DOC QUBIC master

March 26, 2018

This document is an attempt to explain what QUBIC master (author: Pierre Chanial) contains. The aim is for users to be able to modify it.

1 Instrument

The instrument is described in *instrument.py*, the code contains two classes *QubicMultibandInstrument* and *QubicInstrument*. We will first describe *QubicInstrument*. The input of the code are the following

1. calibration : class containing detector position, array of horn, optic properties, primary beam
2. noise properties : $f_{\rm knee}$, $f_{\rm slope}$, $n_{\rm det}^{corr}$, $\mathrm{NEP}_{\rm det}$, $N_{\rm grid}$
3. detector time constant: τ
4. frequency of observation and bandwidth: ν, $\frac{\delta\nu}{\nu}$
5. polarizer: choose to put or not the polarizer grid in the optics set up
6. beam: specification about the primary beam, the secondary beam and the synthetized beam

1.1 Calibration

One input when you run the instrument class is thus the calibration class, it is coded in *calibration.py*. This class read four fits files in *calfiles*

1. *CalQubic_DetArray_v* $*$ *.fits*
2. *CalQubic_HornArray_v* $*$ *.fits*
3. *CalQubic_Optics_v*$*$
4. *CalQubic_PrimBeam_v* $*$ *.fits*

We will write a small script for each of this.

1.1.1 Detector array

The detector layout is created by reading the latest version of *CalQubic_DetArray_v* $*$ *.fits*, we will change it in the futur to the one specify in the global.dict file. The fits file contains the following information.

1. the version of the file
2. the coordinates of the corner (vertex) of each detector, this is specified as a $(\mathrm{id}_{\rm det}^{\rm row}, \mathrm{id}_{\rm det}^{\rm column}, 4, 2)$ array. For the nominal instrument there are 32×32 detectors, which are the values $\mathrm{id}_{\rm det}^{\rm row}$, $\mathrm{id}_{\rm det}^{\rm column}$ can run on. Of course there are four corners, and each corner has two coordinates. The coordinates are given in the order: x and y with units: m.

3. removed: this is a useful table allowing us to remove some detectors, if 1 the detector is removed, 0 otherwise.

4. the detector index

5. the detector quadrant, the detectors are organised as quadrant the four quadrant index are : 0,1,2,3

6. the detector efficiency, set to 0.8 for all the detectors

This info is first read in *instrument.py*, it is then send to the Layout class. The Layout class can be found in the package *pysimulators*, in *packedtables/layouts.py*. The detector layout contain all information in the detector fits files, also including the center of the detector location.

The layout also contain information on detectors angle, using the focal length of the telescope in the following way. It modify the coordinates array (vertex) which become a $(\mathrm{id}_{\mathrm{det}}^{\mathrm{row}}, \mathrm{id}_{\mathrm{det}}^{\mathrm{column}}, 4, 3)$ array, the new coordinates is the focal lengh (0.3m specified in Optics) of the instrument. It then compute the angle of the detector on the sky in the following way

$$\theta = \arctan(\sqrt{x^2 + y^2}/f) \quad (1)$$
$$\phi = \arctan(y/x) \quad (2)$$

Where x and y are the coordinates of the center of the detector. Note that f the focal is chosen to be oriented -0.3m.

1.1.2 Horns array

The horn layout is created by reading the latest version of *CalQubic_HornArray_v* $*$ *.fits*, we will change it in the futur to the one specify in the global.dict file. The fits file contains the following information.

1. the version of the file

2. the spacing between the different horn arranged on a regular grid, the fits file contain the value 0.014 m

3. xreflection and yreflection, possible reflection around the x and y axis, set to False in the code

4. angle counter-clockwise rotation angle in degrees (45).

5. the horn radius (0.0066 m)

This info is first read in *instrument.py*, it is then send to the Layout class.

1.1.3 Optics and Prim Beam

The primary beam file *CalQubic_PrimBeam_v* $*$ *.fits* contains the information on the primary beam FWHM, for the instrument it is 13°. The optic file *CalQubic_Optics_v*$*$ contains information about the different optical elements, namely their Temperature (K), their transmission, their emissivity and the number of polarisation state. Their are 11 optical elements listed in the file, two at 250 K (winB1 and Block2), two at 100 K (12cmEd Block3), five at 6K (8cmEd,HWP,PolGr,Ba2Ba,Combin) and two at 0.1 K (7cmEd,BandFi).

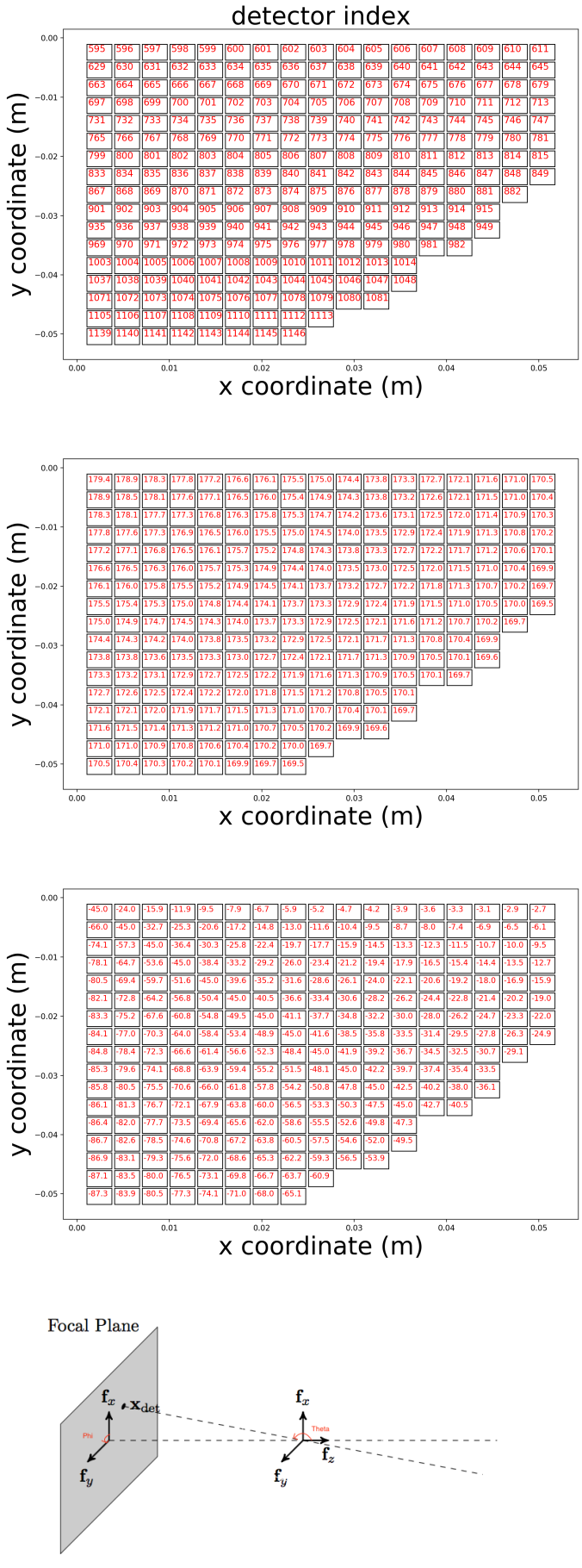


Figure 1: top: detector index, middle top: θ, middle bottom: ϕ, bottom: focal plane angle .

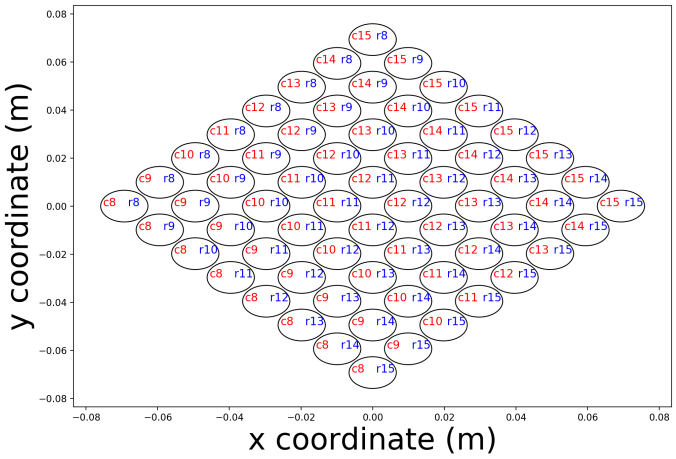


Figure 2: array of horn with row and column displayed

1.2 Noise properties

Now let's discuss a bit the way the code handles the noise. Once you have specified an instrument (see previous subsection), you can simulate the noise seen by the different detectors. For this you have to specify a sampling (pointing) and a scene. For now on we will assume random pointing, the actual pointing will be described in details in the pointing section. The number of detectors for the TD is 248, two source of noise affect the detector:

1. The detector noise
2. The photon noise

In the following we will describe how the code handle each of these items.

1.2.1 Detector noise

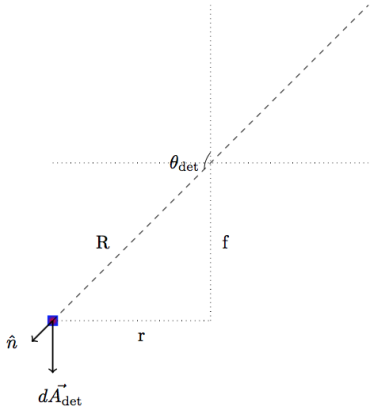
The routine adding detector noise *get_noise_detector* is called in *instrument.py*, it actually refer to the routine *get_noise* coded in *pysimulator* The routine can use a PSD (power spectrum density) and the bandwidth of the instrument to generate a noise realisation. Another usage of the routine is to specify the properties of the noise. The following properties should be used: $f_{\rm knee}$, $f_{\rm slope}$, $\rm NEP_{det}$. Remember: Noise Equivalent Power can be interpreted as the input signal power that produces a signal-to-noise ratio of unity at the output of a given detector at a given data-signaling rate or modulation frequency, and effective noise bandwidth; it is the minimum detectable power per square root bandwidth.

1. If $f_{\rm knee} = 0$, the code generate white noise only, it first computes $\sigma = \frac{\rm NEP_{det}}{\sqrt{2T_s}}$, given that the NEP is in unit $W/\sqrt{\rm Hz}$, σ is in Hz. T_s is the sampling rate, set to 1 s in the code. The noise is added in time-space.
2. if $f_{\rm knee} \neq 0$, the noise would be added in frequency space. The psd is computed as $P(f) = \sigma^2(1 + |f/f_{\rm knee}|^{f_{\rm slope}})/f_s$, then $n^{\rm det}(t) = \mathcal{F}^{-1}[\mathcal{F}[\mathcal{G}^{\rm det}(0,1)(t)]\sqrt{P(f)}]\sqrt{f_s}$

The noise is generated for each sample and each detector and is return as a $(n_{\rm det}, n_{\rm sample})$ array

1.2.2 Photon noise

The photon noise represent the contribution from the loading due to the atmosphere and optic temperature. This is where the scene routine come into play. The code takes as input the atmosphere temperature $T_{\rm atm} = 200K$, the atmosphere emissivity $\epsilon_{\rm atm} = 0.015$ and the atmosphere transmission $\eta_{\rm atm} = 1$, the temperature of the CMB, $T_{CMB} = 2.7255K$ as well as the temperature, emissivity,



$$
\begin{aligned}
d\Omega &= \frac{d\vec{A}_{\rm det}.\hat{n}}{R^2} \\
d\vec{S}.\hat{n} &= dA_{\rm det}\cos(\pi-\theta_{\rm det}) = -dA_{\rm det}\cos(\theta_{\rm det}) \\
R &= f/\cos(\pi-\theta_{\rm det}) \\
d\Omega &= -\frac{dA_{\rm det}}{f^2}\cos^3(\theta_{\rm det}) \\
\Omega &= -\frac{A_{\rm det}}{f^2}\cos^3(\theta_{\rm det})
\end{aligned}
$$

Figure 3: solid angle seen by a detector on the focal plane

transmission, and polarisation states of all optical elements $T_i, \epsilon_i, \eta_i, p_i = (1, 2)$. The emissivity, transmission and polarisation state of the CMB are set to 1. In the default setting of the code, there are 13 components total: (CMB, Atm +11 optics components). Then the code compute an effective transmission for each of the component. Basically the idea is to compute the transmission of a given component on the detector array, for the CMB this transmission is given by $\eta_{\rm eff}^{\rm CMB} = \prod_{i=1}^{11}\eta_i$, the last component transmission between set to 1. Then the code compute the solid angle sustained by a given detector on the sky $\Omega_{\rm det} = -\frac{A_{\rm det}}{f^2}\cos^3(\theta_{\rm det})$ and the effective surface of the horn array $S_{\rm horn} = N_{\rm horn}\pi R_{\rm horn}^2$.

The NEP is then computed in the following way:

$$
\begin{aligned}
{\rm NEP}_{\rm det} &= \sqrt{\sum_i 2h\nu P_\gamma^i\left(1+\frac{P_\gamma^i}{h\nu g_i}\right)} && (3)\\
P_\gamma^i &= \frac{\epsilon_i\eta_{{\rm eff,i}}h\nu}{\exp(h\nu/k_BT_i)-1}g_i\rho_{\rm det}B_{\rm sec}(\theta_{\rm det}) && (4)\\
g_i &= p_iS_{\rm horn}\Omega_{\rm det}\left(\frac{\nu}{c}\right)^2\Delta\nu && (5)
\end{aligned}
$$

We will need a good bibliography reference to explain this formula... $B_{\rm sec}(\theta_{\rm det})$ is the secondary beam transmission. The secondary beam is define just like the primary beam but with the remapping $\theta = \pi - \theta$. $\rho_{\rm det}$ is the detector efficiency, set to 0.8 in the code. The NEP is then computed for each detector and the photon noise is added as a white noise component on the detector array.

1.3 Time constant

The time constant is used in *get_detector_response_operator* in *instrument.py*. It takes the form of an operator than can be used on a $(n_{\rm det}, n_{\rm sample})$ array. The operator perform a convolution of the time ordered data by a truncated exponential. The time constant appears in the following equation:

$$\tau\frac{{\rm do(t)}}{dt} + {\rm o(t)} = i(t) \tag{6}$$

Where i(t) is a source term (forcing function) and o(t) is the recovered signal. A general solution of this equation can be found using a green function

$$o(t) = o_0\exp\left(-\frac{t}{\tau}\right) + \frac{1}{\tau}\int_{0^+}^{t} i(t')\exp\left(-\frac{(t-t')}{\tau}\right)dt' \tag{7}$$

Or more simply in Fourier space

$$\tilde{o}(\omega) = \tilde{i}(\omega)\frac{1-i\omega t}{1+(i\omega t)^2} \tag{8}$$

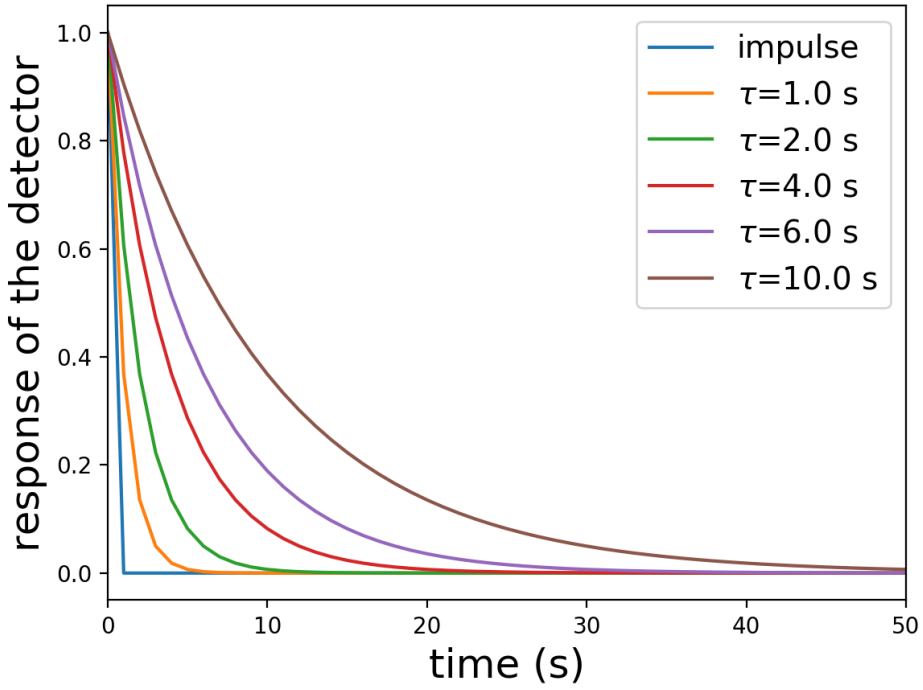


Figure 4: effect of the time constant on an impulse signal

In the code, this convolution is done efficiently using the following difference equation

$$o_n = (1 - \exp(-\frac{1}{f_s \tau}))i_n + \exp(-\frac{1}{f_s \tau})o_{n-1} \tag{9}$$

Where f_s is the sampling frequency $f_s = 1/T_s$ and T_s is set to 1s by default in the code. The time constant convolution is coded up as an operator in the code. The action of the operator is coded up in *pysimulators*, in the fortran routines *operators.f90.src* and *module_operators.f90.src*. We illustrate the effect of time constant on a impulse at t=0 in Figure 4.

1.4 Beam

They are three relevant beam for QUBIC, the primary beam, the secondary beam and the synthetic beam.

1.4.1 Primary and secondary beam

The primary beam and secondary beam are defined as un-normalized Gaussian with $\sigma_{\rm FWHM} = 13^\circ$

$$B_{\rm prim}(\theta, \phi) = \exp(-\frac{\theta^2}{2\sigma^2}) \tag{10}$$

$$B_{\rm sec}(\theta, \phi) = \exp(-\frac{(\pi - \theta)^2}{2\sigma^2}) \tag{11}$$

$$\sigma = \sigma_{\rm FWHM}/\sqrt{8 \log 2} \tag{12}$$

They are returned as healpix map in the code, a figure showing the healpix map as well as the projection on the north (primary) and south (secondary) spherical cap is displayed in Figure 7

1.4.2 Synthetic beam

The computation of the synthetic beam is more subtle as it involves the diffraction by the horn arrays and is computed for each detector on the focal plane. It can be obtained using the *get_synthbeam*

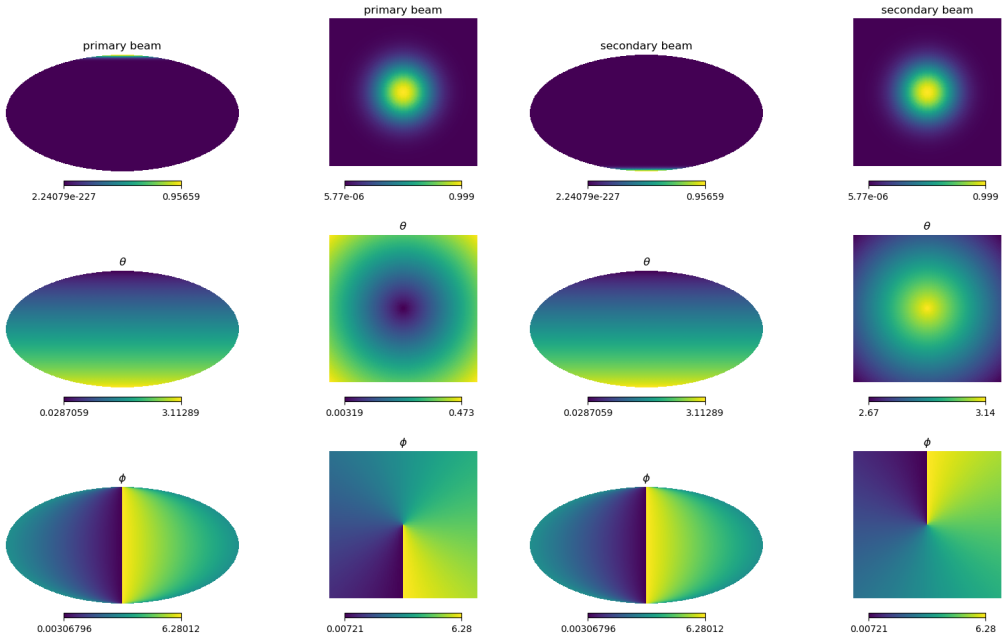


Figure 5: primary and secondary beam respectively projected on the north and south spherical cap

routine in *instrument.py*. The routine takes as argument a scene (which is basically only used to specify the resolution of the healpix map to display the synthetic beam), an index for the detector for which we want to compute the synthetic beam (if not specified, the beam is returned as an ($n_{\rm det}$, $n_{\rm pix}$) array where $n_{\rm pix}$ is the number of pixel of the healpix map, and $\theta_{\rm max}$, the maximum zenithal angle above which the synthetic beam is assumed to be zero. The code then call the static method *_get_synthbeam* still in *instrument.py* with argument scene, the position of the detector (3d coordinate (x_c, y_c, z_c) in m where z_c is the focal lenght), the area of the detector (m^2), ν, $\Delta\nu$, the horn array specifications, the primary and secondary beam and the data type of the synthetic beam (float32 by default). The code then compute the healpix θ and ϕ array corresponding to the $n_{\rm side}$ of the healpix map and call the routine *_get_response*.

One subtlety happen during this call, the bandwidth $\Delta\nu = 37.5$ GHz is called as a spectral irradiance. Note that the primary function of *_get_response* is to return the monochromatic complex field $\sqrt{(W/{\rm Hz})}^{1/2}$ related to the electric field over a specified area of the focal plane created by sources of specified spectral irradiance $W/(m^2.{\rm Hz})$. In the code the sky (created by scene) is assumed to be in $W/(m^2.{\rm Hz})$, by passing the bandwidth, I believe the code assume that the sky is integrated along the bandwidth (but it's a bit weird) and that the beam size is independent of frequencies (weird again).

There are two different effect we need to consider the first one is the light incident on the horn array, then the transmission between the back of the horn array to the detector array.

1. The complex electric amplitude and phase going through each horn can be computed as

$$E^h(\theta,\phi) = E^h_0(\theta,\phi)\exp(\frac{2i\pi\nu}{c}\vec{x}_{\rm horn}.\hat{n}) \quad (13)$$

$$E^h_0(\theta,\phi) = \sqrt{\Delta_\nu B_{\rm prim}(\theta,\phi)\pi r^2_{\rm horn}} \quad (14)$$

Here $\vec{x}_{\rm horn}$ is a 3d vector with cartesian coordinates $(x_{\rm horn}, y_{\rm horn}, z_{\rm horn} = 0)$ and $\hat{n} = (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta)$ is the spherical radial unit vector in cartesian coordinates. The result is given as a $(n_{\rm horn}, n_{\rm pix})$ array.

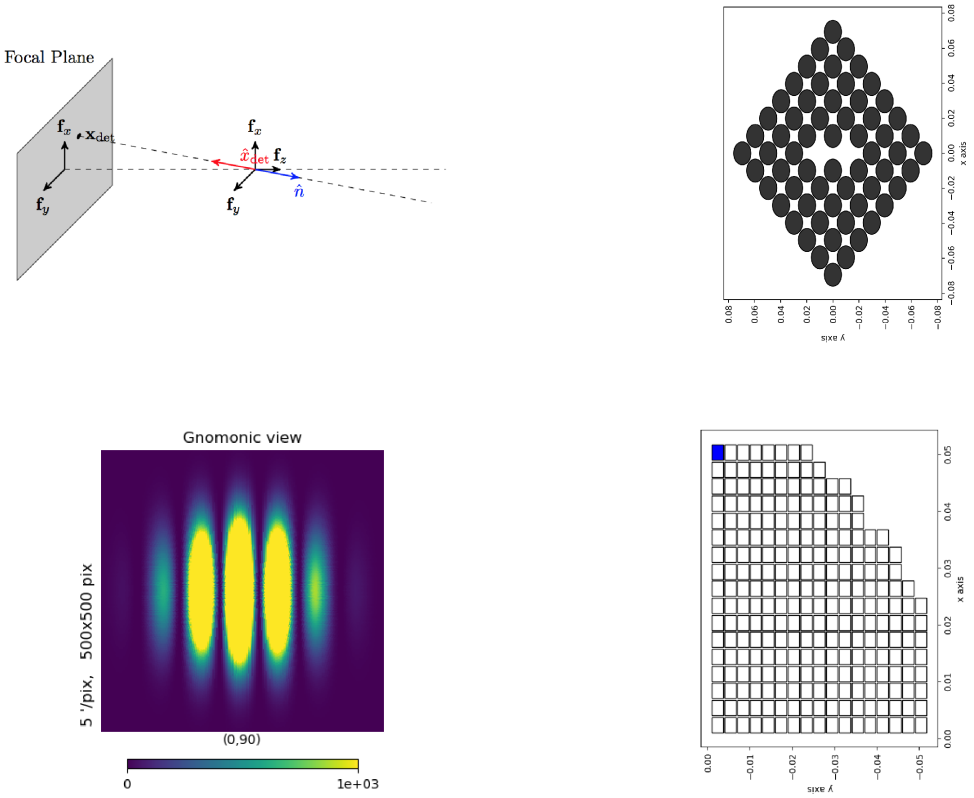


Figure 6: synthetic beam for two horns open

2. The Phase and transmission from the switches to the detectors on the focal plane

$$T^h_{\rm det}(\theta_{\rm det},\phi_{\rm det}) = T^h_{0,\rm det}(\theta_{\rm det},\phi_{\rm det})\exp(\frac{2i\pi\nu}{c}\hat{x}_{\rm det}.\vec{x}_{\rm horn}) \quad (15)$$

$$= \sqrt{B_{\rm sec}(\theta_{\rm det},\phi_{\rm det})\frac{\Omega_{\rm det}}{\Omega_{\rm sec}}} \quad (16)$$

Here $\Omega_{\rm det}$ is the solid angle sustain by the detector, $\Omega_{\rm sec}$ is the solid angle of the secondary beam. $\hat{x}_{\rm det}$ is a cartesian unit vector pointing toward the detector (with $z_{\rm det} = -f$) $T^h_{\rm det}$ is a $(n_{\rm det}, n_{\rm horn})$ array. The code use *ne.evaluate* for the fast evaluation of array expressions elementwise.

The synthetic beam is then returned as

$$B^{\rm synth}_{\rm det}(\hat{n}) = \Delta_\nu B_{\rm prim}(\theta,\phi)\pi r^2_{\rm horn} B_{\rm sec}(\theta_{\rm det},\phi_{\rm det})\frac{\Omega_{\rm det}}{\Omega_{\rm sec}}\left|\sum_{h=1}^{N_h}\exp\left(\frac{2i\pi\nu}{c}\vec{x}_{\rm h}(\hat{x}_{\rm det}+\hat{n})\right)\right|^2 \quad (17)$$

We can comment a bit on the last term which source the interference pattern, let's define $i\vec{\alpha} = \frac{2i\pi\nu}{c}(\hat{x}_{\rm det}+\hat{n})$, for two horns open (j and k) we have

$$(e^{i\vec{\alpha}.\vec{x}_j}+e^{i\vec{\alpha}.\vec{x}_k})(e^{-i\vec{\alpha}.\vec{x}_j}+e^{-i\vec{\alpha}.\vec{x}_k}) = 2(1+\cos(\vec{\alpha}.(\vec{x}_j-\vec{x}_k))) \quad (18)$$

Now let's try to compute this term when we have all horns open except two. For doing this let's first compute the interference pattern when all horns are open

$$S_{\rm tot} = \left|\sum_{h=1}^{N_h} e^{i\alpha\vec{x}_{\rm h}}\right|^2 = N_h + \sum_{j=1}^{N_h}\sum_{k\neq j}\cos(\vec{\alpha}.(\vec{x}_j-\vec{x}_k)) \quad (19)$$

The expression for all horns open but the horn i is:

$$S_{-i} = S_{\rm tot} - 1 - \sum_{k\neq i} 2\cos(\vec{\alpha}.(\vec{x}_i-\vec{x}_k)) \quad (20)$$

The expression for all horns open but the horns i and j is

$$S_{-ij} = S_{\rm tot} - 2 - \sum_{k\neq i} 2\cos(\vec{\alpha}.(\vec{x}_i - \vec{x}_k)) - \sum_{k\neq j} 2\cos(\vec{\alpha}.(\vec{x}_j - \vec{x}_k)) + 2\cos(\vec{\alpha}.(\vec{x}_j - \vec{x}_i)) \quad (21)$$

$$S_{-ij} = S_{-i} + S_{-j} + 2\cos(\vec{\alpha}.(\vec{x}_j - \vec{x}_i)) - S_{tot} \quad (22)$$

Note that this expression differs from the one presented in the self-calib paper, they are aware that they have a typo.

1.5 Multifrequencies

The code also allow for constructing a multiband instrument. The first step is to specify a number of frequencies subband ν, this is done in the code with the routine *compute_freq* in *polyacquisition.py*. The code take an input frequency say $\nu = 150$GHz, a relative band width $\frac{\delta\nu}{\nu} = 0.25$ and the number of frequencies we want to cut this band into $N_{\rm freq} = 10$. The code start by computing $\nu_{\rm min} = \nu(1 - \frac{\delta\nu}{\nu})$ and $\nu_{\rm max}(1 + \frac{\delta\nu}{\nu})$, and use interpolation in log space to define a set of sub-bands between them.

The output of the code are

$$N_{\nu,\rm edges} = N_\nu + 1 \text{ The number of frequency edges} \quad (23)$$

$$\nu_{\rm edges} \quad \text{The location of the edges} \quad (24)$$

$$\nu \quad \text{The center frequency of each sub-band} \quad (25)$$

$$\delta \quad \text{The size of each sub-band} \quad (26)$$

$$\Delta \quad \text{The size of the full band} \quad (27)$$

$$N_\nu \quad \text{The number of frequencies of the sub-band} \quad (28)$$

Note that this routine is used both for simulating the instrument (at the moment this is arbitrary and is chosen to be 10) and for recovering the final product as a function of output frequencies.

2 Pointing

They are two different pointing method implemented in the code, they can be found in *samplings.py*. One of them is realistic: *create_sweeping_pointings* and one of them allow for fast test of the code *create_random_pointings* Both methods return the azimuth (A), the elevation (h), the pitch angle (γ) and the hwp angle (η) for each sample. For both pointing methods you start by specifying the center of the patch in equatorial coordinate ($\rm RA_c$,$\rm DEC_c$). For both methods, the HWP angle is computed as

$$\eta(t) = 11.25\mathcal{I}(0,7)(t) \quad (29)$$

where $\mathcal{I}(0,7)$ represent integer random number between 0 and 7.

2.1 Random Ponting

For random pointing, you have to specify the number of sample you want, this is done with the argument $n_{\rm pointings}$, you also have to specify the maximum angular distance to the center of the patch $d\theta$. Optional arguments are the date of observation (T_0 default being '2016-01-01 00:00:00'), the observer latitude and longitude (default is DOME C: lat = -(75 + 6 / 60) long = 123 + 20 / 60). The code start by generating $n_{\rm pointings}$ random number for θ, ϕ and γ in the following way

$$\theta(t) = \arccos(\cos d\theta + (1 - \cos d\theta)\mathcal{U}_0(0,1)(t)) \quad (30)$$

$$\phi(t) = 2\pi\mathcal{U}_1(0,1)(t) \quad (31)$$

$$\gamma(t) = 2\pi\mathcal{U}_2(0,1)(t) \quad (32)$$

Where $\mathcal{U}_n(0,1)$ are random number uniformly distributed between 0 and 1. It then compute the time at which is sample is taken: $t_i = T_0 + i\delta t$, note that default is $\delta t = 1$s in the code. Finally it use a set of operators:

1. Cartesian2SphericalOperator ($\mathcal{O}_{c\to s}$) : (see *pyoperators/nonlinear.py*)
   Convert cartesian unit vectors into spherical coordinates, The spherical coordinate system is defined by:
   - the zenith direction of coordinate (0, 0, 1)
   - the azimuthal reference of coordinate (1, 0, 0)
   - the azimuth signedness: it is counted positively from the X axis to the Y axis.
   Four conventions define what the two spherical angles are
   - 'zenith, azimuth': angles commonly used in physics or the (colatitude, longitude) angles used in the celestial and geographical coordinate systems
   - 'azimuth, zenith': (longitude, colatitude) convention
   - 'elevation, azimuth: (latitude, longitude) convention
   - 'azimuth, elevation': (longitude, latitude) convention

2. CartesianEquatorial2HorizontalOperator ($\mathcal{R}_{e\to h}$): (see *pysimulators/operators.py*)
   Conversion between equatorial-to-horizontal cartesian coordinates.
   The ICRS equatorial direct referential is defined by:
   - the Earth center as the origin
   - the vernal equinox of coordinates (1, 0, 0)
   - the Earth North pole of coordinates (0, 0, 1)
   The horizontal referential is defined by:
   - the observer geographic position as the origin
   - the azimuth reference (North or South) of coordinates (1, 0, 0)
   - whether the azimuth is measured towards the East or West
   - the zenith of coordinates (0, 0, 1)

3. Rotation3dOperator ($\mathcal{R}_{\rm rot}$): (see *pyoperators/linear.py*)
   Operator for 3-d active rotations about 1, 2 or 3 axes.
   The rotation axes are specified one by one by selecting a convention.
   For intrinsic rotations (in which the coordinate system changes with the rotation), the following conventions are possible:
   X, Y, Z,
   XY', XZ', YX', YZ', ZX', ZY',
   XZ'X", XZ'Y", XY'X", XY'Z", YX'Y", YX'Z",
   YZ'Y", YZ'X", ZY'Z", ZY'X", ZX'Z" and ZX'Y".
   The primes denote the rotated axes after the first elemental rotation and the double primes the rotated axes after the second one.
   And for the extrinsic rotations (in which the original coordinate system remains motionless):
   X, Y, Z,
   XY, XZ, YX, YZ, ZX, ZY,
   XZX, XZY, XYX, XYZ, YXY, YXZ, YZY, YZX, ZYZ, ZYX, ZXZ and ZXY.

4. Spherical2CartesianOperator ($\mathcal{O}_{s\to c}$): (see *pyoperators/nonlinear.py*)
   Convert spherical coordinates into cartesian unit vectors

The full rotation to get azimuth and elevation from θ and ϕ is then

$$A(t), h(t) = \mathcal{O}^{\rm az,el}_{c\to s}(\mathcal{R}^{\rm NE}_{e\to h}(t)(\mathcal{R}^{\rm ZY'}_{\rm rot}[{\rm RA_c}, 90 - {\rm DEC_c}](\mathcal{O}^{\rm zen,az}_{s\to c}(\theta(t), \phi(t))))) \tag{33}$$

2.2 Sweeping Pointing

As for the random pointing, you start by specifying the center of the patch in equatorial coordinate ($\rm RA_c$,$\rm DEC_c$). Then you compute the azimuth and elevation corresponding to the center of the patch as a function of time (given the starting date of observation, the sampling period and the observer latitude and longitude).

$$A_c(t), h_c(t) = \mathcal{R}^{\rm NE}_{e\to h}(t)({\rm RA_c}, {\rm DEC_c}) \tag{34}$$

Where $\mathcal{R}^{\rm NE}_{h\to e}(t)$ is the SphericalEquatorial2HorizontalOperator (see *sampling.py/equ2hor*). For the sweeping pointing routine, we have to specify the duration of the observation $t_{\rm obs}$, in hours, the number of samples is then given by $n_{\rm pointings} = \frac{3600 t_{\rm obs}}{T_s}$ with T_s the sampling period. We also have to specify the azimuth extent of a scan ΔA, and the angular speed ω of the scan. The time it takes to scan back and forth is then given by $t_{bf} = \frac{2\Delta A}{\omega}$. The sweeping index is then trivially given by $i_{\rm sweep}(t) = \left\lfloor \frac{t}{t_{bf}} \right\rfloor$, the azimuth offset for each sample is given by

$$\begin{aligned}
\epsilon(t) &= \omega t \ (\mathrm{mod}\ 2\Delta A) && (35)\\
\tilde{\epsilon}(t) &= \begin{cases} \epsilon(t) & \text{if } \epsilon(t) < \Delta A \\ -\epsilon(t) + 2\Delta A & \text{otherwise} \end{cases} && (36)\\
\delta A(t) &= -\frac{\Delta A}{2} + \tilde{\epsilon}(t) && (37)\\
& && (38)
\end{aligned}$$

For the elevation, you also have to specify the number of sweep at constant elevation $n^h_{\rm sweep}$. The elevation index is given by $i^h_{\rm sweep}(t) = \frac{i_{\rm sweep}(t)}{n^h_{\rm sweep}}$ which is used to specify h(t). Finally we have

$$\begin{aligned}
A(t) &= A_c(t) + \delta A(t) && (39)\\
h(t) &= h(i^h_{\rm sweep}(t)) && (40)
\end{aligned}$$

Finally the pitch angle is computed using a pitch velocity ω_γ (default 0.1 degree/s) and a maximum pitch angle $\gamma_{\rm max}$ (default 10 degree)

$$\begin{aligned}
\epsilon_\gamma(t) &= \omega_\gamma t \ (\mathrm{mod}\ 4\gamma_{\rm max}) && (41)\\
\tilde{\epsilon_\gamma}(t) &= \begin{cases} \epsilon_\gamma(t) & \text{if } \epsilon_\gamma(t) < 2\gamma_{\rm max} \\ -\epsilon_\gamma(t) + 4\gamma_{\rm max} & \text{otherwise} \end{cases} && (42)\\
\gamma(t) &= -\gamma_{\rm max} + \tilde{\epsilon}_\gamma(t) && (43)\\
& && (44)
\end{aligned}$$

I have also added an option in my version of the code for doing scan of a fixed source.

3 Acquisition

For a multifrequencies instrument, they are two different acquisition methods. The polyacquisition method and the multiacquisition method. The multiacquisition method being simply a generalisation where we get observation for different frequencies subbands.

3.1 Multiacquisition

The main function in the the *multiacquisition.py* routine is the *get_observation* function. It takes an array of maps of the form ($N_{\rm sub}$, $n_{\rm pix}$, $n_{\rm stokes}$) where $N_{\rm sub}$ is the number of frequencies sub-bands, $n_{\rm pix} = 12N^2_{\rm side}$ and $n_{\rm stokes}$ is the number of Stokes parameter ($n_{\rm stokes} = 3$ for I,Q,U), and create a TOD for each detector (TOD being the detector signal as function of time). The function has two optional argument, convolution and noiseless. If noiseless=False, no noise will be added to the TOD. In the following I will describe what convolution do and how we go from a map to a TOD.

3.1.1 Convolution

The default value for convolution is convolution=TRUE, the convolution is done using the *get_convolution_peak_operator* defined in *instrument.py*. This routine depends on the argument ripples, if ripples=True the operator is *ConvolutionRippledGaussianOperator* which takes as argument the frequency of the subbands. Otherwise the convolution is done with the *HealpixConvolutionGaussianOperator*,

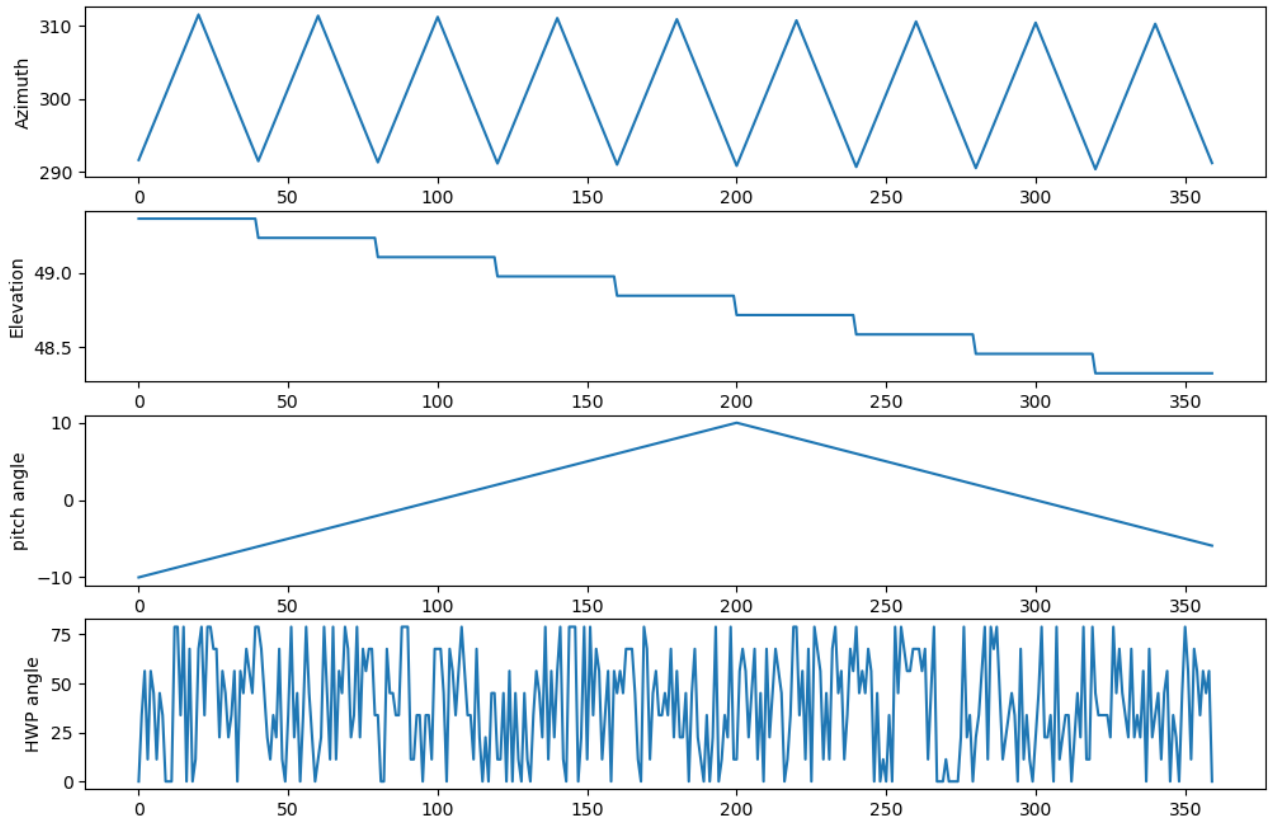


Figure 7: 6 minutes sweeping scan from La Puna, the center of the patch is chosen at RA=0, DEC=0

with $\sigma_{\rm fwhm}(\nu^i) = \sigma^{150}_{\rm fwhm}\frac{150.10^9}{\nu^i}$ The later is simply done in pixel space with the healpy smoothing function, while the former is done in *ripples.py* in harmonic space. The default value in the code is ripples =False, For now we will not describe the ripple convolution in details as I don't think it's extremely relevant for the rest of the analysis. The result is a convolved map for each of the sub-bands, and it is from this set of map that we will generate the TOD.

3.1.2 From map to TOD

This subsubsection could be a section by itself as it is truly the core of the code, how to take a healpix map and project it to a TOD. The main function is *get_operator* in *acquisition.py*, it creates the H operator which consist of the composition of many different operators

$$\mathcal{H} = \mathcal{R}_{\rm det}\mathcal{T}_{\rm inst}\mathcal{I}_{\rm det}\mathcal{P}_{\rm ol}[\mathcal{H}_{\rm WP} * \mathcal{P}_{\rm roj}]\mathcal{F}_{\rm ilt}\mathcal{A}_{\rm p}\mathcal{T}_{\rm atm}\mathcal{U}_{\rm nit}\mathcal{D}_{\rm ist} \tag{45}$$

The TOD is then obtained by the operation TOD $= \mathcal{H}(m)$ where m is the map. In the following, we will describe what does each of these operators.

1. $\mathcal{D}_{\rm ist}$ is a MPI operator. It distributes a global map, of which each MPI process has a copy, to the MPI processes. It is a block column operator whose blocks are identities distributed across the MPI processes.

2. The $\mathcal{U}_{\rm nit}$ operator convert sky temperature into W/m^2/Hz, the conversion applied to healpix pixel is

$$\mathcal{U}_{\rm nit} = \frac{10^{-6}}{T}\frac{2\Omega_{pix}h\nu^3}{c^2}\frac{h\nu}{kT}\frac{e^{\frac{h\nu}{kT}}}{(e^{\frac{h\nu}{kT}}-1)^2} \tag{46}$$

   the operator can be found in *pysimulators/interfaces/healpy/scenes.py*

3. $\mathcal{T}_{\rm atm}$ is the atmosphere transmission, it is set to 1 by default in the code

4. $\mathcal{A}_{\rm p}$ is the aperture integration operator, it integrates flux density in the telescope aperture, this convert signal from $W/m^2/Hz$ into W/Hz. It is coded in *instrument.py*. It is an homothety (a rescaling by a scalar) of value $\lambda = N_{\rm horn} S_{\rm horn} = N_{\rm horn} \pi r_{\rm horn}^2$

5. $\mathcal{F}_{\rm ilt}$ is the filter operator, it converts units from W/Hz to W, it is again an homothety operator with $\lambda = \delta\nu$ (the bandwidth)

6. $\mathcal{P}_{\rm roj}$ is the projection operator, converting W to W/sr, the core of the code *_get_projection_operator* is in *instrument.py*. It takes as argument a rotation, the scene, the central frequency ν, the position (center location of the detectors), the synthetic beam, the horn and the primary beam. This operator takes a $(n_{\rm pix}, n_{\rm stokes})$ map and return a $(n_{\rm det}, n_{\rm sample}, n_{\rm stokes})$ TOD. An example of such TOD can be found in Figure 8. So how does this work?

   The first step is to compute the spherical coordinates ($\theta_i^{\rm det}$, $\phi_i^{\rm det}$) of the beam peaks for each detector, in radians, up to a maximum diffraction order, with the routine *_peak_angles* in *instrument.py*. The diffraction order is used to decide which the peaks are ignored. For instance, a value of $k_{\rm max} = 2$ will model the synthetic beam by $(2k_{\rm max}+1)^2 = 25$ peaks and a value of $k_{\rm max} = 0$ will only sample the central peak. This is done by computing a normed position vector for each detector in the focal plane $\hat{x}^{\rm det} = \frac{\boldsymbol{x^{det}}}{|\boldsymbol{x^{det}}|}$, then the code define a mesh Fourier grid $\boldsymbol{k}$ with k_x and $k_y \in [-k_{\rm max}, k_{\rm max}]$ ($k_{\rm max}$ is set to 8 by default in the code), the location of the peaks is then found as $n_{i,x}^{\rm det} = \hat{x}_x^{\rm det} - \frac{c}{\nu}\frac{k_{i,x}}{d_h}$ where d_h is the horn spacing and similar for n_y. Finally

$$\theta_i^{\rm det} = \arcsin\sqrt{(n_{i,x}^{\rm det})^2 + (n_{i,y}^{\rm det})^2} \tag{47}$$

$$\phi_i^{\rm det} = \arctan\frac{n_{i,y}^{\rm det}}{n_{i,x}^{\rm det}} \tag{48}$$

   This only tells you that the peaks of the synthesized beam are separated by an integer times λ/d_h, while this is sound, we can test if this is the case for the technical demonstrator which has an inhomogeneous horn distribution. After doing the test, we realized there was a mismatch between the inferred beam peak position and the actual beam peak position. This was due to the rotation of 45 degree of the horn array which was not properly propagated in the *_peak_angles_kmax* routine. This is now fixed we have an argument to rotate the mesh grid k_x and k_y. For $k_{max} = 8$, the default value in the code, the number of peaks is 289 however, most of them can not be represented in the code (they give nan in the arcsin routine as the argument takes value > 1) . At the end, only 14 values of θ and ϕ are selected. The code also compute an integration factor at each peak position

$$A_i^{\rm det} = \frac{B(\theta_i^{\rm det}, \phi_i^{\rm det})\Omega_{\rm beam}^{150}(\frac{150}{\nu})^2}{\Omega_{\rm pix}} N_h \tag{49}$$

   Then the code create an array of the form $\hat{n}_{\rm peak}^{\rm det}$ of shape $(n_{\rm det}, n_{\rm peak}, 2)$ where $\hat{n}_{\rm peak}^{\rm det}$= ($\theta_{\rm peak}^{\rm det}$, $\phi_{\rm peak}^{\rm det}$) and change coordinate from the spherical frame to a cartesian frame using the *Spherical2CartesianOper* the resulting array is of the form $\hat{x}_{\rm peak}^{\rm det}$ of shape $(n_{\rm det}, n_{\rm peak}, 3)$ where $\hat{x}_{\rm peak}^{\rm det}$ is the usual normed position vector at the peaks position. For each detector (the parallelization is on the number of detector), a change of coordinate is applied, namely the transpose of *cartesian_galactic2instrument* ($\mathcal{R}_{g\to i}$) .

$$\mathcal{R}_{g\to i}(t) = \mathcal{R}_{\rm rot}^{\rm ZY'Z"}[{\rm az}, 90 - {\rm el}, {\rm pitch}]\mathcal{R}_{e\to h}^{NE}(t)\mathcal{R}_{g\to e} \tag{50}$$

   The position vector in the new coordinates is:

$$\hat{x}_{\rm peak}^{\rm gal,det}(t) = [\mathcal{R}_{g\to i}]^T(t)\hat{x}_{\rm peak}^{\rm det} \tag{51}$$

corresponding to the synthetized beam peaks position of each detectors as a function of time. The resulting product is a series of $n_{\rm det}$ array of shape$(n_{\rm sample}, n_{\rm peak}, 3)$ We then use the cartesian to healpix operator $\mathcal{O}_{c\to h}$, to associate an healpix pixel to the the synthetized beam peaks position

$$\#^{\rm gal,det}_{\rm peak}(t) = \mathcal{O}_{c\to h}\hat{x}^{\rm gal,det}_{\rm peak}(t) \tag{52}$$

$\#^{\rm gal,det}_{\rm peak}(t)$ being a $(n_{\rm sample}, n_{\rm peak}, 3)$ array which can be used to identify the relevant healpix pixel . Now, that we have identified the healpix pixels corresponding to the synthetized beam peaks position of each detectors as a function of time, we will also need to rotate the polarisation.

We define $\hat{e}_\theta$, $\hat{e}_\phi$, the unit vectors defining the plane orthogonal to the pointing vector $\hat{x}^{\rm det}_{\rm peak}$ in the frame of the instrument. In spherical coordinates we have

$$\hat{x}^{\rm det}_{\rm peak} = \begin{bmatrix}\sin\theta\cos\phi\\ \sin\theta\sin\phi\\ \cos\theta\end{bmatrix} \hat{e}^{\rm det}_\theta = \begin{bmatrix}\cos\theta\cos\phi\\ \cos\theta\sin\phi\\ -\sin\theta\end{bmatrix} \hat{e}^{\rm det}_\phi = \begin{bmatrix}-\sin\phi\\ \cos\phi\\ 0\end{bmatrix} \tag{53}$$

Which becomes, in cartesian coordinates

$$\hat{x}^{\rm det}_{\rm peak} = \begin{bmatrix}x\\ y\\ z\end{bmatrix} \hat{e}^{\rm det}_\theta = \begin{bmatrix}xz/\sqrt{1-z^2}\\ yz/\sqrt{1-z^2}\\ -\sqrt{1-z^2}\end{bmatrix} \hat{e}^{\rm det}_\phi = \begin{bmatrix}-y/\sqrt{1-z^2}\\ x/\sqrt{1-z^2}\\ 0\end{bmatrix} \tag{54}$$

We then change frame and now compute the pointing vector as given in the galactic frame

$$\hat{x}^{\rm gal,det}_{\rm peak}(t) = [\mathcal{R}_{g\to i}]^T(t)\hat{x}^{\rm det}_{\rm peak} \tag{55}$$

We then define a new plane orthogonal to the pointing vector in galactic coordinate, in this plane we will have

$$\hat{e}^{\rm gal,det}_\phi = \begin{bmatrix}-y^{\rm gal}/\sqrt{1-(z^{\rm gal})^2}\\ x^{\rm gal}/\sqrt{1-(z^{\rm gal})^2}\\ 0\end{bmatrix} \tag{56}$$

This vector can then be rotated back in the plane of the instrument

$$\hat{e}^{\rm det}_{\phi,\rm rot} = [\mathcal{R}_{g\to i}](t)\hat{e}^{\rm gal,det}_\phi \tag{57}$$

The rotation of the plane can then be expressed using the scalar product of the rotated vector with the initial vector

$$\cos\alpha = \hat{e}^{\rm det}_{\phi,\rm rot}.\hat{e}^{\rm det}_\phi \tag{58}$$

$$\sin\alpha = \hat{e}^{\rm det}_{\phi,\rm rot}.\hat{e}^{\rm det}_\theta \tag{59}$$

This is then assigned to a *FSRRotation3dMatrix*(see *pysimulators/sparse*), with element $(11 : A, 22 : A\cos 2\alpha, 23 : -A\sin 2\alpha)$ the action of this operator on a vector being

$$\begin{bmatrix}I_{\rm out}\\ Q_{\rm out}\\ U_{\rm out}\end{bmatrix} = M\begin{bmatrix}I_{\rm in}\\ Q_{\rm in}\\ U_{\rm in}\end{bmatrix} \tag{60}$$

$$M^{\rm det}_{\rm peak}(t) = \begin{pmatrix}A^{\rm det}_{\rm peak}(t) & 0 & 0\\ 0 & A^{\rm det}_{\rm peak}(t)\cos 2\alpha^{\rm det}_{\rm peak}(t) & A^{\rm det}_{\rm peak}(t)\sin 2\alpha^{\rm det}_{\rm peak}(t)\\ 0 & -A^{\rm det}_{\rm peak}(t)\sin 2\alpha^{\rm det}_{\rm peak}(t) & A^{\rm det}_{\rm peak}(t)\cos 2\alpha^{\rm det}_{\rm peak}(t)\end{pmatrix} \tag{61}$$

Overall we have a $(n_{\rm det}, n_{\rm time}, n_{\rm stokes}) \times (n_{\rm pix}, n_{\rm stokes})$ Projection operator whose action can be summarized as

$$\begin{bmatrix}I^{\rm det}(t)\\ Q^{\rm det}(t)\\ U^{\rm det}(t)\end{bmatrix} = \sum_{\rm peak} M^{\rm det}_{\rm peak}(t)\begin{bmatrix}I^{\rm gal}[\#^{\rm gal,det}_{\rm peak}(t)]\\ Q^{\rm gal}[\#^{\rm gal,det}_{\rm peak}(t)]\\ U^{\rm gal}[\#^{\rm gal,det}_{\rm peak}(t)]\end{bmatrix} \tag{62}$$

$$\#^{\rm gal,det}_{\rm peak}(t) = \mathcal{O}_{c\to h}[\mathcal{R}_{g\to i}]^T(t)\hat{x}^{\rm det}_{\rm peak} \tag{63}$$

An example of this projection is shown in Figure 8.

7. $\mathcal{H}_{\rm WP}$ is the half wave plate operator, it will apply an extra rotation to the polarisation components. The main routine is in the *get_hwp_operator* in *instrument.py*, it call a *Rotation3dOperator* from *pyoperators*. and rotate the polarisation component by $4\psi(t)$ where $\psi(t)$ is the HWP angle.

$$\begin{bmatrix} \tilde{I}^{\rm det}(t) \\ \tilde{Q}^{\rm det}(t) \\ \tilde{U}^{\rm det}(t) \end{bmatrix} = \mathcal{H}_{\rm WP}(t) \begin{bmatrix} I^{\rm det}(t) \\ Q^{\rm det}(t) \\ U^{\rm det}(t) \end{bmatrix} \tag{64}$$

$$\mathcal{H}_{\rm WP}(t) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos 4\psi(t) & \sin 4\psi(t) \\ 0 & -\sin 4\psi(t) & \cos 4\psi(t) \end{pmatrix} \tag{65}$$

The tilde quantities being modulated by the half wave plate.

8. $\mathcal{P}_{\rm ol}$ is the polarized grid operator. It takes a $(n_{\rm det}, n_{\rm time}, n_{\rm stokes})$ array and project it to a $(n_{\rm det}, n_{\rm time})$ array by selecting a polarisation component, essentialy

$$\mathcal{P}_{\rm ol}^{\rm det}(t) = 0.5 \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \tag{66}$$

so in this convention it will only select the $\tilde{Q}^{\rm det}(t)$ component. Using this operator allow to obtain a TOD with the final form

$$\begin{aligned} \mathcal{T}^{\rm det}(t) &= 0.5(I^{\rm det}(t) + \tilde{Q}^{\rm det}(t)) & (67) \\ &= \mathcal{P}_{\rm ol}^{\rm det}(t)\mathcal{H}_{\rm WP}(t) \sum_{\rm peak} M_{\rm peak}^{\rm det}(t) \begin{bmatrix} I^{\rm gal}[\#_{\rm peak}^{\rm gal,det}(t)] \\ Q^{\rm gal}[\#_{\rm peak}^{\rm gal,det}(t)] \\ U^{\rm gal}[\#_{\rm peak}^{\rm gal,det}(t)] \end{bmatrix} & (68) \\ &= \mathcal{O}^{\rm det}(t) \begin{bmatrix} I^{\rm gal}[\#_{\rm peak}^{\rm gal,det}(t)] \\ Q^{\rm gal}[\#_{\rm peak}^{\rm gal,det}(t)] \\ U^{\rm gal}[\#_{\rm peak}^{\rm gal,det}(t)] \end{bmatrix} & (69) \end{aligned}$$

with

$$\mathcal{O}^{\rm det}(t) = 0.5 \sum_{\rm peak} A_{\rm peak}^{\rm det}(t) \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos 4\psi(t) & \sin 4\psi(t) \\ 0 & -\sin 4\psi(t) & \cos 4\psi(t) \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos 2\alpha_{\rm peak}^{\rm det}(t) & \sin 2\alpha_{\rm peak}^{\rm det}(t) \\ 0 & -\sin 2\alpha_{\rm peak}^{\rm det}(t) & \cos 2\alpha_{\rm peak}^{\rm det}(t) \end{pmatrix} \tag{70}$$

9. $\mathcal{I}_{\rm det}$ is an operator to integrate the flux density in the detector solid angles while taking into account the secondary beam transmission, it is defined in *get_detector_integration_operator*, It's a diagonal operator with value

$$\mathcal{I}_{\rm det} = \frac{\Omega_{\rm det}}{\Omega_{\rm beam}} B^{\rm sec}(\phi_{\rm det}, \theta_{\rm det}) \tag{71}$$

The solid angle sustained by the detector $\Omega_{\rm det} = -\frac{A_{\rm det}}{f^2}\cos^3(\theta_{\rm det})$, $\Omega_{\rm beam}$ is the secondary beam solid angle and $B^{\rm sec}(\phi_{\rm det}, \theta_{\rm det})$ is the beam associated to the detector pointing.

10. $\mathcal{T}_{\rm inst}$ Return the operator that multiplies by the cumulative instrumental transmission $\eta_{\rm eff}^{\rm CMB} = \prod_{i=1}^{11} \eta_i$ which is the product of the transmission of all the different optical element.

$$\mathcal{T}_{\rm inst} = \rho_{\rm det} \prod_{i=1}^{11} \eta_i \tag{72}$$

where $\rho_{\rm det}$ is the detector efficiency

11. Finally $\mathcal{R}_{\rm det}$ is the detector response opetator (*get_detector_response_operator*) which apply the time constant by convolving the signal with a truncated exponential.

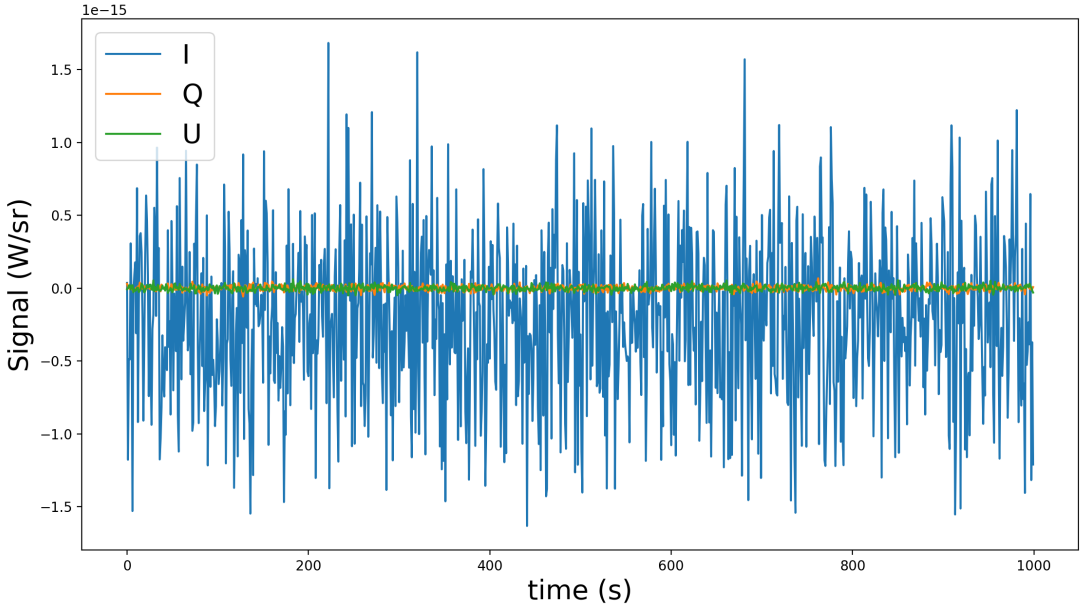


Figure 8: TOD of I,Q and U obtained using the projection operator, this is an intermediate step before obtaining the full signal seen by the detector.

4 Map making

The code uses a maximum likelihood map making algorithm to go from TOD to map. The TOD can be written as

$$\boldsymbol{d} = \boldsymbol{\mathcal{H}m} + \boldsymbol{n} \tag{73}$$

Assuming a gaussian noise with covariance $\boldsymbol{\mathcal{N}}$ we can write down the likelihood function of measuring $\boldsymbol{d}$ given $\boldsymbol{m}$

$$-2\ln\mathcal{L} \propto (\boldsymbol{d} - \boldsymbol{\mathcal{H}m})^T \boldsymbol{\mathcal{N}^{-1}} (\boldsymbol{d} - \boldsymbol{\mathcal{H}m}) \tag{74}$$

We can find the maximum likelihood solution of the system by taking the derivative with respect to $\boldsymbol{m}$

$$\frac{\partial \ln\mathcal{L}}{\partial \boldsymbol{m}} = -2\boldsymbol{\mathcal{H}}^T \boldsymbol{\mathcal{N}^{-1}} (\boldsymbol{d} - \boldsymbol{\mathcal{H}m}) = 0 \tag{75}$$

$$\boldsymbol{m} = (\boldsymbol{\mathcal{H}}^T \boldsymbol{\mathcal{N}^{-1}} \boldsymbol{\mathcal{H}})^{-1} \boldsymbol{\mathcal{H}}^T \boldsymbol{\mathcal{N}^{-1}} \boldsymbol{d} \tag{76}$$

One choice is thus the noise covariance matrix, it should be as close as possible to the noise present on the data. There are two different noise covariance matrices that can be used in the code:

1. *get_diag_invntt_operator* which return a diagonal invariance noise covariance matrix taking into account detector noise and photon noise

$$\mathcal{N}^{-1} = (\sigma^2_{\rm det} + \sigma^2_{\gamma})^{-1} = = 2T_s(\mathrm{NEP}^2_{\rm det} + \mathrm{NEP}^2_{\gamma})^{-1} \tag{77}$$

   Where T_s is the sampling period in s. We can rescale the noise per sample using an effective duration, this can be propagated to the noise covariance matrix in the following way

$$\mathcal{N}^{-1}_{\rm eff} = \mathcal{N}^{-1} \frac{365 \times 24 \times 3600 \times N_{\rm yr}}{N_{\rm sample} T_s} \tag{78}$$

2. *get_invntt_operator* which return the full inverse covariance matrix and can include correlated noise in the time streams

5 Scene

Scene contains routines to do rotation and unit conversion and to specify the atmospheric properties.

6 Input sky

Once the frequency band of the instrument have been specified, we need to simulate the sky in each of the frequency band, for the moment I use *create_input_cmb_sky* that return a copy of I,Q,U for the 10 frequencies band.