

```
Attributes[ConvertDcm] = {Protected, ReadProtected}
```

RotateGradient is an option for ReadDicomDirDiff. If False it
will also output the gradient direction as stored in the dicom header. ≫

```
Attributes[RotateGradient] = {Protected, ReadProtected}
```

ScaleCorrect is an option for ReadDicom, ReadDicomDiff, ReadDicomDir and ReadDicomDirDiff. The
dicom image values are corrected for rescale slope, scale slope and rescale intercept. ≫

```
Attributes[ScaleCorrect] = {Protected, ReadProtected}
```

IVIMTools

Functions

BayesianIVIMFit2[data, bval, init, mask] performs bayesian IVIM fit of data.

data is the data which should be {slice, Ndiff, x, y}.
bval is the bvector whould be length Ndiff.
init is the initalization of the bayesian
    fit which comes from IVIMCalc, (without S0 using 2 compartments).
mask is the region in which the bayesian fit is performed.

output is {f1, dc, pdc1}. ≫

```
Attributes[BayesianIVIMFit2] = {Protected, ReadProtected}
```

```
Options[BayesianIVIMFit2] = {ChainSteps → {20000, 1000, 10}, UpdateStep → {0.5, 0.2, 0.5},
  FixPseudoDiff → False, CorrectPar → True, FixPseudoDiffSD → 0.5, OutputSamples → False,
  FitConstrains → ThetaConv[{{-7.6, 7.6}, {-10., -5.7}, {-7., 0.}}]}
```

```
SyntaxInformation[BayesianIVIMFit2] = {ArgumentsPattern → {_, _, _, _, OptionsPattern[]}}
```

BayesianIVIMFit3[data, bval, init, mask] performs bayesian IVIM fit of data.

data is the data which should be {slice, Ndiff, x, y}.
bval is the bvector whould be length Ndiff.
init is the initalization of the bayesian
    fit which comes from IVIMCalC, (without S0 using 3 compartments).
mask is the region in which the bayesian fit is performed.

output is {f1, f2, dc, pdc1, pdc2}. ≫

```
Attributes[BayesianIVIMFit3] = {Protected, ReadProtected}
```

```
Options[BayesianIVIMFit3] =
 {ChainSteps → {20000, 1000, 10}, UpdateStep → {0.5, 0.5, 0.1, 0.5, 0.5}, FixPseudoDiff → False,
  CorrectPar → True, OutputSamples → False, FixPseudoDiffSD → 0.5, FitConstrains →
   ThetaConv[{{-7.6, 7.6}, {-7.6, 7.6}, {-10., -5.5}, {-6.5, -2.3}, {-5.2, 0.}}]}
```

```
SyntaxInformation[BayesianIVIMFit3] = {ArgumentsPattern → {_, _, _, _, OptionsPattern[]}}
```

CorrectParMap[par, constraints, mask] removes
    the IVIM parameters outside the constraints within the mask.

par is {f1, dc, pdc1} or {f1, f2, dc, pdc1, pdc2}.
constraints are the lower and upper constraints for each parameters {{min, max},...}
mask has the same dimensions as the parameter maps.

output are the corrected paremeter maps. ≫

```
Attributes[CorrectParMap] = {Protected, ReadProtected}
```

```
SyntaxInformation[CorrectParMap] = {ArgumentsPattern → {_, _, _}}
```

FConvert[F] convers the fraction F from log space. ≫

```
Attributes[FConvert] = {Protected, ReadProtected}
```

```
SyntaxInformation[FConvert] = {ArgumentsPattern → {_}}
```

FConverti[f] converts the fraction f to log space. ≫

```
Attributes[FConverti] = {Protected, ReadProtected}
```

```
SyntaxInformation[FConverti] = {ArgumentsPattern → {_}}
```

FracCorrect[fraction, time] corrects the signal fraction calculated
    with the IVIM model for tissue relaxation and acquisition parameters.
After correction the signal fraction can be regarded as volume fraction.
FracCorrect[{fraction1, fraction2}, time] corrects
    the signal fraction1 and fraction2 from a 3 compartement IVIM model.

time is {{te, tr}, {t2t, t21}, {t1t, t11}} or {{te, tr}, {t2t, t21, t22}, {t1t, t11, t12}}
where t2t and t1t are "tissue"
    relaxation times and t11 t12, t21 and t22 the "fluid" relaxation times

output is the corrected fraction maps ≫

```
Attributes[FracCorrect] = {Protected, ReadProtected}
```

```
SyntaxInformation[FracCorrect] = {ArgumentsPattern → {_, _, _.}}
```

HistogramPar[data, {constraints, Nbins}, style, color, range] plots histograms of IVIM solution.
HistogramPar[data, {constraints, Nbins, mu,
    conv}, components, color, range] plots histograms of IVIM solution.

data is {f1, dc, pdc1} or {f1, f2, dc, pdc1, pdc2}.
constraints are the ranges of the x–axes for the plots.
Nbins are the number of histogram bins.
style is the plot type, can be 1, 2, or 3.
color is the color of the histogram.
range are the ranges of the y–axes.

output is a row of histograms. ≫

```
Attributes[HistogramPar] = {Protected, ReadProtected}
```

```
SyntaxInformation[HistogramPar] = {ArgumentsPattern → {_, _, _, _, _.}}
```

IVIMCalc[data, binp, init] calculates the IVIM fit.

data should be 1D ,2D, 3D or 4D.
binp should be full bmatrix which can be calculated from the bvecs en bvals using Bmatrix.
init should are the initialization parameters for 2 components
this is {S0, f, D, Dp} for 3 componentes this is {S0, f1, f2, D, Dp1, Dp2}.

output is {S0, f1, D, pD1} or {S0, f1, f2, D, pD1, pD2}. ≫

```
Attributes[IVIMCalc] = {Protected, ReadProtected}
```

```
Options[IVIMCalc] = {Method → Automatic, Parallelize → True, MonitorIVIMCalc → True,
  IVIMFixed → False, IVIMConstrained → True, IVIMTensFit → False, IVIMComponents → 2,
  IVIMConstrains → {{0.8, 1.2}, {0, 1}, {0.0005, 0.0035}, {0.001, 0.5}, {0.001, 0.5}}}
```

```
SyntaxInformation[IVIMCalc] = {ArgumentsPattern → {_, _, _, _., OptionsPattern[]}}
```

IVIMCorrectData[data, {S0, f, pdc}, bval] removes the ivim signal from the data.

data is the original data.
{S0, f, pdc} are the solution to a 2 compartment IVIM fit using IVIMCalc or BayesianIVIMFit2.
bval are the bvalues.

output is the corrected data. ≫

```
Attributes[IVIMCorrectData] = {Protected, ReadProtected}
```

```
Options[IVIMCorrectData] = {FilterMaps → True, FilterType → Median, FilterSize → 1}
```

```
SyntaxInformation[IVIMCorrectData] = {ArgumentsPattern → {_, {_, _, _}, _, OptionsPattern[]}}
```

IVIMFunction[] gives the IVIM function with 2 comps.
IVIMFunction[components] gives the IVIM function.
IVIMFunction[components, type] gives the IVIM function.

type can be "Normal" or "Exp".
componenets can be 2 or 3.

output is the function with b, S0, f1, f2, D, pD1, pD2 as parameters ≫

```
Attributes[IVIMFunction] = {Protected, ReadProtected}
```

```
SyntaxInformation[IVIMFunction] = {ArgumentsPattern → {_, _}}
```

IVIMResiduals[data, binp, pars] calculates the root mean square
residuals of an IVIM fit ussing IVIMCalc, BayesianIVIMFit2 or BayesianIVIMFit3. ≫

```
Attributes[IVIMResiduals] = {Protected, ReadProtected}
```

ThetaConv[{F1, Fc, pDc}] converts the parameters from Log
space to normal space. Is used in BayesianIVIMFit2 and BayesianIVIMFit3.
ThetaConv[{F1, F2, Dc, pDc1}] converts the parameters from Log space to
normal space. Is used in BayesianIVIMFit2 and BayesianIVIMFit3.
ThetaConv[{F1, F2, Dc, pDc1, pDc2}] converts the parameters from Log space to
normal space. Is used in BayesianIVIMFit2 and BayesianIVIMFit3. ≫

```
Attributes[ThetaConv] = {Protected, ReadProtected}
```

ThetaConvi[{f, dc, pdc}] converts the parameters from Normal
space to Log space. Is used in BayesianIVIMFit2 and BayesianIVIMFit3.
ThetaConvi[{f1, f2, dc, pdc1}] converts the parameters from Normal space
to Log space. Is used in BayesianIVIMFit2 and BayesianIVIMFit3.
ThetaConvi[{f1, f2, dc, pdc1, pdc2}] converts the parameters from Normal space
to Log space. Is used in BayesianIVIMFit2 and BayesianIVIMFit3. ≫

```
Attributes[ThetaConvi] = {Protected, ReadProtected}
```

```
SyntaxInformation[ThetaConvi] = {ArgumentsPattern → {_}}
```

Options

ChainSteps is an option for BayesianIVIMFit2
and BayesianIVIMFit3. It determines how long the algorithm runs.
three values must be given {itterations, burn steps, sample density}. ≫

```
Attributes[ChainSteps] = {Protected, ReadProtected}
```

CorrectPar is an option for BayesianIVIMFit2 and BayesianIVIMFit3.
If True it removes the values outside the contraints using CorrectParMap ≫

```
Attributes[CorrectPar] = {Protected, ReadProtected}
```

FilterMaps is an option for IVIMCorrectData. If
True the IVIM parameter maps are filtered before signal correction ≫

```
Attributes[FilterMaps] = {Protected, ReadProtected}
```

FilterSize is an option for IVIMCorrectData. If FilterMaps is True it gives the kernel size. ≫

```
Attributes[FilterSize] = {Protected, ReadProtected}
```

FilterType is an option for IVIMCorrectData. If FilterMaps
is True it tells which filter to use. can be "Median" of "Gausian" ≫

```
Attributes[FilterType] = {Protected, ReadProtected}
```

FitConstrains is an option for BayesianIVIMFit2
and BayesianIVIMFit3. Gives the contraints of the parameters.
The values are used for displaying the histograms and for the initialization if CorrectPar is True ≫

```
Attributes[FitConstrains] = {Protected, ReadProtected}
```

FixPseudoDiff is an option for BayesianIVIMFit2 and BayesianIVIMFit3.
If the pDc1 and pD2 were fixed in IVIMCalc this value should be True. ≫

```
Attributes[FixPseudoDiff] = {Protected, ReadProtected}
```

FixPseudoDiffSD is an option for BayesianIVIMFit2 and BayesianIVIMFit3.
Gives the standard deviation of pDc1 and pD2 if FixPseudoDiff is True ≫

```
Attributes[FixPseudoDiffSD] = {Protected, ReadProtected}
```

IVIMComponents is an option for IVIMCalc. Default
value is 2, the tissue and the blood component. can also be set to 3. ≫

```
Attributes[IVIMComponents] = {Protected, ReadProtected}
```

IVIMConstrained is an option for IVIMCalc. When set
True the fit wil be constrained to the values given in IVIMConstrains. ≫

```
Attributes[IVIMConstrained] = {Protected, ReadProtected}
```

IVIMConstrains is an option for IVIMCalc. Default values are: {{0.8, 1.2}, {0, 1}, {0.0005,
0.0035}, {0.005, 0.5}, {0.002, 0.015}}. Where {{S0 in percentage},{fractions},{tissue
diffusion},{blood compartment Dp},{third compartment}}. ≫

```
Attributes[IVIMConstrains] = {Protected, ReadProtected}
```

IVIMFixed is an option for IVIMCalc and the default value is False. When
set True the pseudo diffusion wil be fixed to the parameter given as init.
When set to "One" only the fast component of a 3 compartment fit is fixed. ≫

```
Attributes[IVIMFixed] = {Protected, ReadProtected}
```

IVIMTensFit is an option for IVIMCalc. When set
True the tissue diffusion component wil be calculated as a tensor. ≫

```
Attributes[IVIMTensFit] = {Protected, ReadProtected}
```

Method is an option for various algorithm–intensive
functions that specifies what internal methods they should use. ≫

```
Attributes[Method] = {Protected}
```

MonitorIVIMCalc is an option for IVIMCalc. When true the proceses of the calculation is shown. ≫

```
Attributes[MonitorIVIMCalc] = {Protected, ReadProtected}
```

OutputSamples is an option for BayesianIVIMFit2 and BayesianIVIMFit3.
If set True the full marcov chain is given as an additionaln output. ≫

```
Attributes[OutputSamples] = {Protected, ReadProtected}
```

Parallelize[*expr*] evaluates *expr* using automatic parallelization. ≫

```
Attributes[Parallelize] = {HoldFirst, Protected}
```

```
Parallelize[Parallel`Kernels`Private`args$___] :=
 (Parallel`Protected`doAutolaunch[TrueQ[Parallel`Static`$enableLaunchFeedback]];
  Parallelize[Parallel`Kernels`Private`args$])
```

```
Options[Parallelize] = {Method → Automatic, DistributedContexts :→ $Context}
```

UpdateStep is an option for BayesianIVIMFit2 and BayesianIVIMFit3. It determines how
often the parameters are updated. Is optimized during the first 500 burn steps. ≫

```
Attributes[UpdateStep] = {Protected, ReadProtected}
```

JcouplingTools

Functions

GetSpinSystem[name] get a spinsystem that can be used in
SimHamiltonian. Current implementes systems are "glu", "lac", "gaba",
"fatGly", "fatAll", "fatEnd", "fatDouble", "fatStart", and "fatMet". ≫

```
Attributes[GetSpinSystem] = {Protected, ReadProtected}
```

```
Options[GetSpinSystem] = {CenterFrequency → 4.65}
```

```
SyntaxInformation[GetSpinSystem] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

PhaseAlign[spec] automatically phase aligns
the spectrum by maximizing the Real part of the spectrum. ≫

```
Attributes[PhaseAlign] = {Protected, ReadProtected}
```

```
SyntaxInformation[PhaseAlign] = {ArgumentsPattern → {_}}
```

PlotSpectrum[ppm, spec] plots the spectrum, ppm and spec can be generated using SimReadout. ≫

```
Attributes[PlotSpectrum] = {Protected, ReadProtected}
```

```
Options[PlotSpectrum] = {PlotRange → {{0, 6}, Full}, SpectrumColor → GrayLevel[0]}
```

```
SyntaxInformation[PlotSpectrum] = {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

SequencePulseAcquire[din, H] performs a pulsaquire experiment
of the spin system din given the hamiltonian H with a 90 Degree pulse.
SequencePulseAcquire[din, H, b1] performs a pulsaquire experiment of the spin
system din given the hamiltonian H with a 90 Degree pulse and b1.
The output is a new spinsystem dout. ≫

```
Attributes[SequencePulseAcquire] = {Protected, ReadProtected}
```

```
SyntaxInformation[SequencePulseAcquire] = {ArgumentsPattern → {_, _, _.}}
```

SequenceSpinEcho[din, H, te] performs a spin echo experiment with echo time te
of the spin system din given the hamiltonian H with a 90 and 180 Degree pulse.
SequenceSpinEcho[din, H, te, b1] performs a spin echo experiment with echo time te of the
spin system din given the hamiltonian H with a 90 and 180 Degree pulse and b1.
The output is a new spinsystem dout. ≫

```
Attributes[SequenceSpinEcho] = {Protected, ReadProtected}
```

```
SyntaxInformation[SequenceSpinEcho] = {ArgumentsPattern → {_, _, _, _.}}
```

SequenceSteam[din, H, {te, tm}] performs a stimulated echo experiment with echo time te and mixing
time tm of the spin system din given the hamiltonian H with 3 90 Degree pulses.
The output is a new spinsystem dout. ≫

```
Attributes[SequenceSteam] = {Protected, ReadProtected}
```

```
SyntaxInformation[SequenceSteam] = {ArgumentsPattern → {_, _, {_, _}}}
```

SequenceTSE[din ,H, {te, necho}, {ex, ref}] performs a multi echo spin echo
experiment with echo time te with necho echos of the spin system din given
the hamiltonian H using ex Degree exitation and ref Degree refocus pulses.
SequenceTSE[din ,H, {te, necho}, {ex, ref}, b1_:1] performs a multi echo spin echo
experiment with echo time te with necho echos of the spin system din given the
hamiltonian H using ex Degree exitation and ref Degree refocus pulses and b1.
The output is a new spinsystem dout. ≫

```
Attributes[SequenceTSE] = {Protected, ReadProtected}
```

```
SyntaxInformation[SequenceTSE] = {ArgumentsPattern → {_, _, {_, _}, {_, _}, _.}}
```

SimAddPhase[din ,H ,phase] adds phase to the spin system din given the hamiltonian H.
din and H are generated by SimHamiltonian.
The output is a new spinsystem dout. ≫

```
Attributes[SimAddPhase] = {Protected, ReadProtected}
```

```
SyntaxInformation[SimAddPhase] = {ArgumentsPattern → {_, _, _}}
```

SimEvolve[din,H,t] evolves the spin system din given the
hamiltonian H over a time t. din and H are generated by SimHamiltonian.
The output is a new spinsystem dout. ≫

```
Attributes[SimEvolve] = {Protected, ReadProtected}
```

```
SyntaxInformation[SimEvolve] = {ArgumentsPattern → {_, _, _}}
```

SimHamiltonian[sysi] simulates the hamiltionan for
a given spin system. The spinsystem is generated by GetSpinSystem.
The output is the spin system and hamiltonian structure. ≫

```
Attributes[SimHamiltonian] = {Protected, ReadProtected}
```

```
Options[SimHamiltonian] = {FieldStrength → 3}
```

```
SyntaxInformation[SimHamiltonian] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

SimReadout[din, H] performs a readout of a spinsystem din with hamiltonian H.
Output is {time,fids,ppm,spec,dout}, which are the free induction decay fids with
its time, the spectrum spec with its ppm and the evolved spin system dout. ≫

```
Attributes[SimReadout] = {Protected, ReadProtected}
```

```
Options[SimReadout] = {ReadoutOutput → all, ReadoutPhase → 90,
  Linewidth → 5, LinewidthShape → L, ReadoutSamples → 2046, ReadoutBandwith → 2000}
```

```
SyntaxInformation[SimReadout] = {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

SimRotate[din, H ,angle] rotates the spin system
din given the hamiltonian H over angele with phase 90 degrees.
SimRotate[din, H ,angle, phase] rotates the spin system din given
the hamiltonian H over angele with phase.
din and H are generated by SimHamiltonian.
The output is a new spinsystem dout. ≫

```
Attributes[SimRotate] = {Protected, ReadProtected}
```

```
SyntaxInformation[SimRotate] = {ArgumentsPattern → {_, _, _, _.}}
```

SimSignal[din, H] performs a readout of a spinsystem din with hamiltonian H.
Output is the complex signal. ≫

```
Attributes[SimSignal] = {Protected, ReadProtected}
```

```
Options[SimSignal] = {ReadoutOutput → all}
```

```
SyntaxInformation[SimSignal] = {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

SimSpoil[din] spoils all the non zeroth order states of a spin system.
The output is a new spinsystem dout. ≫

```
Attributes[SimSpoil] = {Protected, ReadProtected}
```

```
SyntaxInformation[SimSpoil] = {ArgumentsPattern → {_}}
```

SysTable[sys] shows the spinsystem as a table. The spinsytem is obtained form GetSpinSystem. ≫

```
Attributes[SysTable] = {Protected, ReadProtected}
```

```
SyntaxInformation[SysTable] = {ArgumentsPattern → {_}}
```

Options

CenterFrequency is an option for GetSpinSystem and defines the center frequency. ≫

```
Attributes[CenterFrequency] = {Protected, ReadProtected}
```

FieldStrength is an option for SimHamiltonian. It
defines the field strength for which the hamiltonian is calculated. ≫

```
Attributes[FieldStrength] = {Protected, ReadProtected}
```

Linewidth is an option for SimReadout and defines the spectral linewidth. ≫

```
Attributes[Linewidth] = {Protected, ReadProtected}
```

LinewidthShape is an option for SimReadout and defines the linewidth shape, values can be
"L", "G" or "L", which are Laplacian, Gaussion or a combination, respectively. ≫

```
Attributes[LinewidthShape] = {Protected, ReadProtected}
```

PlotRange is an option for graphics functions
that specifies what range of coordinates to include in a plot. ≫

```
Attributes[PlotRange] = {Protected, ReadProtected}
```

ReadoutBandwith is an option for SimReadout defines the spectral bandwith. ≫

```
Attributes[ReadoutBandwith] = {Protected, ReadProtected}
```

ReadoutOutput is an option for SimReadout and SimSignal and values
can be "all" and "each". When set to "all" the total signal and signal is given,
when set to "each" the signal or spectrum for each peak is given seperately. ≫

```
Attributes[ReadoutOutput] = {Protected, ReadProtected}
```

ReadoutPhase is an option for SimReadout and defines the readout phase. ≫

```
Attributes[ReadoutPhase] = {Protected, ReadProtected}
```

ReadoutSamples is an option for SimReadout
and defines the number of readout samples for the spectrum. ≫

```
Attributes[ReadoutSamples] = {Protected, ReadProtected}
```

SpectrumColor is an option for PlotSpectrum and defines the spectrum color. ≫

```
Attributes[SpectrumColor] = {Protected, ReadProtected}
```

MaskingTools

Functions

GetMaskData[data, mask] retruns the data selected by the mask. ≫

```
Attributes[GetMaskData] = {Protected, ReadProtected}
```

```
Options[GetMaskData] = {GetMaskOutput → All}
```

```
SyntaxInformation[GetMaskData] = {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

HomoginizeData[data, mask] tries to homoginize
the data within the mask by removing intensity gradients. ≫

```
Attributes[HomoginizeData] = {Protected, ReadProtected}
```

```
SyntaxInformation[HomoginizeData] = {ArgumentsPattern → {_, _}}
```

Mask[data] creates a mask by automatically finding a threshold.
Mask[data, min]creates a mask which selects only data above the min value.
Mask[data,{min,max}] creates a mask which selects data between the min and max value. ≫

```
Attributes[Mask] = {Protected, ReadProtected}
```

```
Options[Mask] = {MaskSmoothing → False, MaskComponents → 1, MaskClosing → 5, MaskFiltKernel → 2}
```

```
SyntaxInformation[Mask] = {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

MaskData[data, mask] applies a mask to data. mask can be 2D or 3D, data can be 2D, 3D or 4D. ≫

```
Attributes[MaskData] = {Protected, ReadProtected}
```

```
SyntaxInformation[MaskData] = {ArgumentsPattern → {_, _}}
```

MeanSignal[data] calculates the mean signal per volume of 4D data.
MeanSignal[data, pos] calculates the
mean signal per volume of 4D data for the given list of positions. ≫

```
Attributes[MeanSignal] = {Protected, ReadProtected}
```

```
Options[MeanSignal] = {UseMask → True}
```

```
SyntaxInformation[MeanSignal] = {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

MergeSegmentations[masks, labels] generates an ITKsnap or
slices3D compatible segmentation from individual masks and label numbers.
Output is a labled segmentation. ≫

```
Attributes[MergeSegmentations] = {Protected, ReadProtected}
```

```
SyntaxInformation[MergeSegmentations] = {ArgumentsPattern → {_, _}}
```

NormalizeData[data] normalizes the data to the mean signal of
the data. For 4D data it normalizes to the first volume of the 4th dimension.
NormalizeData[data,{min,max}] normalizes the data between min and max. ≫

```
Attributes[NormalizeData] = {Protected, ReadProtected}
```

```
SyntaxInformation[NormalizeData] = {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

RemoveMaskOverlaps[mask] removes the overlaps
between multiple masks. Mask is a 4D dataset with {z, masks, x, y} ≫

```
Attributes[RemoveMaskOverlaps] = {Protected, ReadProtected}
```

```
SyntaxInformation[RemoveMaskOverlaps] = {ArgumentsPattern → {_}}
```

RescaleSegmentation[data, dim] rescales segmentations to given dimensions.
RescaleSegmentation[data, {vox1, vox2}]
rescales segmentations from voxelsize vox1 to voxelsize vox2. ≫

```
Attributes[RescaleSegmentation] = {Protected, ReadProtected}
```

```
SyntaxInformation[RescaleSegmentation] = {ArgumentsPattern → {_, _}}
```

ROIMask[maskdim, {name–>{{{x,y},slice}..}..}] crates mask from coordinates x and y at slice.
maskdim is the dimensions of the output {zout,xout,yout}. ≫

```
Attributes[ROIMask] = {Protected, ReadProtected}
```

```
SyntaxInformation[ROIMask] = {ArgumentsPattern → {_, _, _.}}
```

SegmentMask[mask, n] devides a mask in n
equal segments along the slice direction. n must be an integer. ≫

```
Attributes[SegmentMask] = {Protected, ReadProtected}
```

```
SyntaxInformation[SegmentMask] = {ArgumentsPattern → {_, _, _.}}
```

SmoothMask[mask] generates one clean masked volume form a noisy mask. ≫

```
Attributes[SmoothMask] = {Protected, ReadProtected}
```

```
Options[SmoothMask] = {MaskComponents → 1, MaskClosing → 5, MaskFiltKernel → 2}
```

```
SyntaxInformation[SmoothMask] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

SmoothSegmentation[masks] smooths segmentations
and removes the overlaps between multiple segmentations. ≫

```
Attributes[SmoothSegmentation] = {Protected, ReadProtected}
```

```
Options[SmoothSegmentation] = {MaskFiltKernel → 2}
```

```
SyntaxInformation[SmoothSegmentation] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

SplitSegmentations[segmentation] splits a lable
mask from ITKsnap or slicer3D in seperate masks and label numbers.
Output is masks and label numbers, {mask, labs}. ≫

```
Attributes[SplitSegmentations] = {Protected, ReadProtected}
```

```
SyntaxInformation[SplitSegmentations] = {ArgumentsPattern → {_}}
```

Options

GetMaskOutput is an option for GetMaskData. Defaul is "Slices" which
gives the mask data per slices. Else the entire mask data is given as output. ≫

```
Attributes[GetMaskOutput] = {Protected, ReadProtected}
```

MaskClosing is an option for Mask and
SmoothMask. The size of the holes in the mask that will be closed ≫

```
Attributes[MaskClosing] = {Protected, ReadProtected}
```

MaskComponents is an option for Mask and
SmoothMask. Determinse the amount of largest clusters used as mask. ≫

```
Attributes[MaskComponents] = {Protected, ReadProtected}
```

MaskFiltKernel is an option for Mask, SmoothMask
and SmoothSegmentation. How mucht the contours are smoothed. ≫

```
Attributes[MaskFiltKernel] = {Protected, ReadProtected}
```

MaskSmoothing is an options for Mask, if set to True
it smooths the mask, by closing holse and smoothing the contours. ≫

```
Attributes[MaskSmoothing] = {Protected, ReadProtected}
```

UseMask is a function for MeanSignal and DriftCorrect ≫

```
Attributes[UseMask] = {Protected, ReadProtected}
```

NiftiTools

Functions

CompressNiiFiles[] promts for a folder. It
then compresses all nii files to .nii.gz files in the selected folder.
CompressNiiFiles[folder] compresses all nii files to .nii.gz files in folder. ≫

```
Attributes[CompressNiiFiles] = {Protected, ReadProtected}
```

```
SyntaxInformation[CompressNiiFiles] = {ArgumentsPattern → {_., _.}}
```

DcmToNii[] converts a dicom folder to nii, you will be prometed for the location of the folders.
DcmToNii[{"input","ouput"}] converts the
"input" dicom folder to nii files which are place in the "output" folder.
For this function to work the dcm2niix.exe file should be present in the QMRITools aplication folder. ≫

```
Attributes[DcmToNii] = {Protected, ReadProtected}
```

```
Options[DcmToNii] = {CompressNii → True, Method → Automatic}
```

```
SyntaxInformation[DcmToNii] = {ArgumentsPattern → {_., _., OptionsPattern[]}}
```

ExportBmat[bmat] exports the diffusion bmatrix to exploreDTI format.
ExportBmat[bmat, "file"] exports the diffusion bmatrix to "file" in the exploreDTI format. ≫

```
Attributes[ExportBmat] = {Protected, ReadProtected}
```

```
SyntaxInformation[ExportBmat] = {ArgumentsPattern → {_, _.}}
```

ExportBval[bvals] exports the diffusion bvalues to exploreDTI format.
ExportBval[bvals, "file"] exports the diffusion bvalues to "file" in the exploreDTI format. ≫

```
Attributes[ExportBval] = {Protected, ReadProtected}
```

```
SyntaxInformation[ExportBval] = {ArgumentsPattern → {_, _.}}
```

ExportBvec[grad] exports the diffusion gradients to exploreDTI format.
ExportBvec[grad, "file"] exports the diffusion gradients to "file" in the exploreDTI format. ≫

```
Attributes[ExportBvec] = {Protected, ReadProtected}
```

```
SyntaxInformation[ExportBvec] = {ArgumentsPattern → {_, _.}}
```

ExportNii[data, vox] exports the nii file and will promt for a file name.
ExportNii[data, vox, "file"] exports the nii file to the location "file". ≫

```
Attributes[ExportNii] = {Protected, ReadProtected}
```

```
Options[ExportNii] = {NiiDataType → Automatic, CompressNii → True}
```

```
SyntaxInformation[ExportNii] = {ArgumentsPattern → {_, _, _., OptionsPattern[]}}
```

ExtractNiiFiles[] promts for a folder. It then extracts all nii.gz files to .nii files in the selected folder.
ExtractNiiFiles[folder] extracts all nii.gz files to .nii files in folder. ≫

```
Attributes[ExtractNiiFiles] = {Protected, ReadProtected}

SyntaxInformation[ExtractNiiFiles] = {ArgumentsPattern → {_., _.}}
```

ImportBmat[] will promt to select the *.txt file containing the bmatrix.
ImportBmat[*.txt] imports the given *.txt file containing the bmatrix. ≫

```
Attributes[ImportBmat] = {Protected, ReadProtected}

SyntaxInformation[ImportBmat] = {ArgumentsPattern → {_., _.}}
```

ImportBval[] will promt to select the *.bval file.
ImportBval[*.bval] imports the given *.bval file. ≫

```
Attributes[ImportBval] = {Protected, ReadProtected}

SyntaxInformation[ImportBval] = {ArgumentsPattern → {_.}}
```

ImportBvalvec[] will promt to select the *.bval and *.bvec files.
ImportBvalvec[file] if file is either a *.bval
or *.bvec it will automatically import the *.bval and *.bvec files.
ImportBvalvec[*.bvec,*.bval] imports the given *.bval and *.bvec files. ≫

```
Attributes[ImportBvalvec] = {Protected, ReadProtected}

Options[ImportBvalvec] = {FlipBvec → False}

SyntaxInformation[ImportBvalvec] = {ArgumentsPattern → {_., _., OptionsPattern[]}}
```

ImportBvec[] will promt to select the *.bvec file.
ImportBvec[*.bvec] imports the given *.bvec file. ≫

```
Attributes[ImportBvec] = {Protected, ReadProtected}

Options[ImportBvec] = {FlipBvec → False}

SyntaxInformation[ImportBvec] = {ArgumentsPattern → {_., OptionsPattern[]}}
```

ImportExploreDTItens["file"] imports the *.nii export for the tensor from explore DTI. ≫

```
Attributes[ImportExploreDTItens] = {Protected, ReadProtected}
```

ImportNii[] promts to select the nii file to import.
ImportNii["file"] imports the nii file.
The default output is {data, vox}, however using NiiMethod various outputs can be given.
The Nii import is also suported using the native Import function from Mathematica. ≫

```
Attributes[ImportNii] = {Protected, ReadProtected}

Options[ImportNii] = {NiiMethod → default, NiiScaling → False}

SyntaxInformation[ImportNii] = {ArgumentsPattern → {_., OptionsPattern[]}}
```

ImportNiiDiff[] will promt for the *.nii, *.bvec and *.bval file to import.
ImportNiiDiff[*.nii] will import the *.nii file and
automatically also imports the *.bvec and *.bval is they have the same name.
ImportNiiDiff[*.nii,*.bvec,*.bval] will import the given files.
The output will be {data,grad,bvec,vox}. ≫

```
Attributes[ImportNiiDiff] = {Protected, ReadProtected}
```

```
Options[ImportNiiDiff] = {RotateGradients → False, FlipBvec → True}
```

```
SyntaxInformation[ImportNiiDiff] = {ArgumentsPattern → {_., _., _., OptionsPattern[]}}
```

ImportNiiDix["file"] imports the dixon nii file which should contain
all possible outputs given by the scanner and corrects them accordingly. ≫

```
Attributes[ImportNiiDix] = {Protected, ReadProtected}
```

ImportNiiT1["file"] imports the T1 file which should contain the echos
and the T1map calculated by the scanner and corrects them accordingly. ≫

```
Attributes[ImportNiiT1] = {Protected, ReadProtected}
```

ImportNiiT2["file"] imports the T2 file which should contain the echos
and the T2map calculated by the scanner and corrects them accordingly. ≫

```
Attributes[ImportNiiT2] = {Protected, ReadProtected}
```

Options

CompressNii is an option for DcmToNii and ExportNii. If set True .nii.gz files will be created. ≫

```
Attributes[CompressNii] = {Protected, ReadProtected}
```

FlipBvec is an option for ImportBvalvec. ≫

```
Attributes[FlipBvec] = {Protected, ReadProtected}
```

Method is an option for various algorithm–intensive
functions that specifies what internal methods they should use. ≫

```
Attributes[Method] = {Protected}
```

NiiDataType is an option of Export Nii. The number
type of Nii file can be "Integer", "Real", "Complex", or "Automatic". ≫

```
Attributes[NiiDataType] = {Protected, ReadProtected}
```

NiiMethod is an option for ImportNIi. Values can be
"data", "dataTR", "header", "scaling", "headerMat", "rotation", "all". ≫

```
Attributes[NiiMethod] = {Protected, ReadProtected}
```

NiiScaling is an option for ImportNii. It scales
the nii values with scale slope and offset for quantitative data. ≫

```
Attributes[NiiScaling] = {Protected, ReadProtected}
```

RotateGradients is an option for ImportNiiDiff. ≫

```
Attributes[RotateGradients] = {Protected, ReadProtected}
```

PhysiologyTools

Functions

AlignRespLog[physLog, respirect, scanTime] aligns respirect and physlog data. physLog is
output from ImportPhyslog.resirect is the first output from ImportRespirect. ≫

```
Attributes[AlignRespLog] = {Protected, ReadProtected}
```

```
Options[AlignRespLog] = {OutputMethod → val, SampleStep → 0.005}
```

```
SyntaxInformation[AlignRespLog] = {ArgumentsPattern → {_, _, _, OptionsPattern[]}}
```

ImportPhyslog[] imports all physlog files from the folder selcted.
ImportPhyslog["forder"] imports all physlog files from "folder" selcted. ≫

```
Attributes[ImportPhyslog] = {Protected, ReadProtected}
```

```
SyntaxInformation[ImportPhyslog] = {ArgumentsPattern → {_.}}
```

ImportRespirect[] impors all the respirect log files from the folder selcted.
ImportRespirect["folder"] impors all the respirect log files from the "folder" selcted. ≫

```
Attributes[ImportRespirect] = {Protected, ReadProtected}
```

```
SyntaxInformation[ImportRespirect] = {ArgumentsPattern → {_.}}
```

PlotPhyslog[{time, resp}, {start, stop}] plots the physlog from ImportPhyslog.
PlotPhyslog[{time, resp}, {start, stop}, scanTime] plots the physlog from ImportPhyslog. ≫

```
Attributes[PlotPhyslog] = {Protected, ReadProtected}
```

```
SyntaxInformation[PlotPhyslog] = {ArgumentsPattern → {_, _, _.}}
```

PlotRespiract[data, dataP, scantimes] plots the respirect data to correct peaks. data and dataP are
the first outputs of ImportResirect. scantimes is the output from AlignRespLog.
PlotRespiract[data, dataP, scantimes, steps]. ≫

```
Attributes[PlotRespiract] = {Protected, ReadProtected}
```

```
SyntaxInformation[PlotRespiract] = {ArgumentsPattern → {_, _, _.}}
```

Options

OutputMethod can be "val" or "plot" ≫

```
Attributes[OutputMethod] = {Protected, ReadProtected}
```

SampleStep is an option for AlignRespiract ≫

```
Attributes[SampleStep] = {Protected, ReadProtected}
```

PlottingTools

Functions

GetSliceData[data, offsets] gets the slices from
the data defined by offsets which are obtained by GetSlicePosisions.
GetSliceData[data, offsets, vox] gets the slices from the data defined by
offsets which are obtained by GetSlicePosisions in mm.
The offsets can also be provided manually which is {{AX,..},{COR,..},{SAG,..}}. ≫

```
Attributes[GetSliceData] = {Protected, ReadProtected}
```

```
SyntaxInformation[GetSliceData] = {ArgumentsPattern → {_, _, _.}}
```

GetSlicePositions[data] finds the position of slices with the maximal signal in voxel index.
GetSlicePositions[data, vox] find the position of slices with the maximal signal in mm. ≫

```
Attributes[GetSlicePositions] = {Protected, ReadProtected}
```

```
Options[GetSlicePositions] =
 {MakeCheckPlot → False, DropSlices → {1, 1, 1}, PeakNumber → {1, 1, 2}}
```

```
SyntaxInformation[GetSlicePositions] = {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

GradientPlot[bvec, bval] plots the given
bvec with position of the gradients scaled according to the bval. ≫

```
Attributes[GradientPlot] = {Protected, ReadProtected}
```

```
Options[GradientPlot] =
 {PlotSpace → bspace, PlotColor → Auto, SphereSize → 0.05, PositiveZ → False}
```

```
SyntaxInformation[GradientPlot] = {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

ListSpherePlot[points] plots 3D points as spheres ≫

```
Attributes[ListSpherePlot] = {Protected, ReadProtected}
```

```
Options[ListSpherePlot] = {SphereSize → 2, SphereColor → Automatic}
```

```
SyntaxInformation[ListSpherePlot] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

MakeSliceImages[imgData] generates images from the imgData which is obtained form GetSliceData.
MakeSliceImages[imgData, vox] generates images from the imgData which is
    obtained form GetSliceData, vox is used for the correct aspect ratio of the images.
MakeSliceImages[imgData, {labData, labels}] generates images from the imgData
    which is obtained form GetSliceData with an overlay of the segmentations in
    labData, which can also be obtained using GetSliceData on the segmentations.
labels should be the label numbers used in the original segmentation
    (to allow correct scaling between slices).
MakeSliceImages[imgData, {labData, labels},vox] generates images from the imgData
    which is obtained form GetSliceData with an overlay of the segmentations
    in labData, which can also be obtained using GetSliceData on the
    segmentations, vox is used for the correct aspect ratio of the images. ≫

```
Attributes[MakeSliceImages] = {Protected, ReadProtected}
```

```
Options[MakeSliceImages] =
  {PlotRange → Automatic, ColorFunction → GrayTones, ImageLegend → False}
```

```
SyntaxInformation[MakeSliceImages] = {ArgumentsPattern → {_, _., _., OptionsPattern[]}}
```

PlotContour[data, vox] creates a contour of the data.
PlotContour[data, vox, scale] creates
    a contour of the data with the surface colored acording to scale.
PlotContour[data, vox, scale, range] creates a contour of the data with the
    surface colored acording to scale with a fixed plotrange. ≫

```
Attributes[PlotContour] = {Protected, ReadProtected}
```

```
Options[PlotContour] = {ContourStyle → {GrayLevel[0.5], 0.25}}
```

```
SyntaxInformation[PlotContour] = {ArgumentsPattern → {_, _, _., _., OptionsPattern[]}}
```

PlotCorrection[w] plots deformation vectors w {w1,w2..} generated by
    Registration2D and Registration3D for multiple datasets or registration steps. ≫

```
Attributes[PlotCorrection] = {Protected, ReadProtected}
```

```
SyntaxInformation[PlotCorrection] = {ArgumentsPattern → {_}}
```

PlotData[data] plots the data.
PlotData[data, vox] plots the data and for 3D and 4D data assumes the voxelsize vox (z,x,y).
PlotData[data1, data2] plots data1 and data2.
PlotData[data1, data2, vox] plots data1
    and data2 and for 3D and 4D data assumes the voxelsize vox (z,x,y). ≫

```
Attributes[PlotData] = {Protected, ReadProtected}
```

```
Options[PlotData] = {PlotRange → Auto, ColorFunction → BlackToWhite}
```

```
SyntaxInformation[PlotData] = {ArgumentsPattern → {_, _., _., OptionsPattern[]}}
```

PlotData3D[data,vox] is a 3D dataviewer, data
is the 3D dataset and voxsize the size of the voxels in mm (z,x,y). ≫

```
Attributes[PlotData3D] = {Protected, ReadProtected}
```

```
SyntaxInformation[PlotData3D] = {ArgumentsPattern → {_, _.}}
```

PlotDefGrid[data, phasemap, shiftpar] plots the dataset on the background
with on top the non deformed and the deformed grid, or arrows or lines. ≫

```
Attributes[PlotDefGrid] = {Protected, ReadProtected}
```

PlotDuty[{grad, bval, ord}, mode] plot the gradient dutycycle ≫

```
Attributes[PlotDuty] = {Protected, ReadProtected}
```

```
SyntaxInformation[PlotDuty] = {ArgumentsPattern → {{_, _, _}, _.}}
```

PlotIVIM[vals, data, bvals] plots the results of the IVIM fits from IVIMCalc or BayesianIVIMFit2 or Baye. ≫

```
Attributes[PlotIVIM] = {Protected, ReadProtected}
```

```
Options[PlotIVIM] =
 {Method → , PlotColor → {RGBColor[1, 0, 0], RGBColor[0, 1, 0], RGBColor[0, 0, 1], GrayLevel[0]},
  NormalizeIVIM → Fit, PlotRange → Auto, ImageSize → 400}
```

```
SyntaxInformation[PlotIVIM] = {ArgumentsPattern → {_, _, _, OptionsPattern[]}}
```

PlotMoments[{G(t),..}, te, t] plots the moments generated by CalculateMoments ≫

```
Attributes[PlotMoments] = {Protected, ReadProtected}
```

```
SyntaxInformation[PlotMoments] = {ArgumentsPattern → {_, _, _}}
```

PlotSequence[seq,var] where seq is the output from GradSeq. ≫

```
Attributes[PlotSequence] = {Protected, ReadProtected}
```

```
SyntaxInformation[PlotSequence] = {ArgumentsPattern → {_, _}}
```

Options

ColorFunction is an option for graphics functions
that specifies a function to apply to determine colors of elements. ≫

```
Attributes[ColorFunction] = {Protected}
```

ContourStyle is an option for contour plots that
specifies the style in which contour lines or surfaces should be drawn. ≫

```
Attributes[ContourStyle] = {Protected}
```

DropSlices is an option for GetSlicePositions and specifies
how many slices from the beginning and and should be ignored. ≫

```
Attributes[DropSlices] = {Protected, ReadProtected}
```

ImageLegend is an option for MakeSliceImages, if set true a barlegend is added to the image. ≫

```
Attributes[ImageLegend] = {Protected, ReadProtected}
```

ImageSize is an option that specifies the overall size of an image to display for an object. ≫

```
Attributes[ImageSize] = {Protected}
```

MakeCheckPlot is an option for GetSlicePositions and if set true gives a plot of the slices locations. ≫

```
Attributes[MakeCheckPlot] = {Protected, ReadProtected}
```

Method is an option for various algorithm–intensive
functions that specifies what internal methods they should use. ≫

```
Attributes[Method] = {Protected}
```

NormalizeIVIM is an option for IVIMplot. If True the signal at b=0 is 1. ≫

```
Attributes[NormalizeIVIM] = {Protected, ReadProtected}
```

PeakNumber is an option of GetSlicePostitions
and specifies how many slices per direction need to be found. ≫

```
Attributes[PeakNumber] = {Protected, ReadProtected}
```

PlotColor is an option for GradientPlot can be any color or gradient color name. ≫

```
Attributes[PlotColor] = {Protected, ReadProtected}
```

PlotRange is an option for graphics functions
that specifies what range of coordinates to include in a plot. ≫

```
Attributes[PlotRange] = {Protected, ReadProtected}
```

PlotSpace is an option for GradientPlot can be "bspace" or "qspace". ≫

```
Attributes[PlotSpace] = {Protected, ReadProtected}
```

PositiveZ is an options for GradientPlot.
If True all Gradients are displayed with a positive z direction. ≫

```
Attributes[PositiveZ] = {Protected, ReadProtected}
```

SphereColor ListSpherePlor. Default value is
Automatic, If a color is given this color will be used for all spheres. ≫

```
Attributes[SphereColor] = {Protected, ReadProtected}
```

SphereSize is an option for GradientPlot and
ListSpherePlor. Sets the size of the spheres thar represent the gradients. ≫

```
Attributes[SphereSize] = {Protected, ReadProtected}
```

ProcessingTools

Functions

CorrectJoinSetMotion[[{dat1,dat2,...}, vox, over] motion correts multiple sets with overlap. Over is the
number of slices overlap between stes. A Translation registration is performed. ≫

```
Attributes[CorrectJoinSetMotion] = {Protected, ReadProtected}

Options[CorrectJoinSetMotion] = {JoinSetSplit → True, PaddOverlap → 2}

SyntaxInformation[CorrectJoinSetMotion] = {ArgumentsPattern → {_, _, _, OptionsPattern[]}}
```

DataTranformation[data,vox,w] transforms a 3D dataset accordint to the affine transformation vector w

```
Attributes[DataTransformation] = {Protected, ReadProtected}

Options[DataTransformation] = {InterpolationOrder → 1}

SyntaxInformation[DataTransformation] = {ArgumentsPattern → {_, _, _, OptionsPattern[]}}
```

DatTot[{data1, data2, ..}, name, vox] calculates the parameter table conating
the volume, mean, std and 95 CI for each of the diffusion parameters. ≫

```
Attributes[DatTot] = {Protected, ReadProtected}

SyntaxInformation[DatTot] = {ArgumentsPattern → {_, _, _}}
```

DatTotXLS[{data1, data2, ..}, name, vox] is the same as
DatTot, but gives the parameters as strings for easy export to excel. ≫

```
Attributes[DatTotXLS] = {Protected, ReadProtected}

SyntaxInformation[DatTotXLS] = {ArgumentsPattern → {_, _, _}}
```

ErrorPlot[data, xdata] plots a errorplot of the data where
the first dim of the data is the xrange which matches the xdata list.
ErrorPlot[data, xdata, range] similar with a given y range. ≫

```
Attributes[ErrorPlot] = {Protected, ReadProtected}

Options[ErrorPlot] =
  {ColorValue → {GrayLevel[0], RGBColor[1, 0, 0]}, PlotLabel → , AxesLabel → , ImageSize → 300}

SyntaxInformation[ErrorPlot] = {ArgumentsPattern → {_, _, _., OptionsPattern[]}}
```

FiberDensityMap[fiberPoins, dim, vox] generates a fiber density map for
the fiberPoins which are imported by LoadFiberTracts. The dimensions dim
should be the dimensions of the tracked datasets van vox its volxel size. ≫

```
Attributes[FiberDensityMap] = {Protected, ReadProtected}
```

```
Options[FiberDensityMap] = {SeedDensity → Automatic}
```

```
SyntaxInformation[FiberDensityMap] = {ArgumentsPattern → {_, _, _, OptionsPattern[]}}
```

FiberLengths[fpoints,flines] calculates the fiber lenght using the output from LoadFiberTacts.
FiberLengths[{fpoints,flines}] calculates the fiber lenght using the output from LoadFiberTacts. ≫

```
Attributes[FiberLengths] = {Protected, ReadProtected}
```

```
SyntaxInformation[FiberLengths] = {ArgumentsPattern → {_, _.}}
```

FindOutliers[data] finds the outliers of a list of data. ≫

```
Attributes[FindOutliers] = {Protected, ReadProtected}
```

```
Options[FindOutliers] = {OutlierMethod → IQR, OutlierOutput → Mask,
  OutlierIterations → 1, OutlierRange → 2, OutlierIncludeZero → True}
```

```
SyntaxInformation[FindOutliers] = {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

FitData[data,range] converts the data into 100 bins within
the +/– range around the mean. Function is used in ParameterFit. ≫

```
Attributes[FitData] = {Protected, ReadProtected}
```

```
SyntaxInformation[FitData] = {ArgumentsPattern → {_, _.}}
```

GetMaskMeans[dat, mask, name] calculates the mean,
std, 5,50 and 95% CI form the given data for each of the given masks.
Mask can be genereated by SplitSegmentations. name is a string that is added to the header. ≫

```
Attributes[GetMaskMeans] = {Protected, ReadProtected}
```

```
Options[GetMaskMeans] = {MeanMethod → SkewNormalDist}
```

```
SyntaxInformation[GetMaskMeans] = {ArgumentsPattern → {_, _, _., OptionsPattern[]}}
```

Hist[data, range] plots a probability density histogram of the data from
xmin to xmax with a fitted (skew)normal distribution. Uses ParameterFit.
Hist[data, range, label] plots a probability density histogram of the data from xmin
to xmax with a fitted (skew)normal distribution and label as x–axis label.
Hist[{data1..,data2,..}, {range1,range2,..}] plots a probability density histogram of the data
from xmin to xmax with a fitted (skew)normal distribution. Uses ParameterFit.
Hist[{data1,data2,..}, {range1,range2,..}, {label1,label2,..}] plots a probability density histogram of the
data from xmin to xmax with a fitted (skew)normal distribution and label as x–axis label. ≫

```
Attributes[Hist] = {Protected, ReadProtected}

Options[Hist] = {ColorValue →
    {{GrayLevel[0], GrayLevel[1]}, RGBColor[1, 0, 0], RGBColor[0, 1, 0], RGBColor[0, 0, 1]},
  Method → SkewNormal, PlotLabel → , AxesLabel → , ImageSize → 300}

SyntaxInformation[Hist] = {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

Hist2[pars, range] plots a probability density histogram of the data
over range with two fitted (skew)normal distribution. Uses ParameterFit2.
Hist2[pars, range, label] plots a probability density histogram of the data over
range with two fitted (skew)normal distribution. Uses ParameterFit2. ≫

```
Attributes[Hist2] = {Protected, ReadProtected}

Options[Hist2] = {Scaling → False}

SyntaxInformation[Hist2] = {ArgumentsPattern → {_, _, _, OptionsPattern[]}}
```

InvertDataset[data] inverts the data along the x y and z axes. In other words
it is rotated aroud the origin such that (x,y,z)=(−x,−y,−z) and (0,0,0)=(0,0,0) ≫

```
Attributes[InvertDataset] = {Protected, ReadProtected}
```

JoinSets[{dat1,dat2,...}, over] joins dat1, dat2, ... with over slices overlap.
JoinSets[{dat1,dat2,dat3...},{over1,over2,...}] joins dat1 and dat2 with
over1 slices overlap, Joins dat2 and dat3 with over2 slices overlap and so on.
JoinSets[{dat1,dat2,...},{{over,drop1,drop2},...}] joins dat1, dat2 with over slices
overlap and drops drop1 slices for dat1 and drop2 from drop 2. ≫

```
Attributes[JoinSets] = {Protected, ReadProtected}

Options[JoinSets] = {ReverseSets → True, ReverseData → True,
  NormalizeSets → True, MotionCorrectSets → False, PaddOverlap → 2, JoinSetSplit → True}

SyntaxInformation[JoinSets] = {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

MeanRange[Range] calculates the medain (50%) and
standard deviation (14% and 86%) range and reports it as a string. ≫

```
Attributes[MeanRange] = {Protected, ReadProtected}
```

MeanStd[data] calculates the mean and standard deviation and reports it as a string. ≫

```
Attributes[MeanStd] = {Protected, ReadProtected}
```

MedCouple[data] calculates the medcouple of a list of data. ≫

```
Attributes[MedCouple] = {Protected, ReadProtected}
```

NumberTableForm[data] makes a right aligned table of the numbers with 3 decimal percision.
NumberTableForm[data, n] makes a right aligned table of the numbers with n decimal percision. ≫

```
Attributes[NumberTableForm] = {Protected, ReadProtected}
```

```
Options[NumberTableForm] = {TableMethod → NumberForm, TableAlignments → Automatic,
  TableDepth → ∞, TableDirections → Column, TableHeadings → None, TableSpacing → Automatic}
```

```
SyntaxInformation[NumberTableForm] = {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

ParameterFit[data] fits a (skew)Normal probability density function to the data.
ParameterFit[{data1, data2,...}] fits a (skew)Normal
probability density function to each of the datasets. Is used in Hist. ≫

```
Attributes[ParameterFit] = {Protected, ReadProtected}
```

```
Options[ParameterFit] = {FitFunction → SkewNormal, FitOutput → Parameters, Method → Automatic}
```

```
SyntaxInformation[ParameterFit] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

ParameterFit2[data] fits two skewNormal probaility density fucntions
to the data. Assuming two compartments, one for fat and one for muscle. ≫

```
Attributes[ParameterFit2] = {Protected, ReadProtected}
```

```
Options[ParameterFit2] = {FitOutput → BestFitParameters}
```

```
SyntaxInformation[ParameterFit2] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

SetupDataStructure[dcmFolder] makes nii folders and generates nii
files for a directory of dmc data where the data is structured per subject. ≫

```
Attributes[SetupDataStructure] = {Protected, ReadProtected}
```

SmartMask[input] crates a smart mask of input, which is either
the tensor or the tensor parameters calculated using ParameterCalc.
SmartMask[input, mask] crates a smart mask of input and used
the mask as a prior selection of the input. ≫

```
Attributes[SmartMask] = {Protected, ReadProtected}
```

```
Options[SmartMask] =
  {Strictness → 0.5, MaskCompartment → Muscle, SmartMethod → Continuous, SmartMaskOutput → mask}
```

```
SyntaxInformation[SmartMask] = {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

SNRCalc[data,masksig,masknoise] calculates the Signal to noise ratio
of the signal selected by masksig and the noise selected by masknoise. ≫

```
Attributes[SNRCalc] = {Protected, ReadProtected}
```

```
SyntaxInformation[SNRCalc] = {ArgumentsPattern → {_, _, _}}
```

SNRMapCalc[data1,noisemap] calcualtes the signal to
noise ratio of the data using MN[data]/(1/sqrt[pi/2] sigma), where sigma
is the local mean of the noise map assuming it is a rician distribution.

SNRMapCalc[{data1,data2}] calcualtes the signal to noise ratio from two
identical images using MN[data1,data2] / (.5 SQRT[2] STDV[data2−data1]).

SNRMapCalc[{data1, .. dataN}] calcualtes the signal to noise ratio of the data
using MN/sigma where the mean signal MN is the average voxe value over all
dynamics N and the sigma is the standard deviation over all dynamics N. ≫

```
Attributes[SNRMapCalc] = {Protected, ReadProtected}
```

```
Options[SNRMapCalc] = {OutputSNR → SNR, SmoothSNR → 2}
```

```
SyntaxInformation[SNRMapCalc] = {ArgumentsPattern → {_, _., _., OptionsPattern[]}}
```

SplitSets[data, Nsets, Nover] splits the data in Nsets with Nover slices overlap. ≫

```
Attributes[SplitSets] = {Protected, ReadProtected}
```

```
Options[SplitSets] = {ReverseSets → False, ReverseData → True, PaddOverlap → 0}
```

```
SyntaxInformation[SplitSets] = {ArgumentsPattern → {_, _, _, OptionsPattern[]}}
```

Options

AxesLabel is an option for graphics functions that specifies labels for axes. ≫

```
Attributes[AxesLabel] = {Protected}
```

ColorValue is an option for Hist and ErrorPlot. Default {Black, Red}. ≫

```
Attributes[ColorValue] = {Protected, ReadProtected}
```

FitFunction is an option for ParameterFit. Options are
"Normal" or "SkewNormal". Indicates which function wil be fitted. ≫

```
Attributes[FitFunction] = {Protected, ReadProtected}
```

FitOutput is an option for ParameterFit and ParameterFit2.
Option can be "Parameters", "Function" or "BestFitParameters". ≫

```
Attributes[FitOutput] = {Protected, ReadProtected}
```

ImageSize is an option that specifies the overall size of an image to display for an object. ≫

```
Attributes[ImageSize] = {Protected}
```

InterpolationOrder is an option for Interpolation, as well as ListLinePlot, ListPlot3D, ListContourPlot,
and related functions, that specifies what order of interpolation to use. ≫

```
Attributes[InterpolationOrder] = {Protected}
```

JoinSetSplit is an option ofr CorrectJoinSetMotion. If True
RegisterDataTransformSplit is used else RegisterDataTransform is used. ≫

```
Attributes[JoinSetSplit] = {Protected, ReadProtected}
```

MaskCompartment is an option for SmartMask. Can be "Muscle" or "Fat". ≫

```
Attributes[MaskCompartment] = {Protected, ReadProtected}
```

MeanMethod is an option for GetMaskMeans.
The option can be "NormalDist", "SkewNormalDist", or "Mean". ≫

```
Attributes[MeanMethod] = {Protected, ReadProtected}
```

Method is an option for various algorithm–intensive
functions that specifies what internal methods they should use. ≫

```
Attributes[Method] = {Protected}
```

MotionCorrectSets is an option for JoinSets. True motion
corrects the individual stacs before joining using CorrectJoinSetMotion. ≫

```
Attributes[MotionCorrectSets] = {Protected, ReadProtected}
```

NormalizeSets is an option for JoinSets. True normalizes the individual stacs before joining. ≫

```
Attributes[NormalizeSets] = {Protected, ReadProtected}
```

OutlierIncludeZero is an option for FindOutliers. If set
to True all values that are zero are ignored and considered outliers. ≫

```
Attributes[OutlierIncludeZero] = {Protected, ReadProtected}
```

OutlierIterations is an option for FindOutliers.
Specifies how many iterations are used to find the outliers.
Each itteration the outliers are reevaluated on the data with the
previously found outliers alread rejected. ≫

```
Attributes[OutlierIterations] = {Protected, ReadProtected}
```

OutlierMethod is an option for FindOutliers. values can be "IQR", "SIQR" or "aIQR". "IRQ" is used for
normly distributed data, "SIQR" or "aIQR" are better for skewed distributions. ≫

```
Attributes[OutlierMethod] = {Protected, ReadProtected}
```

OutlierOutput is an option for FindOutliers. If value is "Mask" it gives
a list of 1 for data and 0 for outliers. Else the output is {data, outliers}. ≫

```
Attributes[OutlierOutput] = {Protected, ReadProtected}
```

OutlierRange is an option for FindOutliers. Specifies how many times the IQR is considred an oulier. ≫

```
Attributes[OutlierRange] = {Protected, ReadProtected}
```

OutputSNR is an option for SNRMapCalc. ≫

```
Attributes[OutputSNR] = {Protected, ReadProtected}
```

PaddOverlap is an option of CorrectJoinSetMotion
and JoinSets. it allows for extra motion in the z direction. ≫

```
Attributes[PaddOverlap] = {Protected, ReadProtected}
```

PlotLabel is an option for graphics functions that specifies an overall label for a plot. ≫

```
Attributes[PlotLabel] = {Protected}
```

ReverseData is an option for JoinSets. Reverses each
individual datset given as input for the JoinSets function. True by default. ≫

```
Attributes[ReverseData] = {Protected, ReadProtected}
```

ReverseSets is an option for JoinSets. Reverses the order of the datsets, False by default. ≫

```
Attributes[ReverseSets] = {Protected, ReadProtected}
```

Scaling is an option for Hist2. Scales the individual fits of the fat and muscle compartment. ≫

```
Attributes[Scaling] = {Protected, ReadProtected}
```

SeedDensity is an option for FiberDensityMap. The seedpoint spacing in mm. ≫

```
Attributes[SeedDensity] = {Protected, ReadProtected}
```

SmartMaskOutput is an option for Smartmask. Can be set to "mask"
to output only the mask or "full" to also output the probability mask. ≫

```
Attributes[SmartMaskOutput] = {Protected, ReadProtected}
```

SmartMethod is an option for SmartMask. This specifies
how the mask is generated. Can be "Continuous" or "Catagorical" ≫

```
Attributes[SmartMethod] = {Protected, ReadProtected}
```

SmoothSNR is an option for SNRMapCalc. ≫

```
Attributes[SmoothSNR] = {Protected, ReadProtected}
```

Strictness is an option for SmartMask value between 0 and 1. Higer values removes more data. ≫

```
Attributes[Strictness] = {Protected, ReadProtected}
```

TableAlignments is an option for TableForm and MatrixForm
which specifies how entries in each dimension should be aligned. ≫

```
Attributes[TableAlignments] = {Protected}
```

TableDepth is an option for TableForm and MatrixForm which specifies
the maximum number of levels to be printed in tabular or matrix format. ≫

```
Attributes[TableDepth] = {Protected}
```

TableDirections is an option for TableForm and MatrixForm which specifies
whether successive dimensions should be arranged as rows or columns. ≫

```
Attributes[TableDirections] = {Protected}
```

TableHeadings is an option for TableForm and MatrixForm which gives
the labels to be printed for entries in each dimension of a table or matrix. ≫

```
Attributes[TableHeadings] = {Protected}
```

TableMethod is an option for NumberTableForm. It specifies which number form
to uses. Values can be NumberForm, ScientificForm or EngineeringForm ≫

```
Attributes[TableMethod] = {Protected, ReadProtected}
```

TableSpacing is an option for TableForm and MatrixForm which specifies
how many spaces should be left between each successive row or column. ≫

```
Attributes[TableSpacing] = {Protected}
```

RelaxometryTools

Functions

CalibrateEPGT2Fit[datan, times, angle]
calculates the Fat T2 ralaxation that will be used in the EPGT2fit.

Outputs the fat T2 value. ≫

```
Attributes[CalibrateEPGT2Fit] = {Protected, ReadProtected}
```

```
Options[CalibrateEPGT2Fit] =
  {EPGRelaxPars → {{0, 100}, {20, 300}, {1400., 365.}}, EPGFitPoints → 50}
```

```
SyntaxInformation[CalibrateEPGT2Fit] = {ArgumentsPattern → {_, _, _, OptionsPattern[]}}
```

CreateT2Dictionary[{T1m, T1f}, {Necho,
echoSpace}, angle] Creates a EPG signal dictionary used for EPGT2fit.
Every dictionary that is defined is cached.

Output is {dictionary, vals} ≫

```
Attributes[CreateT2Dictionary] = {Protected, ReadProtected}
```

```
Options[CreateT2Dictionary] =
 {DictB1Range → {0.5, 1.4, 0.01}, DictT2Range → {10., 70., 0.2}, DictT2fRange → {100., 200., 2.}}
```

```
SyntaxInformation[CreateT2Dictionary] = {ArgumentsPattern → {_, _, _, OptionsPattern[]}}
```

DictionaryMinSearch[dictionary, y] performs dictionary
minimization of data y. dictionary is generated with CreateT2Dictionary.

Output is {{T2, B1}, fwfraction, residualError}. ≫

```
Attributes[DictionaryMinSearch] = {Protected, ReadProtected}
```

```
SyntaxInformation[DictionaryMinSearch] = {ArgumentsPattern → {_, _, _.}}
```

EPGSignal[{Necho, echoSpace}, {T1, T2},
{ex_angle,ref_angle}, B1] generates a EPG T2 curve with stimulated echos.
T1, T2 and echoSpace are in ms, angel is in degree, B1 is between 0 and 1.

Output is the EPG Signal vector. ≫

```
Attributes[EPGSignal] = {Protected, ReadProtected}
```

```
SyntaxInformation[EPGSignal] = {ArgumentsPattern → {_, _, _, _, _.}}
```

EPGT2Fit[data, {Necho, detlaTE}, {exitation, refoucs}] fits the
T2 based on Marty B et.al. Simultaneous muscle water T2 and fat fraction
mapping using transverse relaxometry with stimulated echo compensation.
Exitation and refocus are the RF pulse angles e.g. 90,180. They can also be a
range of angeles over the slice profile as defined by GetSliceProfile.

Output is {{{T2map,B1Map},{wat, fat, fatMap},
residual},callibration} or {{T2map,B1Map},{wat, fat, fatMap}, residual} ≫

```
Attributes[EPGT2Fit] = {Protected, ReadProtected}
```

```
Options[EPGT2Fit] =
 {DictB1Range → {0.5, 1.4, 0.01}, DictT2Range → {10., 60., 0.2}, DictT2fRange → {120., 170., 1.},
  EPGRelaxPars → {1400., 365.}, EPGCalibrate → False, EPGFitPoints → 50,
  EPGMethod → dictionaryM, MonitorEPGFit → True, OutputCalibration → False, EPGSmoothB1 → True}
```

```
SyntaxInformation[EPGT2Fit] = {ArgumentsPattern → {_, _, _, OptionsPattern[]}}
```

NonLinearEPGFit[{vals, T2cons}, y] performs dictionary minimization
of data y. vals = {{T1muscle, T1fat, T2fat}, {Necho, echoSpace, angle}}.

Output is {{T2, B1}, fwfraction, residualError}. ≫

```
Attributes[NonLinearEPGFit] = {Protected, ReadProtected}
```

```
SyntaxInformation[NonLinearEPGFit] = {ArgumentsPattern → {_, _}}
```

T1Fit[data, TR] fits the T1 value to the data using a nonlinear method.

Output is {t1, apar, bpar} ≫

```
Attributes[T1Fit] = {Protected, ReadProtected}
```

T1rhoFit[data, EchoTimes] fits the T1rho value to the data using linear or nonlinear methdos.

Output is {S(0), T1rhomap}. ≫

```
Attributes[T1rhoFit] = {Protected, ReadProtected}
```

```
Options[T1rhoFit] = {Method → Linear}
```

```
SyntaxInformation[T1rhoFit] = {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

T2Fit[data, EchoTimes] fits the T2 value to the data using linear or nonlinear methods.

Output is {S(0), T2}. ≫

```
Attributes[T2Fit] = {Protected, ReadProtected}
```

```
Options[T2Fit] = {Method → Linear}
```

```
SyntaxInformation[T2Fit] = {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

TriExponentialT2Fit[data, EchoTimes] fits the T2 based
on Azzabou N et.al. Validation of a generic approach to muscle water T2
determination at 3T in fat–infiltrated skeletal muscle. J. Magn. Reson. 2015.
The fat T2 parameters are automatically estimated from the high signal voxels from the last echo.

Output is {{S(0), fatFraction, muscleFraction,
T2map},callibration} or {S(0), fatFraction, muscleFranction, T2map}. ≫

```
Attributes[TriExponentialT2Fit] = {Protected, ReadProtected}
```

```
Options[TriExponentialT2Fit] = {OutputCalibration → False}
```

```
SyntaxInformation[TriExponentialT2Fit] = {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

Options

DictB1Range is an option for CreateT2Dictionary and EPGT2Fit. It specifies
the range and step of the B1 values in the dictionary {min, max, step}. ≫

```
Attributes[DictB1Range] = {Protected, ReadProtected}
```

DictT2fRange is an option for CreateT2Dictionary and EPGT2Fit. is specifies the
range and step of the T2 fat values in the dictionary {min, max, step} in ms.
If a single value is given this fixed value is used a long as EPGCalibrate is False. ≫

```
Attributes[DictT2fRange] = {Protected, ReadProtected}
```

DictT2Range is an option for CreateT2Dictionary and EPGT2Fit. is specifies
the range and step of the T2 values in the dictionary {min, max, step} in ms. ≫

```
Attributes[DictT2Range] = {Protected, ReadProtected}
```

EPGCalibrate is an option for EPGT2Fit. If set
to True it does autmatic callibration of the T2 fat relaxation time. ≫

```
Attributes[EPGCalibrate] = {Protected, ReadProtected}
```

EPGFitPoints is a option for CalibrateEPGT2Fit and EPGT2Fit. Number of points is 200 by default. ≫

```
Attributes[EPGFitPoints] = {Protected, ReadProtected}
```

EPGMethod is an optionf for EPGT2Fit. Values can be "NLLS", "dictionary" or "dictionaryM". ≫

```
Attributes[EPGMethod] = {Protected, ReadProtected}
```

EPGRelaxPars is and option for EPGT2Fit. Needs
to be {T1muscl, T1Fat, T2Fat} in ms, defaul is {1400,365,137}. ≫

```
Attributes[EPGRelaxPars] = {Protected, ReadProtected}
```

EPGSmoothB1 is an options for EPGT2Fit. If set to True the B1 map of the fit will be
smoothed after which the minimization if perfomed again but with a fixed B1. ≫

```
Attributes[EPGSmoothB1] = {Protected, ReadProtected}
```

Method is an option for various algorithm–intensive
functions that specifies what internal methods they should use. ≫

```
Attributes[Method] = {Protected}
```

MonitorEPGFit show waitbar during EPGT2Fit. ≫

```
Attributes[MonitorEPGFit] = {Protected, ReadProtected}
```

OutputCalibration is an option for EPGT2Fit
and TriExponentialT2Fit. If true it outputs the calibartion values. ≫

```
Attributes[OutputCalibration] = {Protected, ReadProtected}
```

SimulationTools

Functions

AddNoise[data, noise] ads rician noise to the data with a given sigma or SNR value. ≫

```
Attributes[AddNoise] = {Protected, ReadProtected}
```

```
Options[AddNoise] = {NoiseSize → Sigma}
```

```
SyntaxInformation[AddNoise] = {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

BlochSeries[vectorIn, deltat, freqRange, B1] performs a Bloch simulation of an RF pulse. ≫

```
Attributes[BlochSeries] = {Protected, ReadProtected}
```

CalculateGfactor[factors, sensitivity, Wmat] calculates a gfactor for given sensitivity maps and noise corraltion W. given the sense factors which is a list of three integers. ≫

```
Attributes[CalculateGfactor] = {Protected, ReadProtected}
```

```
Options[CalculateGfactor] = {GRegularization → 0.}
```

```
SyntaxInformation[CalculateGfactor] = {ArgumentsPattern → {_, _, _, _., OptionsPattern[]}}
```

CreateDiffData[sig, eig, bvec, gradients, dim] creates a DTI datasets of dimensions dim with sig as unweighted signal S0 and bvec and gradients. eig can be {l1, l2, l3}, {{l1, l2, l3}, {e1, e2, e3}}, {{l1, l2, l3}, "Random"}, {{l1, l2, l3}, "RandomZ"} or {{l1, l2, l3}, "OrtRandom"}.
Uses Tensor internally. ≫

```
Attributes[CreateDiffData] = {Protected, ReadProtected}
```

```
SyntaxInformation[CreateDiffData] = {ArgumentsPattern → {_, _, _, _, _.}}
```

GetPulseProfile[excitation, refocus] gives the pusl angle profiles for the exitation and refocussing pulses.
a pulse is defined as {"name", flipangle, {G_strnth, Dur, BW}}.
GetPulseProfile[{"name", flipangle, {G_strnth, Dur, BW}}] gives detaile slice profile information of one pulse.

output is {ex_angle_profiel, ref_angel_profile, {plots}}.
output for single pulse is {{distance, Mt, Mz, Mx, My, ang, phase}, plots} ≫

```
Attributes[GetPulseProfile] = {Protected, ReadProtected}
```

```
Options[GetPulseProfile] =
  {MagnetizationVector → {0, 0, 1}, SliceRange → 12, SliceRangeSamples → 25}
```

```
SyntaxInformation[GetPulseProfile] = {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

GfactorSimulation[sensitivity, Wmat, {dir,sense}] calculates the
gfactormaps for given sensitivity maps and noise corraltion W in one direction.
The sensefactors are a list of integers in a given direction: "LR", "FH", or "AP".
GfactorSimulation[sensitivity, Wmat, {dir1,sense1}, {dir2,sense2}] calculates the
gfactormaps for given sensitivity maps and noise corraltion W in two directions. ≫

```
Attributes[GfactorSimulation] = {Protected, ReadProtected}

Options[GfactorSimulation] = {GRegularization → 0., GOutput → Grid}

SyntaxInformation[GfactorSimulation] = {ArgumentsPattern → {_, _, _, _., _., OptionsPattern[]}}
```

PlotSimulation[pars, xval, true, label, color] plots the pars
(output form Parameters). Using label as PlotLabel and xval as x axis Thics.tr
are the true parameter values. color are the color used for the plot. ≫

```
Attributes[PlotSimulation] = {Protected, ReadProtected}

Options[PlotSimulation] = {PlotRange → {{0, 3}, {0, 3}, {0, 3}, {0, 3}, {0, 1}}}

SyntaxInformation[PlotSimulation] = {ArgumentsPattern → {_, _, _, _, _, OptionsPattern[]}}
```

PlotSimulationAngle[par, xdata, label, col] plots pars (output from Anlge Parameters). ≫

```
Attributes[PlotSimulationAngle] = {Protected, ReadProtected}

Options[PlotSimulationAngle] = {PlotRange → {0, 90}}
```

PlotSimulationAngleHist[pars, label, xdata] plots pars (output from Anlge Parameters). ≫

```
Attributes[PlotSimulationAngleHist] = {Protected, ReadProtected}

SyntaxInformation[PlotSimulationAngleHist] =
 {ArgumentsPattern → {_, _, _, _, _., OptionsPattern[]}}
```

PlotSimulationHist[pars, label, xdata, tr] plots the pars (output form Parameters). Using
label as plotlabel and xdata as x axis label. tr are the true parameter values. ≫

```
Attributes[PlotSimulationHist] = {Protected, ReadProtected}

SyntaxInformation[PlotSimulationHist] = {ArgumentsPattern → {_, _, _, _}}
```

PlotSimulationVec[tens, xdata, label] plots the eigenvectors from simulated tensors. ≫

```
Attributes[PlotSimulationVec] = {Protected, ReadProtected}

Options[PlotSimulationVec] = {SortVecs → True}

SyntaxInformation[PlotSimulationVec] = {ArgumentsPattern → {_, _, _, OptionsPattern[]}}
```

Pulses[name] gives the pulse shape of some predefinec Philips pulse shapes. ≫

```
Attributes[Pulses] = {Protected, ReadProtected}
```

Signal[par,TR,TE] calculates the MRI signal at a given TR and TE. Par is defineds as {pd, T1, T2}. ≫

```
Attributes[Signal] = {Protected, ReadProtected}
```

```
SyntaxInformation[Signal] = {ArgumentsPattern → {_, _, _}}
```

SimAngleParameters[tens,vec] caculates the diffusion eigenvectors for tens compared to the true values vec. The output can be used in PlotSimulationAngleHist and PlotSimulationAngle. ≫

```
Attributes[SimAngleParameters] = {Protected, ReadProtected}
```

```
SyntaxInformation[SimAngleParameters] = {ArgumentsPattern → {_, _}}
```

SimParameters[tens] caculates the diffusion parameters for tens. The output can be used in PlotSimulationHist and PlotSimulation. ≫

```
Attributes[SimParameters] = {Protected, ReadProtected}
```

```
Options[SimParameters] = {Reject → False}
```

```
SyntaxInformation[SimParameters] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

SimulateSliceEPG[exitation, refocus, {{T1, T2}, {Necho, echoSp}, b1}] gives a simulated slice profile and EPG singnal plot.
exitation and refocus are generated by GetPulseProfiel. ≫

```
Attributes[SimulateSliceEPG] = {Protected, ReadProtected}
```

```
Options[SimulateSliceEPG] = {ReportFits → False}
```

```
SyntaxInformation[SimulateSliceEPG] = {ArgumentsPattern → {_, _, _, OptionsPattern[]}}
```

Tensor[{l1, l2, l3}] creates a diffuison tensor with vectors {{0,0,1},{0,1,0},{1,0,0}} and eigenvalues {l1, l2, l3}.
Tensor[{l1, l2, l3}, {e1, e2, e3}] creates a diffuison tensor with vectors {e1, e2, e3} and eigenvalues {l1, l2, l3}.
Tensor[{l1, l2, l3}, "Random"] creates a diffuison tensor with random orthogonal eigenvectors {e1, e2, e2} and eigenvalues {l1, l2, l3}.
Tensor[{l1, l2, l3}, "RandomZ"] creates a diffuison tensor with random orthogonal eigenvectors {{1,0,0}, e2, e3} with random eigenvectors and eigenvalues {l1, l2, l3}.
Tensor[{l1, l2, l3}, "OrtRandom"] creates a diffuison tensor with random orthogonal eigenvectors {{1,0,0},{0,1,0},{0,0,1}} and eigenvalues {l1, l2, l3}. ≫

```
Attributes[Tensor] = {Protected, ReadProtected}
```

```
Options[Tensor] = {TensOutput → Vector}
```

```
SyntaxInformation[Tensor] = {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

Options

GOutput is an option for GfactorSimulation. can be "Grid" or "List". ≫

```
Attributes[GOutput] = {Protected, ReadProtected}
```

GRegularization is an option for CalculateGfactor and GfactorSimulation. ≫

```
Attributes[GRegularization] = {Protected, ReadProtected}
```

MagnetizationVector is an option for GetPulseProfile.
It defines the start magnetization vector for the bloch simulation. ≫

```
Attributes[MagnetizationVector] = {Protected, ReadProtected}
```

NoiseSize is an option for AddNoise. Values can be
"Sigma", then the noise sigma is given or "SNR", then the SNR is given. ≫

```
Attributes[NoiseSize] = {Protected, ReadProtected}
```

PlotRange is an option for graphics functions
that specifies what range of coordinates to include in a plot. ≫

```
Attributes[PlotRange] = {Protected, ReadProtected}
```

Reject is an option for EigenvalCalc. It True
then voxels with negative eigenvalues are rejected and set to 0. ≫

```
Attributes[Reject] = {Protected, ReadProtected}
```

ReportFits is an option for SimulateSliceEPG. If True it also reports the fit values ≫

```
Attributes[ReportFits] = {Protected, ReadProtected}
```

SliceRange is an option for GetPulseProfile. It specifies over which range
the slice profile is generated (in mm). the total profile is 2xSliceRange. ≫

```
Attributes[SliceRange] = {Protected, ReadProtected}
```

SliceRangeSamples is an option for GetPulseProfile.
defines how many samples are used to generate half a puls profile. ≫

```
Attributes[SliceRangeSamples] = {Protected, ReadProtected}
```

SortVecs is an option for PlotSimulationVec. ≫

```
Attributes[SortVecs] = {Protected, ReadProtected}
```

TensOutput is an option for Tensor. Values can be "Vector" or "Matrix". ≫

```
Attributes[TensOutput] = {Protected, ReadProtected}
```

TensorTools

Functions

ADCCalc[eigenvalues] caculates the ADC from the given eigenvalues. ≫

```
Attributes[ADCCalc] = {Protected, ReadProtected}
```

```
SyntaxInformation[ADCCalc] = {ArgumentsPattern → {_}}
```

AngleCalc[data, vector] calculates the angel between the
vector and the data. Data shoud be an array of dimensions {xxx,3}. ≫

```
Attributes[AngleCalc] = {Protected, ReadProtected}
```

```
Options[AngleCalc] = {Distribution → 0-180}
```

```
SyntaxInformation[AngleCalc] = {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

AngleMap[data] calculates the zennith and azimuth angles of
a 3D dataset (z,x,y,3) containing vectors relative to the slice direction. ≫

```
Attributes[AngleMap] = {Protected, ReadProtected}
```

```
SyntaxInformation[AngleMap] = {ArgumentsPattern → {_}}
```

ColorFAPlot[tenor] create a color coded FA map from the tensor for l1, l2 and l3. ≫

```
Attributes[ColorFAPlot] = {Protected, ReadProtected}
```

```
SyntaxInformation[ColorFAPlot] = {ArgumentsPattern → {_}}
```

ConcatenateDiffusionData[{{data1, .., dataN}, {grad1, .., gradN},
{bval, .., bvalN}, {vox, .., voxN}}] concatenates the diffusion data sets.
ConcatenateDiffusionData[{data1, .., dataN}, {grad1, .., gradN}, {bval, ..,
bvalN}, {vox, .., voxN}] concatenates the diffusion data sets. ≫

```
Attributes[ConcatenateDiffusionData] = {Protected, ReadProtected}
```

```
SyntaxInformation[ConcatenateDiffusionData] = {ArgumentsPattern → {_, _., _., _.}}
```

Correct[data, phase, shiftpar] corrects the dataset
data using the phasemap and the shiftpar and interpolation order 1.
Correct[data, phase, shiftpar, int] corrects the dataset data using the
phasemap and the shiftpar and interpolation order int. ≫

```
Attributes[Correct] = {Protected, ReadProtected}
```

```
SyntaxInformation[Correct] = {ArgumentsPattern → {_, _, _, _.}}
```

Deriv[disp, vox] calculates the derivative of the displacement along
the three main axes. disp is the displacement field, vox is the voxel size.
Deriv[disp, vox, mask] calculates the derivative of the displacement along the three
main axes. Sharp edges between the background en disp are solved by
the mask. mask is a mask delining the edge of the displacement field. ≫

```
Attributes[Deriv] = {Protected, ReadProtected}
```

```
SyntaxInformation[Deriv] = {ArgumentsPattern → {_, _, _}}
```

DriftCorrect[data, bval] dirft corrects the data using
the signals of the lowest bvalue that has 6 or more unique volumes.
For the function to work optimal it is best to have these volumes evenly spread
througout thet data and for the first and last volume to have this low bvalue. ≫

```
Attributes[DriftCorrect] = {Protected, ReadProtected}
```

```
Options[DriftCorrect] = {NormalizeSignal → True, UseMask → True}
```

```
SyntaxInformation[DriftCorrect] = {ArgumentsPattern → {_, _, _., OptionsPattern[]}}
```

ECalc[eigenvalues] caculates the e from the given eigenvalues. ≫

```
Attributes[ECalc] = {Protected, ReadProtected}
```

```
Options[ECalc] = {MonitorCalc → True}
```

```
SyntaxInformation[ECalc] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

EigensysCalc[tensor] caculates the eigensystem for the given tensor. ≫

```
Attributes[EigensysCalc] = {Protected, ReadProtected}
```

```
Options[EigensysCalc] = {MonitorCalc → False}
```

```
SyntaxInformation[EigensysCalc] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

EigenvalCalc[tensor] caculates the eigenvalues for the given tensor. ≫

```
Attributes[EigenvalCalc] = {Protected, ReadProtected}
```

```
Options[EigenvalCalc] = {MonitorCalc → True, RejectMap → False, Reject → True}
```

```
SyntaxInformation[EigenvalCalc] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

EigenvecCalc[tensor] caculates the eigenvectors for the given tensor. ≫

```
Attributes[EigenvecCalc] = {Protected, ReadProtected}
```

```
Options[EigenvecCalc] = {MonitorCalc → False}
```

```
SyntaxInformation[EigenvecCalc] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

FACalc[eigenvalues] caculates the FA from the given eigenvalues. ≫

```
Attributes[FACalc] = {Protected, ReadProtected}
```

```
SyntaxInformation[FACalc] = {ArgumentsPattern → {_}}
```

ParameterCalc[tensor] caculates the eigenvalues and MD and FA from the given tensor. ≫

```
Attributes[ParameterCalc] = {Protected, ReadProtected}
```

```
Options[ParameterCalc] = {Reject → True, MonitorCalc → False}
```

```
SyntaxInformation[ParameterCalc] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

RemoveIsoImages[data, grad, bval] Romoves the ISO images from
the philips scanner from the data. ISO images have g={0,0,0} and b>0. ≫

```
Attributes[RemoveIsoImages] = {Protected, ReadProtected}
```

```
SyntaxInformation[RemoveIsoImages] = {ArgumentsPattern → {_, _, _}}
```

ResidualCalc[DTI,{tensor,S0},gradients,bvector] calculates the tensor residuals for the given dataset.
ResidualCalc[DTI,{tensor,S0},outlier,gradients,bvector] calculates
the tensor residuals for the given dataset taking in account the outliers.
ResidualCalc[DTI,{tensor,S0},bmat] calculates the tensor residuals for the given dataset.
ResidualCalc[DTI,{tensor,S0},outlier,bmat] calculates
the tensor residuals for the given dataset taking in account the outliers.
ResidualCalc[DTI,tensor,gradients,bvector] calculates the tensor residuals
for the given dataset. Tensor must contain Log[S0].
ResidualCalc[DTI,tensor,outlier,gradients,bvector] calculates the tensor residuals for
the given dataset taking in account the outliers. Tensor must contain Log[S0].
ResidualCalc[DTI,tensor,bmat] calculates the tensor residuals for
the given dataset. Tensor must contain Log[S0].
ResidualCalc[DTI,tensor,outlier,bmat] calculates the tensor residuals for the given
dataset taking in account the outliers. Tensor must contain Log[S0]. ≫

```
Attributes[ResidualCalc] = {Protected, ReadProtected}
```

```
Options[ResidualCalc] = {MeanRes → All}
```

```
SyntaxInformation[ResidualCalc] = {ArgumentsPattern → {_, _, _, _, OptionsPattern[]}}
```

SigmaCalc[DTI,grad,bvec] calculates the
noise sigma based on the tensor residual, using a blur factor of 10.
SigmaCalc[DTI,tens,grad,bvec] calculates the noise sigma based on
the tensor residual, using a blur factor of 10.
SigmaCalc[DTI,grad,bvec,blur] calculates the noise sigma based on
the tensor residual, If blur is 1 ther is no blurring.
SigmaCalc[DTI,tens,grad,bvec,blur] calculates the noise sigma based
on the tensor residual. If blur is 1 ther is no blurring. ≫

```
Attributes[SigmaCalc] = {Protected, ReadProtected}
```

```
Options[SigmaCalc] = {FilterShape → Median}
```

```
SyntaxInformation[SigmaCalc] = {ArgumentsPattern → {_, _, _, _., _., OptionsPattern[]}}
```

SortDiffusionData[data, grad, bval] sorts
the diffusion datasets grad and bval for magnitude of bvalue. ≫

```
Attributes[SortDiffusionData] = {Protected, ReadProtected}
```

```
SyntaxInformation[SortDiffusionData] = {ArgumentsPattern → {_, _, _}}
```

TensorCalc[data, gradients, bvalue] calculates the diffusion tensor
for the given dataset. Allows for one unweighted image and one b value.
Gradient directions must be in the form {{x1,y1,z1}, ..., {xn,yn,zn}}
without the unweighted gradient direction.
bvalue is a singe number indicating the b–value used.
TensorCalc[data, gradients, bvec] calculates the diffusion tensor for the
given dataset. allows for multiple unweighted images and multiple bvalues.
allows for differnt tensor fitting methods. gradient directions must be in the form {{x1,y1,z1},
..., {xn,yn,zn}} with the unweighted direction as {0,0,0}. bvec the bvector, with a
bvalue defined for each gradient direction. b value for unweighted images is 0.
TensorCalc[data, bmatix] calculates the diffusion tensor for the given dataset.
allows for multiple unweighted images and multiple bvalues.
bmat is the bmatrix which can be generated usiong Bmatrix. ≫

```
Attributes[TensorCalc] = {Protected, ReadProtected}
```

```
Options[TensorCalc] = {MonitorCalc → True, Method → iWLLS, FullOutput → True,
  RobustFit → True, Parallelize → True, RobustFitParameters → {0.0001, 6}}
```

```
SyntaxInformation[TensorCalc] = {ArgumentsPattern → {_, _, _., OptionsPattern[]}}
```

TensorCorrect[tensor, phase, shift, vox] corrects the tensor based
on B0 field map. Can perform both translation and rotation of tensor. ≫

```
Attributes[TensorCorrect] = {Protected, ReadProtected}
```

```
Options[TensorCorrect] = {RotationCorrect → False}
```

```
SyntaxInformation[TensorCorrect] = {ArgumentsPattern → {_, _, _, _, _., OptionsPattern[]}}
```

Options

Distribution is an option for AngleCalc. values can be "0–180", "0–90" and "–90–90". ≫

```
Attributes[Distribution] = {Protected, ReadProtected}
```

FilterShape is an option for SigmaCalc. Can be "Gaussian" of "Median". ≫

```
Attributes[FilterShape] = {Protected, ReadProtected}
```

FullOutput is an option for TensorCalc when using bvector. When True also the S0 is given as output. ≫

```
Attributes[FullOutput] = {Protected, ReadProtected}
```

MeanRes is an option for ResidualCalc.
When True the root mean square of the residual is calculated. ≫

```
Attributes[MeanRes] = {Protected, ReadProtected}
```

Method is an option for various algorithm–intensive
functions that specifies what internal methods they should use. ≫

```
Attributes[Method] = {Protected}
```

MonitorCalc is an option for all Calc fucntions. When true the proceses of the calculation is shown. ≫

```
Attributes[MonitorCalc] = {Protected, ReadProtected}
```

NormalizeSignal is an option for DriftCorrect. ≫

```
Attributes[NormalizeSignal] = {Protected, ReadProtected}
```

Parallelize[*expr*] evaluates *expr* using automatic parallelization. ≫

```
Attributes[Parallelize] = {HoldFirst, Protected}
```

```
Parallelize[Parallel`Kernels`Private`args$___] :=
 (Parallel`Protected`doAutolaunch[TrueQ[Parallel`Static`$enableLaunchFeedback]];
  Parallelize[Parallel`Kernels`Private`args$])
```

```
Options[Parallelize] = {Method → Automatic, DistributedContexts :→ $Context}
```

Reject is an option for EigenvalCalc. It True
then voxels with negative eigenvalues are rejected and set to 0. ≫

```
Attributes[Reject] = {Protected, ReadProtected}
```

RejectMap is an option for EigenvalCalc. If Reject is True and RejectMap is True
both the eigenvalues aswel as a map showing je rejected values is returned. ≫

```
Attributes[RejectMap] = {Protected, ReadProtected}
```

RobustFit is an option for TensorCalc. If true outliers will be rejected in the fit, only works with WLLS.
If FullOutput is given the outlier map is given. ≫

```
Attributes[RobustFit] = {Protected, ReadProtected}
```

RobustFitParameters is an option for TensorCalc. gives the threshold for
stopping the itterations and the kappa for the outlier marging, {tr,kappa}. ≫

```
Attributes[RobustFitParameters] = {Protected, ReadProtected}
```

RotationCorrect is an option for TensorCorrect. Default is False. Is a tensor is deformed
setting to True also the shear is accounted for by local rotation of the tensor ≫

```
Attributes[RotationCorrect] = {Protected, ReadProtected}
```

UseMask is a function for MeanSignal and DriftCorrect ≫

```
Attributes[UseMask] = {Protected, ReadProtected}
```

Viste Tools

Functions

DatRead[file] imports data from file (dtitool *.dat format) as binary data using Real32 format. ≫

```
Attributes[DatRead] = {Protected, ReadProtected}
```

```
SyntaxInformation[DatRead] = {ArgumentsPattern → {_}}
```

DatWrite[file, data] exports data to *.dat file as binary data using Real32 format. ≫

```
Attributes[DatWrite] = {Protected, ReadProtected}
```

```
SyntaxInformation[DatWrite] = {ArgumentsPattern → {_, _}}
```

DTItoolExp[tensor, voxsize] exports tensor to {XX.dat, YY.dat, ZZ.dat, XY.dat, XZ.dat,
YZ.dat} and uses XX.dat as background and generates corresponding *dti files.
DTItoolExp[tensor, voxsize, folder] exports tensor to {XX.dat, YY.dat, ZZ.dat, XY.dat, XZ.dat, YZ.dat}
to the given folder and uses XX.dat as background and generates corresponding *dti files.
DTItoolExp[tensor, voxsize, folder, add] exports tensor to {XX.dat, YY.dat, ZZ.dat,
XY.dat, XZ.dat, YZ.dat} to the given folder and uses XX.dat as background
and generates corresponding *dti files adds – add to the filenames.
DTItoolExp[back, tensor, voxsize] exports background to back.dat and tensor to {XX.dat,
YY.dat, ZZ.dat, XY.dat, XZ.dat, YZ.dat} and generates corresponding *dti files.
DTItoolExp[back, tensor, voxsize, folder] exports background to back.dat and tensor to {XX.dat, YY.dat,
ZZ.dat, XY.dat, XZ.dat, YZ.dat} to the given folder and generates corresponding *dti files.
DTItoolExp[back, tensor, voxsize, folder, add] exports background to back.dat and
tensor to {XX.dat, YY.dat, ZZ.dat, XY.dat, XZ.dat, YZ.dat} to the given folder
and generates corresponding *dti files and adds – add to the filenames. ≫

```
Attributes[DTItoolExp] = {Protected, ReadProtected}
```

```
SyntaxInformation[DTItoolExp] = {ArgumentsPattern → {_, _, _., _., _.}}
```

DTItoolExpFile[file, background, add, voxsize] exports a *.dti text file. ≫

```
Attributes[DTItoolExpFile] = {Protected, ReadProtected}
```

```
SyntaxInformation[DTItoolExpFile] = {ArgumentsPattern → {_, _, _, _}}
```

DTItoolExpInd[data, file] exports a 3D array
data to the file filename DTItool format (*.dat) using DatWrite.
DTItoolExpInd[data, file ,folder] exports data to given file and folder.
DTItoolExpInd[data, file ,folder, add]
exports data to given file and folder and adds –add to the filename. ≫

```
Attributes[DTItoolExpInd] = {Protected, ReadProtected}
```

```
SyntaxInformation[DTItoolExpInd] = {ArgumentsPattern → {_, _, _., _.}}
```

DTItoolExpTens[tensor] exports a diffustion tensor array to the DTItool format (*.dat).
DTItoolExpTens[tensor, add] exports tensor and adds – add to the filenames.
DTItoolExpTens[tensor, add, folder]
exports tensor to the given folder and adds – add to the filenames. ≫

```
Attributes[DTItoolExpTens] = {Protected, ReadProtected}
```

```
SyntaxInformation[DTItoolExpTens] = {ArgumentsPattern → {_, _., _.}}
```

ExportVol[filename, data, voxsize] exports a .vol and .raw file which can be loaded in DTItool 3.0. ≫

```
Attributes[ExportVol] = {Protected, ReadProtected}
```

```
Options[ExportVol] = {BinaryType → Integer16}
```

```
SyntaxInformation[ExportVol] = {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

ImportDTI[folder] imports xx.dat, yy.dat, zz.dat, xy.dat, xz.dat and yz.dat from the given folder.
ImportDTI[folder, add] imports xx–add.dat, yy–add.dat,
zz–add.dat, xy–add.dat, xz–add.dat and yz–add.dat from the given folder.
ImportDTI[{file1, file2, ..}] imports the given *.dat files. ≫

```
Attributes[ImportDTI] = {Protected, ReadProtected}
```

```
SyntaxInformation[ImportDTI] = {ArgumentsPattern → {_, _.}}
```

ImportVol[] promts for a vol file to open.
ImportVol["file"] inpormts the file.
the function returns data and voxsize. ≫

```
Attributes[ImportVol] = {Protected, ReadProtected}
```

```
Options[ImportVol] = {}
```

```
SyntaxInformation[ImportVol] = {ArgumentsPattern → {_., OptionsPattern[]}}
```

LoadFiberTracts[] promts for a .fbs to open
LoadFiberTracts["file"] imports the file. ≫

```
Attributes[LoadFiberTracts] = {Protected, ReadProtected}
```

```
SyntaxInformation[LoadFiberTracts] = {ArgumentsPattern → {_.}}
```

Options

BinaryType is an option for ExportVol and must be
"Integer16" for an integer array and "Real32" for a Double array. ≫

```
Attributes[BinaryType] = {Protected, ReadProtected}
```