Description
Returns the string "imageprofilefitter".

Example

```
- pf := imageprofilefitter(infile='ngc1313.cube')
- pf.type()
imageprofilefitter
```

1.2.17 imagepol - Tool

Package general
Module images

Polarimetric analysis of images
include "imagepol.g"

Constructors

imagepol	Create an Imagepol tool from an AIPS++ image file
imagepoltestimage	Create an Imagepol tool and a test image file

Functions

complexfraclinpol	Complex fractional linear polarization
complexlinpol	Complex linear polarization
depolratio	Linear depolarization ratio
done	Destroy this Imagepol tool
fourierrotationmeasure	Find Rotation Measure (Fourier approach)
fraclinpol	Fractional linear polarization
fractotpol	Fractional total polarization
id	Return the fundamental identifier of this tool
linpolint	Linearly polarized intensity
linpolposang	Linearly polarized position angle
makecomplex	Make a Complex image
pol	Polarized quantities
rotationmeasure	Find Rotation Measure (traditional approach)
sigma	Find best guess at thermal noise
sigmadepolratio	Error in linear depolarization ratio
sigmafraclinpol	Error in fractional linear polarization
sigmafractotpol	Error in fractional total polarization

sigmalinpolint	Error in linearly polarized intensity
sigmalinpolposang	Error in linearly polarized position angle
sigmastokes	Find standard deviation of specified Stokes data
sigmastokesi	Find standard deviation of Stokes I data
sigmastokesq	Find standard deviation of Stokes Q data
sigmastokesu	Find standard deviation of Stokes U data
sigmastokesv	Find standard deviation of Stokes V data
sigmatotpolint	Error in total polarized intensity
stokes	Stokes
stokesi	Stokes I
stokesq	Stokes Q
stokesu	Stokes U
stokesv	Stokes V
summary	Summarise Imagepol tool
totpolint	Total polarized intensity

Summary
An Imagepol `tool` provides specialized polarimetric analysis of images. Some of these things could be done with the Lattice Expression Language (LEL; see note223) but are more conveniently offered separately.

General
You construct an Imagepol tool from an image (AIPS++, FITS, and Miriad formats are supported) with a Stokes coordinate axis. What you can do with your Imagepol `tool` depends on exactly which Stokes parameters you have in the image. You must have some combination of Stokes I, Q, U and V on the Stokes axis. These refer to total intensity, two components of linear polarization, and circular polarization, respectively. Therefore, if you ask for linear polarization and the image only has Stokes I and V, you will get an error.
The Imagepol `tool` functions generally return, by default, an Image `tool` as their output. In most cases, this is a “virtual” image. There are a range of different sorts of “virtual” images in AIPS++ (see Image). But the Imagepol `tool functions` generally return reference Image `tools`. That is, these reference different pieces of the original image that was used to construct the Imagepol `tool`, either directly, or as mathematical expressions (e.g. the polarized intensity). If you delete the construction image, you render your Imagepol `tool` and its outputs useless. If you wish, rather than return a virtual image `tool`, you can fill in the `outfile` argument of most Imagepol `tool` functions and write the image out to disk, associating the Image `tool` with the disk file.
In some of the functions, the standard deviation of the thermal noise is needed. This is for debiasing polarized intensity images or working out statistical error images. By default it is worked out for you from the construction image with outliers from the mean discarded. Since it is the

thermal noise it is trying to find, it is worked out from the V, Q & U, and finally I data in that order of precedence. This is because Stokes V is much less likely to contain source signal than Stokes I. You can supply the noise level if you know it better. For example, for small images or images with few signal-free pixels, the theoretical estimate may be better.

Analysis and Display

Traditionally, when generating secondary and tertiary images (e.g. position angle, fractional polarization, rotation measure etc), one masks the putput image according to some statistical test. For example, if the error in the output image is greater than some value, or the errors in the input images are greater than some value. Imagepol `tools` do not offer this kind of masking. It does provide you with the error images for the derived images. By using LEL when you analyze your images, you can mask the images however you want when you use them. That is, we defer the error interpretation as long as possible. Here is an example.

Example

```
include 'imagepol.g'
ip := imagepol('stokes.image')       # Construct Imagepol tool from image
lpa := ip.linpolposang()             # Linearly polarized position angle image
lpaerr := ip.sigmalinpolposang()     # Error in linearly polarized position angle image
lpa.view(mask='$lpaerr<5')           # Display when p.a. error < 5 degrees
```

Display is handled via the Viewer `tool`. It can display and overlay combinations of raster, contour and vector representations of your data. Please see the view function of the Image `tool` and the document Getting Results) for high-level information on how to display images in general.
It is common to display linear polarization data via vectors where the position angle of the vector is the position angle of the linear polarized radiation, and the amplitude of the vector is proportional to either the total polarized intensity or fractional polarized intensity.
The data source of a vector display is either a Complex or a Float image. If it is a Complex image (e.g. the complex linear polarization $Q + iU$) then both the amplitude and the phase (position angle) are available. If it is just a Float image, then it is assumed to be the position angle and an amplitude of unity will be provided. The angular units are given by image brightness units which you can set with function setbrightnessunits. If the units are not recognized as angular, degrees are assumed.
The position angle is measured positive North through East when you display a plane holding a celestial coordinate (the usual astronomical convention). For other axis/coordinate combinations, a positive position angle is measured from +x to +y in the absolute world coordinate frame.

The Imagepol `tool` can create Complex disk images for you via functions complexlinpol (complex linear polarization), complexfraclinpol (complex fractional linear polarization) and makecomplex (takes amp/phase or real/imag). As well as these Complex images, you can also make Float images of the linearly polarized intensity, linearly polarized position angle, and the fractional linearly polarized intensity (see below).
Now, the Image `tool` cannot yet deal with Complex images (it will in the future). This means that you cannot currently do

```
include 'imagepol.g'
p := imagepol('iquv.im')  # Make Imagepol tool
p.complexlinpol('clp')    # Make complex image of linear polarization disk file
clp := image('clp')       # Error
```

which is a bit annoying presently. However, the Viewer `tool` is able to read Complex images so that you are able to display them ok.

```
include 'imagepol.g'
include 'viewer.g'
p := imagepol('iquv.im')  # Make Imagepol tool
p.complexlinpol('clp')    # Make complex image of linear polarization disk file
dv.gui()                  # Start data manager to give access to Complex image
```

If you wanted to make a vector map display you would select the appropriate image in the Viewer's data manager GUI, click 'Vector Map' on the right hand side and it would appear in the display. Note that the Viewer's Vector map display capability also offers you amplitude noise debiasing and the On-The-Fly mask.

Overview of Imagepol `tool` functions

- Access to the different Stokes subimages is via functions stokes, stokesi, stokesq, stokesu, and stokesv.
- Create the standard secondary and tertiary polarization images with
  - complexfraclinpol - complex fractional linear polarization
  - complexlinpol - complex linear polarization
  - makecomplex - make complex image from amp/phase or real/imag
  - pol - polarized quantities as specified
  - linpolint - linearly polarized intensity
  - linpolposang - linearly polarized position angle
  - totpolint - total polarized intensity
  - fraclinpol - fractional linear polarization

- fractotpol - fractional total polarization
- depolratio - linear depolarization ratio

- Create the standard secondary and tertiary polarization error images (simple propagation of Gaussian errors) with
  - sigma - best guess at thermal noise
  - sigmastokes - specified Stokes parameter
  - sigmastokesi - Stokes I
  - sigmastokesq - Stokes Q
  - sigmastokesu - Stokes U
  - sigmastokesv - Stokes V
  - sigmalinpolint - linearly polarized intensity
  - sigmalinpolposang - linearly polarized position angle
  - sigmatotpolint - total polarized intensity
  - sigmafraclinpol - fractional linear polarization
  - sigmafractotpol - fractional total polarization
  - sigmadepolratio - linear depolarization ratio
- You can compute Rotation Measure images via either the traditional techniques involving a number of different frequencies (regularly or irregularly spaced) with function

  rotationmeasure or via a new Fourier Technique for many regularly-spaced frequencies with function fourierrotationmeasure.

imagepol - Constructor

Package	**general**
Module	**images**
Tool	**imagepol**

Create an Imagepol tool from an AIPS++ image file

Synopsis
imagepol(infile)

Arguments

infile	in	Input image (file name or tool) Allowed:String or Image tool

Returns
Imagepol `tool` or fail

Description

The constructor takes an image as its input. This can be specified as either the name of a disk file, or you can give a pre-existing Image `tool`. If you give a disk file name, the disk image file may be in AIPS++, FITS or Miriad format. The input image must have a Stokes axis. The exact collection of Stokes that the image has, determines what the Imagepol `tool` can compute. Stokes I, Q, U, and V refer to total intensity, two components of linear polarization, and circular polatization, respectively. Therefore, if you ask for linear polarization and the image only has Stokes I and V, you will get an error.
The input image may contain data at many frequencies. For example, the image may be a 4D image with axes RA, DEC, Stokes and Frequency (order not important) where the Frequency axis is regularly sampled. However, the image may also contain many frequencies at irregular intervals. Such an image may be created with the Image `tool` constructor called imageconcat. It enables you to concatenate images along an axis, and it allows irregular coordinate values along that axis.

Example

```
- im := image('$AD/myimage')
- p1 := imagepol(im)
- p2 := imagepol('$AD/myimage')
```

We create two Imagepol `tools`, one from an Image `tool` and one directly from the disk file.

Example

```
- im := imageconcat(infiles="im.f1 im.f2 im.f3 im.f4 im.f5", axis=4)
- p := imagepol(im)
- p.rotationmeasure(rm='rm', rmerr='rmerr', rmmax=500, maxpaerr=10)
```

Say we have 5 images, each with axes RA, DEC, Stokes, and Frequency in that order. We make an Image `tool` which concatenates these images along the frequency axis - you have ordered them in increasing or decreasing frequency order. Note that the Image `tool` is virtual - it is not written to an output file. This just means that the data are read from the input images as needed. The Imagepol `tool` is then made from this virtual Image `tool`. We then compute the Rotation Measure and Rotation Measure error images with the traditional method and write them out to disk.

---

imagepoltestimage - Constructor

Package	**general**
Module	**images**
Tool	**imagepol**

Create an Imagepol tool and a test image file

Synopsis
imagepoltestimage(outfile, rm, pa0, sigma, nx, ny, nf, f0, bw)

Arguments

outfile	in	Output image file name Allowed:String Default: imagepol.iquv
rm	in	Rotation Measure (rad/m/m) Allowed:Vector of float Default: Auto no-ambiguity determine
pa0	in	Position angle (degrees) at zero wavelength Allowed:Float Default: 0
sigma	in	Fractional noise level Allowed:Float Default: 0.001
nx	in	Shape of image in x direction Allowed:Integer Default: 32
ny	in	Shape of image in y direction Allowed:Integer Default: 32
nf	in	Shape of image in frequency direction Allowed:Integer Default: 32
f0	in	Reference frequency (Hz) Allowed:Float Default: 1.4e9
bw	in	Bandwidth (Hz) Allowed:Float Default: 128e6

Returns
Imagepol `tool` or fail

Description

The constructor can be used to generate a test image and then create an Imagepol `tool` from it.
The test image is 4-dimensional (RA, DEC, Stokes and Frequency). The Stokes axis holds I,Q,U and V. The source is just a constant I (if you don't add noise all spatial pixels will be identical) and V. Q and U vary with frequency according to the specified Rotation Measure components (no attempt to handle bandwidth smearing within channels is made). The actual values of I,Q,U, and V are chosen arbitrarily otherwise (could be added as arguments if desired).
You can use this image, in particular, to explore the Rotation Measure algorithms in functions rotationmeasure and fourierrotationmeasure.
If you don't specify the Rotation Measure, then it is chosen for you so that there is no position angle ambiguity between adjacent channels (the value will be sent to the Logger).
The noise added to the image is specified as a fraction of the total intensity (constant). Gaussian noise with a standard deviation of `sigma *` I_max is then added to the image.

Example

```
- p1 := imagepoltestimage(rm=200)
- p1.rotationmeasure(rm='rm.out', rmmax=250)
- rm := image('rm.out')
- rm.view()
```

---

imagepol.complexlinpol - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Complex linear polarization

Synopsis
complexlinpol(outfile)

Arguments

outfile	in	Output image file name Allowed:String

Returns
T or fail

Description

This function (short-hand name `clp`) produces the complex linear polarization; $Q + iU$ and writes it to a disk image file.
The Image `tool` cannot yet handle Complex images. You must therefore write the Complex image to disk. The Viewer can display Complex images. Also the Table tool can access Complex images.

Example

```
- include 'imagepol.g'
- include 'table.g'
- p := imagepol('stokes.cube')
- p.complexlinpol('cplx')
- t := table('cplx')
- t.browse()
```

---

imagepol.complexfraclinpol - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Complex fractional linear polarization

Synopsis
complexfraclinpol(outfile)

Arguments

outfile	in	Output image file name Allowed:String

Returns
T or fail

Description

This function (short-hand name `clp`) produces the complex fractional linear polarization; $(Q+iU)/I$ and writes it to a disk image file.
The Image `tool` cannot yet handle Complex images. You must therefore write the Complex image to disk. The Viewer can display Complex images. Also the Table tool can access Complex images.

Example

```
- include 'imagepol.g'
- include 'table.g'
- p := imagepol('stokes.cube')
- p.complexfraclinpol('cplx')
- t := table('cplx')
- t.browse()
```

---

imagepol.depolratio - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Linear depolarization ratio

Synopsis
depolratio(infile, debias, clip, sigma, outfile)

Arguments

infile	in	Other image Allowed:String or Image tool
debias	in	Debias the linearly polarized intensity ? Allowed:T or F Default: F
clip	in	Clip level for auto-sigma determination Allowed:Float Default: 10
sigma	in	Standard deviation of thermal noise Allowed:Float Default: Auto determined
outfile	in	Output image file name Allowed:String Default: Unset

Returns
Image tool or fail

Description

This function (short-hand name `dr`) returns the linear depolarization ratio computed from two frequencies; this is the ratio of the fractional linear polarization at the two frequencies. Generally this is done when you have generated two images, each at a different frequency (continuum work). Thus if the fractional linear polarization images are $m_{\nu 1}$ and $m_{\nu 2}$ then the depolarization ratio is $m_{\nu 1}/m_{\nu 2}$.
This function operates with two images; the first (at frequency $\nu 1$) is that with which you constructed your Imagepol `tool`. The second (at frequency $\nu 2$) is supplied via the argument `infile`, which can be an Image `tool`, or a String holding the name of the `image file`.
In generating the depolarization ratio, you may optionally debias the linearly polarized intensity. This requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.
You can get the depolarization ratio error image with function sigmadepolratio.

Example

```
- im1 := image('stokes.4800')
- im2 := image('stokes.8300')
- p := imagepol(im1)
```

```
- dpr := p.depolratio(im2);           # m_4800 / m_8300
- edpr := p.sigmadepolratio(im2);
```

---

imagepol.done - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Destroy this Imagepol tool

Synopsis
done()

Returns
T or fail

Description

If you no longer need to use an Imagepol `tool`, calling this function will free up its resources. That is, it destroys the `tool`. You can no longer call any functions on the `tool` after it has been `done`. Note that the underlying `image file` with which the `tool` was constructed is not deleted. In addition, any images that you created with the Imagepol `tool functions` will also still be viable.

Example

```
- p := imagepol('stokes.cube')
- p.done()
- p
F
```

---

imagepol.fourierrotationmeasure - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Find Rotation Measure (Fourier approach)

Synopsis
fourierrotationmeasure(complex, amp, pa, real, imag, zerolag0)

Arguments

complex	in	Output complex linear polarization image file name Allowed:String Default: Unset
amp	in	Output linear polarization amplitude image file name Allowed:String Default: Unset
pa	in	Output linear polarization position angle (degrees) image file name Allowed:String Default: Unset
real	in	Output linear polarization real image file name Allowed:String Default: Unset
imag	in	Output linear polarization imaginary angle image file name Allowed:String Default: Unset
zerolag0	in	Force zero lag to 0 ? Allowed:T or F Default: F

Returns
T or fail

Description

This function (short-hand name `frm`) will only work if you constructed the Imagepol `tool` from an image containing Stokes Q and U, and a regular frequency axis. It Fourier transforms the complex linear polarization (Q+iU) over the spectral axis to the rotation measure axis. Thus, if your input image contained RA/DEC/Stokes/Frequency, the output image would be RA/DEC/RotationMeasure. The Rotation Measure axis has as many pixels as the spectral axis.
This method enables you to see the polarization as a function of Rotation Meausure. Its main use is when searching for large RMs. See Killeen, Fluke,

Zhao and Ekers (1999, preprint) for a description of this method (or http://www.atnf.csiro.au/~nkilleen/rm.ps) and its advantages over the traditional method (rotationmeasure) of extracting the Rotation Measure. Although you can write out the complex polarization image with the argument `complex`, you can't do much with it because Image `tools` cannot yet be constructed from complex images. Hence you can also write out the complex linear polarization image in any or all of the other forms.
The argument `zerolag0` allows you to force the zero lag (or central bin) of the Rotation Measure spectrum to zero (effectively by subtracting the mean of Q and U from the Q and U images). This may avoid Gibbs phenomena from any strong low Rotation Measure signal which would normally fall into the central bin.

Example

```
- p := imagepoltestimage(outfile='iquv.im', rm=[5.0e5,1e6], nx=8, ny=8, nf=512,
                         f0=1.4e9, bw=8e6)
- p.frm(amp='amp')
- amp := image('amp')
- amp.view()                    # And reorder to put RM along X-axis
```

In this example we give two RM components and recover them.

---

imagepol.fraclinpol - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Fractional linear polarization

Synopsis

fraclinpol(debias, clip, sigma, outfile)

Arguments

debias	in	Debias the linearly polarized intensity ? Allowed:T or F Default: F
clip	in	Clip level for auto-sigma determination Allowed:Float Default: 10
sigma	in	Standard deviation of thermal noise Allowed:Float Default: Auto determined
outfile	in	Output image file name Allowed:String Default: Unset

Returns
Image tool or fail

Description
This function (short-hand name `flp`) returns the fractional linear polarization; $\sqrt{(Q^2+U^2)}/I$.
You may optionally debias the polarized intensity. This requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

Example

```
- p := imagepol('stokes.cube')
- flp := p.fraclinpol()
```

---

imagepol.fractotpol - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Fractional total polarization

Synopsis
fractotpol(debias, clip, sigma, outfile)

Arguments

debias	in	Debias the total polarized intensity ? Allowed:T or F Default: F
clip	in	Clip level for auto-sigma determination Allowed:Float Default: 10
sigma	in	Standard deviation of thermal noise Allowed:Float Default: Auto determined
outfile	in	Output image file name Allowed:String Default: Unset

Returns
Image tool or fail

Description
This function (short-hand name `ftp`) returns the fractional linear polarization; $\sqrt{(Q^2 + U^2 + V^2)}/I$.
You may optionally debias the polarized intensity. This requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.
If your image contains only Q and U, or only V, then just those components contribute to the total polarized intensity.

Example

```
- p := imagepol('stokes.cube')
- ftp := p.fractotpol()
```

---

imagepol.id - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Return the fundamental identifier of this tool

Synopsis
id()

Description

This function should be of little interest to users. It returns the fundamental identifier of the tool.

Returns
ToolID - a record

Example

```
- p := imagepol('stokes.cube')
- p.id()
[sequence=15, pid=20665, time=941768782, host=elara, agentid=3]
```

---

imagepol.linpolint - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Linearly polarized intensity

Synopsis
linpolint(debias, clip, sigma, outfile)

Arguments

debias	in	Debias the linearly polarized intensity ? Allowed:T or F Default: F
clip	in	Clip level for auto-sigma determination Allowed:Float Default: 10
sigma	in	Standard deviation of thermal noise Allowed:Float Default: Auto determined
outfile	in	Output image file name Allowed:String Default: Unset

Returns
Image tool or fail

Description
This function (short-hand name `lpi`) returns the linearly polarized intensity; $\sqrt{(Q^2 + U^2)}$.
You may optionally debias the polarized intensity. This requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

Example

```
- p := imagepol('stokes.cube')
- lpi := p.linpolint()
```

---

imagepol.linpolposang - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Linearly polarized position angle

Synopsis
linpolposang()

Returns
Image tool or fail

Description

This function (short-hand name `lppa`) returns the linearly polarized position angle image ($0.5 \tan^{-1}(U/Q)$) in degrees.

Example

```
- p := imagepol('stokes.cube')
- lppa := p.linpolposang()
```

---

imagepol.makecomplex - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Make a Complex image

Synopsis
makecomplex(complex, real, imag, amp, phase)

Arguments

complex	in	Output complex image file name Allowed:String
real	in	Input real image file name Allowed:String Default: Unset
imag	in	Input imaginary image file name Allowed:String Default: Unset
amp	in	Input amplitude image file name Allowed:String Default: Unset
phase	in	Input phase image file name Allowed:String Default: Unset

Returns
T or fail

Description

This function generates a Complex `image file` from either a real and imaginary, or an amplitude and phase pair of images. If you give a linear position angle image for the phase, it will be multipled by two before the real and imaginary parts are formed.

Example

```
- p := imagepol('iquv')
- p.complexlinpol('qu.cplx1')
-
- q := p.stokesq()
- u := p.stokesu()
- q2 := q.subimage('q')
- u2 := u.subimage('u')
- p.makecomplex('qu.cplx2', real='q', image='u')
```

In this example we make two complex linear polarization images which should be identical.

---

imagepol.pol - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Polarized quantities

Synopsis
pol(which, debias, clip, sigma, outfile)

Arguments

which	in	Specify operation Allowed:String: One of 'lpi', 'tpi', 'lppa', 'flp', 'ftp' (case insensitive)
debias	in	Debias the polarized intensity ? Allowed:T or F Default: F
clip	in	Clip level for auto-sigma determination Allowed:Float Default: 10
sigma	in	Standard deviation of thermal noise Allowed:Float Default: Auto determined
outfile	in	Output image file name Allowed:String Default: Unset

Returns
Image tool or fail

Description

This function just packages the other specific polarization functions into one where you specify an operation with the argument `which` (can be useful for scripts). This argument can take the values:

- 'lpi' - linearly polarized intensity (function linpolint)
- 'tpi' - total polarized intensity (function totpolint)
- 'lppa' linearly polarized position angle (function linpolposang)
- 'flp' - fractional linear polarization (function fraclinpol)
- 'ftp' - fractional total polarized intensity (function fractotpol)

Example

```
- p := imagepol('stokes.cube')
- lpi := p.pol('lpi')
```

---

imagepol.rotationmeasure - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Find Rotation Measure (traditional approach)

Synopsis
rotationmeasure(rm, rmerr, pa0, pa0err, nturns, chisq, sigma, rmfg, rmmax, maxpaerr, plotter, nx, ny)

Arguments

rm	in	Output Rotation Measure image file name Allowed:String Default: Unset
rmerr	in	Output Rotation Measure error image file name Allowed:String Default: Unset
pa0	in	Output position angle (degrees) at zero wavelength image file name Allowed:String Default: Unset
pa0err	in	Output position angle (degrees) at zero wavelength error image file name Allowed:String Default: Unset
nturns	in	Output number of turns image file name Allowed:String Default: Unset
chisq	in	Output reduced chi squared image file name Allowed:String Default: Unset
sigma	in	Estimate of the thermal noise Allowed:Float Default: Auto
rmfg	in	Foreground Rotation Measure (rad/m/m) to subtract Allowed:Float Default: 0
rmmax	in	Maximum Rotation Measure (rad/m/m) to solve for Allowed:Float Default: 0
maxpaerr	in	Maximum input position angle error (degrees) to allow Allowed:Float Default: None
plotter	in	Name of plotter Allowed:String Default: None
nx	in	Number of plots in x direction Allowed:Integer Default: 5
ny	in	Number of plots in y direction Allowed:Integer Default: 5

Returns
T or fail

Description

This function (short-hand name `rm`) generates the Rotation Measure image from a collection of different frequencies. It will only work if the Imagepol `tool` was constructed from an image containing Stokes Q and U, and a frequency axis (regular or irregular) with at least 2 pixels. It will work out the position angle images for you.
See also the fourierrotationmeasure function for a new Fourier-based approach.
Rotation Measure algorithms that work robustly are not common. The main problem is in trying to account for the $n - \pi$ ambiguity (see Leahy et al, Astronomy & Astrophysics, 156, 234 or Killeen et al; http://www.atnf.csiro.au/~nkilleen/rm.ps).
The algorithm that this function uses is that of Leahy et al. Please refer to their paper for details (their Appendix A.1). But as in all these algorithms, the basic process is that for each spatial pixel, a vector of position angles (i.e. at the different frequencies) is fit to determine the Rotation Measure and the position angle at zero wavelength (and their errors). You can write out an image containing the number of $n - -\pi$ turns that were added to the data at each spatial pixel and for which the best fit was found. You can also write out the reduced chi-squared image for the fits.
Note that as yet no assessment of curvature (i.e. deviation from the simple linear position angle - λ^2 functional form) is made.
You can write out any or all of the output images.
The argument `sigma` gives the themal noise in Stokes Q and U. By default it is worked out for you from the image data. But if it proves to be inaccurate (maybe not many signal-free pixels), then you can input it here. This is used for working out the error in the position angles (propagation of Gaussian errors).
The argument `maxpaerr` specifies the maximum allowable error in the position angle that is acceptable. The default is an infinite value. From the standard propagation of errors, the error in the linearly polarized position angle is determined from the Stokes Q and U images (at each spatial pixel for each frequency). At each spatial pixel we do a fit to the position angle vector (i.e. at the different frequencies) to determine the Rotation Measure. If the position angle error for any pixel in the vector exceeds the specified value, it is dropped from the fit. The process generates an error for the fit and this is used to compute the errors in the output images.
Note that `maxpaerr` is *not* used to specify that any pixel for which the output position angle error exceeds this value should be masked out.
The argument `rmfg` is used to specify a foreground RM value. For example, you may know the mean RM in some direction out of the Galaxy, then including this can aid the algorithm (reduces ambiguity).
The argument `rmmax` specifies the maximum absolute RM that should be solved for. This quite an important parameter. If you leave it at the default, zero, basically, no ambiguity handling will be invoked. So you must have some

idea of what you are looking for - this is the basic problem with Rotation Measure algorithms.
Plotting can be done with a PGPLOT device via argument `plotter`. The syntax is `plotter=name/type`. For example `plotter='plot1.ps/ps'` (disk postscript file) or `plotter='1/xs'` (X-windows device) or `plotter='plot/glish'` (*Glish* PGplotter). The plots show the used position angle data (after ambiguity adjustments), position angle errors, and fit for each profile. The pixel location of each profile and reduced chi-square of the fit are written in the title.

Example

```
- im := imageconcat(infiles="im.f1 im.f2 im.f3 im.f4 im.f5", axis=4)
- p := imagepol(im)
- ok := p.rm(rm='rm', rmerr='rmerr', rmmax=800, maxpaerr=10)
```

Say we have 5 images, each with axes RA, DEC, Stokes, and Frequency in that order. We make an Image `tool` which concatenates these images along the frequency axis - you have ordered them in increasing or decreasing frequency order. Note that the Image `tool` is virtual - it is not written to an output file. This just means that the data are read from the input images as needed. The Imagepol `tool` is then made from this virtual Image `tool`. We then compute the Rotation Measure and Rotation Measure error images with the traditional method and write them out to disk.

---

imagepol.sigma - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Find best guess at thermal noise

Synopsis
sigma(clip)

Arguments

clip	in	Clip level for auto-sigma determination Allowed:Float Default: 10

Returns
Float or fail

Description

This function returns the standard deviation from V, Q&U or I in that order of precedence. It is attempting to give you the best estimate of the thermal noise it can from the data. Outliers from the mean are clipped at the specified level.

Example

```
- p := imagepol('stokes.cube')
- sigma := p.sigma()
```

---

imagepol.sigmadepolratio - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Error in linear depolarization ratio

Synopsis
sigmadepolratio(infile, debias, clip, sigma, outfile)

Arguments

infile	in	Other image Allowed:String or Image tool
debias	in	Debias the linearly polarized intensity ? Allowed:T or F Default: F
clip	in	Clip level for auto-sigma determination Allowed:Float Default: 10
sigma	in	Standard deviation of thermal noise Allowed:Float Default: Auto determined
outfile	in	Output image file name Allowed:String Default: Unset

Returns
Image tool or fail

Description

This function (short-hand name `edr`) returns the error in the linear depolarization ratio computed from two frequencies; this is the ratio of the fractional linear polarization at the two frequencies. Generally this is done when you have generated two images, each at a different frequency (continuum work). Thus if the fractional linear polarzation images are $m1$ and $m2$ then the depolarization ratio is $m1/m2$.
This function operates with two images; the first is that with which you constructed your Imagepol `tool`. The second is supplied via the argument `infile`, which can be an Image `tool`, or a String holding the name of the `image file`.
In generating the depolarization ratio, and hence its error, you may optionally debias the linearly polarized intensity. This requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.
You can get the depolarization ratio image with function depolratio.

Example

```
- im1 := image('stokes.4800')
- im2 := image('stokes.8300')
- p := imagepol(im1)
- dpr := p.depolratio(im2);
- edpr := p.sigmadepolratio(im2);
```

imagepol.sigmafraclinpol - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Error in fractional linear polarization

Synopsis
sigmafraclinpol(clip, sigma, outfile)

Arguments

clip	in	Clip level for auto-sigma determination Allowed:Float Default: 10
sigma	in	Standard deviation of thermal noise Allowed:Float Default: Auto determined
outfile	in	Output image file name Allowed:String Default: Unset

Returns
Image tool or fail

Description

This function (short-hand name `sflp`) returns the error (standard deviation) of the fractional linear polarization. This result comes from standard propagation of errors. The result is an Image tool as the error is signal-to-noise ratio dependent.
This function requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

Example

```
- p := imagepol('stokes.cube')
- sigflp := p.sigmafraclinpol()
```

---

imagepol.sigmafractotpol - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Error in fractional total polarization

Synopsis
sigmafractotpol(clip, sigma, outfile)

Arguments

clip	in	Clip level for auto-sigma determination Allowed:Float Default: 10
sigma	in	Standard deviation of thermal noise Allowed:Float Default: Auto determined
outfile	in	Output image file name Allowed:String Default: Unset

Returns
Image tool or fail

Description

This function (short-hand name `sftp`) returns the error (standard deviation) of the fractional total polarization. This result comes from standard propagation of errors. The result is an Image tool as the error is signal-to-noise ratio dependent.
This function requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

Example

```
- p := imagepol('stokes.cube')
- sigftp := p.sigmafractotpol()
```

---

imagepol.sigmalinpolint - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Error in linearly polarized intensity

Synopsis
sigmalinpolint(clip, sigma, outfile)

Arguments

clip	in	Clip level for auto-sigma determination Allowed:Float Default: 10
sigma	in	Standard deviation of thermal noise Allowed:Float Default: Auto determined
outfile	in	Output image file name Allowed:String Default: Unset

Returns
Float or fail

Description

This function (short-hand name `slpi`) returns the error (standard deviation) of the linearly polarized intensity; $\sqrt{(Q^2 + U^2)}$. This result comes from standard propagation of statistical errors. The result is a float as the error is not signal-to-noise ratio dependent
This function requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

Example

```
- p := imagepol('stokes.cube')
- siglpi := p.sigmalinpolint()
```

---

imagepol.sigmalinpolposang - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Error in linearly polarized position angle

Synopsis
sigmalinpolposang(clip, sigma, outfile)

Arguments

clip	in	Clip level for auto-sigma determination Allowed:Float Default: 10
sigma	in	Standard deviation of thermal noise Allowed:Float Default: Auto determined
outfile	in	Output image file name Allowed:String Default: Unset

Returns
Image tool or fail

Description

This function (short-hand name `slppa`) returns the error (standard deviation) of the linearly polarized position angle $(0.5\tan^{-1}(U/Q)\sqrt{(Q^2+U^2)})$ in degrees. This result comes from standard propagation of errors. The result is an Image tool as the error is signal-to-noise ratio dependent.
This function requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

Example

```
- p := imagepol('stokes.cube')
- siglppa := p.sigmalinpolposang()
```

---

imagepol.sigmastokes - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Find standard deviation of specified Stokes data

Synopsis
sigmastokes(which, clip)

Arguments

which	in	Specify Stokes Allowed:String: One of 'I', 'Q', 'U', 'V' (case insensitive)
clip	in	Clip level for auto-sigma determination Allowed:Float Default: 10

Returns
Float or fail

Description

This function (short-hand name `ss`) returns the standard deviation of the noise for the specified Stokes. Outliers from the mean are clipped at the specified level.

Example

```
- p := imagepol('stokes.cube')
- sigq := p.sigmastokes('q', 10.0)
```

---

imagepol.sigmastokesi - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Find standard deviation of Stokes I data

Synopsis
sigmastokesi(clip)

Arguments

clip	in	Clip level for auto-sigma determination Allowed:Float Default: 10

Returns
Float or fail

Description

This function (short-hand name `ssi`) returns the standard deviation of the noise for the Stokes I data. Outliers from the mean are clipped at the specified level.

Example

```
- p := imagepol('stokes.cube')
- sigi := p.sigmastokesi(10.0)
```

---

imagepol.sigmastokesq - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Find standard deviation of Stokes Q data

Synopsis
sigmastokesq(clip)

Arguments

clip	in	Clip level for auto-sigma determination Allowed:Float Default: 10

Returns
Float or fail

Description

This function (short-hand name `ssq`) returns the standard deviation of the noise for the Stokes Q data. Outliers from the mean are clipped at the specified level.

Example

```
- p := imagepol('stokes.cube')
- sigq := p.sigmastokesq(10.0)
```

---

imagepol.sigmastokesu - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Find standard deviation of Stokes U data

Synopsis
sigmastokesu(clip)

Arguments

clip	in	Clip level for auto-sigma determination Allowed:Float Default: 10

Returns
Float or fail

Description

This function (short-hand name `ssu`) returns the standard deviation of the noise for the Stokes U data. Outliers from the mean are clipped at the specified level.

Example

```
- p := imagepol('stokes.cube')
- sigu := p.sigmastokesu(10.0)
```

---

imagepol.sigmastokesv - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Find standard deviation of Stokes V data

Synopsis
sigmastokesv(clip)

Arguments

clip	in	Clip level for auto-sigma determination Allowed:Float Default: 10

Returns
Float or fail

Description

This function (short-hand name `ssv`) returns the standard deviation of the noise for the Stokes V data. Outliers from the mean are clipped at the specified level.

Example

```
- p := imagepol('stokes.cube')
- sigv := p.sigmastokesv(10.0)
```

---

imagepol.sigmatotpolint - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Error in total polarized intensity

Synopsis

sigmatotpolint(clip, sigma)

Arguments

clip	in	Clip level for auto-sigma determination Allowed:Float Default: 10
sigma	in	Standard deviation of thermal noise Allowed:Float Default: Auto determined

Returns

Float or fail

Description

This function (short-hand name `stpi`) returns the error (standard deviation) of the total polarized intensity; $\sqrt{(Q^2+U^2+V^2)}$. This result comes from

standard propagation of statistical errors. The result is a float as the error is not signal-to-noise ratio dependent
This function requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

Example

```
- p := imagepol('stokes.cube')
- sigtpi := p.sigmatotpolint()
```

---

imagepol.stokes - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Stokes

Synopsis
stokes(which, outfile)

Arguments

which	in	Specify Stokes Allowed:String: One of 'I', 'Q', 'U', 'V' (case insensitive)
outfile	in	Output image file name Allowed:String Default: Unset

Returns
Image tool or fail

Description

This function returns an image containing the specified Stokes only. This interface can be useful for scripts.

Example

```
- p := imagepol('stokes.cube')
- q := p.stokes('q')
```

---

imagepol.stokesi - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Stokes I

Synopsis
stokesi()

Returns
Image tool or fail

Description
This function returns an image containing Stokes I only.

Example

```
- p := imagepol('stokes.cube')
- i := p.stokesi()
```

---

imagepol.stokesq - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Stokes Q

Synopsis
stokesq()

Returns
Image tool or fail

Description
This function returns an image containing Stokes Q only.

Example

```
- p := imagepol('stokes.cube')
- q := p.stokesq()
```

---

imagepol.stokesu - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Stokes U

Synopsis
stokesu()

Returns
Image tool or fail

Description
This function returns an image containing Stokes U only.

Example

```
- p := imagepol('stokes.cube')
- u := p.stokesu()
```

---

imagepol.stokesv - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Stokes V

Synopsis
stokesv()

Returns
Image tool or fail

Description
This function returns an image containing Stokes V only.

Example

```
- p := imagepol('stokes.cube')
- v := p.stokesv()
```

---

imagepol.summary - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Summarise Imagepol tool

Synopsis
summary()

Returns
Image tool or fail

Description

This function just lists a summary of the Imagepol `tool` to the logger. Currently it just summarizes the image with which it was constructed.

Example

```
- p := imagepol('stokes.cube')
- p.summary()
Image name       : Concatenation :
Telescope        : ATCA
Observer         : Fluke
Date observation : 1998/05/18/09:35:54
Image mask       : None
Image units      : Jy/beam

Direction system : J2000
Frequency system : LSR
Velocity  system : RADIO

Name             Proj Shape Tile   Coord value at pixel    Coord incr Units
---------------------------------------------------------------------------
Right Ascension   SIN   201   67  11:40:48.400   101.00 -2.000000e+00 arcsec
Declination       SIN   201   67 -26.29.30.000   101.00  2.000000e+00 arcsec
Frequency               13    7  1.432000e+09     1.00 -8.000000e+06 Hz
Velocity                                 Fail     1.00          Fail km/s
Stokes                   4    1          IQUV
```

---

imagepol.totpolint - Function

Package	**general**
Module	**images**
Tool	**imagepol**

Total polarized intensity

Synopsis
totpolint(debias, clip, sigma, outfile)

Arguments

debias	in	Debias the total polarized intensity ? Allowed:T or F Default: F
clip	in	Clip level for auto-sigma determination Allowed:Float Default: 10
sigma	in	Standard deviation of thermal noised Allowed:Float Default: Auto determined
outfile	in	Output image file name Allowed:String Default: Unset

Returns
Image tool or fail

Description
This function (short-hand name `tpi`) returns the total polarized intensity; $\sqrt{(Q^2+U^2+V^2)}$. If your image contains only Q and U, or only V, then just those components contribute to the total polarized intensity.
You may optionally debias the polarized intensity. This requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

Example

```
- p := imagepol('stokes.cube')
- tpi := p.totpolint()
```

1.2.18 is_imagepol - Function

Package general
Module images

Is this variable an imagepol tool ?

Synopsis
is_imagepol(thing)

Arguments

thing	in	Variable to test Allowed:Glish variable Default: None

Returns
T or F

Description

Determine if this variable is an Imagepol `tool`. This is very useful in scripts; for robustness, you check that the tool is an Imagepol `tool` before you start applying Imagepol `tool functions`.

Example

```
- p:=imagepol('myimage')
- is_imagepol(p)
T
- p2 := [10,20]
- is_imagepol(p2)
F
```

1.2.19 imagepoltest - Function

Package general
Module images

Run test suite for Imagepol tool

Synopsis
imagepoltest(which)

Arguments