ChatGPT

TimeSeriesMatrix/FrequencySeriesMatrix と GWpy TimeSeries/FrequencySeries の機能比較

基本プロパティ・データ構造

- **GWpy TimeSeries/FrequencySeries:** 単一チャンネルの時系列/周波数データを保持するクラスです。主要な属性として、TimeSeries はサンプリング間隔 `dt`、開始時刻 `t0`（= epoch）、全サンプルの時間軸 `times` プロパティ、サンプル数、サンプルレート `sample_rate` などを持ちます。一方 FrequencySeries は周波数解像度 `df`、開始周波数 `f0`、周波数軸 `frequencies` などを持ち、`unit`（物理単位）、`name`、`channel` メタデータも含まれます [1] [2]。
- **TimeSeriesMatrix/FrequencySeriesMatrix:** 複数の TimeSeries（または FrequencySeries）を格子状（行列状）にまとめたコンテナクラスです。内部的には3次元配列（形状が (N行, M列, サンプル数)）でデータを保持し、各要素に対応するメタデータ行列も管理します。TimeSeriesMatrixでは GWpy の TimeSeries 相当の属性をプロパティとして提供しており、`dt`（サンプリング間隔）、`t0`（開始時刻）、`times`（時間軸配列）などは内部の共通軸情報にエイリアスされています [3] [4]。例えば `tsm.dt` は全要素に共通なサンプリング間隔（= `dx`）を返し、`tsm.times` は共有の時間配列を返します。`sample_rate` プロパティも定義されており、`dt` からHz単位に換算して取得・設定できます [5] [6]。FrequencySeriesMatrixも同様に、`df`（周波数間隔）や `f0`、`frequencies` プロパティを備えています（内部実装は FrequencySeries の `dx` / `x0` / `xindex` に対応）。
- **実装状況:** TimeSeriesMatrixは基本的なコンストラクタ引数を GWpy TimeSeriesと同様に受け付けます（例えば `dt` または `sample_rate`、`t0` または `epoch` のいずれか一方を指定 [7] [8]、`times` 配列を直接指定して不規則または特定の時間軸に対応可能 [9]）。GWpyと同じ引数ルール（mutual exclusivity）も実装されており、矛盾する引数を与えた場合は適切に `ValueError` を投げます [10] [11]。また、`times` 指定時には他の時間関連引数を無視するという GWpy のセマンティクスも再現しており、警告メッセージを出した上で無視する実装になっています [12] [13]（テストでも警告が確認されています [14] [15]）。TimeSeriesMatrix生成時に `epoch` と `t0` の重複設定を避け、内部的には `epoch` は `t0` のエイリアスとして扱われます [16] [17]。基本プロパティについてのテストでは、生成した TimeSeriesMatrix の `shape`、`t0`、`dt`、`times` 長、`sample_rate` が期待通り設定されることが確認されています [18] [19]。さらに、`sample_rate` プロパティのセッターも用意され、既存の時間軸を基にサンプルレートを変更すると自動的に新しい時間軸（xindex）が再構築されます [6] [20]。例えば、`tsm.sample_rate = 2.0*Hz` と設定すると時刻配列の間隔が0.5秒に再計算され、データ長に応じた新しい `times` が割り当てられます [21] [22]。`sample_rate = None` と設定すれば時間軸情報をクリア（xindexをNoneに）し、未定義状態にする挙動もGWpyに倣って実装されています [23] [24]。
- **拡張ポイント:** TimeSeriesMatrixは**複数系列の同時管理**に特化しているため、全要素が**共通の時間軸**を持つことを前提としています。内部では各要素の軸情報を抽出・比較し、長さ・サンプリングレート・開始時刻が一致しない場合はエラーを発生させる設計です [25] [26]。この厳密な軸整合性チェックは、単一Seriesには不要ですが、行列全体で意味のある演算を行うために追加された**独自仕様**と言えます。逆に言えば、GWpyではTimeSeries同士の直接演算時に明示的な軸チェックはありませんが、TimeSeriesMatrixでは異なる時間サンプル数・間隔のシリーズを混在させることはできません [27]。また、本クラスはオプションで**行名・列名によるインデックス**指定をサポートしており、コンストラクタに `rows` や `cols` 引数でラベルを渡せます。これにより `tsm["r0","c1"]` のように文字列キーで特定の要素を取り出すことが可能で、該当する単一の TimeSeries オブジェクトが返ります [28] [29]。このような**行列的なラベル管理**はGWpy本家には存在しない独自拡張です。

インデックス操作と演算の互換性

- **単一Seriesの場合 (GWpy):** TimeSeriesおよびFrequencySeriesは `numpy.ndarray` を継承しており、添字アクセスやスライス、ブールインデクシング等が可能です。例えば `series[i]` で i番目の要素の `astropy.units.Quantity` を取得、`series[start:end]` で部分系列(TimeSeries)を得ることができます [30] [31]。加減算やスカラーとの演算もNumpy互換で、単位が合う場合には結果も TimeSeries 型で返されます（異なる場合はエラー）。比較演算（>や==など）は bool値のStateTimeSeriesを返すようオーバーロードされており、マスク処理等に利用できます [32]。
- **行列クラスの場合:** TimeSeriesMatrix/FrequencySeriesMatrixもNumpy ndarrayを拡張したクラス（SeriesMatrix派生）として実装されており、**三次元配列**としての基本的な添字操作をサポートします。`tsm[i, j]` と**行列的に2次元インデックス指定**することで、その位置に対応する個別の TimeSeries オブジェクトが取得されます [33] [34]。一方、`tsm[i0:i1, j0:j1]` のようにスライス指定を行うと、元の行列から該当部分を切り出した TimeSeriesMatrix が返されます [35]。内部では `__getitem__` をオーバーライドしており、スカラー指示の場合は高速化のため直接内部配列とメタデータから TimeSeries を生成し、スライスの場合は基底のSeriesMatrixのスライス結果を TimeSeriesMatrix型にビュー変換して返す実装です [36] [37]。テストでも、例えば2x2行列 `tsm` に対し `tsm["r0","c1"]` が TimeSeries インスタンスを返し、長さや値が対応する行列内部の1チャンネル分と一致することが確認されています [29]。このとき取得した TimeSeries の `times` プロパティは元の行列と**同じオブジェクトを参照**しており、時間軸データの余計なコピーを避けています [38]。
- **配列演算の互換性:** 現行のTimeSeriesMatrixでは、Numpy由来の演算子オーバーロード（例えば `+` や `*`）について明示的な拡張は行っていません。そのため、要素同士の加減算を行う場合、一度各要素を取り出して演算するか、内部の `.value` 配列を直接操作する必要があります（将来的な演算オーバーロード拡張の余地あり）。一方、比較演算については、TimeSeriesMatrix全体に対して `tsm > x` のような操作を行うと、現状はbool値の通常のnumpy配列が返るに留まります。GWpyの StateTimeSeriesに相当するような「Matrix版のブールSeries」クラスは提供されていません。つまり、**基本的なスライス・インデックス操作は互換**ですが、**要素間演算の結果を再びMatrixとして得る機能**は未整備です。この点は「未実装（将来的に必要となる可能性がある機能）」に該当します。

前処理・フィルタリング機能

- **GWpy TimeSeries:** 時系列データに対する各種前処理メソッドが充実しています。例えば、`detrend()` で直流成分やトレンドの除去、`taper()` で両端を滑らかにゼロに収束させるプランク窓処理 [39] [40]、`whiten()` でパワースペクトルを用いたホワイトニング処理 [41]、デジタルフィルタによる周波数帯域フィルタリング（`lowpass` 低域通過、`highpass` 高域通過、`bandpass` 帯域通過、`notch` ノッチ除去）、また補間付きのダウンサンプリング/アップサンプリングを行う `resample()` などが用意されています。これらはいずれもTimeSeriesオブジェクトに対して**インスタンスメソッド**として提供され、適切なパラメータ（カットオフ周波数やフィルタ次数等）を指定して呼び出すと、新たなTimeSeries（またはオプションでインプレースに自身を書き換え）を返します。例えば `TimeSeries.whiten()` はFFT長やウィンドウ、さらには既知のASD（Amplitude Spectral Density）を引数で渡すことでホワイトニングを調整可能です [42] [43]。
- **TimeSeriesMatrix:** これら単一Series向け前処理機能の**大部分をカバー**しています。具体的には、上記の主要メソッド（`detrend`, `taper`, `whiten`, `filter`（Butterworthフィルタのエイリアス）, `lowpass`, `highpass`, `bandpass`, `notch`, `resample`）について、TimeSeriesMatrixクラス生成時に動的に**同名のメソッド**を追加する実装となっています [44] [45]。各メソッドは内部で対応する TimeSeriesメソッドを全要素に対して順次呼び出し、その結果得られる複数のTimeSeriesを再び一つのTimeSeriesMatrixにまとめて返す仕組みです [46] [47]。例えば、`tsm.detrend()` を呼ぶと行列内の各シリーズに対し順に `TimeSeries.detrend()` が適用され、**同じ形状 (N×M×サンプル数) の新たな TimeSeriesMatrix** が返ります [48]。このとき時間軸は各要素で変化しないため、元の `tsm.times` がそのまま新Matrixにも引き継がれます [49]（軸長が変化しない処理のため）。一方、`tsm.resample(new_rate)` のようにサンプル数や時間刻みが変わる処理では、各要素のTimeSeriesが

それぞれ新レートでリサンプリングされますが、内部で**共通の新時間軸**を算出し直し、それを全要素で共有させています [50] [51]。このため、リサンプリング後も行列内で時間配列が統一され、形状の整合性が保たれます（例えば2×2行列1024Hz→512Hzに半減すると、サンプル数も半分になりshape[-1] が減少しますが、全要素で同一長に揃っています）。また、`inplace=True` オプションがサポートされており、例えば `tsm.detrend(inplace=True)` とすると新たな行列を返さず**内部のデータ配列を書き換える**動作になります [52] [53]。テストでも、インプレース実行後に元オブジェクト自体が変化し、返り値が元と同一であることが確認されています [54] [55]。さらに、各要素に付随する単位・名前・チャンネル情報 (MetaData) も前処理後に**忠実に保持**されます [47] [56]。たとえば名前や物理単位は入力TimeSeriesからそのままコピーされ、行列全体のメタデータ配列に格納されています。

- **仕様上の差異:** 単一TimeSeriesでは軸の概念は一つだけですが、行列の場合は上記の通り**共通軸の前提**があります。そのため、TimeSeriesMatrixの前処理メソッドでは**処理後も全要素で軸が一致しているか**を検証し、もし長さや `dt` が食い違えば `ValueError` を発生させるようになっています [57] [27]（もっとも、標準的な前処理では各要素の長さが変わることは通常ないため、エラーが起こるのは異なる長さの系列を持っていた場合など異常系のみです）。基本的な挙動はGWpy本家のメソッドと**同等**であり、例えばデトレンド後は平均0になること、whiten後は平均0・分散1にスケールされること、フィルタでは所定の周波数成分が除去・減衰することなど、**各要素ごとの結果は元のTimeSeriesにメソッドを適用した場合と同一**です。拡張的な挙動としては、TimeSeriesMatrix経由で各メソッドに引数を渡す際、その引数が単一値や単一オブジェクトの場合は各要素に同じものが適用されます。例えば GWpy の `TimeSeries.whiten(asd=ref_asd)` は1つの系列を特定ASDで白色化しますが、`tsm.whiten(asd=ref_asd)` とすれば全要素が同じ参照ASD（FrequencySeries）を使って白色化されます。このように**共通パラメータを一括適用**できるのは行列ラッパーを通す利点の一つです（もっとも物理的には各系列でノイズスペクトルが異なる場合も多いため、実用上は必要に応じ個別ASDを渡すことになるでしょう）。
- **未実装の機能:** GWpy TimeSeries に存在するメソッドのうち、TimeSeriesMatrixで未実装のものも一部あります。その代表が `crop()` です。TimeSeriesの `crop(start, end)` は指定時間区間にデータを切り抜く機能ですが、TimeSeriesMatrixには直接対応するメソッドがありません。これは設計上大きな問題ではなく、例えば行列の時間軸が定期的であればスライス演算（例：`tsm[:, :, start_index:end_index]`）で代用可能であり、必要な場合は各要素を取り出して個別にcropすることもできます。したがって現状では**未実装だが大きな支障はない機能**と評価できます。また、GWpy 0.17現在でTimeSeriesに実装されていない機能（例：高度なシグナル処理）がMatrix側で新規に拡張されていることもありません。基本的には**GWpyに存在する主要な前処理は全てMatrix側でカバー**されていると言えます。

スペクトル変換・周波数領域解析

- **GWpy TimeSeries:** 周波数領域への変換やスペクトル解析のメソッドが豊富に用意されています。`fft()` は高速フーリエ変換を行い、複素数の FrequencySeries を返す基本メソッドです [58] [59]。またウェルチ法による平均パワースペクトルを計算する `psd()` (Power Spectral Density)や、その平方根である `asd()` (Amplitude Spectral Density)も提供されています [60] [61]。いずれもFrequencySeries（周波数軸を持つデータ列）として出力され、適切な物理単位（例えばASDでは元の単位/√Hz）とメタデータが付与されます。さらに**二系列間のスペクトル解析**として、相互スペクトル密度を求める `csd(other)` やコヒーレンスを計算する `coherence(other)`、伝達関数比を求める `transfer_function(other)` があり、自己コヒーレンス（自己相関的なコヒーレンス）を計算する `auto_coherence(dt)` も実装されています [62] [63]。例えば `ts1.csd(ts2)` は ts1とts2の間のクロススペクトル密度を FrequencySeries として返し、`ts1.coherence(ts2)` は周波数ごとの相関係数(0〜1の実数)を算出します。GWpy実装ではサンプルレートが異なる2系列に対してこれらの演算を行う場合、自動的に高い方を低い方に**ダウンサンプリング**してから計算する処理が入っています [64]（例えば 2048 Hz と 1024 Hz のデータなら後者に合わせて前者を間引き）。また、FFT長や重なり率、ウィンドウ関数など細かなオプションもパラメータ経由で指定可能です [65] [66]。FrequencySeries 側に

は逆フーリエ変換で時系列に戻す `ifft()` や、フィルタ設計用に零点・極・利得に分解する `zpk()` といった周波数領域特有のメソッドも用意されています [67]。

- **TimeSeriesMatrix:** スペクトル解析機能についても、多くは**同等の操作が可能**です。TimeSeriesMatrixには `fft()`, `psd()`, `asd()` メソッドが実装されており、行列内の各TimeSeriesに対してこれらを適用した結果を一つの FrequencySeriesMatrix にまとめて返します [68] [69]。例えば `fsm = tsm.fft()` と呼ぶと、各要素 `tsm[i,j]` に対して GWpy の TimeSeries.fft() を実行し、それら結果（すべて FrequencySeries）を要素とする 3次元配列を生成します。その際、**全要素で共通の周波数軸**を持つよう注意が払われています。 [70] に示すように各結果の frequency 配列やdf,f0を収集し、`_validate_common_frequency_axis` で長さ・周波数間隔が一致しているかチェックした上で共通軸を採用しています [71] [72]。長さが異なる場合は短い方に合わせてtruncateする処理も入っており（例えばサンプルレート差による周波数分解能差異が出た場合など）、最終的に FrequencySeriesMatrix 全体で一貫した `frequencies` プロパティと `df` を持つように調整します [64] [73]。テストでも `tsm.fft()` の結果得られる FrequencySeriesMatrix の `shape` が期待通りであり、周波数配列長とも整合することが確認されています [74] [75]（例：100サンプルの時系列FFTで出力長51の周波数系列になる）。同様に、`tsm.psd()` や `tsm.asd()` も各要素に対し PSD/ASD を計算し FrequencySeriesMatrix を返します（内部的にはまず1要素で代表計算し、出力配列を確保後に全要素計算する実装で効率化） [76] [77]。
- **二系列スペクトル解析:** TimeSeriesMatrix では、GWpyの二変量メソッドにも対応しています。`csd`, `coherence`, `transfer_function` は**行列同士**または**行列と単一TimeSeries**を引数に取れるようラッパーが実装されています [78] [79]。例えば `out = A.csd(B)` とすれば、同じ形状(N,M)のTimeSeriesMatrix同士について要素ごとに CSD を計算し、対応する(N,M)形状の FrequencySeriesMatrix を得ます [80]。また片方に単一のTimeSeriesを渡すこともでき、その場合はそのSeriesとのCSDを各要素ごとに計算して行列化します [81]。このとき自動的にサイズの放送（broadcast）を行い、単一Seriesが渡された場合は行列の全要素に対して同一の相手Seriesを適用する設計です [82] [83]。二系列メソッドでは、TimeSeriesMatrix側でまず引数 `other` を検証し、Matrix型同士なら形状(N,M)が一致するか、TimeSeries型なら受け入れ可能かをチェックして不正なら例外を投げます [84] [85]。形状不一致（例：行列サイズが異なる）では `ValueError`、対応型でない場合（数値や異種オブジェクト）では `TypeError` を送出し、テストでも期待通りエラーになることが確認されています [86] [87]。**実装上、異なるサンプルレートを持つ行列**同士の解析も可能で、その場合は内部の各要素ペアの計算でGWpy側が高レートを低レートにリサンプリングして処理するため、結果として問題なく計算できます [64]（実際のテストでは片方4096サンプル/1024 Hz、もう片方8192サンプル/2048 Hzの行列で `transfer_function` を計算しエラーにならないことを確認 [88] [89]）。`coherence` や `transfer_function` の結果もFrequencySeriesMatrix（コヒーレンスの場合は値が0〜1の実数）として取得でき、全要素で `epoch`（開始時刻）が元のTimeSeriesMatrixから継承されていることがテストで検証されています [90] [91]。なお、auto_coherence(自己コヒーレンス)についてもTimeSeriesにメソッドが存在すればMatrix側で利用可能です [92]（TimeSeries.auto_coherenceは単一系列内で時間シフトしたコピー同士のコヒーレンスを算出）。

- **FrequencySeriesMatrix:**　周波数領域データの行列版で、TimeSeriesMatrixからのFFTやCSD計算によって得られます。FrequencySeriesMatrix自体もSeriesMatrixを継承しており、基本的な構造・インデックス操作はTimeSeriesMatrixに類似しています。例えば `fsm[i,j]` で FrequencySeries オブジェクト（複素スペクトル列）を取り出せ、`fsm[:,1]`　で2列目の全行を含む部分行列を取得できます。FrequencySeriesMatrixには `df`・`f0`・`frequencies` プロパティが実装され、各要素で共通の周波数軸情報を参照できます（例えば `fsm.df` はその行列スペクトルの周波数分解能を表します）。一方で、FrequencySeriesMatrix　特有の処理機能は限定的です。**逆フーリエ変換**（ifft）については、現時点で FrequencySeriesMatrix に直接の `ifft()` メソッドは実装されていません（※2025年現在）。したがって周波数行列を時系列行列に戻すには、各要素をイテレートして個別に `ifft()` した結果を TimeSeriesMatrix に再構築するか、必要に応じてTimeSeriesMatrix側で対応メソッドを拡張する必要があります。この点は**未実装で将来的に必要となる可能性がある機能**です。もっとも、通常の解析パイプラインでは行列全体で一括逆変換する場面は少なく、必要なら上記のように手動で対応可能なため、深刻な欠落とは言えません。その他、FrequencySeries固有の `zpk()`（ゼロ・ポール・ゲイン取

得）や、スペクトル上での演算（例：2つのFrequencySeriesの比を取る等）については、FrequencySeriesMatrixには専用実装がありません。必要な場合は各要素を取り出して処理する形になります。

- **未実装・非対応の機能:** GWpy TimeSeriesが持つ**スペクトログラム関連**の機能は、TimeSeriesMatrixではサポートしていません。例えば、TimeSeriesの `spectrogram(stride, fftlength, ...)` メソッドはそのシリーズの時間-周波数パワースペクトログラム（Spectrogramオブジェクト）を生成します [93] が、複数Series同時のスペクトログラムを直接得る機能は行列クラスに用意されていません。これは設計上明示的に不要と判断された部分です。仮に複数チャンネルそれぞれのスペクトログラムが必要な場合は、各Seriesを取り出して個別に計算するか、あるいはSpectrogram自体が多次元配列（周波数×時間×チャンネル）を扱えるよう将来拡張する検討余地はありますが、現状そのニーズは高くないと考えられます。したがって**Spectrogram系機能は未実装**であり、必要になった際に対処する方針です。同様に、コヒーレンススペクトログラム（TimeSeries.coherence_spectrogram）等の高度な時刻–周波数解析についてもMatrixではサポート外ですが、これも本質的には**各チャンネル独立に計算すべき**性質のものなので、設計上省かれています。

可視化 (プロット) 機能

- **GWpy 単体Series:** TimeSeriesやFrequencySeriesは、それぞれ `plot()` メソッドによりデータを簡易に可視化できます。TimeSeries.plot()はmatplotlibベースのPlotオブジェクトを返し、時系列グラフを描画します。デフォルトではx軸はGPS秒（もしくは適宜人間読みやすい時刻表示）で、y軸は物理単位付きでプロットされます。FrequencySeries.plot()は既定でx軸対数スケール、y軸もパワースペクトル系（Hzのべき乗が単位に含まれる場合）では対数スケールで描かれる工夫があります [94] 。また複数のSeriesを同じPlotに描画することも可能で、Plotクラスに複数Seriesを渡したり、TimeSeriesList/FrequencySeriesListをplotする機能も提供されています。
- **TimeSeriesMatrix/FrequencySeriesMatrix:** 行列クラスでも簡便なプロット機能が用意されています。TimeSeriesMatrixには独自に `plot()` メソッドが実装されており、内部では基底のSeriesMatrix.plotを呼び出す前に `xscale="auto-gps"` を自動セットしています [95] 。これにより、時間軸がGPS秒の場合に人間に理解しやすい日付時刻表示へ自動調整されます（GWpy Plotの“auto-gps”機能を利用）。複数チャンネルの可視化については、現行実装では**全要素を同一プロットに重ねて描画**する挙動になります。例えば2×2のTimeSeriesMatrixをplotすると、合計4本の時系列が色分けされて一つの軸に描かれます（必要に応じて凡例に行・列名が表示されます）。この動作は内部でTimeSeriesMatrixがIterableとして各Seriesを渡すことでGWpyのPlotが複数系列プロットモードになるためです。FrequencySeriesMatrixについて専用のplot実装はありませんが、SeriesMatrix継承により `fsm.plot()` で全スペクトルを重ね描画できます。FrequencySeries自体のplotはy軸ログ表示など単体での体裁調整がありますが、Matrix経由でも基本的に同様の表示ルールが適用されます（各要素がPSD/ASDなら対数表示、コヒーレンスなら0-1範囲の線など）。
- **評価:** 標準的なプロット用途について、TimeSeriesMatrix/FrequencySeriesMatrixは**実用上問題ない互換性**を備えています。単一Seriesと同様のコードで可視化でき、行列内の全データを一括で見られるため、複数チャンネルの挙動比較に有用です。ただし、プロットの細かなカスタマイズ（例えばチャンネルごとに別サブプロットに描く等）は、自動では行われません。その場合はユーザがPlotオブジェクトを調整するか、行ごと・列ごとにMatrixをスライスして個別にplotする必要があります。このあたりは**GWpy本家に依存する部分**であり、Matrix独自の可視化拡張は最小限となっています。

入出力(I/O)機能

- **GWpy TimeSeries/FrequencySeries:** データ入出力に関する充実した機能があります。TimeSeriesの場合、クラスメソッド `TimeSeries.read(source, format=..., ...)` により様々な形式のデータファイル（例えばGWフレームファイルGWFやCSV、WAV等）からデータを読み取って TimeSeries を生成できます [96] [97] 。複数チャンネルを同時に読み込む `TimeSeriesDict.read()` も用意されてお

り、一度に複数のTimeSeriesを取得可能です [98] 。また、NDSサーバからデータを取得する `fetch()` や、Open Science Centerから公募データを取り寄せる `fetch_open_data()` も備わっています [99] [100] 。書き出しについても、`series.write(filename, format=...)` でHDF5やASCII形式でファイル保存が可能です [101] 。FrequencySeriesも同様に `read` / `write` クラスメソッドがあり、例えばGWpyで計算したPSDを保存・読み込みすることができます [102] [103] 。

- **TimeSeriesMatrix/FrequencySeriesMatrix:** 現時点で**独自のI/O機能は実装されていません**。ファイルやリモートから直接TimeSeriesMatrixを生成するメソッド（例えば `TimeSeriesMatrix.read(...)` や `fetch_matrix(...)` ）は用意されておらず、必要な場合は**まずGWpyの既存機能**でTimeSeriesを読み込み、それらを用いてTimeSeriesMatrixを構築する手順になります。例えば、複数チャンネルのデータを取得したい場合、`TimeSeriesDict.fetch([...])` で辞書形式で読み込んだ後、その中のTimeSeriesを取り出してTimeSeriesMatrixに渡す、という使い方になります。実装上も、I/OはGWpy側に任せる設計となっており、Matrixクラス内でファイルフォーマット処理等は一切行っていません。この判断は**設計上明示的に不要とされた機能**に該当します。すなわち、「データ取得・保存」は既存のTimeSeries/FrequencySeriesやそのDict/Listで十分担えるため、Matrixクラスで重複して実装することを避けている状況です。
- **評価:** I/O機能の非実装については、実用面で大きな問題とはなっていません。というのも、重力波データ解析のシナリオではまず特定のデータ源（NDSサーバやローカルファイル）から**チャンネルごとに**データを取得し、それらを組み合わせて解析する流れが一般的だからです。TimeSeriesMatrixは後段の解析・可視化を効率化する目的で導入されたコンテナであり、前段のデータ取得部分はGWpy本家の堅牢な実装に委ねるのが合理的と考えられます。そのため、Matrix自体が直接ファイル入出力を扱えないことは**設計上の割り切り**であり、もし要求が高まれば将来的にTimeSeriesDictからMatrixへの変換メソッドを追加する程度の拡張で対応可能でしょう。

完成度の総合評価

以上をまとめると、ユーザー実装の TimeSeriesMatrix および FrequencySeriesMatrix は、**GWpy の TimeSeries/FrequencySeries が持つ主要機能の大部分を網羅・対応している**と言えます。単一Seriesに存在するプロパティやメソッドは行列全体に対してほぼ等価に利用可能であり、互換性は高いです。特に**基本的なデータ保持・演算・フィルタリング・スペクトル解析・プロット**に関しては、既存テストでカバーされている通り「本家と同等の機能」を発揮しています。実装済み機能の挙動に大きな不整合は見られず、軸整合性チェックや行列ラベルといった拡張も妥当に機能しています。

一方で、**未実装**の機能もいくつかありますが、それらは設計上の判断で優先度を下げたものです。I/O周り（ファイル読み書きやデータ検索）は本家GWpyに依存し、Spectrogram系の出力も省略されています。逆フーリエ変換（ifft）のように将来的に追加が望まれる機能もありますが、現時点では大きな支障とはなっていません。

総合的に見て、**現段階のTimeSeriesMatrix/FrequencySeriesMatrixの完成度は非常に高く、日常の解析業務において実用上問題ないレベル**です。基盤部分はGWpyの信頼性に乗っており、拡張部分も基本的なテストが一通り整備されているため、安定した運用が可能でしょう [104] [105] 。今後ニーズに応じて一部機能を補完すれば、公式クラスに劣らない包括的なツールとなると評価できます。

---

[1] [2] [30] [31] [32] [39] [40] [41] [42] [43] [58] [59] [60] [61] [62] [63] [64] [65] [66] [67] [73] [93] [94] [96] [97] [98] [99] [100] [101] [102] [103] gwpy_source.txt
file://file-EK9VaHJCoPKkfCkHmBbibH

[3] [4] [5] [6] [7] [8] [9] [12] [13] [16] [17] [20] [25] [26] [33] [34] [35] [36] [37] [44] [45] [46] [47] [52] [53] [57] [68] [69] [70] [71] [72] [76] [77] [78] [79] [82] [84] [92] [95] timeseries.py
file://file_00000000728c71fabca67311db7c1662

10 11 14 15 18 19 21 22 23 24 28 29 38 74 104 test_tsm_basic.py

file://file_000000002eac71faba8693a801950898

27 48 49 50 51 54 55 56 test_tsm_preprocess.py

file://file_0000000055587207b97cbb1d1a2c2b16

75 80 81 83 85 86 87 88 89 90 91 105 test_tsm_bivariate.py

file://file_00000000e45871fa865acb221c7c8323