A Synthetic LISA Manual

Michele Vallisneri

January 5, 2006

0.1 Implementation and usage of Synthetic LISA

In this section, we describe the implementation of all the *Synthetic LISA* objects. In Sec. 0.1.1, we give common notations and conventions; in Sec. 1.1, we discuss the LISA geometry objects; in Sec. **??**, we turn to GW source objects, and their use in the `TDIsignal` class; in Sec. 1.4, we describe LISA noise objects, and their use in the `TDInoise` class. Note that the implementation of TDI in *Synthetic LISA* is not based on fixed-timestep sequences, but rather on methods that can return the noises and GW signals at arbitrary nonquantized times, as needed by the TDI observables. However, some objects (such as pseudorandom or sampled noise sequences) are maintained internally by *Synthetic LISA* as arrays, and then interpolated to the TDI delayed times: see Sec. 1.6.2 for a discussion of the interpolation and bandlimitedness of pseudorandom noises and of presampled noises and GW signals. The syntax of *Synthetic LISA*'s commands is described in detail in Sec. 1.7.

0.1.1 Conventions

Our convention, introduced in Ref. [1], is to label the three LISA spacecraft as 1, 2, 3, and to label the (oriented) LISA arm l corresponding to transmission from spacecraft s to r as $l = \{1, 2, 3\}$ for $(s, r) = \{(3, 2), (1, 3), (2, 1)\}$, and $l = \{-1, -2, -3\}$ for $(s, r) = \{(2, 3), (3, 1), (1, 2)\}$ (thus, $\mathrm{sgn}(l) = \epsilon_{s|l|r}$). The motion of the LISA *guiding center* (the baricenter of the array) is approximately contained in the plane of the ecliptic; for this and other reasons [2], it is convenient to work in a Solar-system–baricentric ecliptic coordinate system (for short, SSB frame); we take the x axis of this system to be directed toward the vernal point. Throughout *Synthetic LISA*, we take $G = c = 1$, and 1 yr = 31,536,000 s.[1]

[1] This default can be altered by modifying the constants `secondsperyear` and `yearspersecond` in the header file `lisasim-lisa.h`.

1 Synthetic LISA classes

1.1 `LISA`: the LISA geometry objects

There are different levels of complexity at which the motion of the LISA array can be modeled in a simulation of the LISA science process; correspondingly, increasingly sophisticated TDI observables are needed to cancel laser noise once the added complexity is taken into account. In *Synthetic LISA*, these levels correspond to different derived classes of the base class `LISA` (in C++, a derived class inherits the data content and behavior of its base class, and can add enhancements or customizations); users can choose the `LISA` class that best fits their needs.

All the `LISA` classes define the methods `putp`, `putn`, and `armlength`, which return $\vec{p}_i(t)$, $\hat{n}_l(t)$, and $L_l(t)$ (see Table 1.1 for their syntax). The armlengths and photon-propagation vectors computed by these functions account for the aberration effects caused by the finite speed of light and by the spacecraft motion intervening between the events of pulse emission and reception.

The derived `LISA` classes currently defined in *Synthetic LISA* are the following; the syntax of the corresponding constructors is given in Table 1.1.

1.1.1 `OriginalLISA`: static LISA geometry

`OriginalLISA(`$L_1 = L_{\rm std}$`,`$L_2 = L_{\rm std}$`,`$L_3 = L_{\rm std}$`)`

Returns a static LISA object with armlengths given by L_1, L_2, L_3 (in seconds). If omitted, the L_i take the standard value $L_{\rm std}$ (16.6782 s in the current version of *Synthetic LISA*, but otherwise found in `lisasim-lisa.h`).

At the simplest level, we ignore the LISA motion altogether, and put the spacecraft in a stationary triangular configuration in the plane of the ecliptic; this model was used implicitly in the development of first-generation TDI. The class `OriginalLISA` implements such a stationary configuration, with spacecraft positions $\vec{p}_i$ set up according to the following rules:

1. the three armlengths are set to arbitrary values L_1, L_2, and L_3;
2. $\vec{p}_1 + \vec{p}_2 + \vec{p}_3 = 0$, so the baricenter of the constellation lies in the origin of the SSB ecliptic coordinate system;
3. $p_1^z = p_2^z = p_3^z = 0$, so the constellation lies in the $z = 0$ plane;

4. looking from above (from positive z's), the spacecraft sequence $1 \to 2 \to 3 \to 1$ traces a clockwise path (x toward $-y$), while the arms labeled by positive indices trace a counterclockwise path (x toward y);

5. $p_1^y = 0$, so spacecraft 1 lies along the x axis.

1.1.2 `ModifiedLISA`: stationary, rotating

`ModifiedLISA`($L_1 = L_{\rm std}$, $L_2 = L_{\rm std}$, $L_3 = L_{\rm std}$)

Returns a stationary LISA object set equal to `OriginalLISA`(L_1, L_2, L_3) at time 0, but rotating around its baricenter (at the SSB origin) with angular speed 2π/yr. The armlengths L_1, L_2, L_3 are given in seconds; if omitted, they take the default value $L_{\rm std}$. Because of the rotation, the armlengths depend on the direction of light propagation, but they are specified without this Sagnac correction.

At the next level of complexity, we model the cartwheeling motion of the LISA array, but not its orbital motion around the Sun. To this purpose, we place the spacecraft in the plane of the ecliptic, and we put them in rotation around their baricenter (which coincides with the origin of the SSB ecliptic coordinate system) with the angular frequency $\Omega = 2\pi/\mathrm{yr}$. The class `ModifiedLISA` implements such a rotating configuration.

At the time $t = 0$, the positions of the three spacecraft coincide with their stationary `OriginalLISA` positions; at subsequent times, they are computed by rotating the initial configuration counterclockwise (x toward y) through the angle Ωt. This model introduces a distinction between the armlengths experienced by light propagating along or against rotation (an example of the Sagnac effect [3]). Modified or second-generation TDI observables are then needed to cancel laser and optical-bench noise. Elementary kinematics shows that the corotating and counterrotating armlengths are given by

$$L_{arm} = L_{arm}^{\rm stat} + \vec{\Omega} \cdot (\vec{p}_r \times \vec{p}_s) + O(\Omega^2), \tag{1.1}$$

for light propagating from spacecraft s to r, where $\vec{\Omega} = \Omega \hat{z}$, and $L_{arm}^{\rm stat} = |\vec{p}_r - \vec{p}_s|$ is evaluated for stationary spacecraft positions (i.e., without rotation). With equal armlengths, the correction reduces[1] to $\pm\Omega |L_{arm}^{\rm stat}|^2/(2\sqrt{3})$, or about 1.6×10^{-5} s for the standard LISA armlength.

[1] According to the standard formula for the Sagnac effect [3], the difference in travel times for corotating and counterrotating paths enclosing the area A should be $4\Omega A/c^2$; in our case, we get a travel-time difference $\sqrt{3}|L_{arm}^{\rm stat}|^2$, consistent with area of the LISA equilateral-triangle configuration.

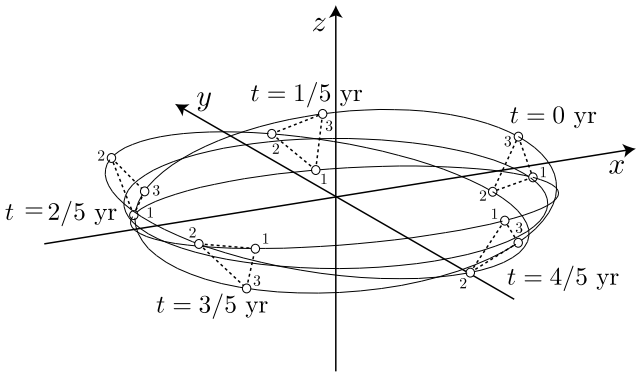

Figure 1.1: Orbital motion of the LISA detector, shown in a Solar-system–baricentric ecliptic coordinate system. The trajectories shown correspond to setting $\zeta = -\pi/6$, $\Omega = 2\pi/\mathrm{yr}$, and $\xi_0 = \eta_0 = 0$. For this choice, at time $t = 0$ the LISA guiding center lies on the positive x axis of the SSB system, while $\vec{p}_1$ lies on the negative y axis. Figure excerpted from Ref. [4].

1.1.3 `CircularRotating`: qualitative solar orbits

`CircularRotating`($\eta_0 = 0$,$\xi_0 = 0$,$sw = 1$,$t_0 = 0$)
`CircularRotating`(L,η_0,ξ_0,sw,t_0)

Return rigid LISA objects where the armlengths are equal and constant before aberration is taken into account. The parameters η_0 and ξ_0 are the true anomaly of the LISA guiding center and the initial phase of the LISA array at $t = t_0$ (seconds); $sw < 0$ will swap spacecraft 2 and 3, so that the spacecraft sequence $1 \to 2 \to 3 \to 1$ goes counterclockwise as seen from above. If given (in which case all parameters must be specified), L is the common armlength; if not given, it is set to L_{std}.

`CircularRotating()` is a legal constructor and replaces `stdlisa()`. The LISA baricenter moves on a circular orbit in the ecliptic, while the constellation rotates around the guiding center with the same angular velocity. The orbits follow Krolak et al., PRD **70**, 022003 (2004).

At the next level of complexity, we include all the qualitative features of the LISA motion using the simplest possible geometry where the armlengths are constant (before aberration is taken into account). That is, we move the baricenter of the LISA array on a circular orbit in the ecliptic plane, while the array itself, inclined with respect to the ecliptic, rotates around the guiding center with the angular frequency Ω. The class `CircularRotating` implements such a rotating configuration.

This model was used by Królak and colleagues [4] to derive analytic expressions of the LISA GW responses. Following the conventions of Ref. [4], in the coordinate frame where the spacecraft are at rest, we set without loss of generality $\vec{p}_i^{\,L} =$

$(L/\sqrt{3})(-\cos 2\sigma_i, \sin 2\sigma_i, 0)$, and $\hat{\vec{n}}_i^L = (\cos\sigma_i, \sin\sigma_i, 0)$, where $\sigma_i = 3\pi/2 - 2(i-1)\pi/3$. A realistic set of spacecraft orbits, shown in Fig. 1.1, is then obtained by letting $\vec{p}_i(t) = \vec{r}(t) + \mathsf{O}_2 \cdot \vec{p}_i^L$, where $\vec{r}$ is the vector from the origin of the SSB coordinate system to the LISA guiding center, as described by the SSB components $\vec{r} = R(\cos\eta, \sin\eta, 0)$; here $R = 1\,\mathrm{AU} = 499.004\,\mathrm{s}$ and $\eta = \Omega t + \eta_0$ is the true anomaly of the LISA guiding center in its orbit around the Sun. The rotation matrix O_2 models the cartwheeling motions of the spacecraft along their inclined orbits, shown in Fig. 1.2; it is given by

$$\mathsf{O}_2 = \begin{pmatrix} \sin\eta\cos\xi - \cos\eta\sin\zeta\sin\xi & -\sin\eta\sin\xi - \cos\eta\sin\zeta\cos\xi & -\cos\eta\cos\zeta \\ -\cos\eta\cos\xi - \sin\eta\sin\zeta\sin\xi & \cos\eta\sin\xi - \sin\eta\sin\zeta\cos\xi & -\sin\eta\cos\zeta \\ \cos\zeta\sin\xi & \cos\zeta\cos\xi & -\sin\zeta \end{pmatrix}; \tag{1.2}$$

here $\xi = -\Omega t + \xi_0$ is the phase of the motion of each spacecraft around the guiding center, while ζ sets the inclination of the orbital plane with respect to the ecliptic. For the LISA trajectory, $\zeta = -\pi/6$ [5]. These spacecraft orbits can be mapped approximately to those used in the LISA Simulator [6] by setting $\eta_0 = \kappa$, $\xi_0 = 3\pi/2 - \kappa + \lambda$, where κ and λ are the parameters defined below Eqs. (56) and (57) of Ref. [6], and by choosing $sw < 0$ in the `CircularRotating` constructor (Table 1.1), which has the effect of exchanging spacecraft 2 and 3.

The armlengths experienced by light propagating between the spacecraft can be found by solving Eq. (3) of Ref. [1] numerically. However, for reasons of efficiency, *Synthetic LISA* employs the lowest-order approximation to the solution,

$$L_{arm} = L + (\mathrm{sgn}\, arm) \times (\Omega R L) \sin(\Omega t - \delta_{|arm|}), \tag{1.3}$$

where $\delta_i \equiv \{\xi_0, \xi_0 + 4\pi/3, \xi_0 + 2\pi/3\}$. For the standard LISA orbits and configuration, the amplitude of this oscillating correction is about 1.6×10^{-3}, or 100 times larger than the correction due to the Sagnac effect alone for `ModifiedLISA`.

1.1.4 `EccentricInclined`: realistic solar orbits

`EccentricInclined`($\eta_0 = 0$,$\xi_0 = 0$,$sw = 1$,$t_0 = 0$)

Returns a realistic Keplerian LISA geometry modeled up to second order in the eccentricity, following Cornish and Rubbo, PRD **67**, 022001 (2003), but with the approximate parametrization of `CircularRotating`: η_0 and ξ_0 true anomaly of baricenter and array phase at $t0 = 0$; $sw < 0$ swaps spacecraft.

The most accurate description of the LISA motion available in *Synthetic LISA* models the eccentric orbits of the spacecraft up to second order in the eccentricity e. For these orbits, implemented by the class `EccentricInclined`, the dominant (and evolving) differences between the armlengths are caused by the flexing motion of the array [7] due to orbital eccentricities. Following Ref. [6], we set

$$\begin{bmatrix} p_i^x \\ p_i^y \\ p_i^z \end{bmatrix} = (1\,\mathrm{AU}) \begin{bmatrix} \cos\alpha + e[\sin\alpha\cos\alpha\sin\beta_i - (1+\sin^2\alpha)\cos\beta_i] + O(e^2) \\ \sin\alpha + e[\sin\alpha\cos\alpha\cos\beta_i - (1+\cos^2\alpha)\sin\beta_i] + O(e^2) \\ -\sqrt{3}e\cos(\alpha - \beta) + O(e^2) \end{bmatrix}, \tag{1.4}$$

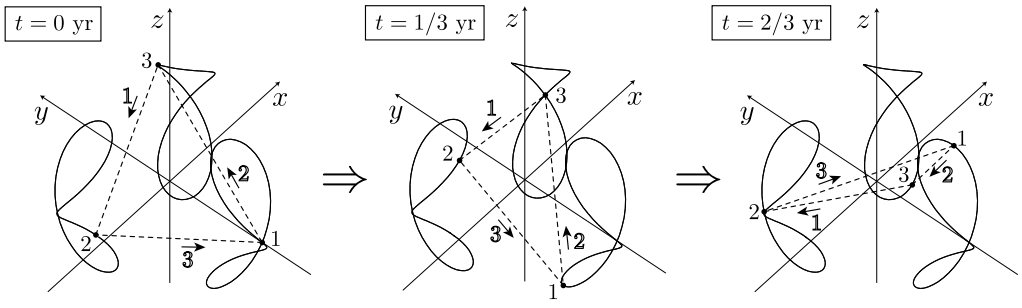

Figure 1.2: Cartwheeling motion of the LISA array, as plotted in a frame with center in the LISA guiding center and axes parallel to the SSB ecliptic frame. We show three snapshots at different times along the LISA orbital period, 1 yr. Figure excerpted from Ref. [4].

$$\alpha = \Omega t + \eta_0, \quad \beta_i = \eta_0 + \xi_0 - \sigma_i, \tag{1.5}$$

where $\sigma_i = 3\pi/2 - 2(i-1)\pi/3$ and $e = 0.00964838$, yielding an effective $L \simeq 16.6782$ s. The parameters η_0 and ξ_0 have approximately the same meaning as in `CircularRotating`. The correspondence to the LISA Simulator orbital parameters is again $\eta_0 = \kappa$, and $\xi_0 = 3\pi/2 - \kappa + \lambda$ (with $sw < 1$).

The armlengths experienced by light propagating along the arms can be found by solving Eq. (3) of Ref. [1]. For efficiency, *Synthetic LISA* employs the lowest-order approximation

$$L_{arm} = L + \frac{1}{32}(eL)\sin(3\Omega t - 3\xi_0) + \left[(\mathrm{sgn}\, arm)(\Omega RL) - \frac{15}{32}(eL)\right]\sin(\Omega t - \delta_{|arm|}), \tag{1.6}$$

where $\delta_i \equiv \{\xi_0, \xi_0 + 4\pi/3, \xi_0 + 2\pi/3\}$. The amplitude of the flexing correction is about 7.5×10^{-2} s, or 0.5% of the nominal LISA armlength; the rate of change of the armlengths is about 1.5×10^{-8} s/s, which requires second-generation TDI to yield sufficient cancellation of laser phase noise.

1.1.5 `PyLISA`: user-provided nominal armlengths

`PyLISA(baseLISA,pythonFunction)`

Returns a `LISA` object with physical geometry (used for light propagation and GW and noise responses) given by `baseLISA`, but with nominal armlengths (as used for the TDI delays) given by `pythonFunction(`i`,`t`)`, with $i = \pm 1, \pm 2, \pm 3$ and the time t given in seconds.

`PyLISA.baseLISA` (instance variable) points to the `baseLISA` object passed to the constructor, which provides the physical geometry.

PyLISA attempts to provide a more general (if less efficient) mechanism for the nominal-armlength computations previously performed with NoisyLISA, NominalLISA, LinearLISA, MeasureLISA (all experimental and not included with the current release).

An example of PyLISA usage that superimposes white noise (with 1 s rms and 1 s bandlimit) to the value of the armlength 1 would be

```
>>> b = synthlisa.CircularRotating()
>>> p = ...
```

1.1.6 CacheLISA: hash-cached armlengths

CacheLISA(baseLISA)

Returns a LISA object that works by routing all putp and putn calls to the LISA object referenced by baseLISA. However, CacheLISA interposes a layer of its own making for retard() calls, effectively caching chained retardations for the most recently accessed time.
CacheLISA, defined in lisasim-retard.h, might improve performance for complicated (or multiple) TDI-variable evaluations, especially when baseLISA points to a PyLISA object.

...

1.1.7 SampledLISA: user-provided armlengths given as arrays

SampledLISA(p1,p2,p3,Δt,prebuffer,interp)

Returns a LISA object that takes the positions of its spacecraft from the two-dimensional Numeric arrays p1, p2, p3; each of these consists of three columns that give the SSB coordinates of corresponding LISA spacecraft.
The array data is spaced by intervals Δt [s], and offset by prebuffer [also s]; interp (> 1) sets the semiwidth of the data window used in Lagrange interpolation of the positions.

More precisely,

$$\begin{bmatrix} p_i^x \\ p_i^y \\ p_i^z \end{bmatrix} = \begin{bmatrix} \texttt{pi}[(t-\texttt{prebuffer})/\Delta t, 0] \\ \texttt{pi}[(t-\texttt{prebuffer})/\Delta t, 1] \\ \texttt{pi}[(t-\texttt{prebuffer})/\Delta t, 2] \end{bmatrix}, \tag{1.7}$$

where $i = 1, 2, 3$ is the spacecraft index. SampledLISA makes copies of the position arrays, so these can be safely destroyed after calling the SampledLISA constructor; on the other hand, modifying the arrays passed to the constructor has no effect on the positions returned by a previously constructed SampledLISA.

The file positions.txt, installed in share/synthlisa, contains positions for the LISA spacecraft along a period of ten years, given at intervals of 1 day, as computer by Ted Sweetser on July 23, 2005. The function stdLISApositions() returns four

arrays for the time axis (beginning at 0) and for $\vec{p}_1$, $\vec{p}_2$, and $\vec{p}_3$, all expressed consistently with the `SampledLISA` conventions. The function `stdSampledLISA()` returns a `SampledLISA` object constructed with these arrays; its optional argument, defaulting to 1, sets the semiwidth of the Lagrange interpolation kernel for the `SampledLISA` object.

1.2 The LISA GW response

In *Synthetic LISA*, the GW response of the TDI observables is obtained from the class `TDIsignal`, which inherits from the base class `TDI` the definitions of the observables in terms of the basic Doppler responses, but which also defines the method `y` for y_{slr}, according [2] to Eq. (1) of Ref. [1]. A `TDIsignal` object is created (see Table 1.4) by specifying a LISA geometry object (Sec. 1.1) and a GW source object; the latter is always derived from the base class `Wave`.

1.3 The `Wave` classes

`Wave` (abstract base class)
`putk()` (instance method) returns the normalized SSB components of the plane-GW propagation vector. `putwave(`t`)` (instance method) returns the SSB components of the gravitational strain tensor at the SSB at time t [s].
All derived classes must provide: `hp(`t`)` and `hc(`t`)`, which return the GW polarizations $h_+(t)$ and $h_\times(t)$ at the SSB at time t [s].

All `Wave` classes share the same geometrical setup, which follows the conventions of Ref. [4]. At the position $\vec{x}$ in the SSB frame, the spatial part of the transverse–traceless metric perturbation associated with a plane GW can be written as

$$\mathsf{h}(t) = h_+(t - \hat{k}\cdot\vec{x})\,\mathsf{e}_+ + h_\times(t - \hat{k}\cdot\vec{x})\,\mathsf{e}_\times; \tag{1.8}$$

here the functions $h_+(t)$ and $h_\times(t)$ express the two polarization components of the wave at time t, measured at the origin of the SSB frame. For a GW source at ecliptic latitude β and ecliptic longitude λ, the unit propagation vector $\hat{k}$ is

$$\hat{k} \equiv -(\cos\beta\cos\lambda, \cos\beta\sin\lambda, \sin\beta). \tag{1.9}$$

The two polarization tensors e_+ and $\mathsf{e}_\times$ that appear in Eq. (1.8) can be defined without

[2] The `TDIsignal` method `z` for z_{slr} always returns zero, since the GW response of the intraspacecraft Doppler observable z_{slr} is null, at least if the lasers are not phase-locked to a master.

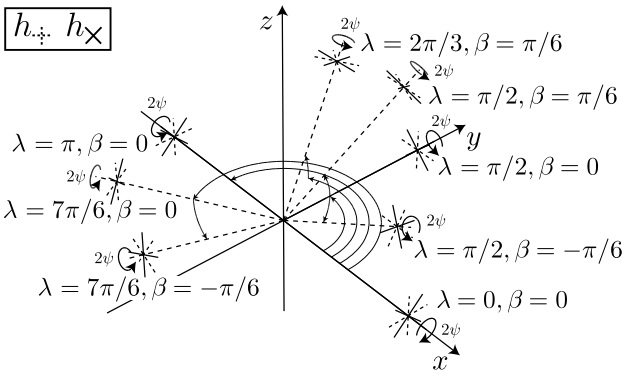

Figure 1.3: Conventional definition of the GW polarizations + (dashed) and × (solid) for various ecliptic latitudes β and longitudes λ. Figure excerpted from Ref. [4].

loss of generality as

$$\mathsf{e}_+ \equiv \mathsf{E} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \cdot \mathsf{E}^T, \qquad \mathsf{e}_\times \equiv \mathsf{E} \cdot \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \cdot \mathsf{E}^T, \tag{1.10}$$

where the orthogonal matrix E,

$$\mathsf{E} \equiv \begin{pmatrix} \sin\lambda\cos\psi - \cos\lambda\sin\beta\sin\psi & -\sin\lambda\sin\psi - \cos\lambda\sin\beta\cos\psi & -\cos\lambda\cos\beta \\ -\cos\lambda\cos\psi - \sin\lambda\sin\beta\sin\psi & \cos\lambda\sin\psi - \sin\lambda\sin\beta\cos\psi & -\sin\lambda\cos\beta \\ \cos\beta\sin\psi & \cos\beta\cos\psi & -\sin\beta \end{pmatrix}, \tag{1.11}$$

expresses an Euler rotation sequence, whereby the β and λ terms can be understood as enforcing the transversality of the GW, while the polarization angle ψ encodes a rotation around the direction of wave propagation, $\hat{k}$, setting the convention used to define the two polarizations. The polarizations corresponding to $\psi = 0$ are shown in Fig. 1.3 for various source positions in the sky. The positional parameters β, λ, and ψ can be mapped to the parameters θ, ϕ, and ψ used in the LISA Simulator [6] by setting $\beta = \pi/2 - \theta$, $\lambda = \phi$, and $\psi = -\psi$.

All the `Wave` classes define the methods `hp(`t`)` and `hc(`t`)`, which return $h_+(t)$ and $h_\times(t)$. The base `Wave` class defines the method `putk()`, which returns $\hat{k}$, and `putwave(`t`)`, which returns $\mathsf{h}(t)$. The derived `Wave` classes currently defined in *Synthetic LISA* are described in the following sections.

1.3.1 `SimpleBinary`: plane GWs from monochromatic binaries

`SimpleBinary`(f,ϕ_0,ι,A,β,λ,ψ)

Returns a `Wave` object implementing the sinusoidal GW emitted from a simple monochromatic binary. The wave is incoming from ecliptic latitude β, longitude λ, with polarization parameter ψ; it has frequency f [Hz] (= twice the orbital frequency of the binary) as observed in the SSB frame, initial phase ϕ_0 (at $t = 0$, and + and × amplitudes $A(1 + \cos^2 \iota)$ and $2 \cos \iota$, consisting with a binary inclination ι.

More precisely,

$$\begin{bmatrix} h_+(t) \\ h_\times(t) \end{bmatrix} = A \begin{bmatrix} (1 + \cos^2 \iota) \times \cos(2\pi f t + \phi_0) \\ (2 \cos \iota) \times \sin(2\pi f t + \phi_0) \end{bmatrix}. \tag{1.12}$$

The inclination angle ι is measured from the direction of the binary's orbital angular momentum to the direction of GW propagation. The standard value[3] of A (using geometrical units) is $(2m_1m_2/d\,R)$ with m_1, m_2 the two masses, d the luminosity distance, and R the orbital separation.

1.3.2 `SimpleMonochromatic`: simple sinusoidal GWs

`SimpleMonochromatic`(f,ϕ,γ,A,β,λ,ψ)

Returns a `Wave` object implementing a simple sinusoidal GW incoming from ecliptic latitude β, longitude λ, with polarization parameter ψ. The wave has frequency f [Hz] as observed in the SSB frame, + and × amplitudes $A \sin \gamma$ and $A \cos \gamma$, and a relative shift ϕ between the polarizations.

More precisely,

$$\begin{cases} h_+(t) = A \sin \gamma \sin(2\pi f t + \phi), \\ h_\times(t) = A \cos \gamma \sin(2\pi f t). \end{cases} \tag{1.13}$$

1.3.3 `GaussianPulse`: impulsive GWs with Gaussian profile

`GaussianPulse`(t_0,$t_{\rm fold}$,γ,A,β,λ,ψ)

Returns a `Wave` object implementing an impulsive plane GW incoming from ecliptic latitude β, longitude λ, with polarization parameter ψ. The pulse is centered at SSB time t_0 [s] with e-folding time $t_{\rm fold}$ [s], and has + and × amplitudes $A \sin \gamma$ and $A \cos \gamma$ (consistently with the `SimpleMonochromatic` convention).

More precisely,

$$\begin{pmatrix} h_+(t) \\ h_\times(t) \end{pmatrix} = A \begin{pmatrix} \sin \gamma \\ \cos \gamma \end{pmatrix} \exp \frac{-(t - t_0)^2}{t_{\rm fold}^2}. \tag{1.14}$$

[3] The common amplitude h_0 used in Ref. [4] differs by a factor of two ($h_0 = 2A$), absorbed in $h_0^\times$ and h_0^+.

For efficiency, the pulse is cut to zero for $|t - t_0| > 10\,t_{\rm fold}$ (this is set by the `const GaussianPulse::sigma_cutoff` in `lisasim-wave.cpp`).

1.3.4 `SineGaussian`: impulsive GWs with Gaussian profile

`SineGaussian(`t_0`,`$t_{\rm fold}$`,`f`,`ϕ_0`,`γ`,`A`,`β`,`λ`,`ψ`)`

Returns a `Wave` object implementing an impulsive plane GW incoming from ecliptic latitude β, longitude λ, with polarization parameter ψ. The pulse is centered at SSB time t_0 [s] with e-folding time $t_{\rm fold}$ [s], and has + and × amplitudes $A\sin\gamma$ and $A\cos\gamma$ (consistently with the `SimpleMonochromatic` convention); in addition, the pulse is modulated by a sinusoid of frequency f, centered at t_0, with relative phase ϕ_0 between the h_+ and $h_\times$ polarizations (also consistently with the `SimpleMonochromatic` convention).

More precisely,

$$\begin{pmatrix} h_+(t) \\ h_\times(t) \end{pmatrix} = A \begin{pmatrix} \sin\gamma \\ \cos\gamma \end{pmatrix} \exp\frac{-(t-t_0)^2}{t_{\rm fold}^2} \begin{pmatrix} \sin[2\pi f(t-t_0)+\phi_0] \\ \sin[2\pi f(t-t_0)] \end{pmatrix}. \tag{1.15}$$

For efficiency, the pulse is cut to zero for $|t - t_0| > 10\,t_{\rm fold}$ (this is set by the `const SineGaussian::sigma_cutoff` in `lisasim-wave.cpp`).

1.3.5 `NoiseWave`: noise-like GWs

`NoiseWave(hpnoise,hcnoise,`β`,`λ`,`ψ`)`
`NoiseWave(`Δt`,prebuffer,psd,filter,interp,`β`,`λ`,`ψ`)`

Returns a `Wave` object implementing a noise-like plane GW incoming from ecliptic latitude β, longitude λ, with polarization parameter ψ.

If the first form of the constructor is used, `hpnoise` and `hcnoise` are `Noise` objects (previously instantiated, e.g., from `InterpolateNoise`) that provide the two GW polarizations $h_+(t)$ and $h_\times(t)$ directly from the methods `hpnoise.noise(`t`)` and `hpnoise.noise(`t`)`.

If the second form is used, $h_+(t)$ and $h_\times(t)$ are fed from two independent pseudorandom noise streams (implemented with `InterpolateNoise`, where Δt [s] is the time spacing of the pseudorandom samples at generation, `prebuffer` [s ???] sets the length of the ring buffer used for the streams, `psd` and `filter` set the one-sided PSD of the pseudorandom noise stream to $\texttt{psd} \times (f/\text{Hz})^{\texttt{filter}}\,\text{Hz}^{-1}$ (currently only `filter`s of −2, 0, and 2 are implemented), and `interp` (> 1) sets the semiwidth of the data window used in the Lagrange interpolation of the discretely generated pseudorandom deviates (1 yields linear interpolation).

`NoiseWave` objects can also be created from arrays of previously generated waveform samples, using the `SampledWave` constructor syntax (see below).

The exact import of the `prebuffer` parameter is that after pseudorandom noise has been requested and computed at time t, the earliest value guaranteed to be available will be at time $t -$ `prebuffer`. This parameter should generally be set higher for a `NoiseWave` object than it would be for an `InterpolateNoise` object, because $h_+(t)$ and $h_\times(t)$ are defined in terms of SSB time, so the polarization values needed to build (say) the TDI observable X at $t = 0$ may require accessing h_+ and $h_\times$ at SSB times $t \simeq -500$ s; on the other hand, when Synthetic LISA builds new `InterpolateNoise` objects it fills the ring buffer with values in the interval $[-$`prebuffer`$, 0]$.

1.3.6 `SampledWave`: plane GWs from equispaced arrays of samples

`SampledWave(hparray,hcarray,len,`Δt`,prebuffer,`$\mathcal{N}$`,filter,interp,`β`,`λ`,`ψ`)`

Returns a `Wave` object implementing a sampled plane GW incoming from ecliptic latitude β, longitude λ, with polarization parameter ψ. The 1-D `Numeric` double-precision arrays `hparray` and `hcarray`, of length `len` and spacing Δt [s] contain time series for h_+ and $h_\times$, offset so that, e.g., $h_+(t) =$ `hparray`$[(t -$ `prebuffer`$)/\Delta t]$. The parameter `interp` (> 1) sets the semiwidth of the data window used in the Lagrange interpolation of the samples (1 yields linear interpolation).

In addition, the sampled time series is multiplied by the normalization constant $\mathcal{N}$ and filtered by the *Synthetic LISA* `Filter` object `filter` (this can be set to 0 if no filtering is requested).

Synthetic LISA implements `SampledWave` internally using `NoiseWave`, after creating sampled `SampledSignal` objects. [`SampledWave` replaces `InterpolateMemory`, which is still supported (but undocumented) for backward compatibility.]

The offsetting governed by `prebuffer` is useful because $h_+(t)$ and $h_\times(t)$ are defined in terms of SSB time, so the polarization values needed to build (say) the TDI observable X at $t = 0$ may require accessing the value of h_+ and $h_\times$ at SSB times $t \simeq -500$ s.

??? What is the exact prebuffering formula?

1.3.7 `PyWave`: generic plane GWs with Python-defined h_+ and $h_\times$

`PyWave(hpfunc,hcfunc,`β`,`λ`,`ψ`)`

Returns a `Wave` object implementing a generic plane GW incoming from ecliptic latitude β, longitude λ, with polarization parameter ψ, and with polarizations $h_+(t)$ and $h_\times(t)$ [with t in seconds] corresponding to the output of the Python functions hpfunc and hcfunc.

Defining $h_+(t)$ and $h_\times(t)$ as Python functions incurs a considerable runtime overhead in the conversions between C++ and Python types, but it may be the simplest way to create new `Wave` types, at least for prototyping purposes. Consider for instance the

following definition, which can be used to access (via `putwave`) the $\mathbf{e}_+$ polarization tensor of a `Wave` object:[4]

```
>>> def myhp(t):
...     return 1
...
>>> def myhc(t):
...     return 0
...
>>> wave = synthlisa.PyWave(myhp,myhc,math.pi/3,math.pi/2,0)
>>> wave.putwave(0)
((1.0, 0.0, 0.0), (0.0, -0.75, 0.43), (0.0, 0.43, -0.25))
```

1.3.8 Writing user-defined `Wave` classes

If more flexibility or performance is needed than allowed by `SampledWave` or `PyWave`, users can implement their own C++ derived `Wave` classes: the only requirement is that the derived class define the methods `hp(`t`)` and `hc(`t`)` [with t in seconds]. Users might find it expedient to modify the code for `SimpleBinary`, found in the source file `lisasim-wave.cpp`.

1.3.9 `WaveArray`: superpositions of `Wave`s

`WaveArray(waves[])`

Returns a `WaveArray` object, which represents the linear superposition of a set of `Wave` objects, each of which can be of a different kind, and can have a different sky position and polarization angle. Here `waves[]` is a Python list or tuple of `Wave` objects.

Note that if `WaveArray` is initialized with a Python list,[5] successive modifications to the list will not be mirrored in the contents of the `WaveArray`, and could result in single `Wave` objects being deallocated when they should not be.

1.4 The `Noise` class and the LISA noise response

In *Synthetic LISA*, the response of the TDI observables to the fundamental LISA noises is obtained from the class `TDInoise`, which inherits from the base class `TDI` the definitions of the observables in terms of the basic Doppler responses, but which also defines the methods `y` and `z` for y_{slr} and z_{slr} according to Eqs. (8) and (9) of Ref. [1]. A `TDInoise` object is created by specifying a LISA geometry object (Sec. 1.1) and a

[4] In the following code snippet, I have reduced the digits printed by the Python interpreter for the result of `putwave`.

[5] This problem does not occur for initialization with a Python tuple, which is *immutable*.

prescription for the 18 fundamental noise variables discussed in Ref. [1]. In *Synthetic LISA*, fundamental noises are represented by derived classes of the base class `Noise`.

All the `Noise` objects available in *Synthetic LISA* are based on the following three-stage model, implemented in `InterpolateNoise`; the syntax of the corresponding constructors is given in Table 1.3.

1. *Generation.* In the first stage, a sequence of noise samples with user-defined sampling timestep Δt is obtained as a user-provided array (e.g., from experimental measurements), or by means of a pseudorandom number generator. *This noise is always understood to be bandlimited at the Nyquist frequency* $f_c = (2\Delta t)^{-1}$; thus, it is completely specified (at any time between the samples) by the sequence of samples. *Synthetic LISA* employs Luescher's lagged Fibonacci generator [8], as implemented in the *GNU Scientific Library*,[6] to generate independent, uniformly distributed deviates; the uniform deviates are then converted into Gaussian deviates by means of the Box–Muller transform [9]. The result is white Gaussian noise with zero mean and unit variance.

2. *Filtering.* In the second stage, the noise sequences are filtered, if so desired, through simple digital time-domain filters to reshape their power spectra. The current version of *Synthetic LISA* provides the finite-difference filter $y[n\Delta t] = x[n\Delta t] - x[(n-1)\Delta t]$ (with x the original noise sequence), which has the power transfer function $|1-\exp(2\pi i f\Delta t)|^2 = 4\sin^2(\pi f\Delta t)$, and the damped-integrator filter $y[n\Delta t] = \alpha y[(n-1)\Delta t]+x[n\Delta t]$, which has power transfer function $(1/4)\sin^{-2}(\pi f\Delta t)$ in the limit $\alpha \to 1$ (the damping, set by default to $\alpha = 0.9999$, is needed to control the DC component of the integrated noise).

   In *Synthetic LISA*, pseudorandom white noise filtered through the finite-difference and damped-integrator filters is used to approximate the standard optical-path and proof-mass noises defined in Ref. [1], which have spectral densities proportional to $f^{\pm 2}$ (rather than $[4\sin^2(\pi f\Delta t)]^{\pm 1}$). The approximation is good if f_c is chosen sufficiently higher than the highest frequency at which one wishes to analyze the TDI noise responses (but of course lower than the Nyquist frequency used to sample the TDI observables, to avoid aliasing).

3. *Interpolation.* In the third stage, the samples are normalized and interpolated to provide the properly dimensioned value of the noise process at arbitrary times (not necessarily multiples of Δt), as required by the TDI observables. Linear and higher-order Lagrange interpolation schemes are available in *Synthetic LISA*. The frequency response of the interpolation procedure is crucial to the simulation and is discussed in detail in Sec. 1.6.2. The basic fact is that interpolation spreads the power of the bandlimited sampled noise to frequencies higher than f_c; some of this power will alias back into the spectrum of the TDI observables, because these are built in the time domain with exact time delays, and thus they can pick up the spurious high-frequency noise components. It is then essential to control the frequency response of interpolation, and to choose the sampling timestep, for both the fundamental noises and TDI, as needed to reduce aliasing.

[6] GNU Scientific Library, v. 1.4 (2003): `www.gnu.org/software/gsl`.

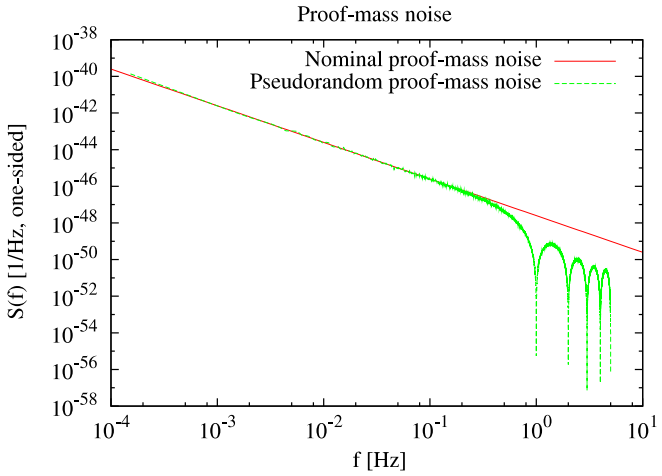

Figure 1.4: Pseudorandom proof-mass noise generated by `InterpolateNoise(1.0,256.0,2.5e-48,-2.0,1)` (dashed), and ideal proof-mass–noise spectral density (solid). The spectrum was produced by the *Synthetic LISA* example script `test-proofnoise`.

1.5 LISA TDI observables

By way of the base class `TDI`, the `TDIsignal` and `TDInoise` classes define the methods *obs*`(`*t*`)`, which return the value of the TDI observables (first generation, modified, and second generation) listed in Tab. 1.4. The user can define additional observables by adding new definitions to the `TDI` class (in C), or by using explicit combinations of the `y` and `z` methods (in Python).

1.6 Tests and applications of Synthetic LISA

In this section we present the results of simple tests and experiments performed with *Synthetic LISA*: in Secs. 1.6.1 and 1.6.2 we examine the spectral densities of the fundamental noises; in Sec. 1.6.3 we compute noise spectra for the TDI observables; in Sec. 1.6.4 we examine monochromatic GW signals as observed with LISA. The *Synthetic LISA* distribution includes all the scripts that were used to run these tests and simulations. See Ref. [1] for more simulations dealing with important issues in the implementation of TDI.

1.6.1 Standard LISA noises

As a first check on the implementation of *Synthetic LISA*, we use the `InterpolateNoise` class to generate pseudorandom series of the LISA fundamental noises with the stan-

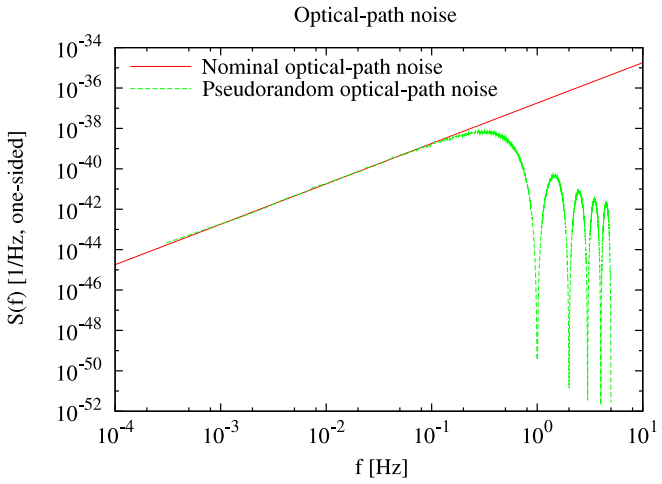

Figure 1.5: Pseudorandom optical-path noise generated by `InterpolateNoise(1.0,256.0,1.8e-37,2.0,1)` (dashed), and ideal optical-path–noise spectral density $S_n^{\mathrm{op}} = 1.8 \times 10^{-37}[f/\mathrm{Hz}]^2\mathrm{Hz}^{-1}$ (solid). The spectrum was produced by the *Synthetic LISA* example script `test-optnoise`.

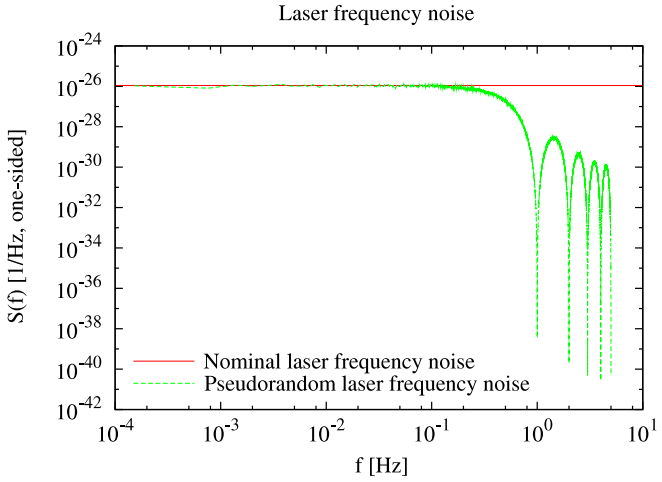

Figure 1.6: Pseudorandom laser frequency noise generated by `InterpolateNoise(1.0,256.0,1.1e-26,0.0,1)` (dashed), and ideal laser-frequency–noise spectral density $S_n^{\mathrm{ls}} = 1.1 \times 10^{-26}\,\mathrm{Hz}^{-1}$ (solid). The spectrum was produced by the *Synthetic LISA* example script `test-lasernoise`.

dard spectral densities described in Ref. [1], and we compute their spectra. Figures 1.4–1.6 show our results for proof-mass noise, optical-path noise, and laser frequency noise respectively. For these tests, we generated pseudorandom noise with a timestep of 1 s (and therefore an ideal bandlimit frequency $f_b = 0.5$ Hz), and we resampled the process to a timestep of 0.1 s (and therefore a Nyquist frequency $f_N = 5$ Hz) using linear interpolation.

We see that the power spectra of the pseudorandom time series (dashed) adhere rather faithfully to the theoretical curves (solid), except at frequencies comparable to f_b, where the effect of interpolation is that noise power is not cut off sharply, but rather drops off smoothly (if rapidly), with nulls at the harmonics[7] of f_b; interpolation is discussed further in the next section. For the optical-path and proof-mass noises, the effect of interpolation is compounded by the effect of the finite-difference and finite-integration time-domain filters, whose transfer function is proportional to $\sin^{\pm 2}[\pi f/(2f_b)]$.

1.6.2 Frequency response of noise interpolation

In *Synthetic LISA*, pseudorandom noise is created by generating a sequence of uncorrelated Gaussian deviates, which are then interpreted as the sampled values at times $t_n = n\Delta t$, for $n = 0, 1, \ldots$ of a continuous random noise process. The process is assumed to be bandlimited below[8] $f_b = 1/(2\Delta t)$: by the sampling theorem (see, e.g., Ref. [10]), the noise process can then be reconstructed perfectly at any intervening time t by convolving the sampled sequence with the interpolating kernel

$$\mathrm{sinc}[\pi(t - t_n)/\Delta t] = \frac{\sin[\pi(t - t_n)/\Delta t]}{[\pi(t - t_n)/\Delta t]}. \tag{1.16}$$

Since the sinc kernel has infinite extent, it is of limited utility; however, a vast class of pratical interpolation schemes, including the linear and Lagrange interpolators implemented in *Synthetic LISA*, can be formulated as the convolution of the sampled sequence with an interpolating kernel that is in some sense an approximation to a sinc.

The tradeoff in the approximation is between the number of samples used to interpolate and the sharpness of the spectral response. The correct sampling of a bandlimited process preserves all the spectral information below the Nyquist frequency, but it populates Fourier space with infinitely many exact replicas of the original spectrum, centered at frequencies $k/\Delta t$, for $k = \pm 1, \pm 2, \ldots$ The effect of sinc interpolation is to multiply this composite spectrum by the Fourier transform of the sinc, which is a perfect square window of height 1 and width $1/\Delta t$, centered at $f = 0$. Thus, sinc

[7] It is interesting to notice that for the f^2 optical-path noise (Fig. 1.5) the locations of the nulls conspire to reduce the aliasing of power to low frequencies, where the nominal f^2 power is lower than the envelope of interpolation-error power above f_b.

[8] The application of time-domain filters does not change this picture: since the filters are linear, they do remodel the initially white spectrum, but the resulting time series can still be interpreted as the sampled version of a bandlimited continuous process.

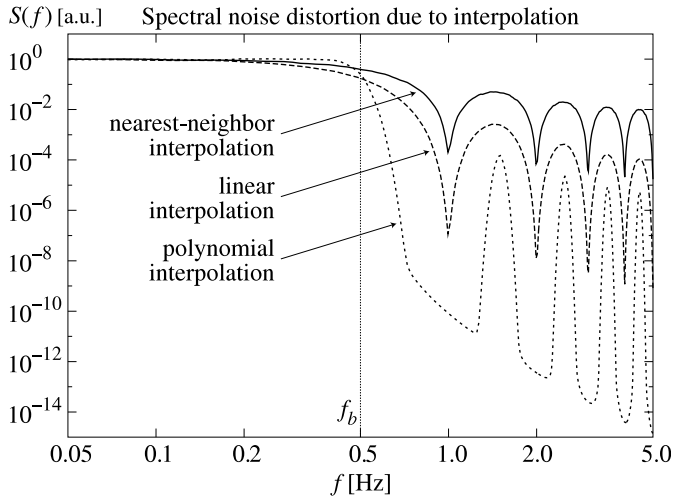

Figure 1.7: Noise distortion for 0.5 Hz-bandlimited white noise, with different interpolation schemes. Higher-order interpolation corresponds to a sharper transition at the bandlimit frequency and to lower ripples and deeper valleys between them (in this graph, the valleys are cut off by spectral leakage from the main platform at power = 1). The spectra were produced by the *Synthetic LISA* example script `test-interpolation`.

interpolation achieves perfect signal reconstruction by selecting only the original spectrum and deleting all unwanted replicas. Practical schemes with kernels of finite extent cannot have such sharp frequency response, so they distort (amplify or suppress, depending on frequency) the original spectral content in the passband below f_b, and they allow some of the power of the unwanted spectral replicas to creep back into the interpolated process (either directly, if the process is sampled with a sufficiently high Nyquist frequency, or by aliasing to frequencies in the passband).

These effects can be observed in Fig. 1.7, which shows the spectrum of pseudorandom white noise, generated with a timestep of 1 s, and resampled to a timestep of 0.1 s, using no interpolation (i.e., defaulting to the nearest 1-s sample), linear interpolation, and Lagrange-polynomial interpolation of order 4, 8, and 32. In all cases, power begins to drop before the nominal bandlimit frequency of 0.5 Hz, but the drop is sharper for higher-order interpolation methods. Spurious power above the bandlimit frequency appears as *ripples* between the f_b harmonics: the height of the ripples decreases with interpolation order, while the valleys among them become wider. In Fig. 1.7, the valleys appear to be cut off by a common downgrading envelope; this is an artifact of spectral estimation due to the residual leakage from the platform below the passband; spectral leakage also smears out the nulls at the f_b harmonics to a finite height.

We conclude that the pseudorandom noise sequences generated by *Synthetic LISA* can be accurate representations of the idealized LISA noises defined in Ref. [1], and therefore can be used to study the noise response of the TDI observables, as long as we take into account the effect of interpolation at frequencies comparable to f_b. Using linear interpolation (the default), it is probably safe to draw conclusions for frequencies $\lesssim f_b/5$; using higher-order interpolation it becomes possible to push our inferences to higher frequencies (however, when using our pseudorandom secondary noises, one should also be mindful of the fact that their spectral shape near f_b is $\sin^{\pm 2} \pi f/(2f_b)$ rather than $f^{\pm 2}$). The arguments made in this section apply also to the interpolated noise obtained from user-provided sampled-noise sequences, as long as the sampled noise can be considered bandlimited below its nominal Nyquist frequency.

See Ref. [11] for a discussion of the use of interpolation in reconstructing the TDI observables on Earth from the y_{slr} and z_{slr} data, sampled onboard at a limited rate that can be transmitted affordably to Earth.

1.6.3 Secondary noises in the TDI observables

We now move on to full-fledged TDI, and we examine the spectra of the first-generation TDI observables for a stationary LISA configuration (`OriginalLISA`), with the purpose of verifying our results against analytical expectations, and of providing an explicit demonstration of laser-noise cancellation.

Figure 1.8 shows the spectrum of X in the case of equal, nominal arms ($L = 16.6782$ s); the solid line plots the analytical expression (4.1) of Ref. [12], while the dashed line plots the spectrum of *Synthetic LISA*'s pseudorandom time series. The agreement is excellent, except at frequencies $\simeq f_b$ (0.5 Hz), where the effects of linear interpolation and of the finite-difference filter become noticeable. [For both curves, the nulls at the armlength-travel frequencies k/L have finite depth, for two different reasons: for the

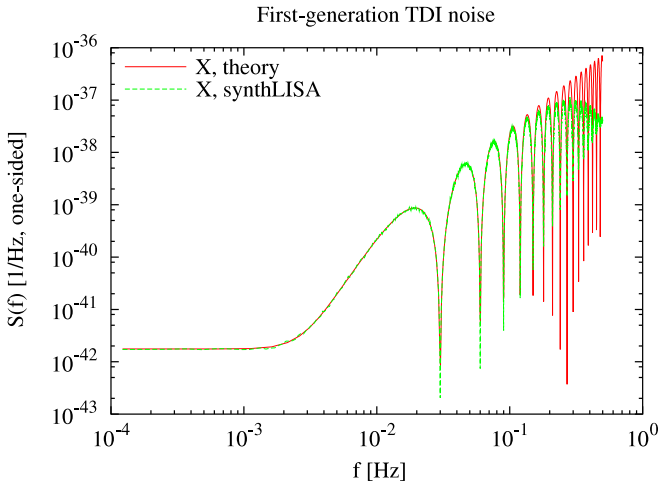

Figure 1.8: Spectral density of the first-generation TDI observable X for a stationary, equal-arm interferometer with $L_i = 16.6782$ s (solid curve), as compared with the analytical expression (4.1) of Ref. [12] (dashed). At the high-frequency end of the plot, the *Synthetic LISA* curve is suppressed by the effects of linear noise interpolation and of the finite-difference time-domain filter; the envelope traced out by the nulls is an artifact of spectral leakage and of the number of points used to draw the theoretical curve. These spectra were produced by the *Synthetic LISA* example script `test-tdiequal-X`.

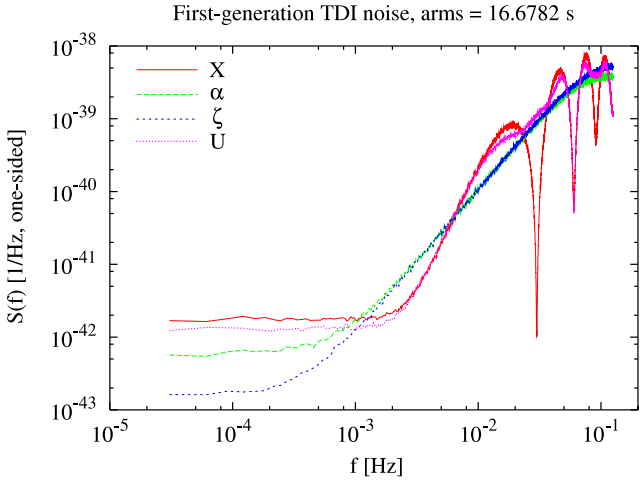

Figure 1.9: Spectral densities of the first-generation TDI observables X, α, ζ, and U, for a stationary, equal-arm interferometer with $L_i = 16.6782$ s. Compare with Fig. 1 of Ref. [13]. The spectra are suppressed at high frequencies by the effects of linear noise interpolation and of the finite-difference time-domain filter. The spectra were produced by the *Synthetic LISA* example script `test-tdiequal`.

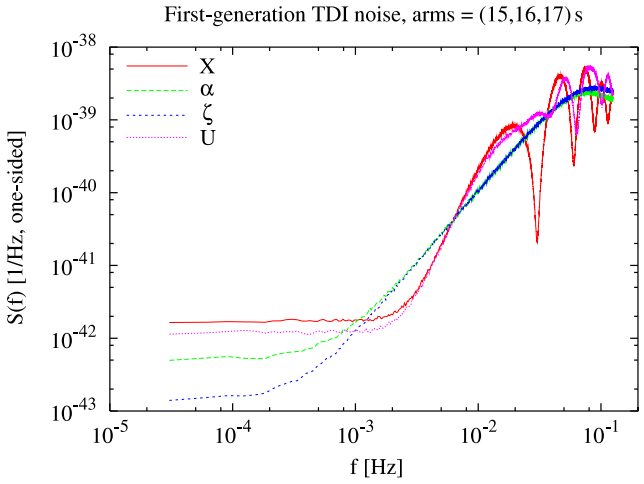

Figure 1.10: Spectral densities of the first-generation TDI observables X, α, ζ, and U, for a stationary, unequal-arm interferometer with $L_1 = 15$ s, $L_2 = 16$ s, $L_3 = 17$ s. Compare with Fig. 1.9. The spectrum was produced by the *Synthetic LISA* example script `test-tdiunequal`.

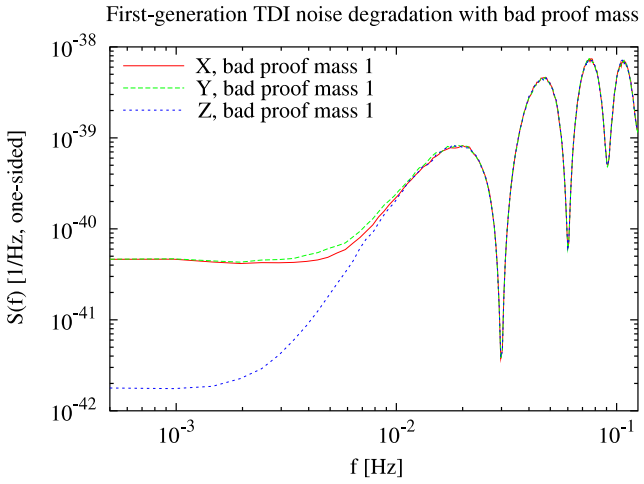

Figure 1.11: Spectral density of the first-generation TDI observables X, Y, and Z in a stationary, equal-arm interferometer with $L = 16.6782$ s, for standard noises (solid), and for $\times 10$ noise in proof mass 1. The spectra were produced by the *Synthetic LISA* example script `test-tdibadmass`.

analytical expression, because the finite set of frequencies used to plot the curve does not include k/L; for the *Synthetic LISA* time series, because of spectral leakage.]

Figure 1.9 compares the spectra of various first-generation TDI observables, again for an equal-arm configuration: this plot compares favorably with Fig. 1 of Ref. [13].

In producing both Figs. 1.8 and 1.9 (and generally for all the spectra shown in this paper, except where otherwise specified), laser frequency noise at the standard nominal level of $1.1 \times 10^{-26}\,\mathrm{Hz}^{-1}$ was explicitly included in the simulation; thus, these plots confirm that for stationary LISA configurations the first-generation TDI observables cancel frequency laser noise to the level of the secondary noises, and by up to 160 dB. In fact, if we were to show the spectra of the basic y_{slr} measurements, these would appear in these figures as straight lines at the $10^{-26}\,\mathrm{Hz}^{-1}$ level. Prior to this work, laser-noise cancellation had been simulated explicitly only by setting the LISA armlength to a multiple of the sampling time [14], which removes the need to interpolate noise. Of course, the simulation of laser-noise cancellation is much more interesting for realistic LISA geometries that include the rotation and flexing of the array: such tests are presented in the next section.

Last, Figs. 1.10 and 1.11 are meant to exemplify the ease with which *Synthetic LISA* can probe the effects of modified LISA configurations. Figure 1.10 shows noise spectra for the first-generation TDI observables in a stationary, *unequal-arm* interferometer: observe the resulting distortion of the noise curves with respect to Fig. 1.9. Figure 1.11 shows X, Y, and Z noise spectra for bad ($\times 10$ rms) noise in proof mass 1; in this example, first shown in Ref. [13], the comparison of the three eight-pulse TDI

observables pinpoints the origin of the excess noise to the faulty proof mass.

1.6.4 GW signals

Synthetic LISA can synthesize the TDI responses to arbitrary GW sources,[9] either by interpolating user-provided h_+ and $h_\times$ time series (using the `Wave` class `InterpolateMemory`), or by incorporating custom-coded `Wave` classes (see Tab. 1.2). The first approach was used by the author in a study [15] of detection prospects for the GW signals from compact stellar objects inspiraling into the supermassive black holes at the centers of galaxies: time series for h_+ and $h_\times$ were produced using the Glampedakis-Hughes-Kennefick quasiadiabatic orbit integrator [16], and then fed to *Synthetic LISA*, which computed the corresponding time series of TDI observables; these were used to derive the expected S/Ns for the capture sources.

In this paper we limit our tests to GW signals from monochromatic binary sources, which already demonstrate the full effects of frequency and amplitude modulation, and for which accurate analytical expressions are available [4].

In the following, we characterize the closeness of two signals $s_1(t)$ and $s_2(t)$ by computing (one minus) the normalized correlation product[10]

$$\frac{\int s_1(t) s_2(t)\, dt}{\int |s_1(t)|^2\, dt}. \tag{1.17}$$

Królak and colleagues [4] give analytical expressions valid at all frequencies for the GW responses of the first- and second-generation TDI variables; these expressions were obtained using the same approximated LISA orbits as implemented in the `LISA` class `CircularRotating` (see Sec. 1.1), but disregarding the causal laser-beam *look-ahead* implicit in Eq. (3) of Ref. [1], so that all six $L_l(t)$ stay equal and constant. Multiple checks conducted for monochromatic binaries with several GW frequencies between 10^{-4} and 10^{-1}, and with several random values of source latitude, longitude, and polarization, show that the TDI responses computed by *Synthetic LISA* (with `CircularRotating`) always match the theoretical expressions of Ref. [4] to a few parts in 10^4 [i.e., Eq. (1.17) is of order one minus a few 10^{-4}].

Running *Synthetic LISA* with the more realistic LISA orbits implemented in `EccentricInclined`, the matching to the theoretical expressions of Ref. [4] is still equally good at low frequencies, but the error grows to a few parts in 10^3 at $f \simeq 10^{-2}$ Hz, and to one percent or more at $f \simeq 10^{-1}$ Hz. These numbers characterize the accuracy of the approximation used in Ref. [4] (see also Ref. [17] for a discussion of the closely related *rigid adiabatic* approximation).

[9] As long as they can be written as superpositions of plane waves with definite propagation vectors.
[10] Note that the denominator of Eq. (1.17) contains the norm of only one of the two signals; thus, this correlation product is an index of amplitude matching as well as phase matching.

1.7 Synthetic LISA usage reference

Tables 1.1–1.4 specify the syntax of the class constructors used to create the *Synthetic LISA* array-geometry (`LISA`), GW (`Wave`), fundamental-noise (`Noise`), and TDI (`TDI`) objects. In C++, these constructors can be used as shown in class declarations ("`OriginalLISA lisa(16,16,16);`") or in the `new` construct ("`LISA *plisa = new OriginalLISA(16,16,16);`"); in Python, they are called directly to return a class reference ("`lisa = OriginalLISA(16,16,16)`"). Tables 1.1–1.4 also describe the class methods available in C++ and Python, which can be accessed in both languages using the standard `class.method()` syntax ("`L = lisa.armlength(0)`"). [Under the hood, the *Synthetic LISA* C++ calls are interfaced to Python by wrappers generated automatically by SWIG.[11] C++ arrays are mapped to `Numeric`[12] array objects in Python.] Additional methods are available from C++, but they are likely to change in future versions of *Synthetic LISA* and should not be considered part of the standard package interface.

In the following, a `fixed-length font` indicates C++/Python methods and objects; by contrast, except where otherwise indicated, the arguments given in *italics* or as Greek letters are either integer indexes (C++ `int`s) or real quantities (C++ `double`s). When function parameters are indicated with a default value ("`CircularRotating(`$\eta_0 = 0$`,`$\xi_0 = 0$`,`$sw = 1$`,`$t_0 = 0$`)`"), they can be omitted, starting from the right, according to C++ custom.

Table 1.1: `LISA` objects

	Constructors	
`OriginalLISA(`L_1`,`L_2`,`L_3`)`	L_1, L_2, L_3 [s]:	armlengths
`ModifiedLISA(`L_1`,`L_2`,`L_3`)`	L_1, L_2, L_3 [s]:	armlengths
`CircularRotating(`L`,`η_0`,`ξ_0`,`sw`,`t_0`)`, `CircularRotating(`$\eta_0 = 0$`,`$\xi_0 = 0$`,`$sw = 1$`,`$t_0 = 0$`)`	L [s]:	common armlength before aberration; se 16.6782 s in the second form of the construct
	η_0 [rad]:	true anomaly of LISA guiding center at $t = t$
	ξ_0 [rad]:	initial phase of LISA array rotation at $t = t_0$
	sw:	set positive (negative) value for clockwise (c terclockwise) s/c sequence $1 \to 2 \to 3 \to 1$ seen from $z > 0$)
	t_0 [s]:	reference time for LISA orbital parameters (s 0 implicitly in Sec. 1.1)
`EccentricInclined(`$\eta_0 = 0$`,`$\xi_0 = 0$`,`$sw = 0$`,`$t_0 = 0$`)`	η_0, ξ_0 [rad]:	approximately as in `CircularRotating` see 1.1 for correspondence to the LISA Simulator
	sw:	set negative to exchange spacecraft 2 and 3
	t_0 [s]:	reference time for LISA orbital parameters

[11] SWIG website: `www.swig.org`.
[12] Numerical Python project website: `sourceforge.net/projects/numpy`.

Table 1.1: `LISA` objects, continued

`NoisyLISA(lisa,`Δt`,`S_n`)`		*returns a* `LISA` *object based on* `lisa`*, but uncorrelated Gaussian, white noise processes turbing the six nominal armlengths used to pute the TDI delays (but not the physical lengths used to model laser-beam propagation*
	Δt [s]:	generation timestep of the pseudorandom length noise
	S_n [1/Hz]:	one-sided power spectral density of armle noise
	METHODS	
`armlength(`*arm*`,`t`)`		returns the light-propagation length $L_{arm}(t$ seconds) experienced along *arm* for recep at time t (seconds); for `CircularRotating` `EccentricInclined`, the armlength is appr mated by Eqs. (1.3) and Eq. (1.6), respective
`putp(p,`*spacecraft*`,`t`)` [C++ only]		fills the `Vector` object `p` with the position of *sp craft* (given in SSB ecliptic coordinates, in ometrized units of seconds) at time t (second
`putn(n,`*arm*`,`t`)` [C++ only]		fills the `Vector` object `n` with the pho propagation unit vector (in SSB ecliptic coo nate components) for transmission along *arm* reception at time t (seconds)
`retardation(lisa,`$d_1,\ldots,d_8$`,`t`)` [helper function]		returns the retarded time $t_{;\{d_i\}}$ (seconds); to fewer than eight delays, set $d_i = 0$ from the r

Table 1.2: `Wave` objects

	CONSTRUCTORS	
`InterpolateMemory(hparray,hcarray,` *samples*`,`Δt`,`*lookback*`,`β,λ,ψ`)`	`hparray`, `hcarray`:	pointers to C `double` arrays (or references `Numeric 'd'` arrays) containing the h_+ and sequences
	samples:	total number of samples in the h_+ and $h_\times$ quences
	Δt [s]:	time spacing of the samples
	lookback [s]:	displaces the time indexing of the arrays so t the nth elements correspond to the time[a] $n\Delta t - lookback$
	β, λ, ψ [rad]:	source ecliptic latitude, ecliptic longitude, and larization angle; see Sec. **??** for correspondenc the LISA Simulator [6]
`SimpleBinary(`$f,\ \phi_0,\ \iota,\ A,\ \beta,\ \lambda,\ \psi$`)`	f [Hz]:	GW frequency, as observed in the SSB frame

Table 1.2: `Wave` objects, continued

	ϕ_0 [rad]:	phase at $t = 0$
	ι [rad]:	binary inclination angle
	A [strain]:	common h_+ and $h_\times$ amplitude
	β, λ, ψ [rad]:	as in `InterpolateMemory`
User-defined `Wave` classes		*constructors should call the base class constru* `Wave(`β`,`λ`,`ψ`)`; *the new class must define the m ods* `hp(t)` *and* `hc(t)`. *Users might find it exped to modify the code for* `SimpleBinary`, *found in source file* `lisasim-wave.cpp`
	METHODS	
`hp(`t`)`, `hc(`t`)`		return GW polarizations $h_+(t)$ and $h_\times(t)$ at t t (seconds)

[a] This offset is inserted because the expressions for the TDI observables contain two kind of delays: the first kind corresponds to retarding (or advancing) the phase of the plane GW to the current position of the LISA array (for the baseline LISA orbit this offset can be up to R, or ~ 500 s, depending on $\hat{k}$); the second kind correspond to displacing the basic Doppler observables by one or more armlengths, as required by the TDI observables (thi offset can be up to $8 \times L \simeq 8 \times 16.7$ s, for second-generation TDI and for the baseline LISA array formation) Thus, computing the TDI responses to a GW at time $t = 0$ can involve the values of h_+ and $h_\times$ at times as early as $t \simeq -635$ s, or as late as $t \simeq 500$ s. To compute the LISA response in the time interval $[0, T]$, the user should provide at least $1 + (T + 2R + 8L)/\Delta t$ samples [$\sim 1 + (T + 1135\,\mathrm{s})/\Delta t$ for the baseline LISA configuration], and set *lookback* to at least $(\Delta t + R + 8L)$.

Table 1.3: `Noise` objects

	CONSTRUCTORS	
`InterpolateNoise(noisearray,`*samples*`,` Δt`,` *prebuf*`,`*norm*`,`*filter* $= 0$`,`*wind* $= 1$`)`	`noisearray`:	pointer to a C `double` array (or reference t `Numeric 'd'` array) containing the raw noise quence
	samples:	total number of samples in the noise sequence
	Δt [s]:	time spacing of the samples
	prebuf [s]:	displaces the indexing of the array so that the element corresponds[a] to the time $t = n\Delta t - pr$ (with $n = 0, \ldots,$ *samples* $- 1$)
	norm:	multiplies the raw noise sequence
	filter:	sets filtering of the raw noise sequence ($-$ damped-integrator filter; 2.0: finite-difference ter; 0.0: no filtering)
	wind [> 1]:	sets semiwidth of the data window used in grange interpolation (a value of 1 yields linear terpolation)

Table 1.3: `Noise` objects, continued

`InterpolateNoise(`Δt`,`*buf*`,`S_n`,`e_n`,` *wind* `=` `1)`	Δt [s]:	time spacing of the pseudorandom samples at g eration
	buf [s]:	sets the minimum internal buffering of the ps dorandom sequences: after the noise has been quested at time t, the earliest noise value guar teed to be available will be at time[b] $t - buf$
	S_n, e_n:	sets the one-sided power spectral density of pseudorandom noise to $S_n \times f^{e_n}$, where $e_n = -$ 0.0, and 2.0 are currently implemented (S_n is gi in units of Hz, Hz^{-1}, and Hz^{-3}, respectively)
	wind [> 1]:	sets semiwidth of the data window used in grange interpolation (a value of 1 yields lir interpolation)
	METHODS	
`noise(`t`)`		returns the filtered, normalized value of the n at time t (seconds)
`reset()`		resets the `Noise` object to time $t = 0$ by resett buffer counters or reseeding pseudorandom nu ber generators

[a] The offset is needed because the LISA noises enter the TDI observables with delays up to four (for first generation TDI) or eight (for second-generation TDI) armlengths; thus, if the user will be computing the LISA noise response in the time interval $[0, T]$, they should provide at least $1 + (T + 8L)/|\Delta t|$ noise samples in each noise object ($\sim 1 + (T + 136\,\mathrm{s})$ for the baseline LISA configuration).
[b] Again, it should be set to $4L$ (for first-generation TDI) or $8L$ (for second-generation TDI).

Table 1.4: `TDI` objects

	CONSTRUCTORS	
`TDIsignal(lisa,wave)`	`lisa`:	C++ pointer (or Python reference) to `LISA` ge etry object used to compute GW responses
	`wave`:	C++ pointer (or Python reference) to GW ob
`TDInoise(lisa,`Δt^{pm}`,`S^{pm}`,`Δt^{op}`,`S^{op}`,` Δt^{ls}`,`S^{ls}`)`	`lisa`:	C++ pointer (or Python reference) to `LISA` ge etry object used to compute noise responses
	$\Delta t^{\mathrm{pm}}, \Delta t^{\mathrm{op}}, \Delta t^{\mathrm{ls}}$ [s]:	set the time spacings used to generate the 18 proof mass, six optical path, six laser) stand pseudorandom noises sequences (done intern with `InterpolateNoise` and $wind = 1$)
	S^{pm} [Hz]:	sets the one-sided power spectral density of the proof-mass noises to $S^{\mathrm{pm}} \times f^{-2}$

Table 1.4: `TDI` objects, continued

	$S^{\rm op}$ [Hz^{-3}]:	sets the one-sided power spectral density of the optical-path noises to[a] $S^{\rm op} \times f^2$
	$S^{\rm ls}$ [Hz^{-1}]:	sets the one-sided power spectral density of the laser frequency noises to $S^{\rm ls}$
`TDInoise(lisa,`$\Delta t^{\rm pm}$`[6],`$S^{\rm pm}$`[6],`$\Delta t^{\rm op}$`[6],` $S^{\rm op}$`[6],`$\Delta t^{\rm ls}$`[6],`$S^{\rm ls}$`)[6]`		as above, but takes parameters as C `dou` arrays (or Python *sequences*) to set the tributes of the six noises of each type se rately; the ordering of the parameters co sponds to $\{pm_1, pm_1^*,\ pm_2, pm_2^*, pm_3, pm$ $\{y_{12}^{\rm op}, y_{21}^{\rm op}, y_{23}^{\rm op}, y_{32}^{\rm op}, y_{31}^{\rm op}, y_{13}^{\rm op}\}$, $\{C_1, C_1^*, C_2, C_2^*, C_3,\ C_3^*\}$
`TDInoise(lisa,proofnoise[6],` `opnoise[6], lasernoise[6])`	`lisa`:	as above
	`proofnoise[6]`:	array of C++ pointers (or Python sequence references) to six `Noise` objects used to prov $\{pm_1, pm_1^*,\ pm_2, pm_2^*, pm_3, pm_3^*\}$
	`opnoise[6]`:	array of C++ pointers (or Python sequence references) to six `Noise` objects used to prov $\{y_{12}^{\rm op}, y_{21}^{\rm op}, y_{23}^{\rm op}, y_{32}^{\rm op}, y_{31}^{\rm op}, y_{13}^{\rm op}\}$
	`proofnoise[6]`:	array of C++ pointers (or Python sequence references) to six `Noise` objects used to prov $\{C_1, C_1^*, C_2, C_2^*, C_3,\ C_3^*\}$
	METHODS	
`y(`s`,`l`,`r`,`d_1`,`d_2`,...,`d_7`,`t`)`		basic LISA inter-spacecraft Doppler respo $y_{slr;d_1\ldots d_n}$(t) [emission at spacecraft s, transm sion through arm l, and reception at r, retar by the armlengths d_n to d_1] at time t (seconds) use fewer than seven delays, set $d_i = 0$ from right[b]
`z(`s`,`l`,`r`,`d_1`,`d_2`,...,`d_8`,`t`)`		basic LISA intra-spacecraft Doppler respo $z_{slr;d_1\ldots d_n}$(t) [made on the same bench as y_{slr} time t (seconds); to use fewer than eight dela set $d_i = 0$ from the right
`alpha(`t`)`, `beta(`t`)`, `gamma(`t`)`, `zeta(`t`)`, `X(`t`)`, `Y(`t`)`, `Z(`t`)` `P(`t`)`, `E(`t`)`, `U(`t`)`		return first-generation TDI observables α, β ζ, X, Y, Z, P, E, and U at time t (seconds) defined in Refs. [18, 12]
`alpha1(`t`)`, `alpha2(`t`)`, `alpha3(`t`)`, `Xm(`t`)`, `Ym(`t`)`, `Zm(`t`)`		return the modified TDI observables α_1, α_2, X, Y, Z, as defined[c] in Refs. [2, 19]
`X1(`t`)`, `X2(`t`)`, and `X3(`t`)`		return the second-generation TDI observables X_2, and X_3, as defined in[c] Refs. [19, 2]

Table 1.4: `TDI` objects, continued

`reset()` [`TDInoise` only]	resets all `Noise` objects used in `TDInoise`
`lock(`*master*`)` [`TDInoise` only]	phase-locks all six lasers to a master laser (*ma* $= 1, 2, 3$ for $C_{1,2,3}$ and *master* $= -1, -2, -3$ $C^*_{1,2,3}$); `Xmlock1(`t`)`, `Xmlock2(`t`)`, and `Xmlock3` then provide simplified (two-way–one-way hyb TDI expressions [20]
`stdproofnoise(lisa,`$\Delta t^{\rm pn}$`,`$S^{\rm pn}$`)`, `stdopticalnoise(lisa,`$\Delta t^{\rm op}$`,`$S^{\rm op}$`)`, `stdlasernoise(lisa,`$\Delta t^{\rm ls}$`,`$S^{\rm ls}$`)` [helper functions]	return C++ pointers (or Python references) pseudorandom `InterpolateNoise` objects w time spacings $\Delta t^{\rm pn}$, $\Delta t^{\rm op}$, and $\Delta t^{\rm ls}$ and one-si power spectral densities $S^{\rm pn} \times f^{-2}$, $S^{\rm op} \times f^2$, $S^{\rm ls}$; the `LISA` object `lisa` is used to choose a su ciently long noise buffer to accommodate seco generation TDI time delays

[a] Since the finite-difference time-domain filter has a spectral transfer function proportional to $\sin^2(\pi f \Delta t^{\rm op})$ the spectrum of the synthesized optical-path noise will only be accurately proportional to f^2 at relatively lo frequency [with respect to the Nyquist frequency *of noise generation* $(2\Delta t^{\rm op})^{-1}$]. Thus, this frequency shoul be set comfortably higher than the highest frequency at which one wishes to analyze the TDI noise response but lower than the Nyquist frequency used to sample the TDI observables, to avoid aliasing.

[b] `TDIsignal` synthesizes the $y^{\rm gw}_{slr;d_1\ldots d_n}$(t) response as follows: (a) it computes the reception time $t_r = t_{;d_1,\ldots,d_}$ by calling repeatedly the `LISA` method `armlength`, according to the retardation chain rule (16) of Ref. [1]; (b) i determines the emission time $t_s = t_r - L_l(t_r)$ and the causal photon propagation vector $\hat{n}_l(t_r) \propto \vec{p}_r(t_r) - \vec{p}_s(t_s)$ by calling the `LISA` method `putn`; (c) it obtains the GW retardations $\hat{k} \cdot \vec{p}_s(t_s)$ and $\hat{k} \cdot \vec{p}_r(t_r)$, by calling the `LIS` method `putp` and accessing the `Vector` object `k` of the class `Wave` [Eq. (1.9)]; (d) it requests the gravitationa tensors $\mathsf{h}[t_s - \hat{k} \cdot \vec{p}_s(t_s)]$ and $\mathsf{h}[t_r - \hat{k} \cdot \vec{p}_r(t_r)]$, by calling the `Wave` method `putwave` [Eq. (1.8)]; (e) it assemble the Doppler response according to Eq. (1) of Ref. [1].

[c] In those references, primed link indices correspond in this paper to positive indices.

Bibliography

[1] M. Vallisneri, *Phys. Rev. D* **71**, 022001 (2005).

[2] M. Tinto, F. B. Estabrook, and J. W. Armstrong, *Phys. Rev. D* **69**, 082001 (2004).

[3] N. Ashby, *Living Rev. Relativity* **6**, 1 (2003), `www.livingreviews.org/lrr-2003-1`.

[4] A. Królak, M. Tinto, and M. Vallisneri, *Phys. Rev. D* **70**, 022003 (2004).

[5] LISA Study Team, *LISA: Laser Interferometer Space Antenna for the detection and observation of gravitational waves, Pre-Phase A Report*, 2nd ed. (Max Planck Institut für Quantenoptik, Garching, Germany, 1998).

[6] N. J. Cornish and L. J. Rubbo, *Phys. Rev. D* **67**, 022001 (2003); erratum, *ibid.*, 029905 (2003). See also LISA Simulator v. 2.0, `www.physics.montana.edu/lisa`.

[7] N. Cornish and R. W. Hellings, *Class. Quant. Grav.* **20**, 4851 (2003).

[8] M. Luescher, *Comp. Phys. Comm.* **79**, 100 (1994).

[9] G. E. P. Box and M. E. Muller, *Ann. Math. Stat.* **29**, 610 (1958).

[10] W. H. Press et al., *Numerical Recipes in C*, 2nd ed. (Cambridge University Press, Cambridge, MA, 1992).

[11] D. A. Shaddock, B. Ware, R. E. Spero, and M. Vallisneri, *Phys. Rev. D* **70**, 081101(R) (2004).

[12] F. B. Estabrook, M. Tinto, and J. W. Armstrong, *Phys. Rev. D* **62**, 042002 (2000).

[13] J. W. Armstrong, F. B. Estabrook, and M. Tinto, *Class. Quant. Grav.* **20**, S283 (2003).

[14] M. Vallisneri and J. W. Armstrong, "Synthetic LISA: simulating Time Delay Interferometry in a model LISA", whitepaper for the LISA Mission Science Office, v. 1.0 (Oct 2003), `www.srl.caltech.edu/lisa/documents.html`.

[15] J. R. Gair et al., *Class. Quant. Grav.* **21**, S1595 (2004) (2004); LIST WG1 EMRI taskforce, "Estimates of detection rates for LISA capture sources," technical report for the LISA Science Team Working Group I (Dec 2003), `www.vallis.org/publications`.

[16] K. Glampedakis, S. A. Hughes, and D. Kennefick, *Phys. Rev. D* **66**, 064005 (2002).

[17] L. J. Rubbo, N. J. Cornish, O. Poujade, Phys. Rev. D **69**, 082003 (2003).

[18] J. W. Armstrong, F. B. Estabrook, and M. Tinto, *Astrophys. J.* **527**, 814 (1999).

[19] D. A. Shaddock, M. Tinto, F. B. Estabrook, and J. W. Armstrong, *Phys. Rev. D* **68**, 061303 (2003).

[20] M. Tinto, D. A. Shaddock, J. Sylvestre, and J. W. Armstrong, *Phys Rev. D* **67**, 122003 (2003).