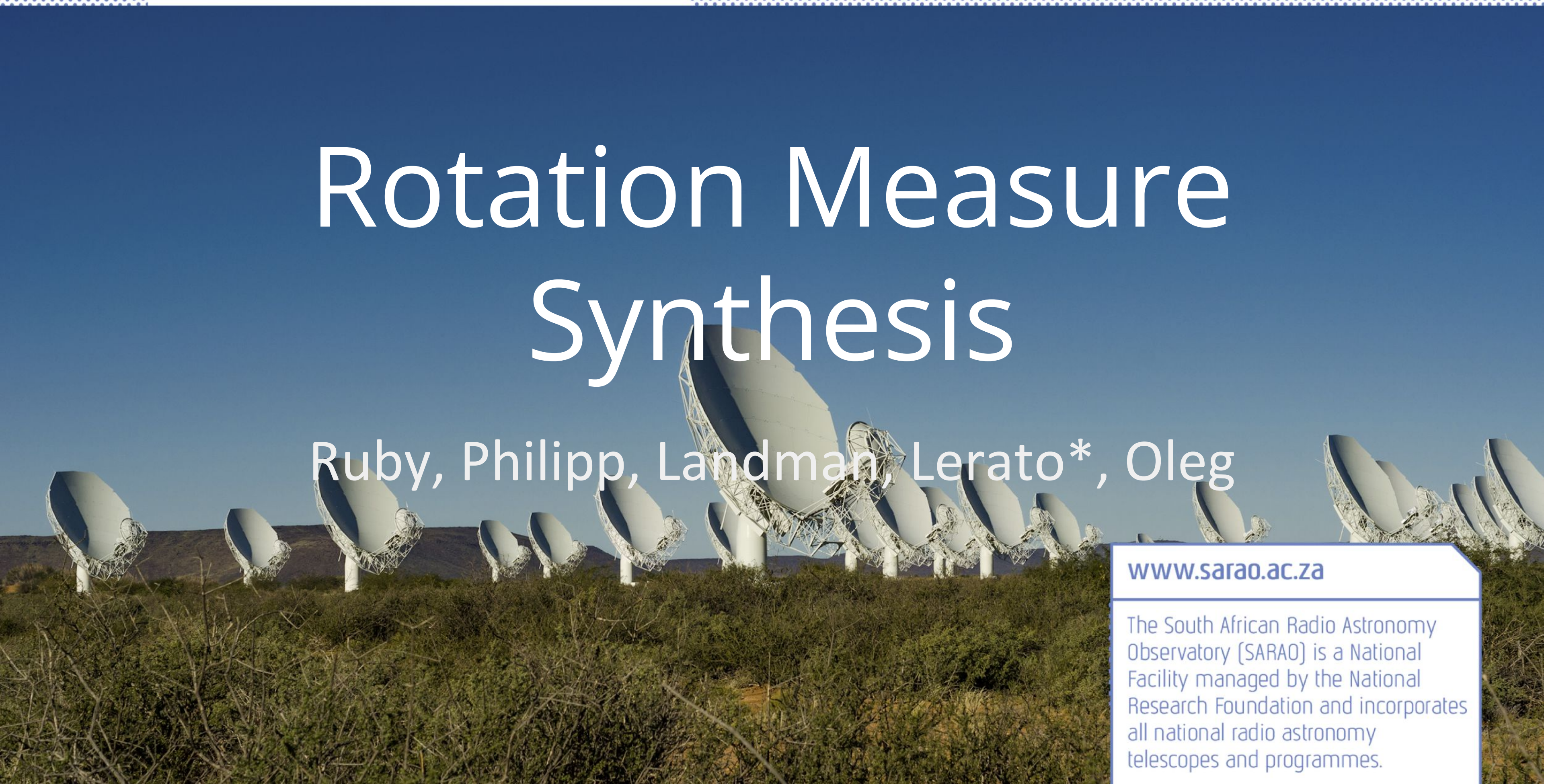
science & innovation
Department: Science and Innovation
REPUBLIC OF SOUTH AFRICA

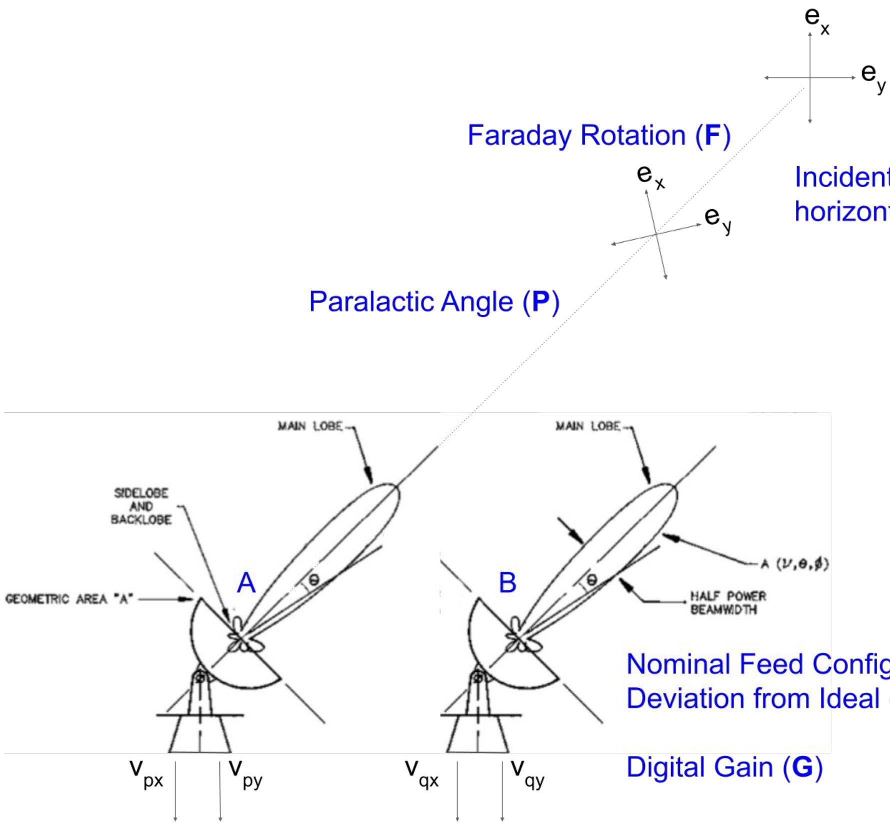
NRF National Research Foundation | SARAO South African Radio Astronomy Observatory

Rotation Measure Synthesis

Ruby, Philipp, Landman, Lerato*, Oleg



Signal detection



Incident electromagnetic field has both a horizontal and a vertical component.

Interferometer / **Measurement Equation**

$$V_{pq} = g_p(t) g_q{}^H(t) G_{pq}(t) X_{pq}(t) \ + \ \varepsilon_{pq}(t) \ + \ \zeta_{pq}(t)$$

$$V_{pq} \approx g_p(t) g_q{}^H(t) X_{pq}(t) \ + \ \varepsilon_{pq}(t)$$

$$X_{pq}(t) \ = \ F_{pq}(\nu) V_{pq}{}^{true}$$

Stokes visibilities

Total intensity

$$I \ = \ < |v_{XX}|^2 + |v_{YY}|^2 >$$

Linear polarisation

$$Q = \ < |v_{XX}|^2 - |v_{YY}|^2 >$$

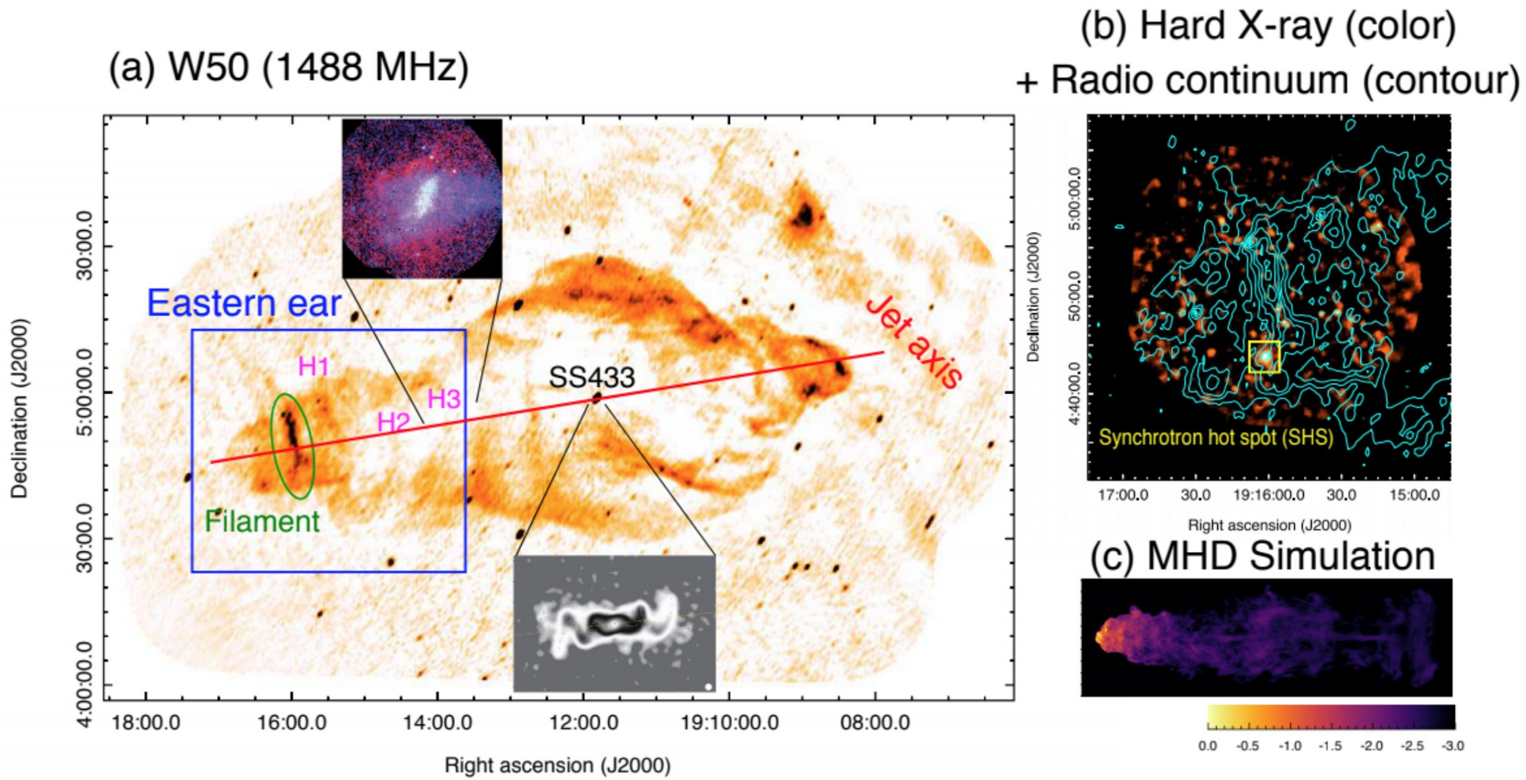
Orthogonal linear polarisation

$$U \ = \ 2 < Re|v_{XY}| >$$

circular polarisation

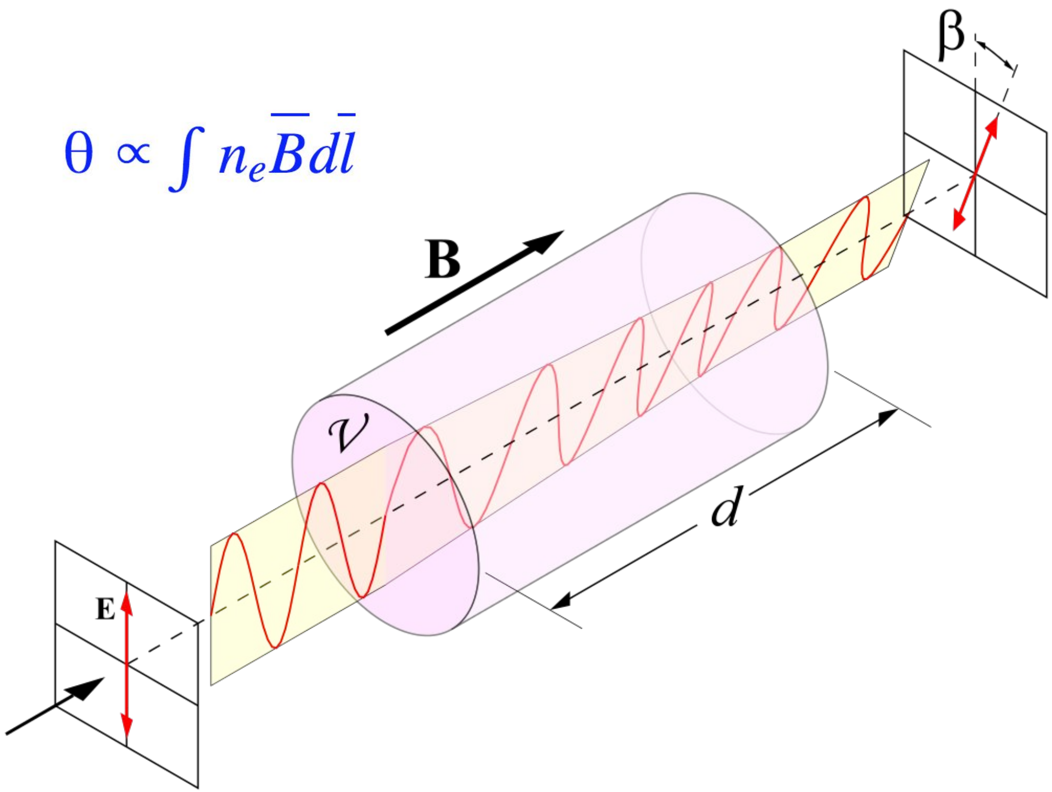
$$V \ = \ 2 < Im|v_{XY}| >$$

Astrophysical jets



Faraday rotation

$\theta \propto \int n_e \overline{B} d\overline{l}$



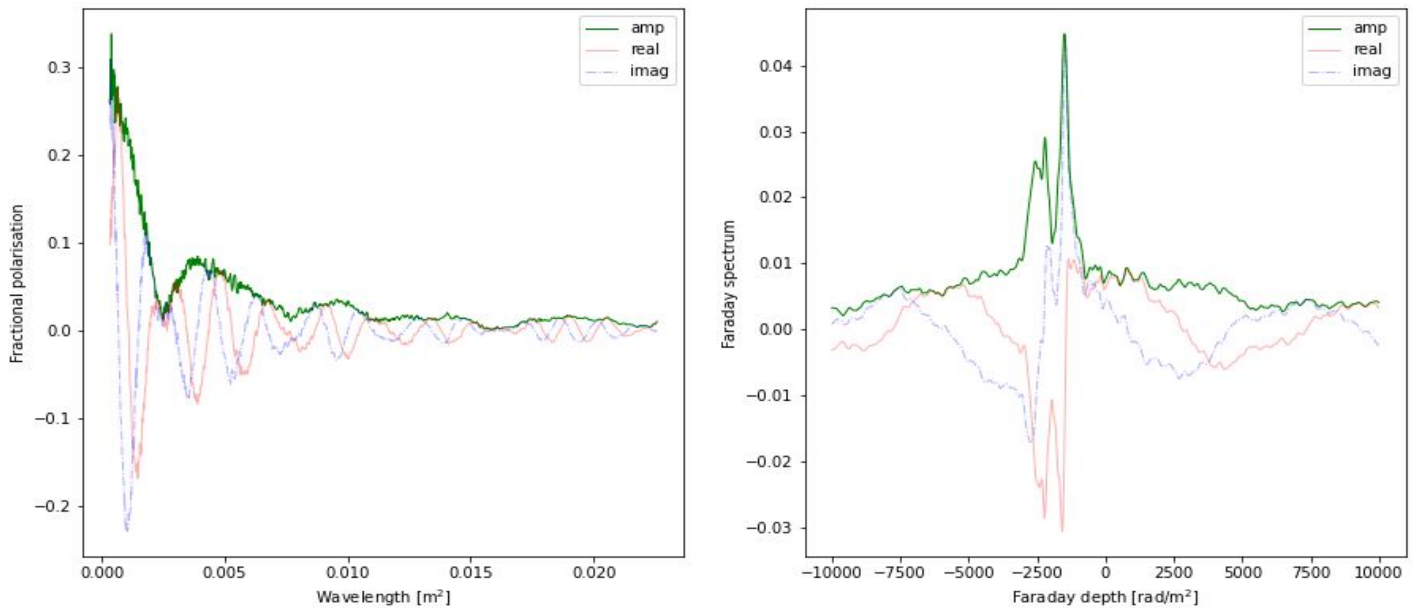
$$P(\lambda^2) = \int F(\varphi) e^{2i\varphi\lambda^2} d\varphi$$

$$\hat{P}(\lambda^2) = W(\lambda^2) \int F(\varphi) e^{2i\varphi\lambda^2} d\varphi$$

$$\hat{F}(\varphi) = K \int \hat{P}(\lambda^2) e^{-2i\varphi(\lambda^2 - \lambda_0^2)} d\lambda^2$$

$$= F(\varphi) * R(\varphi)$$

Implementation



Algorithm

s = Faraday spectrum

d = fractional polarised emission (Q+1j U)

R = Maps a Faraday spectrum to the respective data
R: domain(s) -> domain(d)

Model f: s = f(\xi) such that a priori \xi is standard normal distributed

NIFTy: Approximate P(\xi | d) in some fancy fashion (KL-divergence) with a Gaussian

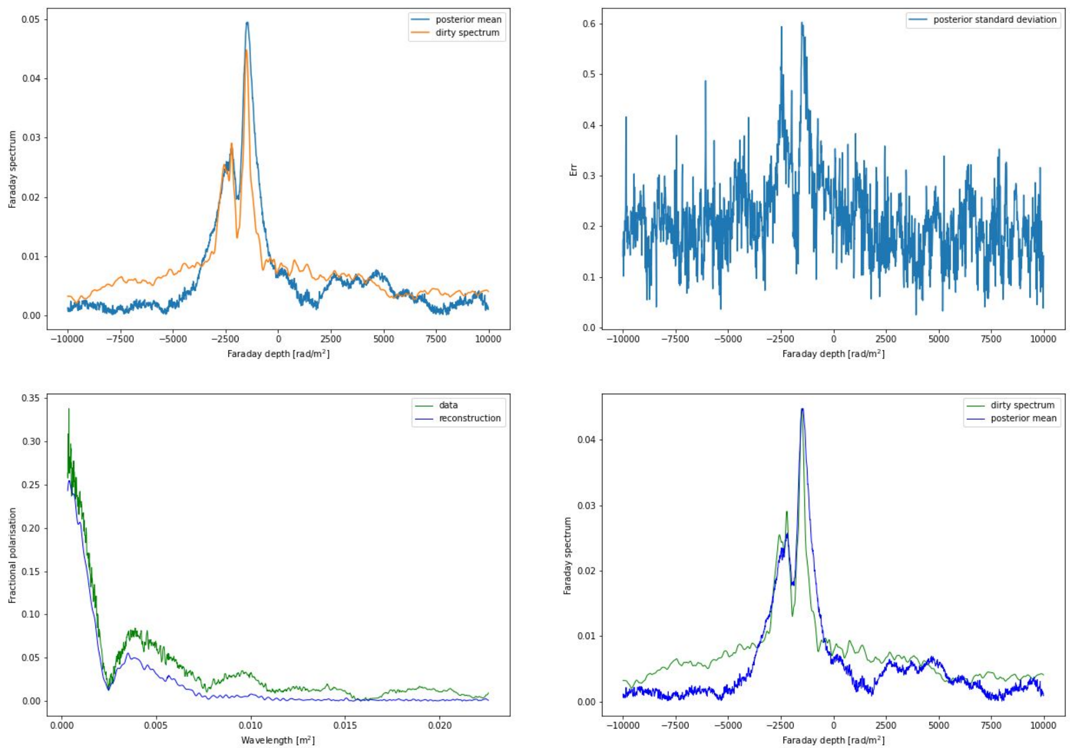
```
class NFFT(ift.LinearOperator):
    def __init__(self, domain, fourier_sampling_points):
        self._domain = ift.DomainTuple.make(domain)  # signal space
        myassert(len(self._domain.shape) == 1)
        myassert(isinstance(self._domain[0], ift.RGSpace))
        # signal space is not harmonic because the output of CorrelatedField
        # will be an RGSpace which is non-harmonic
        myassert(not self._domain[0].harmonic)
        myassert(isinstance(fourier_sampling_points, np.ndarray))
        myassert(fourier_sampling_points.ndim == 1)
        mi = np.min(fourier_sampling_points)
        ma = np.max(fourier_sampling_points)
        nyquist = 1/2/self._domain[0].distances[0]
        myassert(mi > -nyquist)
        myassert(ma <= nyquist)
        myassert(self._domain.size % 2 == 0)
        tgt = ift.UnstructuredDomain(len(fourier_sampling_points))
        self._ks = fourier_sampling_points
        self._target = ift.DomainTuple.make(tgt)  # data space
        self._capability = self.TIMES | self.ADJOINT_TIMES

    def apply(self, x, mode):
        self._check_input(x, mode)
        dom = self._domain[0]
        tgt = self._target[0]

        x = x/tgt.size
        phis = (np.arange(dom.size)-dom.shape[0]//2)*dom.distances[0]

        if mode == self.TIMES:
            res = np.empty(tgt.shape, dtype=np.complex128)
            for ii, k in enumerate(self._ks):
                res[ii] = np.sum(x.val * np.exp(2j * phis * k))
        else:
            res = np.empty(dom.shape, dtype=np.complex128)
            for ii, phi in enumerate(phis):
                res[ii] = np.sum(x.val * np.exp(-2j * phi * self._ks))
        return ift.makeField(self._tgt(mode), res)
```

Results



posterior mean
dirty spectrum
Faraday spectrum
Faraday depth [rad/m^2]
posterior standard deviation
Err
Faraday depth [rad/m^2]
data
reconstruction
Fractional polarisation
Wavelength [m^2]
dirty spectrum
posterior mean
Faraday spectrum
Faraday depth [rad/m^2]