l'Observatoire de Paris | LESIA

Laboratoire d'Études Spatiales et d'Instrumentation en Astrophysique

nenupy

NenuFAR Python package

Alan Loh & the NenuFAR team, User workshop, Station de radioastronomie de Nançay, 2021-11-17

Installation

Getting ready for the hands-on!

- Installation
  - Guest users: OK
  - For the others with a nancep account:

```
python3.8 -m pip install setuptools
python3.8 -m pip install --upgrade jupyter
python3.8 -m pip install --upgrade --user nbconvert
python3.8 -m pip install h5py --upgrade
python3.8 -m pip install ephem --upgrade
python3.8 -m pip install colorama --upgrade
python3.8 -m pip install traitlets --upgrade
python3.8 -m pip install healpy --upgrade
python3.8 -m pip install nenupy --upgrade
python3.8 -m ipykernel install --user
python3.8 -m pip install jupyter ipython --upgrade
python3.8 -m pip install /databf2/nenufar/workshop/pygsm/PyGSM/ --user
```

- Copy the notebook in your home directory

```
/databf2/nenufar/workshop/nenupy_notebook/nenupy.ipynb    # or https://github.com/AlanLoh/nenupy → notebooks
```

- Launch the notebook

```
export PATH=$PATH:~/.local/bin
python3.8 -m jupyter notebook --browser firefox
```

nenupy package

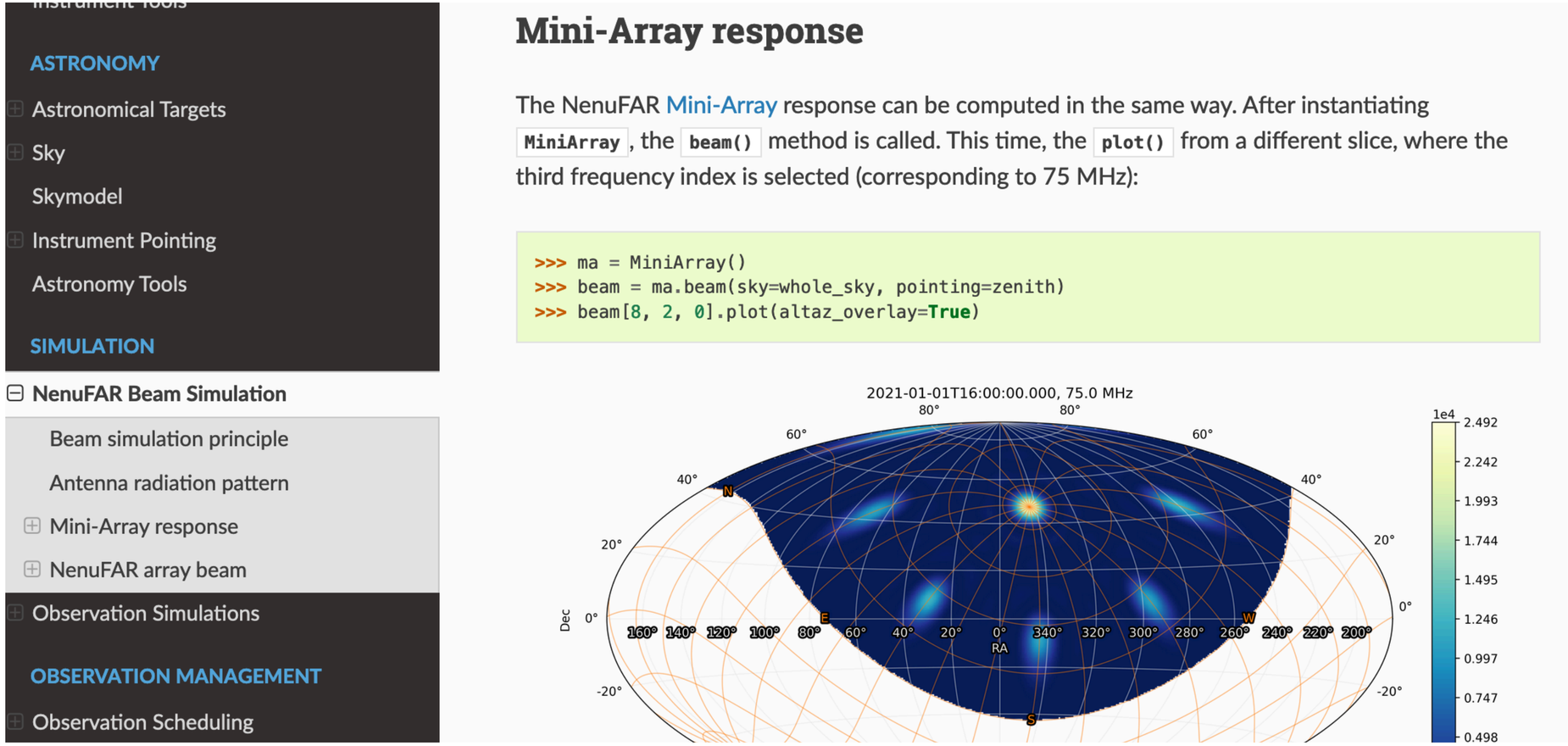
Overview

- Data reading and analysis
  - 'Statistics' data (SST, BST, XST)
  - UnDySPuTeD TF
- VCR tools
  - Apparent source positions and meridian transits
  - NenuFAR-TV and near-field images from XST
  - Observation database (work in progress)
- NenuFAR instrument properties
  - **Array configurations**
  - **Effective area, angular resolution...**
  - **Sensitivity**
  - **Pointing**
- Astronomy
  - **Source ephemerides**
  - **HEALPix or discrete sky**
- Simulations
  - **Beam**
  - **Skymodel**
  - **Observation simulations**
- Observation management
  - **Automatic scheduling**
  - Parset creation API
  - Receiver configuration and data volume estimation

Documentation

Work in progress

https://nenupy.readthedocs.io/



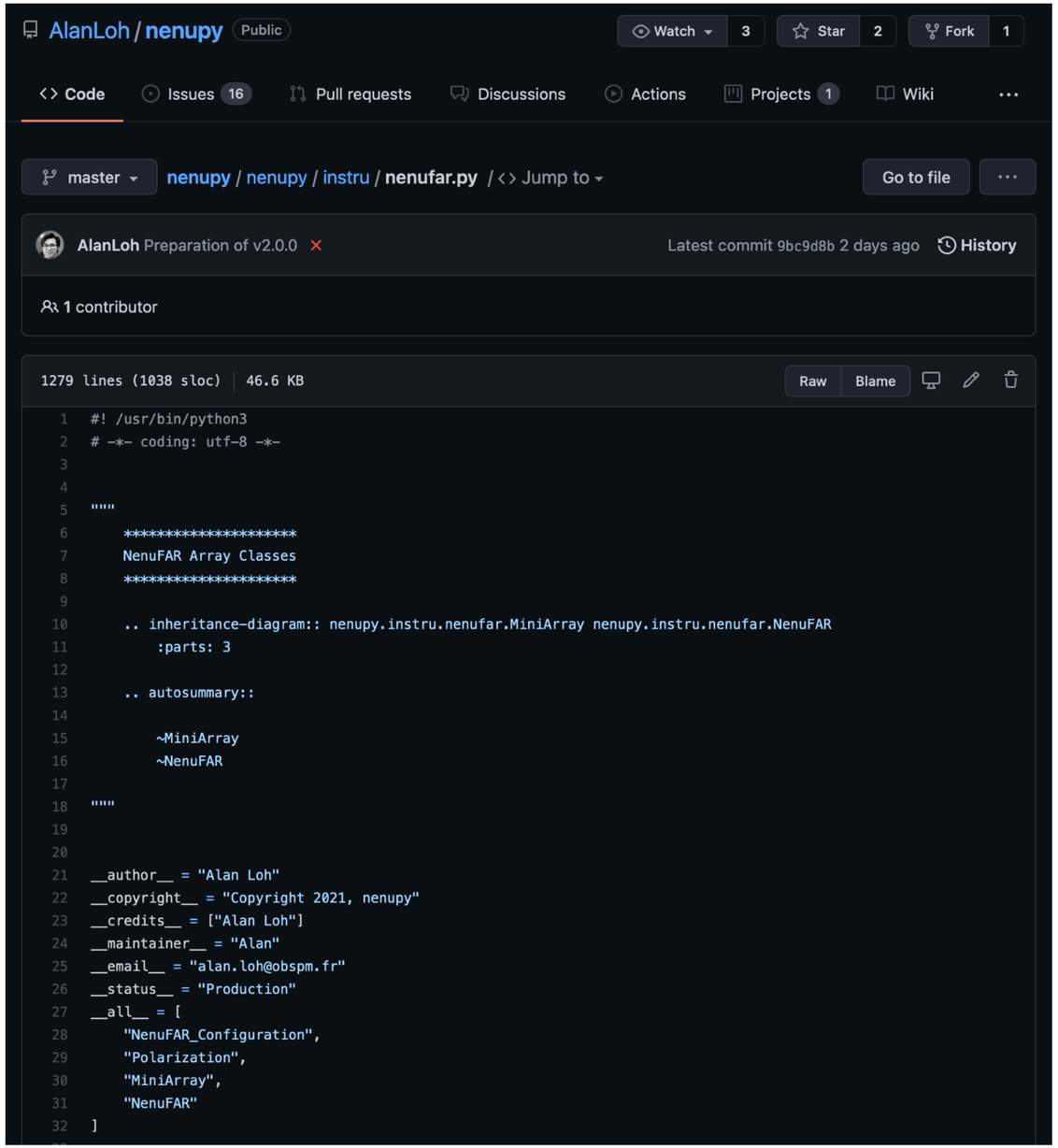
Github

https://github.com/AlanLoh/nenupy

- Have a look at the code

- Post issues (do not hesitate!)
  - Bugs
  - Improvements
  - Feature suggestions

- Contribute
  - Test new features
  - Correct bugs (if I am too slow to respond...)
  - Add your newly developed functions



Quick astropy reminder

astropy

astropy.time

- Load the module

```
>>> from astropy.time import Time, TimeDelta
```

- *Time* instance, and attributes printing

```
>>> t = Time("2021-11-17 16:00:00")
>>> t.isot
'2021-11-17T16:00:00.000'
>>> t.jd
2459536.1666666665
>>> t.mjd
59535.666666666664
```

- *Time* difference: *TimeDelta*

```
>>> dt = Time("2021-11-17 16:00:00") - Time("2021-11-17 14:00:00")
>>> type(dt)
astropy.time.core.TimeDelta
>>> dt.sec
7199.999999999994
>>> ( Time("2021-11-17 16:00:00") + TimeDelta(3600, format="sec") ).isot
'2021-11-17T17:00:00.000'
```

astropy

astropy.units

- Load the module (as an alias)

```
>>> import astropy.units as u
```

- *Quantity* instance, and unit conversions

```
>>> u.Quantity(180, unit="deg")
>>> angle = 10*u.deg    # Other method to instantiates a Quantity
>>> angle.to(u.arcmin)
600'
>>> ( 10*u.cm * 1*u.m ).to(u.m**2)
0.1 m²
>>> frequencies = numpy.linspace(15, 85, 20) * u.MHz    # Define an array Quantity
```

- Operations on *Quantity* objects

```
>>> speed_of_light = 3e5*u.km/u.s
>>> frequency = 30*u.MHz
>>> wavelength = speed_of_light/frequency
>>> wavelength
10000 km / MHz / s
>>> wavelength.to(u.m)
10 m
```

```
>>> frequency = 30*u.MHz
>>> wavelength = frequency.to(
        u.m, equivalencies=u.spectral()
    )
>>> wavelength
9.9930819 m
```

astropy

astropy.coordinates

- Load the module

```
>>> from astropy.coordinates import SkyCoord
```

- *SkyCoord* instance, frame transformation

```
>>> vir_a = SkyCoord.from_name("Virgo A")    # Get a source coordinates from Simbad
>>> vir_a.ra.deg, vir_a.dec.deg              # Print RA/Dec in degrees
(187.70593076, 12.39112329)
>>> source = SkyCoord(ra=300*u.deg, dec=45*u.deg)                  # Arbitrary sky position
>>> source_galactic_coords = source.transform_to("galactic")      # Transformation to Galactic frame
>>> source_galactic_coords.l.deg, source_galactic_coords.b.deg    # Galactic longitude/latitude
(79.92173411857246, 7.8719395931795155)
```

- Operations on *SkyCoord* objects

```
>>> source = SkyCoord(ra=300*u.deg, dec=45*u.deg)    # Compute the angular separation between two SkyCoords
>>> separation = source.separation(vir_a)
>>> type(separation)
<class 'astropy.coordinates.angles.Angle'>
>>> separation.deg
96.3306375837498
```

Observation scheduling

Observation scheduling

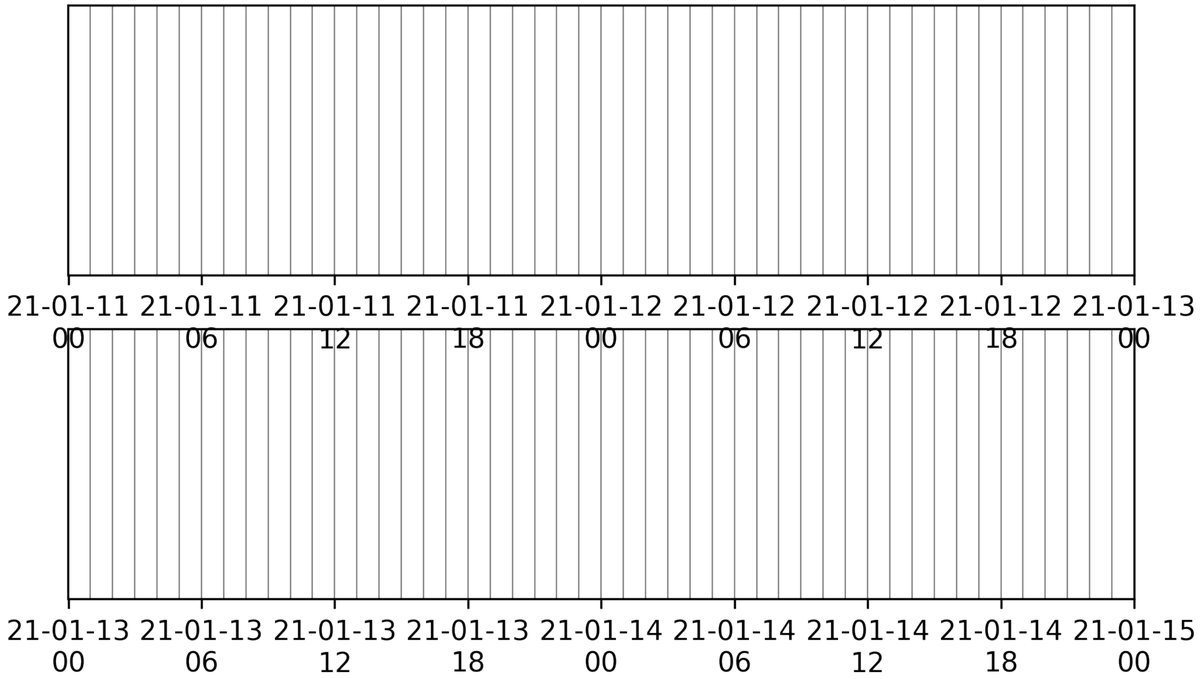
(~automatically)

- Virtual Control Room
  - Allocated time: booking time windows
  - 'Coordinates' tab: source apparent position at Nançay
  - Live observation setup
- nenupy
  - Plan ahead observations (time allocation requests)
  - Optimally schedule several observations within a booking window
  - Based on observing constraints (elevation threshold, meridian transit, local time, etc.)
  - 'Deterministic' algorithm
  - Genetic algorithm
    - Evolves a population of individuals while selecting the most adapted genomes at each generation
      - Individual = observation schedules
      - Genome = list of starting time for each observation
      - Selection : observing constraint score over the schedule

Observation scheduling

Create an empty schedule

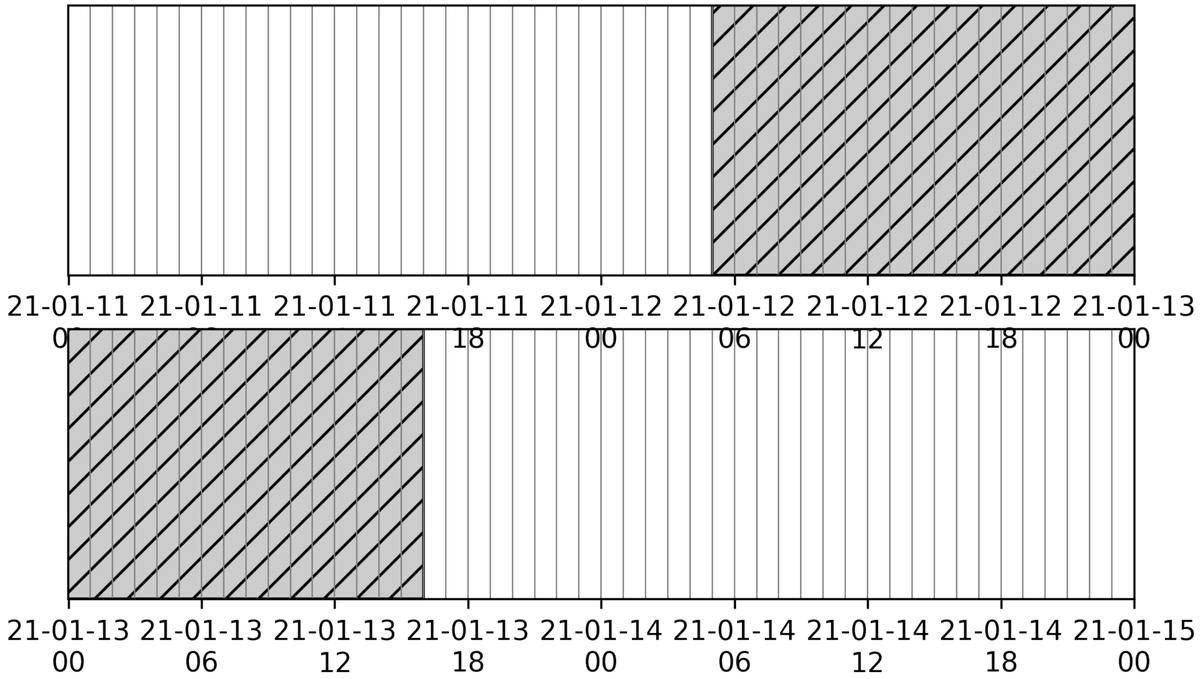
```
>>> from nenupy.schedule import Schedule
>>> schedule = Schedule(
>>>     time_min=Time('2021-01-11 00:00:00'),    # Schedule start time
>>>     time_max=Time('2021-01-15 00:00:00'),    # Schedule stop time
>>>     dt=TimeDelta(3600, format='sec')         # Schedule granularity / time slot duration
>>> )
>>> schedule.plot(days_per_line=2)
```



Observation scheduling

Add unavailable/'reserved' time slots

```
>>> from nenupy.schedule import Schedule, ReservedBlock

>>> schedule = Schedule( time_min=Time('2021-01-11 00:00:00'), time_max=Time('2021-01-15 00:00:00'),
>>>     dt=TimeDelta(3600, format='sec') )
>>> maintenance = ReservedBlock( time_min=Time('2021-01-12 05:00:00'), time_max=Time('2021-01-13 16:00:00') )
>>> schedule.insert(maintenance)
>>> schedule.plot(days_per_line=2)
```



Observation scheduling

Create 'Observation Block(s)'

- Observation block:

```
>>> from nenupy.schedule import ObsBlock, ESTarget

>>> vira = ObsBlock(
>>>     name='Virgo A',
>>>     program='es00',
>>>     target=ESTarget.fromName('Vir A'),         # Or SSTarget (Solar System Target)
>>>     duration=TimeDelta(3600, format='sec')     # Observation duration → will be set at schedule.dt
>>> )
```

- Combining several observation blocks:

```
>>> cyga = ObsBlock(
>>>     name='Cyg A',
>>>     program='es00',
>>>     target=ESTarget.fromName('Cyg A'),
>>>     duration=TimeDelta(3600, format='sec'),
>>> )
>>> my_observations = vira*2 + cyga*3     # Multiply and/or add ObsBlocks together
```

Observation scheduling

Add observing constraints

- Define constraints:

```
>>> from nenupy.schedule import (
        Constraints, ElevationCnst, MeridianTransitCnst
    )

>>> obs_constraints = Constraints(
        ElevationCnst(elevationMin=10*u.deg),
        MeridianTransitCnst()
    )
```

`ElevationCnst`	([elevationMin, weight])
`MeridianTransitCnst`	([weight])
`AzimuthCnst`	(azimuth[, weight])
`LocalTimeCnst`	(hMin, hMax[, weight])
`TimeRangeCnst`	(time_min, time_max[, weight])

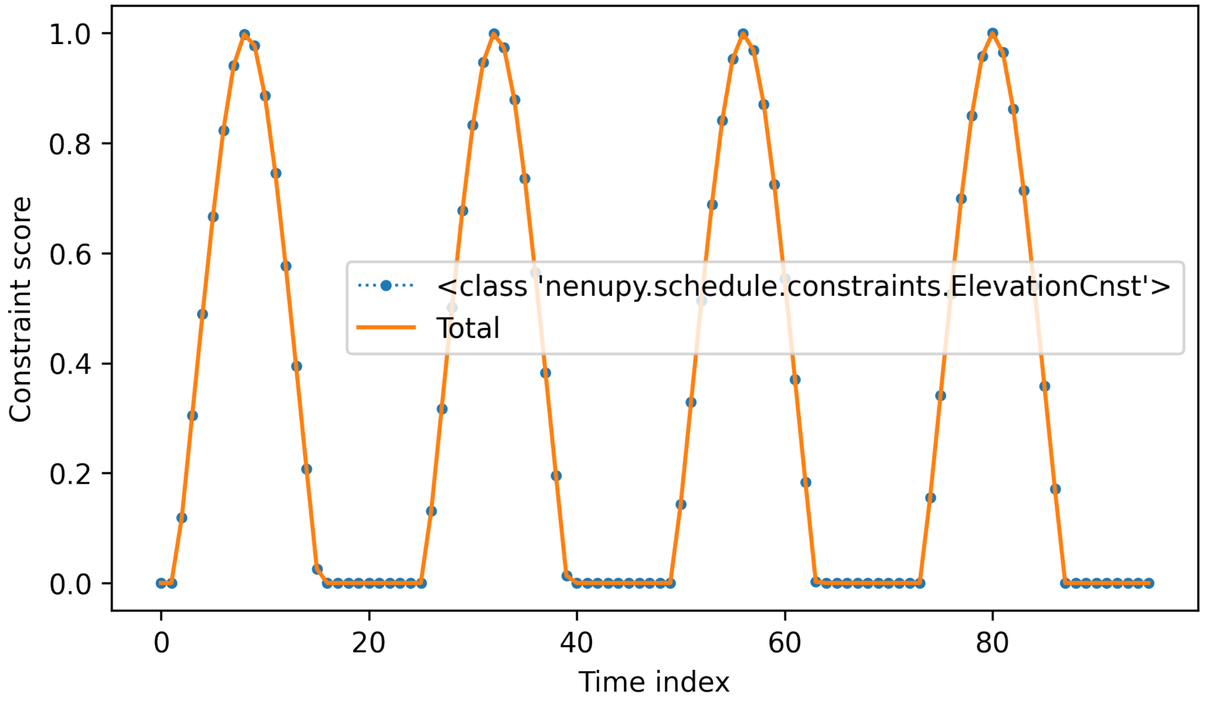
- Add constraints to an observation block:

```
>>> vira = ObsBlock(
>>>     name='Virgo A',
>>>     program='es00',
>>>     target=ESTarget.fromName('Vir A'),
>>>     duration=TimeDelta(3600, format='sec'),
>>>     constraints=obs_constraints
>>> )
```

Observation scheduling

Evaluating the observing constraints over the schedule (1)

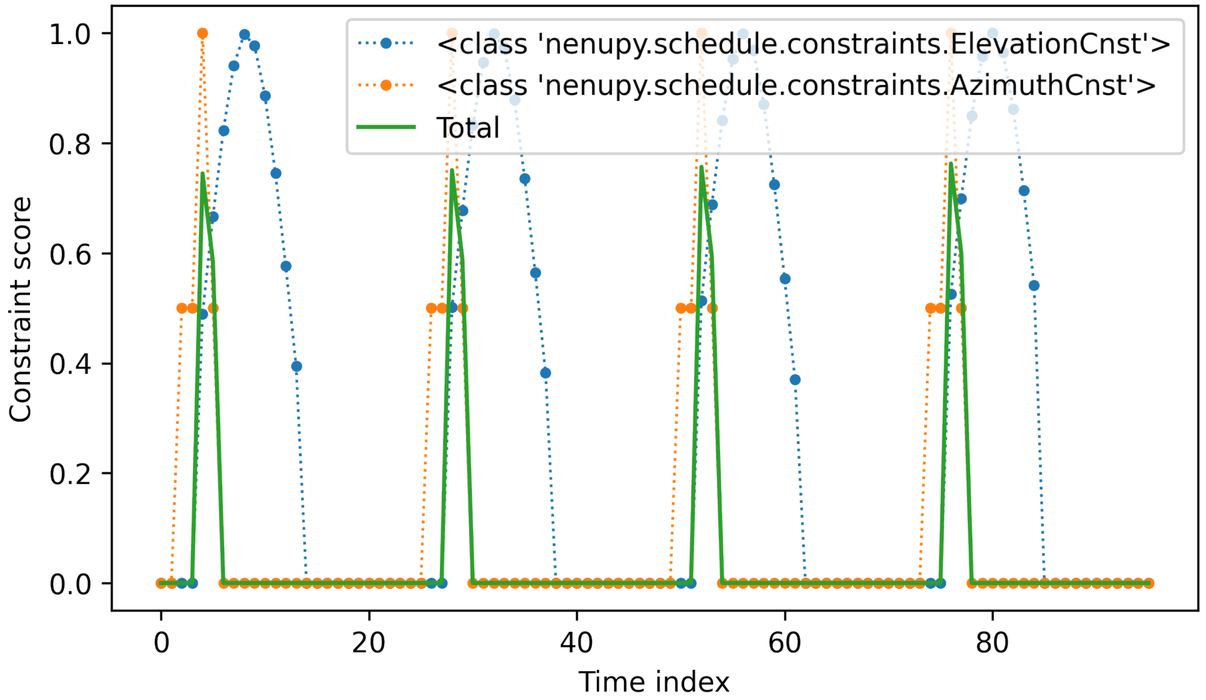
```
>>> schedule = Schedule( time_min=Time('2021-11-15 00:00:00'), time_max=Time('2021-11-19 00:00:00') )
>>> vira = ObsBlock( name='Virgo A', program='es00', target=ESTarget.fromName('Vir A'), duration=TimeDelta(3600*4,
format='sec') )    # By default ElevationCnst(elevationMin=0*u.deg) is selected as standard constraint
>>> schedule.insert(vira)
>>> schedule.book( optimize=False )
>>> schedule.observation_blocks[0].constraints.plot()
```



Observation scheduling

Evaluating the observing constraints over the schedule (2)

```
>>> schedule = Schedule( time_min=Time('2021-11-15 00:00:00'), time_max=Time('2021-11-19 00:00:00') )
>>> vira = ObsBlock( name='Virgo A', program='es00', target=ESTarget.fromName('Vir A'), duration=TimeDelta(3600*4,
format='sec'), constraints=Constraints(ElevationCnst(20*u.deg), AzimuthCnst(100*u.deg)) )
>>> schedule.insert(vira)
>>> schedule.book( optimize=False )
>>> schedule.observation_blocks[0].constraints.plot()
```



Observation scheduling

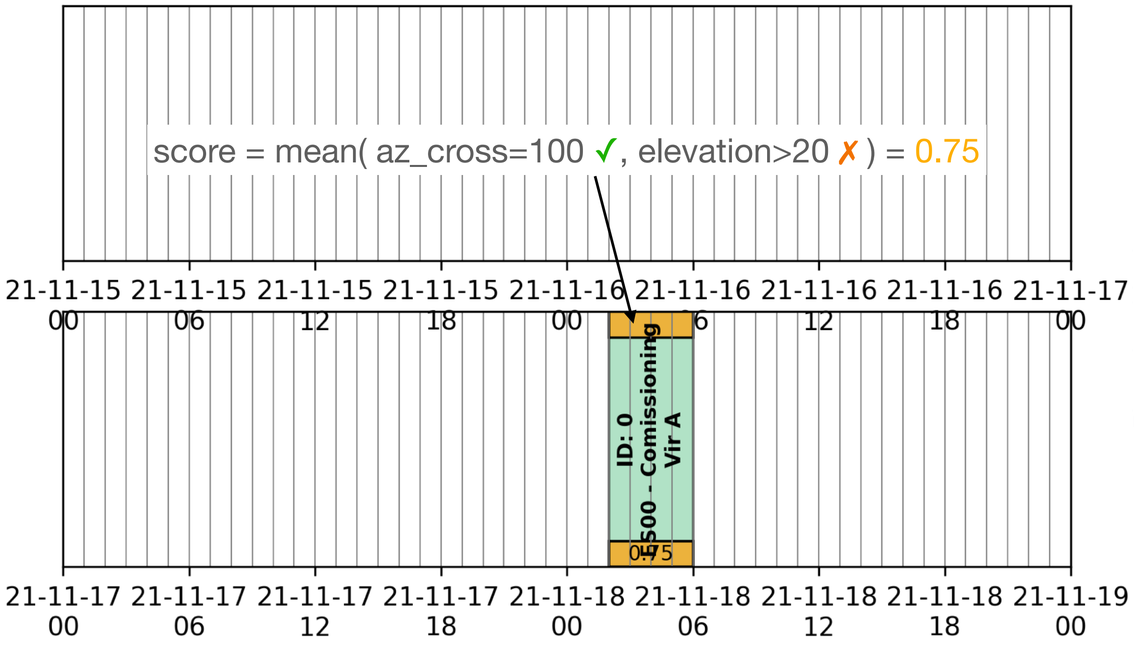
Booking visualization & checking

- Distribute the observations over the schedule

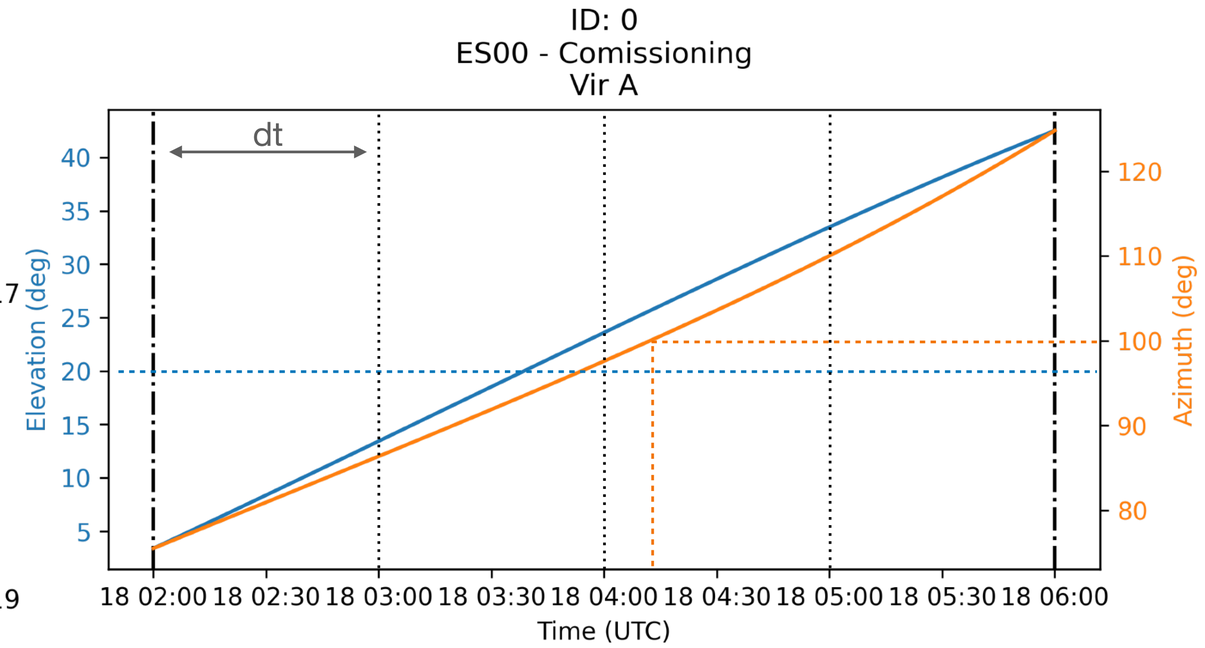
```
>>> schedule.book( optimize=False )     # Use deterministic algorithm to solve
```

- Plot the schedule, show the apparent source position during the booking time found:

```
>>> schedule.plot(days_per_line=2)
```




```
>>> schedule.observation_blocks[0].plot()
```

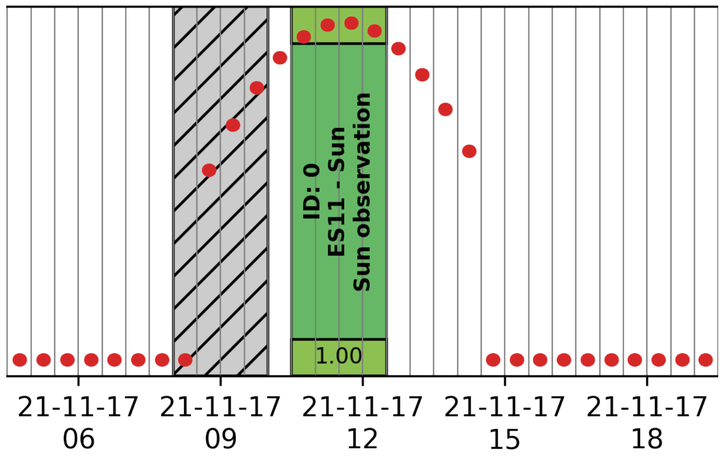


Observation scheduling

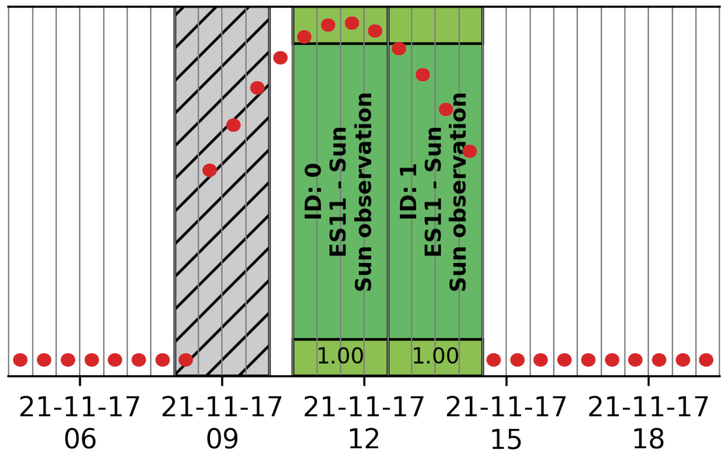
Limit of the deterministic algorithm...

```
>>> schedule = Schedule(time_min=Time('2021-11-17 00:00:00'), time_max=Time('2021-11-18 00:00:00'))
>>> schedule.insert( ReservedBlock(time_min=Time('2021-11-17 08:00:00'), time_max=Time('2021-11-17 10:00:00')))
>>> sun = ObsBlock( name='Sun observation', program='es11', target=SSTarget.fromName('Sun'),
        duration=TimeDelta(2*3600, format='sec'), constraints=Constraints( ElevationCnst(elevationMin=12) ) )
```

```
>>> schedule.insert(sun)
>>> schedule.book(optimize=False)
```

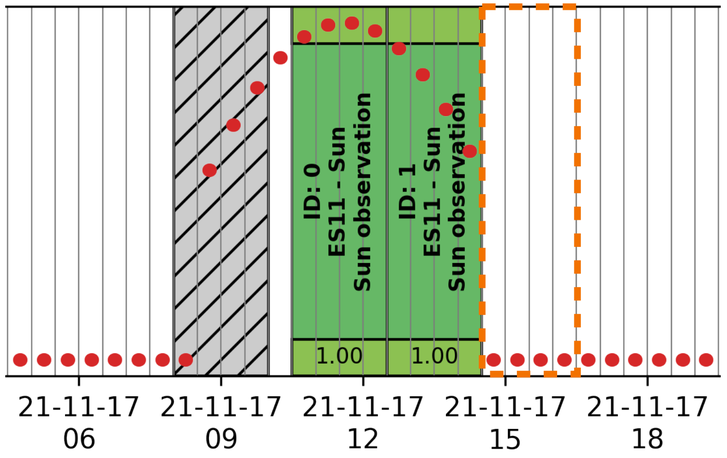



```
>>> schedule.insert(sun * 2)
>>> schedule.book(optimize=False)
```




```
>>> schedule.insert(sun * 3)
>>> schedule.book(optimize=False)
```

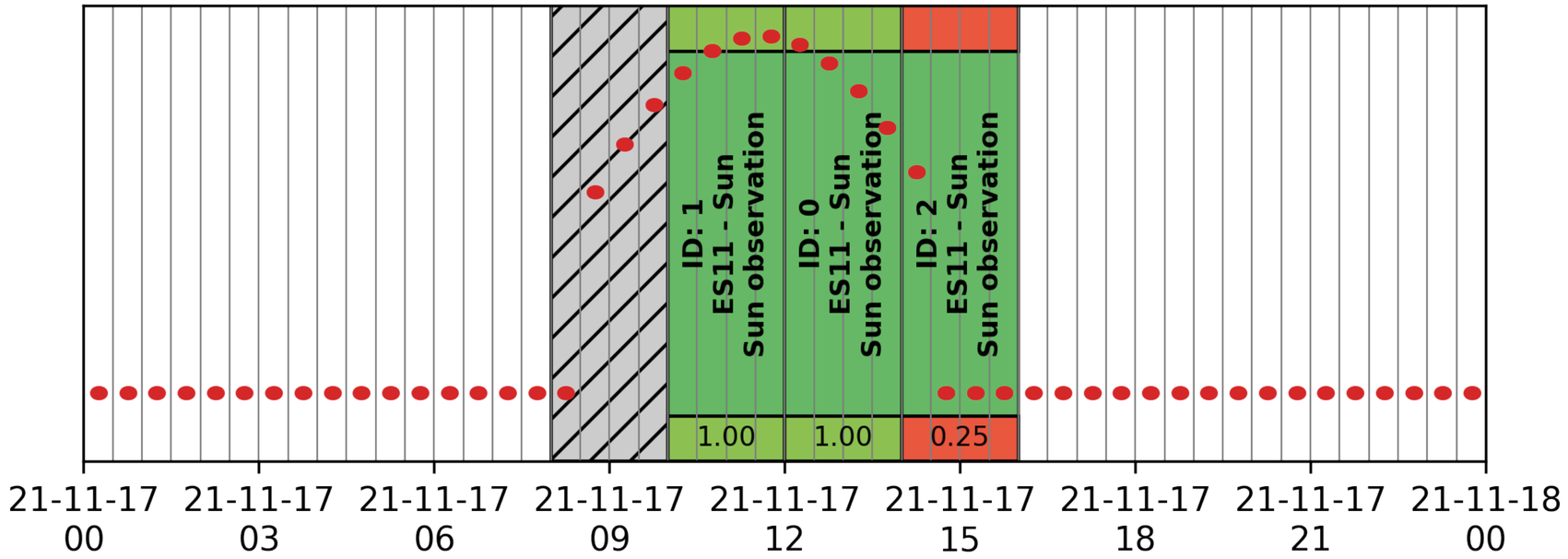
```
WARNING: <ObsBlock> #2 'Sun observation' cannot be scheduled.
INFO: 2/3 observation blocks scheduled (0 impossible to fit).
```



Observation scheduling

Genetic algorithm

```
>>> schedule = Schedule(time_min=Time('2021-11-17 00:00:00'), time_max=Time('2021-11-18 00:00:00'))
>>> schedule.insert( ReservedBlock(time_min=Time('2021-11-17 08:00:00'), time_max=Time('2021-11-17 10:00:00')))
>>> sun = ObsBlock( name='Sun observation', program='es11', target=SSTarget.fromName('Sun'),
        duration=TimeDelta(2*3600, format='sec'), constraints=Constraints( ElevationCnst(elevationMin=12) ) )

>>> schedule.insert(sun * 3)
>>> schedule.book( optimize=True )    # Use genetic algorithm to solve
```

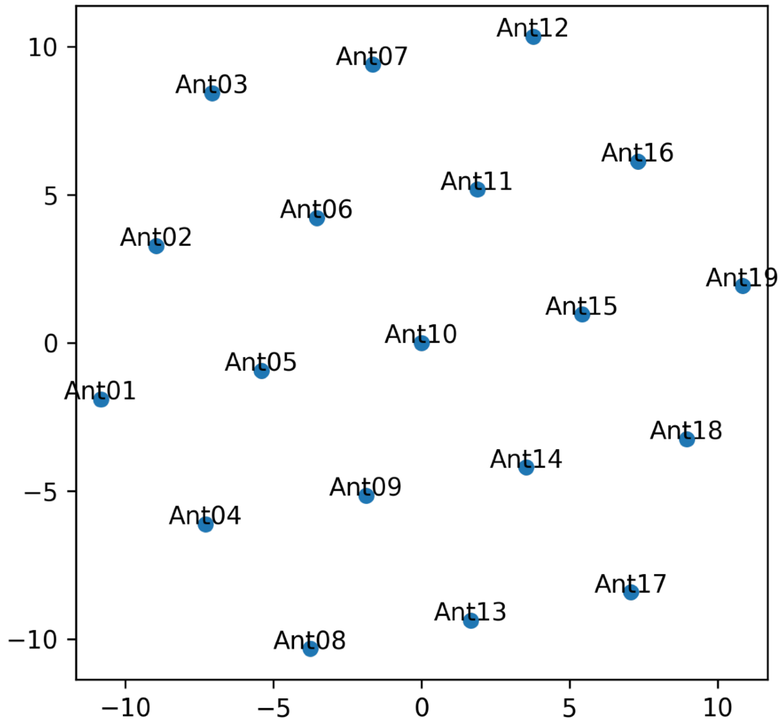


NenuFAR array configurations & properties

Array configuration

Mini-Array

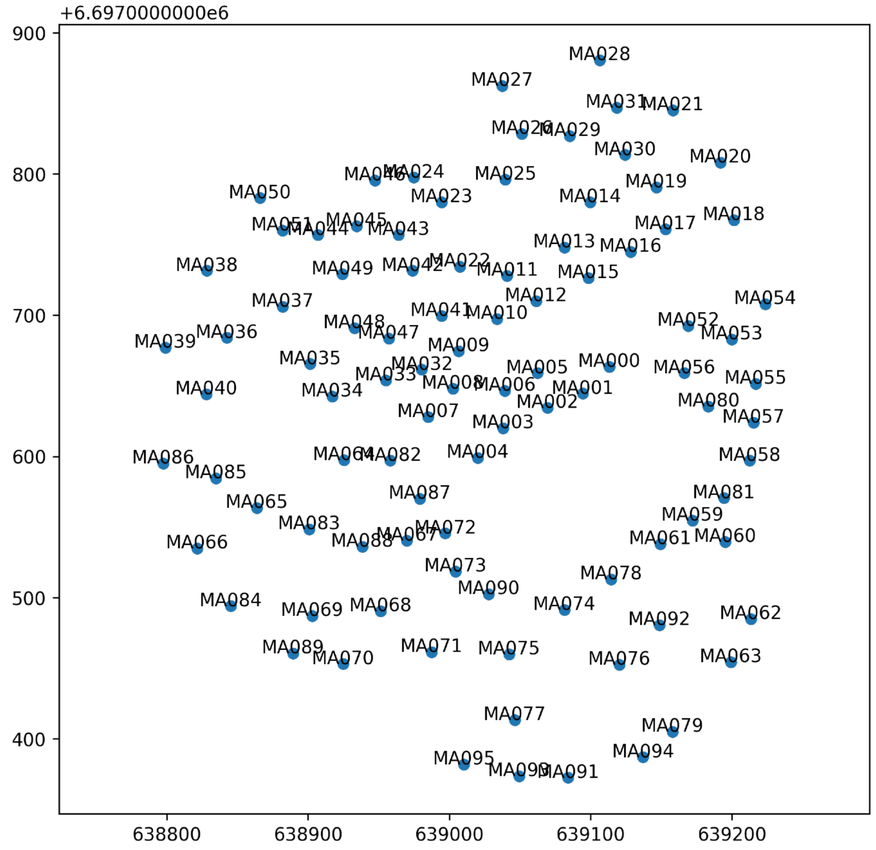
```
>>> from nenupy.instru import MiniArray

>>> ma = MiniArray(index=10)
>>> ma.rotation
290°
>>> ma.plot()
```



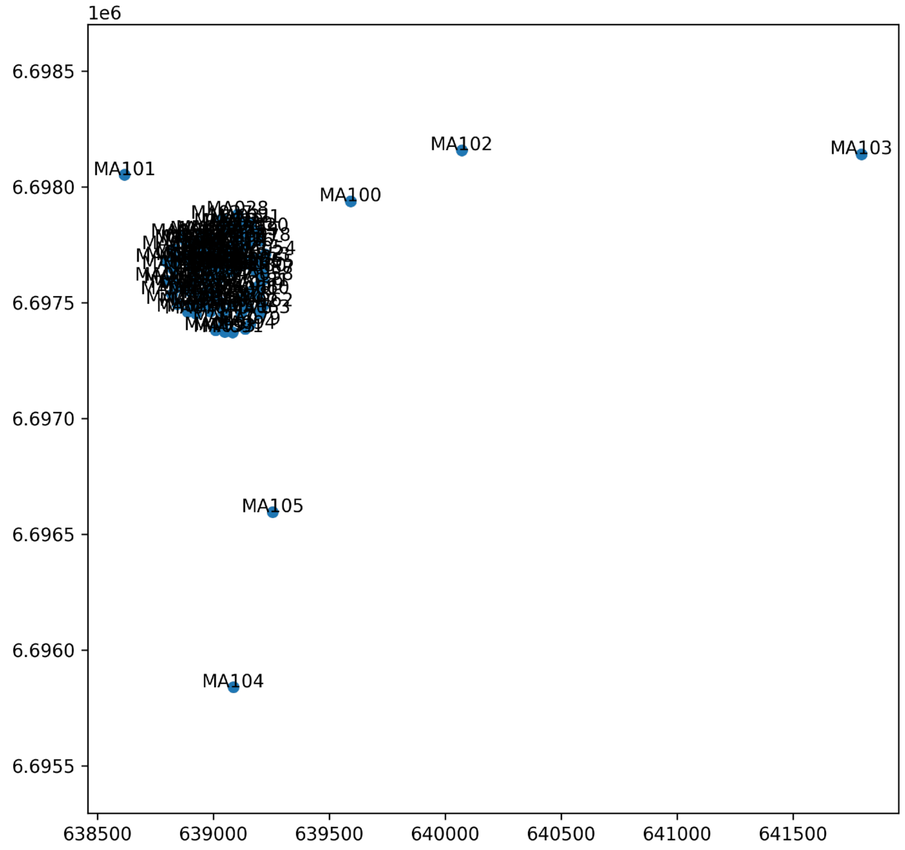
Array configuration

NenuFAR

```
>>> from nenupy.instru import NenuFAR

>>> nenufar_core = NenuFAR()
>>> nenufar_core.plot()
```



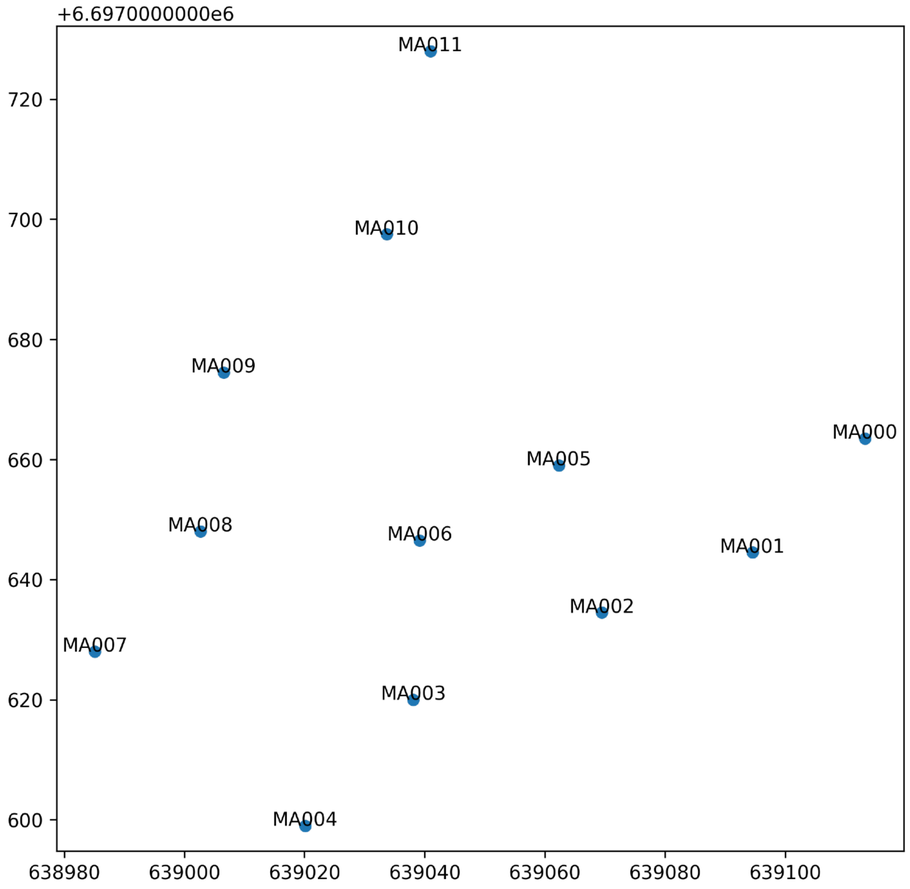

```
>>> from nenupy import NenuFAR

>>> nenufar_remote = NenuFAR(include_remote_mas=True)
>>> nenufar_remote.plot()
```



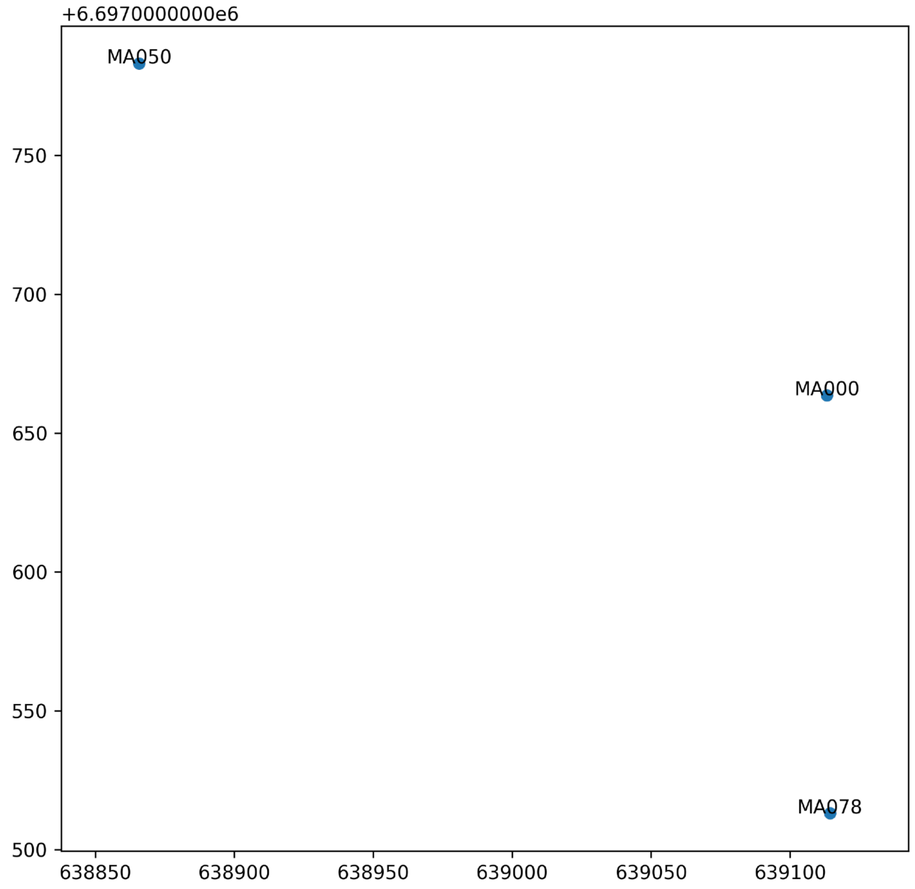
Array configuration

Sub-array

```
>>> nenufar_12mas = NenuFAR()[:12]
>>> nenufar_12mas.plot()
```




```
>>> nenufar_3ma = NenuFAR()["MA000", "MA050", "MA078"]
>>> nenufar_3ma.plot()
```



Telescope properties

Angular resolution

- Mini-Array

```
>>> ma = MiniArray()
>>> ma.angular_resolution(frequency=[15, 27, 48, 85]*u.MHz)
[52.051038, 28.917243, 16.265949, 9.1854773]°
```

- NenuFAR (subset of core Mini-Arrays)

```
>>> nenufar_subset = NenuFAR()[10:30]
>>> nenufar_core.angular_resolution(frequency=[15, 27, 48, 85]*u.MHz)
[5.0052596, 2.7806998, 1.5641436, 0.88328111]°
```

- NenuFAR core MAs

```
>>> nenufar_core = NenuFAR()
>>> nenufar_core.angular_resolution(frequency=[15, 27, 48, 85]*u.MHz)
[2.1118006, 1.1732225, 0.65993768, 0.37267069]°
```

- NenuFAR core + remote MAs

```
>>> nenufar_remote = NenuFAR(include_remote_mas=True)
>>> resolutions = nenufar_remote.angular_resolution(frequency=[15, 27, 48, 85]*u.MHz)
>>> resolutions.to(u.arcmin)
[19.312458, 10.729143, 6.0351432, 3.4080808]'
```

Telescope properties

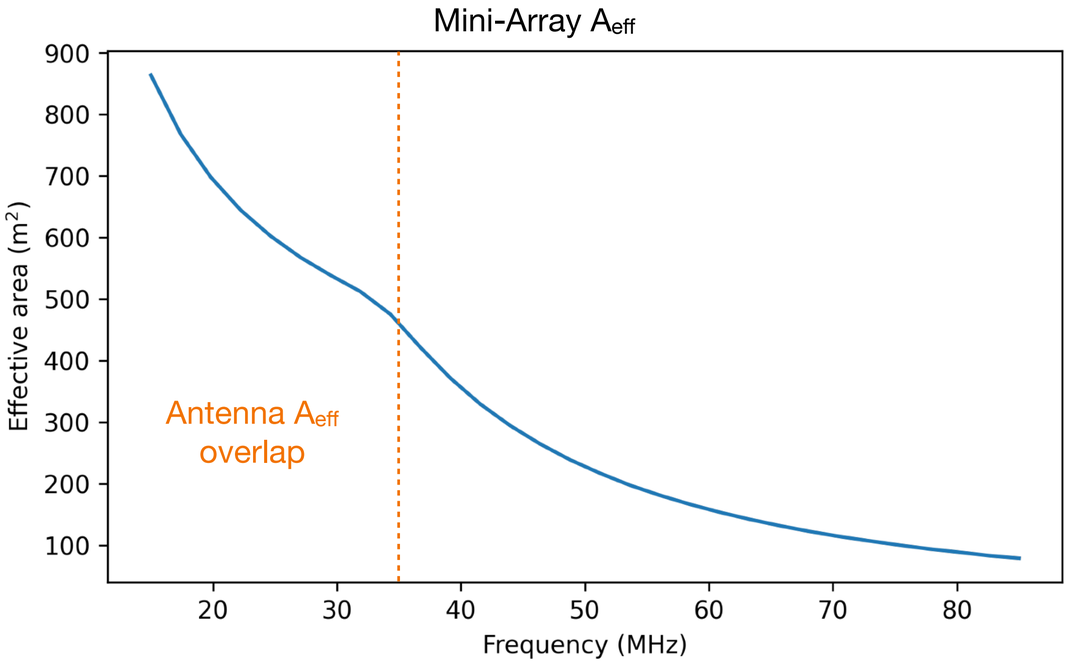
Effective area

- Antenna

```
>>> antenna = MiniArray()["Ant10"]
>>> antenna.effective_area(frequency=[15, 27, 48, 85]*u.MHz)
[133.10199, 41.117441, 13.009539, 4.1531393]m2
```

- Mini-Array

```
>>> ma = MiniArray()
>>> ma.effective_area(frequency=[15, 27, 48, 85]*u.MHz)
[863.77425, 568.4938, 247.12258, 78.99406]m2
```



- NenuFAR-core

```
>>> nenufar_core = NenuFAR()
>>> nenufar_core.effective_area(frequency=[15, 27, 48, 85]*u.MHz)
[82922.328, 54575.405, 23723.768, 7583.4298]m2
```

Telescope properties

Sensitivity

- Imaging mode: $\sigma_{\mathrm{im}} = \dfrac{S_{\mathrm{sys}}}{\sqrt{N_{\mathrm{ant}}(N_{\mathrm{ant}}-1)2\Delta\nu\Delta t}}$
- Beamforming mode: $\sigma_{\mathrm{bf}} = \dfrac{S_{\mathrm{sys}}}{\sqrt{2\Delta\nu\Delta t}}$

with $S_{\mathrm{sys}} = \xi\dfrac{2k_{\mathrm{B}}}{\eta A_{\mathrm{eff}}}T_{\mathrm{sys}}$

and $T_{\mathrm{sys}} = T_{\mathrm{inst}} + T_{\mathrm{sky}} + \left(\sum_{\mathrm{src}} T_{\mathrm{src}}\right)$

```
>>> from nenupy.instru.interferometer import ObservingMode
>>> ma = MiniArray()                          # Define an array, can also be NenuFAR()...
>>> ma.sensitivity(
>>>     frequency=50*u.MHz,
>>>     mode=ObservingMode.BEAMFORMING,       # ObservingMode.BEAMFORMING / ObservingMode.IMAGING
>>>     dt=3600*u.s,                          # Exposure Δt
>>>     df=25*u.MHz,                          # Bandwidth Δν
>>>     elevation=70*u.deg,                   # Mean elevation during the observation
>>>     efficiency=1.,                        # Effective area efficiency (η <= 1)
>>>     decoherence=1.,                       # Imperfect phasing (ξ >=1)
>>>     lna_filter=0                          # Filter (0='no filter', 3='25 MHz high-pass filter')
>>> )
0.19055667 Jy
```

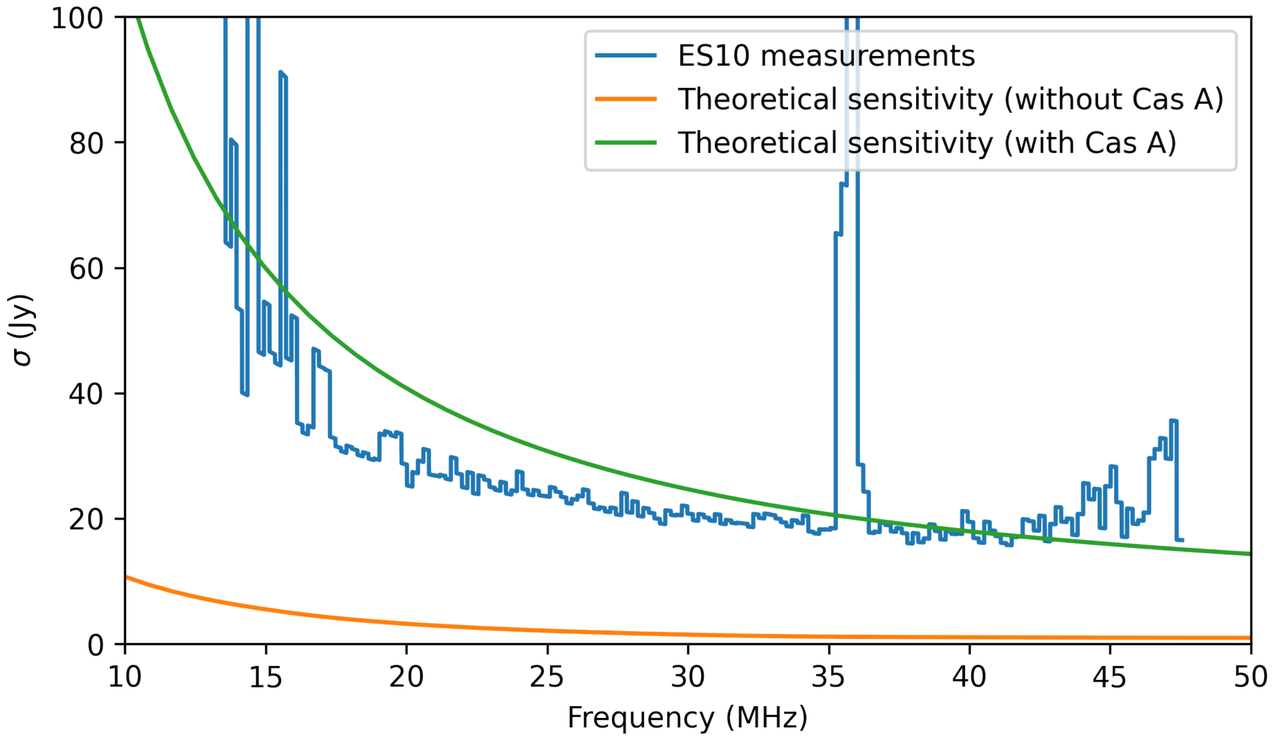
Telescope properties

Sensitivity, pointing towards a bright radio source

```
>>> def casa_spectrum(frequency):
>>>    """ Cas A spectrum """
>>>    a0 = 3.3584
>>>    a1 = -0.7518
>>>    a2 = -0.0347
>>>    a3 = -0.0705
>>>    log_nu=np.log10(frequency.to(u.GHz).value)
>>>    return 0.7 * np.power(10, (a0 + a1*log_nu +
a2*log_nu**2 + a3*log_nu**3) )*u.Jy
```

De Gasperin et al. (2020)

```
>>> nenufar = NenuFAR()[:56]
>>> rms = nenufar.sensitivity(
>>>     frequency=np.linspace(10, 50, 50)*u.MHz,
>>>     mode=ObservingMode.BEAMFORMING,
>>>     dt=1.953*u.h,
>>>     df=190.7*u.Hz,
>>>     elevation=60*u.deg,
>>>     efficiency=1,
>>>     decoherence=1.,
>>>     source_spectrum={"Cas A": casa_spectrum}
>>> )
```



Measurements: courtesy of ES10 'Radio recombination lines'

Simulations

Beam simulation

Principle

- Set up an array configuration
  - MiniArray, NenuFAR or a subset of these classes
- Define the instrumental pointing

```
>>> from nenupy.astro.pointing import Pointing
```

- Configure the desired output
  - *Sky* instance (either HEALPix or discrete sky positions)

```
>>> from nenupy.astro.sky import HpxSky, Sky
```

  - Set up the *time*, *frequency*, *polarization* and *coordinates* arrays

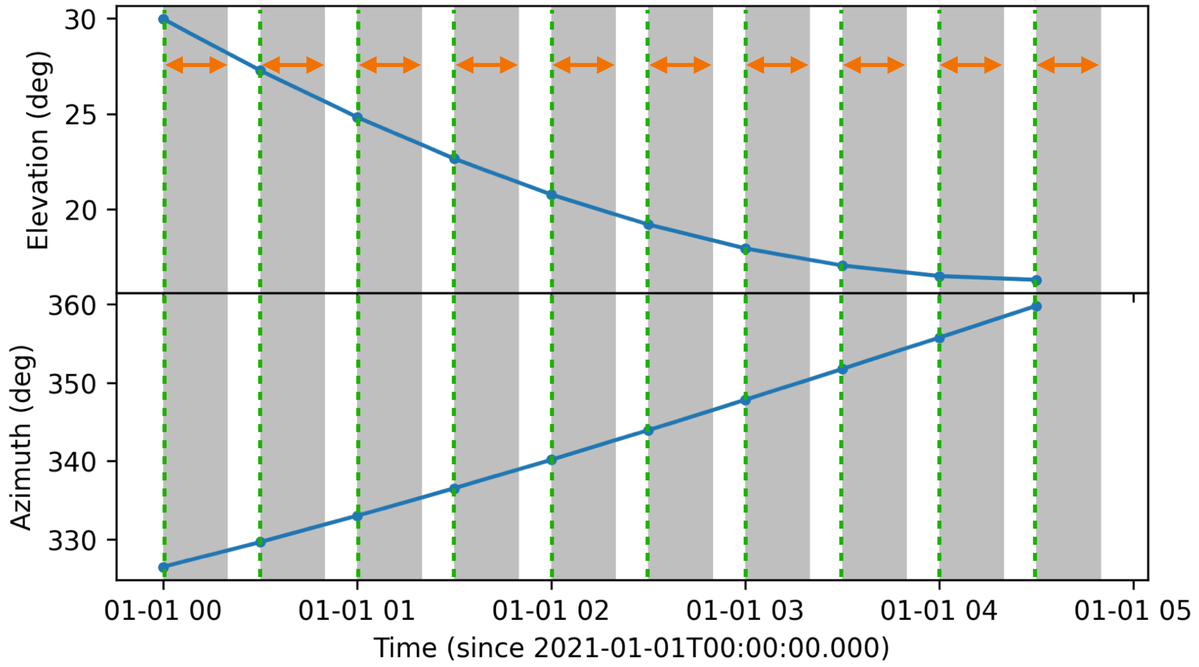
```
>>> simulation_times = Time("2021-01-01 12:00:00") + np.arange(12)*TimeDelta(1800, format="sec")
>>> whole_sky = HpxSky(
        resolution=0.5*u.deg,                       # HEALPix resolution (defines the coordinates array)
        frequency=np.array([25, 50, 75])*u.MHz,     # Frequency array (3 elements)
        polarization=Polarization.NW,               # Single polarization (from nenupy.instru import Polarization)
        time=simulation_times                       # Time array (12 elements)
    )
```

- Run the simulation

Pointing

Source tracking

```
>>> from nenupy.astro.target import FixedTarget
>>> from nenupy.astro.pointing import Pointing
>>> cas_a = FixedTarget.from_name("Cas A")
>>> pointing = Pointing.target_tracking( target=cas_a,
        time=Time("2021-01-01 00:00:00") + np.arange(10)*TimeDelta(1800, format="sec"),
        duration=TimeDelta(np.ones(10)*1200, format="sec") )
>>> pointing.plot(display_duration=True)
```

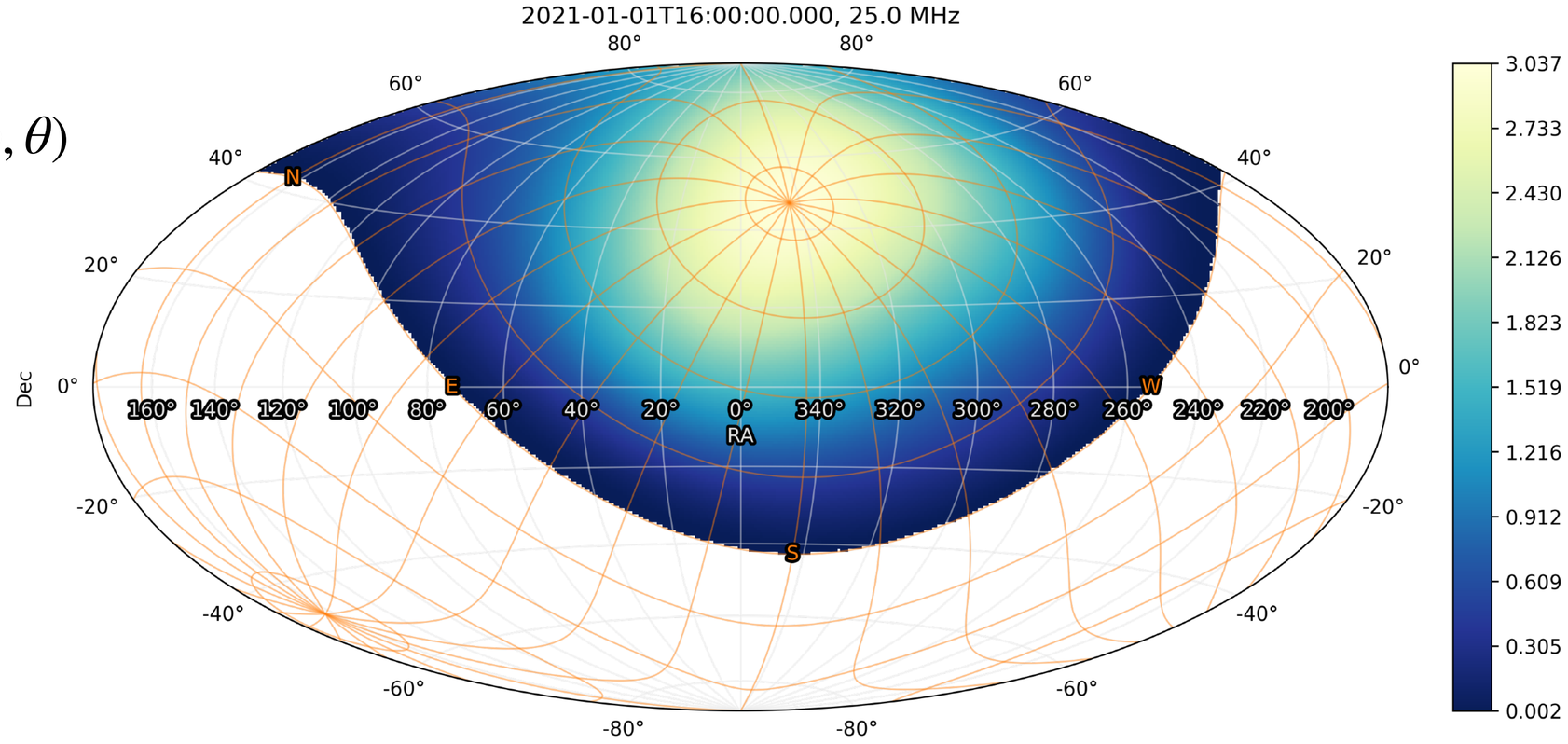


Beam simulation

Antenna radiation pattern

```
>>> antenna = MiniArray()["Ant10"]
>>> antenna_response = antenna.beam(sky=whole_sky, pointing=zenith)    # Pointing doesn't matter here
>>> antenna_response[8, 0, 0].plot(altaz_overlay=True)
```

$\mathcal{G}_{\text{ant}}(\nu, \phi, \theta)$



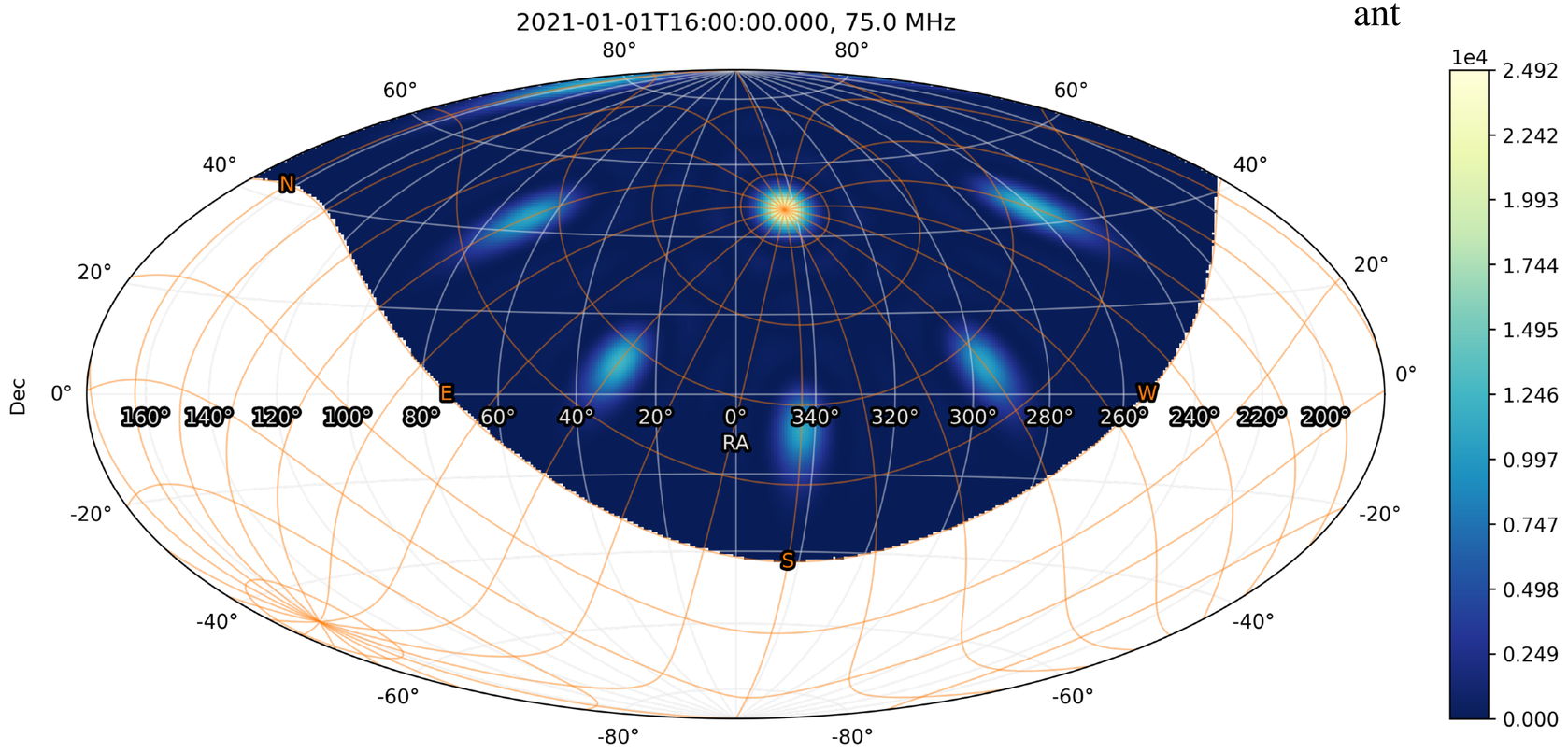
Beam simulation

Mini-Array response

```
>>> ma = MiniArray()
>>> beam = ma.beam(sky=whole_sky, pointing=zenith)
>>> beam[8, 2, 0].plot(altaz_overlay=True)
```

$$\mathcal{G}_{\mathrm{MA}}(\nu,\phi,\theta) = \mathcal{F}_{\mathrm{MA}}(\nu,\phi,\theta)\mathcal{G}_{\mathrm{ant}}(\nu,\phi,\theta)$$

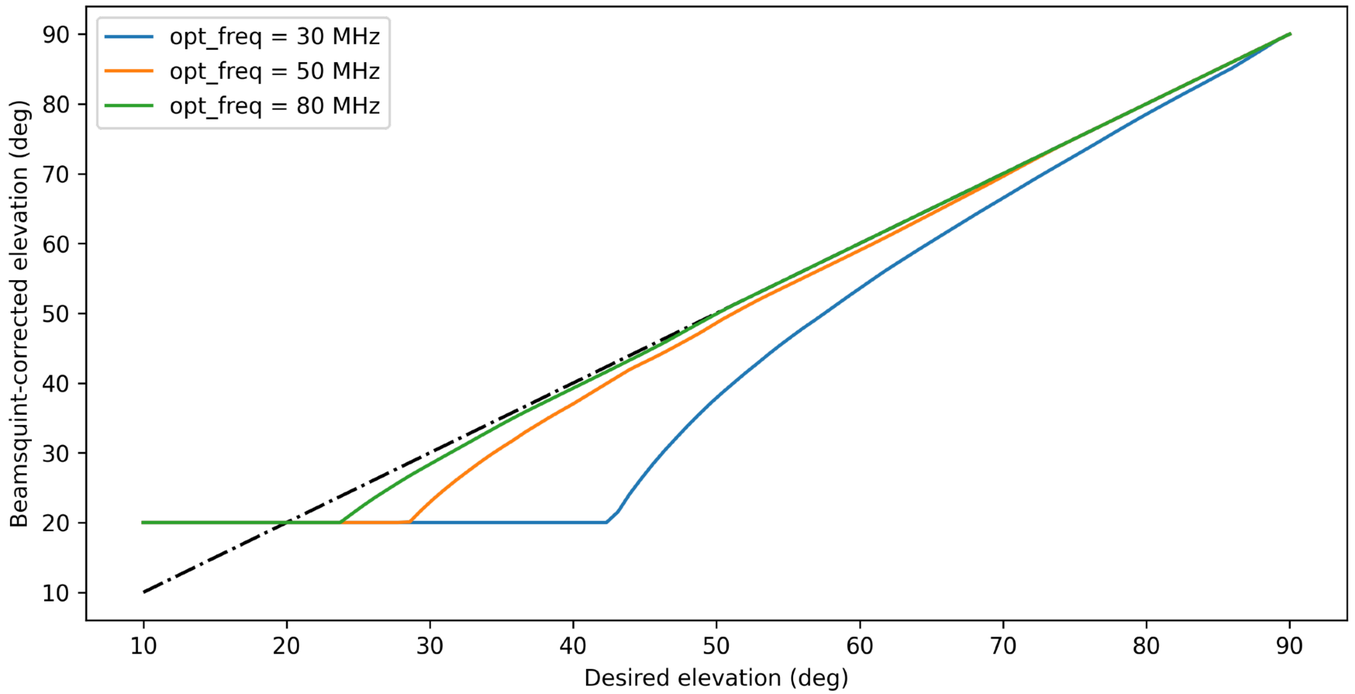
with $\mathcal{F}(\nu,\phi,\theta) = \sum_{\mathrm{ant}} e^{i\mathbf{k}(\nu,\phi,\theta)\cdot\mathbf{r}_{\mathrm{ant}}}$



Beam simulation

Mini-Array beamsquint

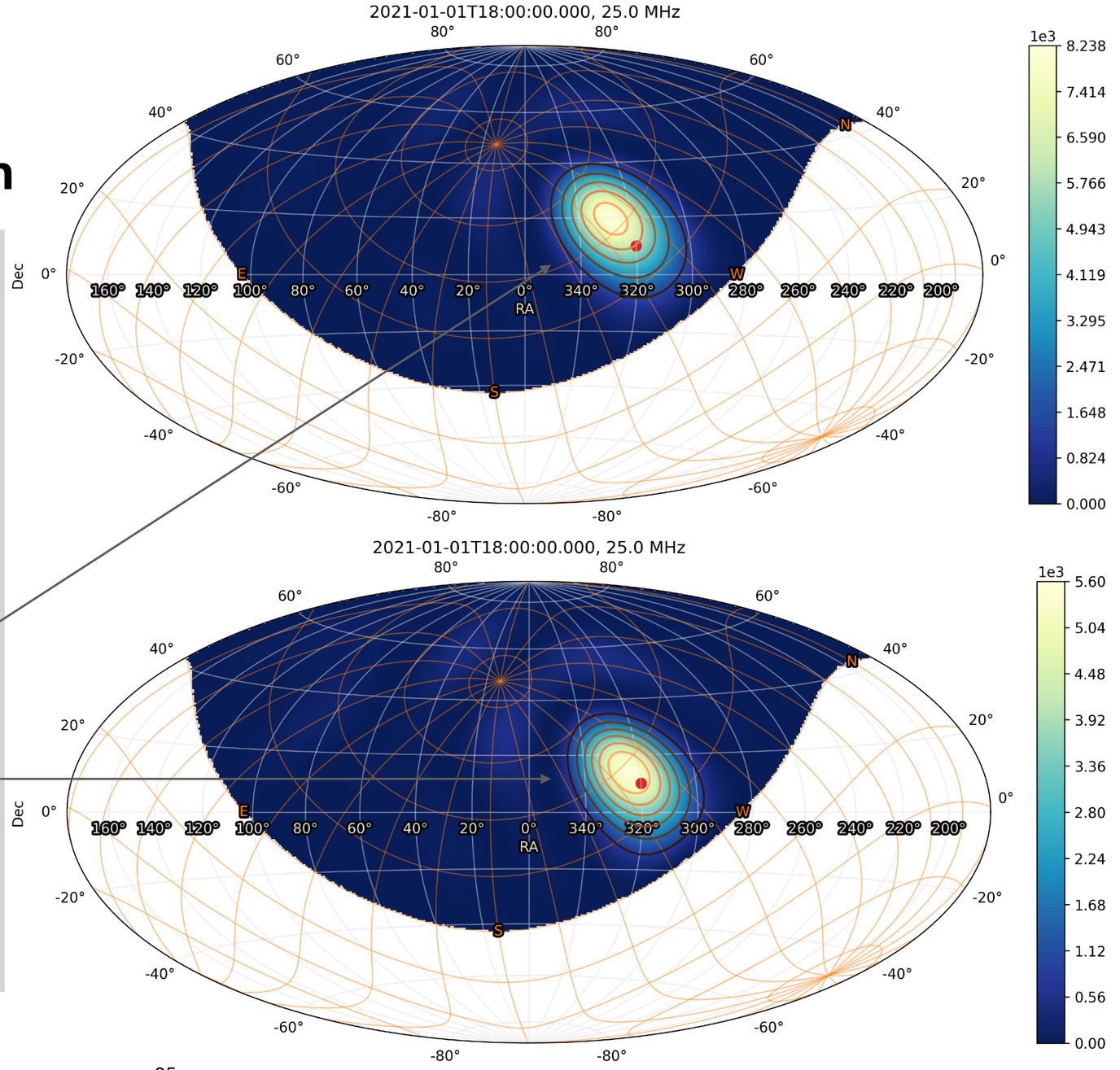
- Combination of:
  - Antenna radiation pattern (maximal at the zenith)
  - Array factor of the antenna distribution (beam towards the target direction)
  - → shift maximal sensitivity towards higher elevations



Beam simulation

Mini-Array beamsquint correction

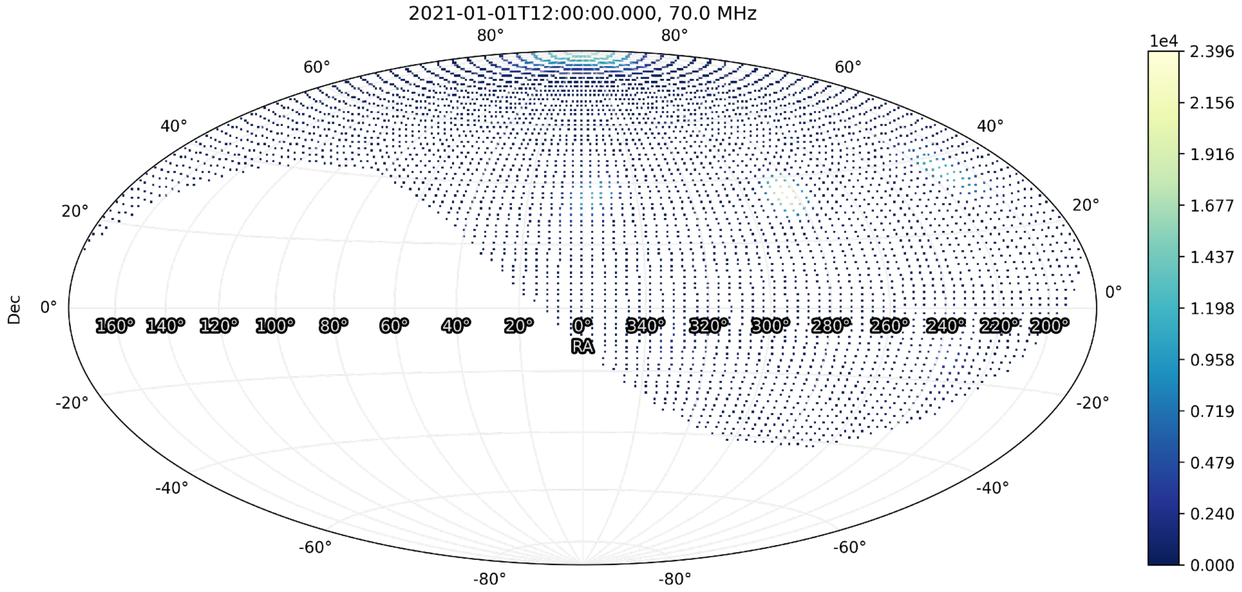
```
>>> # Source tracking
>>> source = SkyCoord(320, 10, unit="deg")
>>> source_tracking = Pointing.target_tracking(
        target=FixedTarget(coordinates=source),
        time=Time("2021-01-01 18:00:00")
    )
>>> # Output configuration
>>> whole_sky = HpxSky(
        resolution=0.5*u.deg,
        frequency=25*u.MHz,
        polarization=Polarization.NW,
        time=Time("2021-01-01 18:00:00")
    )
>>> # Array configuration
>>> ma = MiniArray()
>>> conf = NenuFAR_Configuration(
        beamsquint_correction=False, # or True
    )
>>> beam = ma.beam(
        sky=whole_sky,
        pointing=source_tracking,
        configuration=conf
    )
```



Beam simulation

'Discrete' sky simulation

```
>>> # North Celestial Pole tracking
>>> ncp = FixedTarget.from_name("North Celestial Pole")
>>> ncp_tracking = Pointing.target_tracking(
        target=ncp,
        time=obs_times,
        duration=dt
    )
>>> # Discrete sky grid
>>> ra, dec = np.meshgrid(
        np.linspace(0, 360, 100),
        np.linspace(-90, 90, 100)
    )
>>> sky = Sky(
        coordinates=SkyCoord(
            ra, dec, unit="deg"
        ).ravel(),    # needs to be 1D
        time=time_array,
        frequency=frequency_array,
        polarization=Polarization.NW
    )
>>> # Array definition
>>> ma = MiniArray()
>>> # Beam simulation
>>> beam = ma.beam(sky=sky, pointing=ncp_tracking)
```

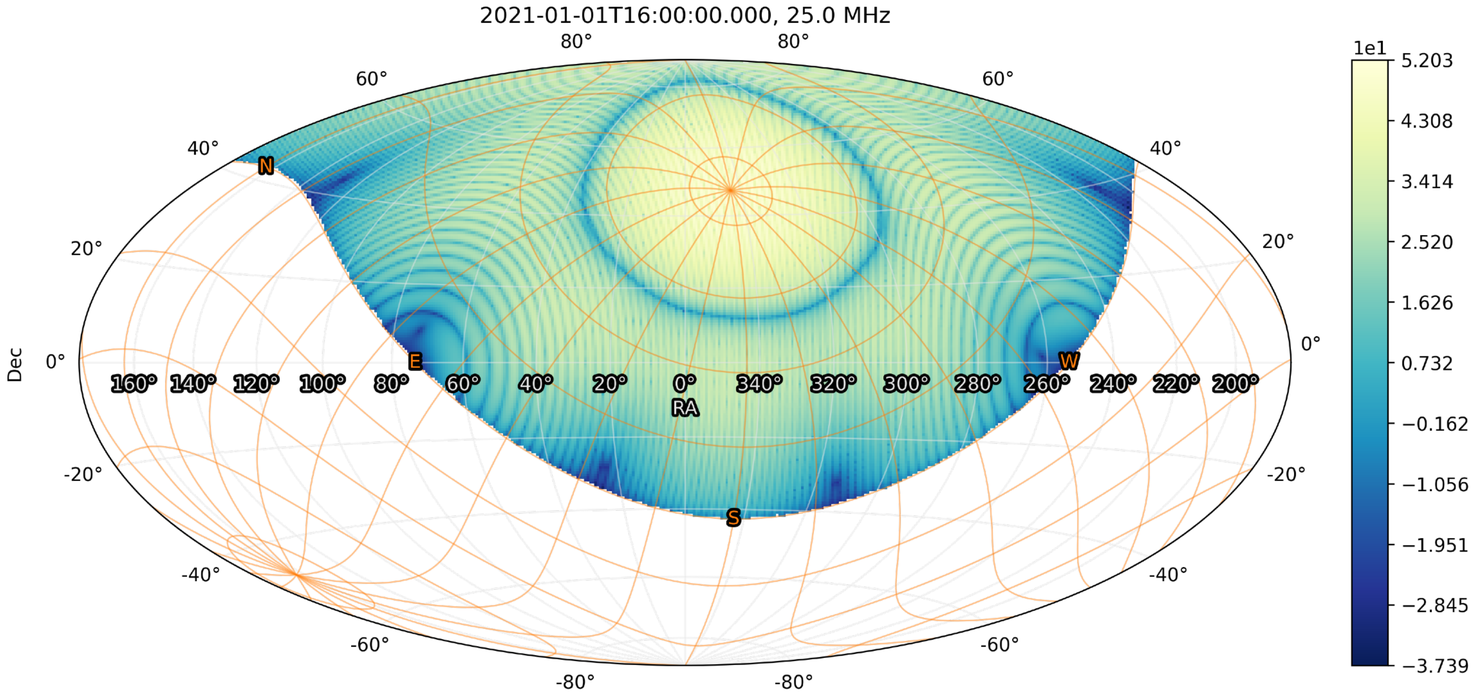


Beam simulation

NenuFAR (two Mini-Arrays)

```
>>> nenufar = NenuFAR()["MA040", "MA055"]    # East-West axis Mini-Arrays
>>> beam = nenufar.beam(sky=whole_sky, pointing=zenith)
>>> beam[8, 0, 0].plot(altaz_overlay=True, decibel=True)
```

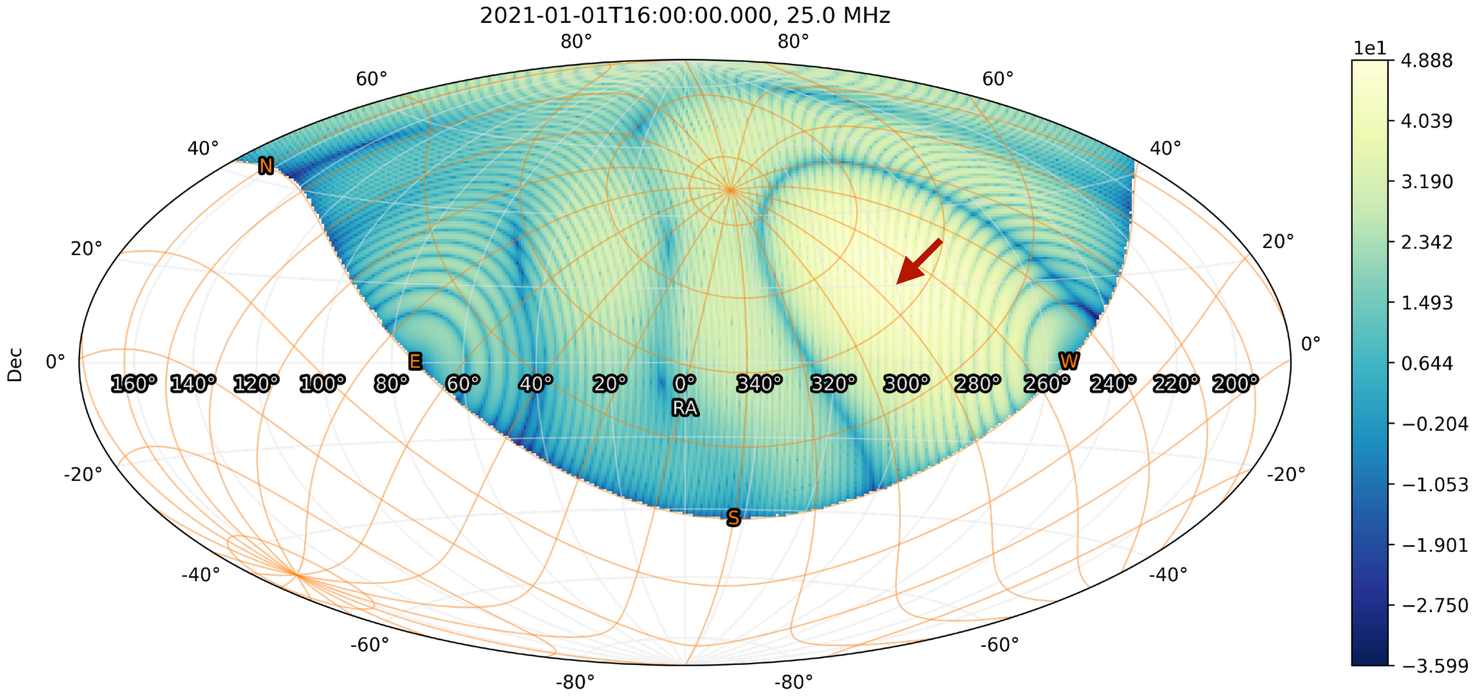
$$\mathcal{G}_{\text{NenuFAR}}(\nu,\phi,\theta) = \mathcal{F}_{\text{NenuFAR}}(\nu,\phi,\theta) \sum_{\text{MA}} \mathcal{G}_{\text{MA}}(\nu,\phi,\theta) \quad (\text{ with } \mathcal{F}(\nu,\phi,\theta) = \sum_{\text{ant}} e^{i\mathbf{k}(\nu,\phi,\theta)\cdot\mathbf{r}_{\text{ant}}} \ )$$



Beam simulation

NenuFAR different analog/numerical pointings

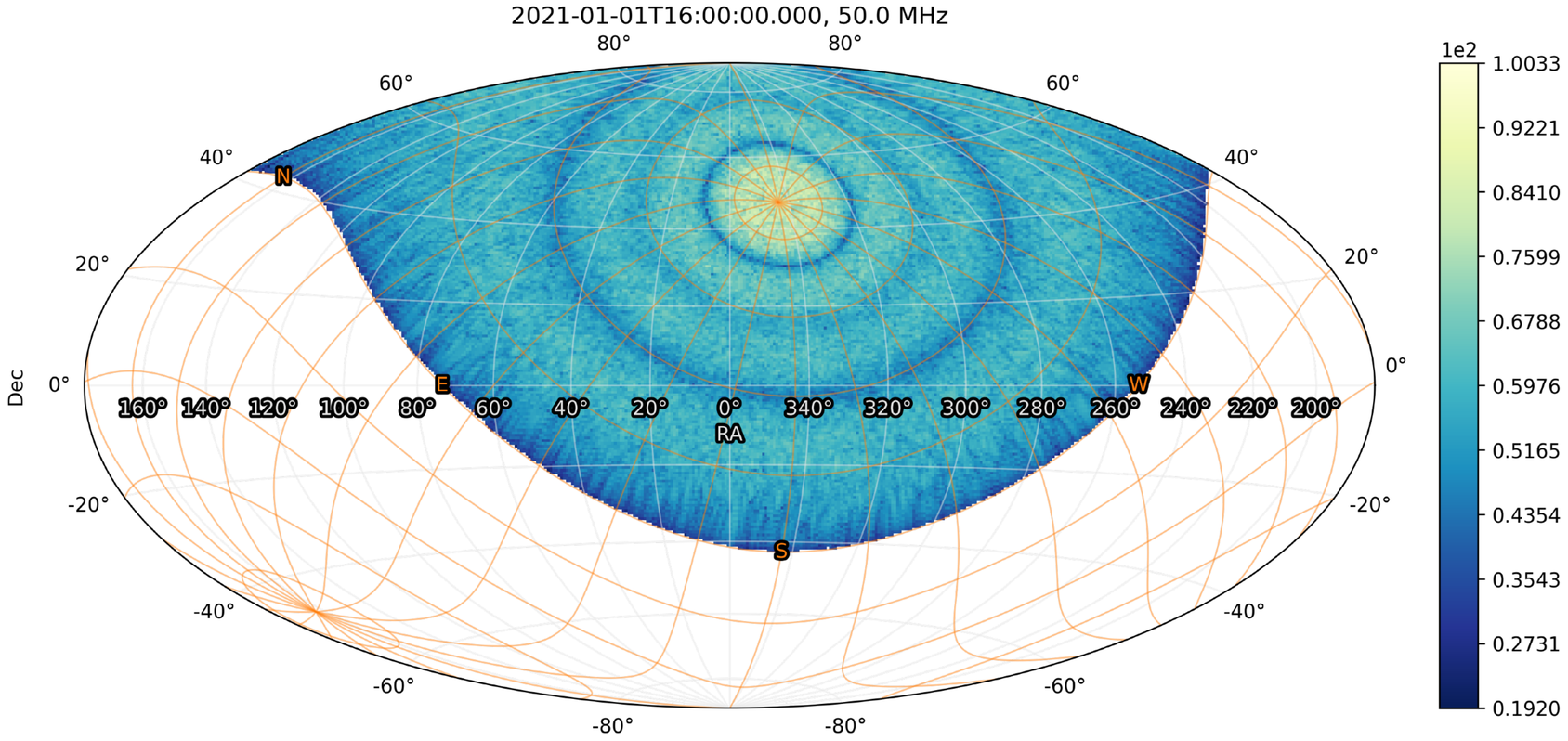
```
>>> nenufar = NenuFAR()["MA040", "MA055"]
>>> other_pointing = Pointing.target_tracking(
        target=FixedTarget(coordinates=SkyCoord(300, 20, unit="deg")),
        time=simulation_times,
        duration=simulation_dt
    )
>>> beam = nenufar.beam( sky=whole_sky, pointing=zenith, analog_pointing=other_pointing )
```



Beam simulation

Mini-Array response

```
>>> nenufar = NenuFAR()     # 96 Mini-Arrays
>>> beam = nenufar.beam(sky=whole_sky, pointing=zenith)
>>> beam[8, 1, 0].plot(altaz_overlay=True, decibel=True)
```



Observation simulation

Exercise: Time profile simulation of Vir A transit

- Search for the next Virgo A meridian transit time
- Set up a time range: from $t_{\mathrm{transit}} - 30\,\mathrm{min}$ to $t_{\mathrm{transit}} + 30\,\mathrm{min}$, 31 time steps with $\delta t = 2\,\mathrm{min}$
- Define two *Pointings*:
  - Analog pointing: Vir A tracking
  - Numerical pointing: transit pointing (fixed at the apparent Vir A meridian crossing time position)
- Prepare simulation output: HpxSky, time range, 50 MHz, NW polarization
- Simulate the beam of NenuFAR core Mini-Arrays
- Retrieve a corresponding skymodel
- Compute the time profile

Observation simulation

Exercise: Time profile simulation of Vir A transit

- Search for the next Virgo A meridian transit time

```
>>> from nenupy.astro.target import FixedTarget
>>> from astropy.time import Time

>>> vira = FixedTarget.from_name("Vir A")

>>> vira_transit = vira.next_meridian_transit(
        time=Time("2021-11-17 16:00:00")
    )

>>> vira_transit
<Time object: scale='utc' format='iso' value=2021-11-18 08:32:42.199>
```

Observation simulation

Exercise: Time profile simulation of Vir A transit

- Set up a time range: from $t_{\mathrm{transit}} - 30\,\mathrm{min}$ to $t_{\mathrm{transit}} + 30\,\mathrm{min}$, 31 time steps with $\delta t = 2\,\mathrm{min}$

```
>>> from astropy.time import TimeDelta
>>> import numpy as np

>>> dt = TimeDelta(2*60, format="sec")

>>> times = vira_transit - TimeDelta(1800, format="sec") + np.arange(31)*dt

>>> times.isot
array(['2021-11-18T08:02:42.199', '2021-11-18T08:04:42.199', '2021-11-18T08:06:42.199', '2021-11-18T08:08:42.199',
       '2021-11-18T08:10:42.199', '2021-11-18T08:12:42.199', '2021-11-18T08:14:42.199', '2021-11-18T08:16:42.199',
       '2021-11-18T08:18:42.199', '2021-11-18T08:20:42.199', '2021-11-18T08:22:42.199', '2021-11-18T08:24:42.199',
       '2021-11-18T08:26:42.199', '2021-11-18T08:28:42.199', '2021-11-18T08:30:42.199', '2021-11-18T08:32:42.199',
       '2021-11-18T08:34:42.199', '2021-11-18T08:36:42.199', '2021-11-18T08:38:42.199', '2021-11-18T08:40:42.199',
       '2021-11-18T08:42:42.199', '2021-11-18T08:44:42.199', '2021-11-18T08:46:42.199', '2021-11-18T08:48:42.199',
       '2021-11-18T08:50:42.199', '2021-11-18T08:52:42.199', '2021-11-18T08:54:42.199', '2021-11-18T08:56:42.199',
       '2021-11-18T08:58:42.199', '2021-11-18T09:00:42.199', '2021-11-18T09:02:42.199'], dtype='<U23')
```

Observation simulation

Exercise: Time profile simulation of Vir A transit

- Define two *Pointings*:
  - Analog pointing: Vir A tracking

```
>>> from nenupy.astro.pointing import Pointing

>>> analog_pointing = Pointing.target_tracking(
        target=vira,    # FixedTarget previously define
        time=times,     # Time range (31 elements)
        duration=dt,    # 2min for each of the 31 sky positions
    )
```

  - Numerical pointing: transit pointing (fixed at the apparent Vir A meridian crossing time position)

```
>>> import astropy.units as u

>>> vira_transit_pointing = Pointing.target_transit(
        target=vira,                                # FixedTarget previously defined
        t_min=Time("2021-11-17 16:00:00"),          # Start time while searching for the next transit
        duration=TimeDelta(7200, format="sec"),     # Frozen pointing for 2 hours, centered on transit time
        azimuth=180*u.deg                           # az=180° → meridian transit
    )
```

Observation simulation

Exercise: Time profile simulation of Vir A transit

- Prepare simulation output: HpxSky, time range, 50 MHz, NW polarization

```
>>> from nenupy.astro.sky import HpxSky
>>> from nenupy.instru import Polarization

>>> whole_sky = HpxSky(
        resolution=0.5*u.deg,
        frequency=50*u.MHz,
        polarization=Polarization.NW,
        time=times
    )

>>> print(whole_sky)
<class 'nenupy.astro.sky.HpxSky'> instance
value: (31, 1, 1, 196608)
  * time: (31,)
  * frequency: (1,)
  * polarization: (1,)
  * coordinates: (196608,)
```

Observation simulation

Exercise: Time profile simulation of Vir A transit

- Simulate the beam of NenuFAR core Mini-Arrays

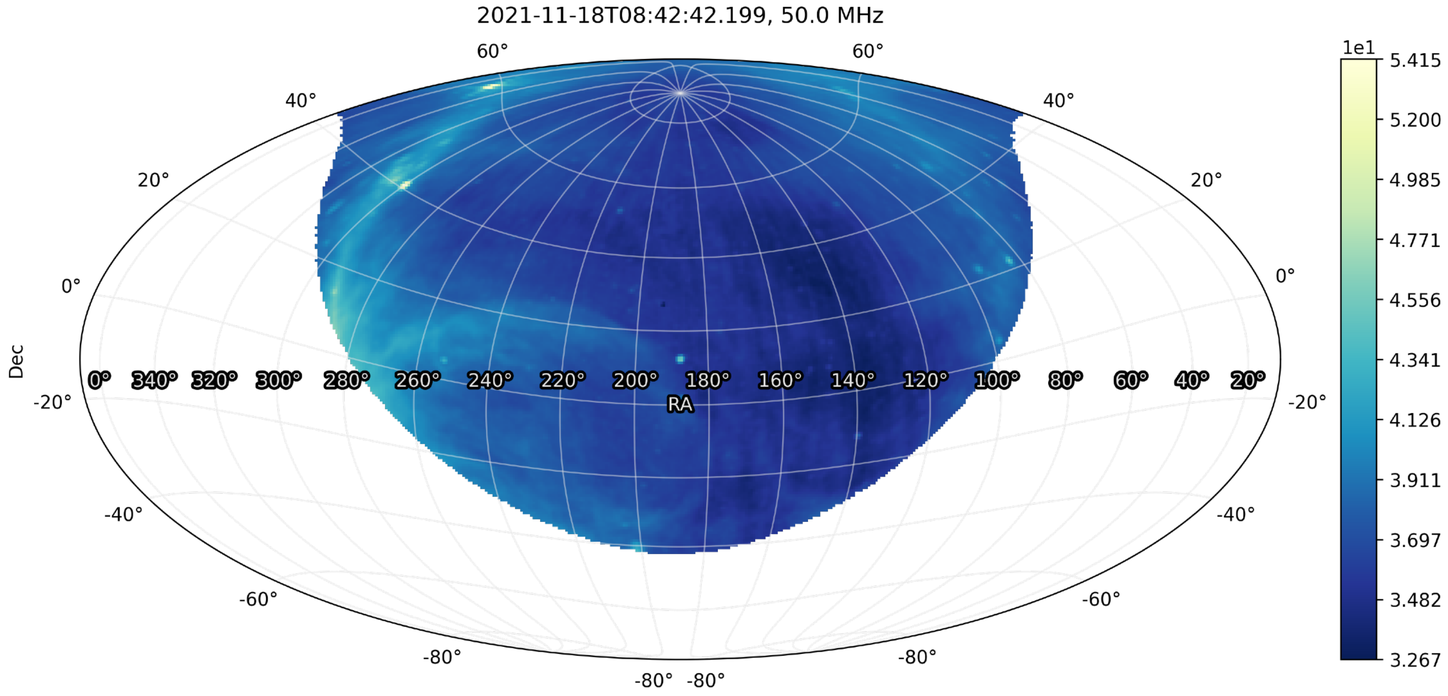
```
>>> from nenupy.instru import NenuFAR

>>> nenufar = NenuFAR()
>>> beam = nenufar.beam(
        sky=whole_sky,
        pointing=vira_transit_pointing,
        analog_pointing=analog_pointing
    )    # ~30 seconds on my computer

>>> print(beam)
<class 'nenupy.astro.sky.HpxSky'> instance
value: (31, 1, 1, 196608)
  * time: (31,)
  * frequency: (1,)
  * polarization: (1,)
  * coordinates: (196608,)
```

Observation simulation

Exercise: Time profile simulation of Vir A transit

- Retrieve a corresponding skymodel

```
>>> from nenupy.astro.skymodel import HpxGSM

>>> skymodel = HpxGSM.shaped_like(beam)

>>> skymodel[20, 0, 0].plot( decibel=True, center=vira.coordinates )
```

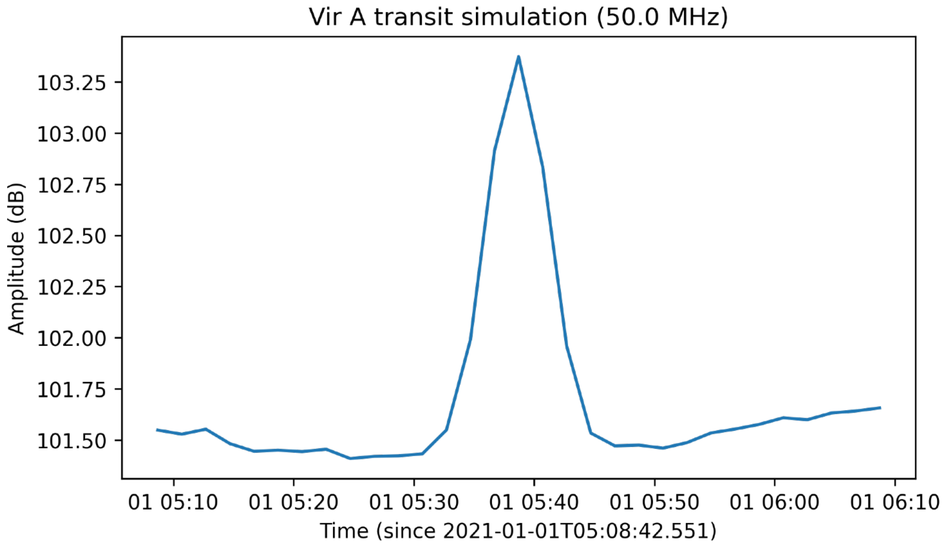


Observation simulation

Exercise: Time profile simulation of Vir A transit

- Compute the time profile

```
>>> import matplotlib.pyplot as plt

>>> sky_through_beam = beam * skymodel
>>> profile = np.nanmean(sky_through_beam.value, axis=(1, 2, 3))

>>> plt.plot(times.datetime, 10*np.log10(profile))
>>> plt.ylabel("Amplitude (dB)")
>>> plt.xlabel(f"Time (since {times[0].isot})")
>>> plt.title(f"Vir A transit simulation ({beam.frequency[0]})")
```



Thanks