HERA Memo #098
Received: 27 May 2021

Mathematical description of the `vis_cpu` visibility simulator

Phil Bull, May 25, 2021

In this document, we set out the formalism, modelling choices, and approximations used by the `vis_cpu` visibility simulator. The original code was written by Aaron Parsons and has since been added to by Hugh Garsden, Phil Bull, and others. The latest code and releases of `vis_cpu` can be found on GitHub at `https://github.com/HERA-Team/vis_cpu`.

Formulation of the visibility calculation

Source location in Cartesian equatorial coordinates: First, source locations in equatorial coordinates (RA= α and Dec= δ) on the unit sphere are converted to a Cartesian system (where sources are assumed to be on a unit sphere), with coordinates

$$\vec{x}_n = (\cos(\alpha_n)\cos(\delta_n), \quad \sin(\alpha_n)\cos(\delta_n), \quad \sin(\delta_n)) \tag{1}$$

for source n. This results in an array of Cartesian positions of the sources call `crd_eq`, which has shape `(3, Nsrcs)`. This factor is time- and frequency-independent.

Rotation from equatorial to topocentric coordinates: Next, a rotation matrix that converts Cartesian equatorial coordinates to topocentric coordinates is calculated. This depends on time (LST), and the latitude of the array, l. For an hour angle $H = -t$, where t is the LST,

$$\mathbf{R}(t) = \begin{pmatrix} \sin(H) & \cos(H) & 0 \\ -\sin(l)\cos(H) & \sin(l)\sin(H) & \cos(l) \\ \cos(l)\cos(H) & -\cos(l)\sin(H) & \sin(l) \end{pmatrix}. \tag{2}$$

For each time sample, the rotation of source positions from equatorial to topocentric coordinates can be calculated as

$$\vec{p}_n(t) = \mathbf{R}(t) \cdot \vec{x}_n, \tag{3}$$

which in the code is given by `crd_top = np.dot(eq2top, crd_eq)`. We can unpack the components of the topocentric coordinate vector as $\vec{p} = (p_x, p_y, p_z)$, which are referred to as `tx, ty, tz` in the code. In this coordinate system, $p_z < 0$ denotes a source that is below the horizon, and so its flux should be set to zero. In the code, this is achieved by setting its beam factor $A_n = 0$.

Geometric delay for each source and antenna: Next, the geometric delay, τ_0, is calculated for each source. This depends on the position of source n on the sky, $\vec{p}_n$, and the antenna position vector $\vec{X}_i$, and in these coordinates can be found directly by taking a dot product,

$$\tau_0^{(i,n)} = \vec{X}_i \cdot \vec{p}_n / c, \tag{4}$$

where c is the speed of light. In the code, this is written as `tau = np.dot(antpos, crd_top) / c.value`. We use the subscript 0 to denote that this is the geometric delay between the antenna position and the origin of the array at $\vec{X} = (0, 0, 0)$.

Complex phase factor: We are now in a position to calculate the complex phase factor for each source, for each antenna i, which depends on frequency and time,

$$\Phi_0^{(i,n)} = \exp\left(2\pi\nu\tau_0^{(i,n)}\right) \tag{5}$$

Note that the calculation in the code uses the angular frequency in this expression, defined as $\omega = 2\pi\nu$.

Product with the sky brightness distribution: The `vis_cpu` code has an outer loop over frequency, ν, and an internal loop over LST or time, t. For each iteration of those loops, we have so far calculated a phase factor $\Phi_0(\nu, t)$, which has shape `(Nants, Nsrcs)`. Next, we can multiply by the sky brightness, which is constructed from the flux of each source at each frequency. As a computational trick, we take the *square root* of the flux to obtain $\sqrt{f_n(\nu)}$ for source n at frequency ν. The flux is real and positive semi-definite, so the square root is always defined. Note that the SED (the flux as a function of frequency) for each source is calculated outside the main `vis_cpu` function, and so any arbitrary frequency spectrum can be given to each source.

Per-antenna visibility factor: We can now take the product of the phase factor (per source n and antenna i) with the sky brightness factor per source to get

$$v_n^{(i)}(\nu, t) = \sqrt{f_n}\,\Phi_0^{(i,n)}. \tag{6}$$

The quantity $v_n^{(i)}$ is the contribution of antenna i and source n to a visibility V_{ij}, before modulation by the antenna pattern. Originally, the beam factor was also attached directly to the per-antenna visibility factor, v.

Polarized antenna factor (conversion to Az-ZA coordinates): We must also include the primary beam antenna factor for each source and antenna, evaluated at the location of each source in the beam pattern. Since the HERA antennas are zenith-pointing drift-scan telescopes, the appropriate coordinate system to evaluate the source positions is always an Alt-Az (or actually azimuth-zenith angle) coordinate system. The Az-ZA coordinates (ϕ and θ respectively) for each source are a function of time.

To convert from Cartesian to Alt-ZA coordinates, we can use the p_x and p_y coordinates calculated earlier, which can simply be identified as the angle cosines ℓ and m, such that

$$\begin{aligned}
L^2 &= \ell^2 + m^2 && (7)\\
\zeta &= \begin{cases} \sqrt{1 - L^2} & \text{if } L^2 < 1 \\ 0 & \text{otherwise} \end{cases} && (8)\\
\phi &= -\arctan_{(2)}(m, \ell) && (9)\\
\theta &= \pi/2 - \arcsin(\zeta). && (10)
\end{aligned}$$

Polarized antenna factor (beam evaluation): With the Az-ZA coordinates of each source in hand, the complex (E-field) beam factor can be evaluated at the location of each source. Originally, this was done by first constructing a 2D spline (`RectBivariateSpline`) of orders `kx = 1` and `ky = 1`, i.e. bilinear interpolation, for each antenna. The interpolation data could be precomputed once for each antenna and frequency, with new evaluations of the interpolation function needed for each time as the sources moved on the sky (but no need to recompute the interpolation data itself). The resolution of the interpolation grid can be chosen by the user, with finer grids requiring increased precomputation and spline evaluation time.

This 'pixel beam' mode is still supported. In addition, the code can now evaluate the beam directly, using the `interp` method on a `UVBeam` object. Since `UVBeam` objects generally have their own internal interpolation data and functions, this can be used natively, avoiding the need to build another interpolator and thus increased computation time and interpolation errors. This is particularly useful for analytic beams models, which can be evaluated precisely, without any interpolation. This all happens transparently to `vis_cpu`, which only needs to know how to call the `interp()` method on the `UVBeam` object.

Note that we allow the antenna beam to be different for each antenna, i.e. we require no assumption of identical antennas.

Polarized antenna factor (Jones matrix): In both of the cases above, we evaluate the antenna pattern at the position of each source. There is an unpolarized beam mode which calculates only the `ee` polarization (the East-East or `xx` polarization), but this is just a special case of the full polarized beam mode, which computes a 2×2 Jones matrix of the antenna response for each source. The dimensions of this matrix are the E-field axes – by convention, $(\hat{\phi}, \hat{\theta})$, the unit vectors in the azimuth and zenith angle directions – and the feeds, which are taken to be `n` (North) and `e` (East) feeds, with both dimensions being ordered in the order stated here. Each beam factor $\mathbf{A}_\mathbf{n}^{(\mathbf{i})}$ therefore has shape (`Naxes`, `Nfeeds`, `Nsrcs`).

Note that the sky model is currently assumed to be (pseudo-)Stokes I only; we do not currently have Stokes Q and U channels in the sky model.

Final multiplication to calculate the visibilities: With the polarized beams and per-antenna visibility factors in hand, we can now perform a series of array/matrix products to form the visibilities for all antenna pairs. Because of the way this multiplication is vectorized, we necessarily calculate the visibilities for all antenna pairs; we cannot arbitrarily calculate some pairs and not others.

We use Einstein summation convention (implemented through an `einsum` function call) to state the expression for the product, which handles the cross-multiplications of the visibility factors, the Jones matrices for each antenna, and the sum over source contributions to the visibility. The expression is

$$V_{ij}^{ab} = F^*_{acin} F_{cbjn} \tag{11}$$

where $F_{abin} = A_{abin} \circ v_{in}$, i.e. an elementwise product in the i and n indices. The Jones matrix indices are a, b, c (with only $a = (n, e)$ and $b = (n, e)$ retained after doing the matrix product implied by the repeated c indices, so these become feed indices), the antennas are labelled by i, j, and the sources are labelled by n (which is summed over). Recall that there are two outer loops over frequency and time, so this product is evaluated for each frequency and time separately. In the code, the expression is actually evaluated in a triangular fashion, via a single loop over antennas i to evaluate the product with antennas $j \geq i$ at each iteration.

Note the trick that has been used here. The per-antenna visibility factors contain only part of the phase information for each visibility, referenced against the phase to the origin of the array. By taking the product of them for antennas i and j, with an appropriate complex conjugate on one of the factors, the common factor of the phase to the origin cancels out, leaving only the difference in the delays between the two antennas, which is the correct phase factor for the baseline (i, j). The square-rooting of the sky brightness is also undone by this procedure; since the sky brightness is real and positive semi-definite, the product simply gives us the expected factor of the flux, $\sqrt{f}^* \sqrt{f} = f$.