**Проверка базовых классов**

**1. IF (frequencies.py)**

* **Работоспособность**: Класс полностью функционален. Методы для установки и получения частоты, полосы пропускания и активности работают корректно.
* **Исключения**: Все проверки типов и значений присутствуют (например, отрицательные частоты, нулевая длина волны при вычислении). Ошибка деления на ноль в get\_freq\_wavelength отрабатывается.
* **Сериализация**: Методы to\_dict и from\_dict реализованы правильно, включают все поля.
* **Полнота**: Код покрывает все заявленные функции (включая пересчет частоты через длину волны с использованием константы C\_MHZ\_CM).
* **Замечания**: Нет проблем, класс готов к использованию.

**2. Frequencies (frequencies.py)**

* **Работоспособность**: Управление списком объектов IF работает: добавление, удаление, получение активных/неактивных частот.
* **Исключения**: Проверки на тип IF и валидность индексов присутствуют. Ошибки вроде пустого списка в activate\_all/deactivate\_all отрабатываются.
* **Сериализация**: to\_dict и from\_dict корректно обрабатывают список IF.
* **Полнота**: Все методы реализованы, включая \_\_len\_\_ и \_\_repr\_\_.
* **Замечания**: Нет явных проблем, класс полностью готов.

**3. Source (sources.py)**

* **Работоспособность**: Создание источника, установка координат (включая пересчет из градусов), активация/деактивация работают.
* **Исключения**: Проверки на типы (str для имен) и диапазоны координат (RA: 0-23h, DEC: -90°..90°) присутствуют и отрабатывают.
* **Сериализация**: to\_dict и from\_dict включают все поля, включая опциональные name\_J2000 и alt\_name.
* **Полнота**: Методы для получения координат в разных форматах (hms/dms и градусы) реализованы.
* **Замечания**: Нет ошибок, класс полностью функционален.

**4. Sources (sources.py)**

* **Работоспособность**: Управление списком источников работает (добавление, удаление, замена).
* **Исключения**: Проверка уникальности имени источника в add\_source и валидность индексов отрабатывают корректно.
* **Сериализация**: to\_dict и from\_dict обрабатывают список источников без проблем.
* **Полнота**: Реализованы все методы, включая \_\_len\_\_ и \_\_repr\_\_.
* **Замечания**: Нет проблем, класс готов.

**5. Scan (scans.py)**

* **Работоспособность**: Создание скана, установка параметров и получение данных (включая пересчет в MJD и datetime) работают.
* **Исключения**: Проверки на типы (Source, Telescopes, Frequencies) и положительность длительности присутствуют.
* **Сериализация**: to\_dict и from\_dict корректно сериализуют вложенные объекты.
* **Полнота**: Все методы реализованы, включая вычисление времени окончания.
* **Замечания**: Нет ошибок, класс полностью готов.

**6. Scans (scans.py)**

* **Работоспособность**: Управление списком сканов работает, включая проверку пересечений в add\_scan и set\_scan.
* **Исключения**: Проверка пересечений сканов по времени и источнику отрабатывает, ошибки индексов тоже.
* **Сериализация**: to\_dict и from\_dict корректно обрабатывают список сканов.
* **Полнота**: Реализованы все методы, включая \_\_len\_\_ и \_\_repr\_\_.
* **Замечания**: Логика пересечения может быть избыточной для некоторых приложений (например, если допустимы пересечения), но это зависит от требований. В текущем виде работает без ошибок.

**7. Telescope (telescopes.py)**

* **Работоспособность**: Создание наземного телескопа и управление его параметрами работают.
* **Исключения**: Проверки на типы (str для code и name) и числовые значения координат/скоростей присутствуют.
* **Сериализация**: to\_dict и from\_dict включают все поля, с указанием "type": "Telescope".
* **Полнота**: Все методы реализованы, включая \_\_repr\_\_.
* **Замечания**: Нет проблем, класс готов.

**8. SpaceTelescope (telescopes.py)**

* **Работоспособность**: Загрузка орбитальных данных из OEM-файла, интерполяция (Чебышев и кубический сплайн) и вычисление по Кеплеровым элементам работают.
* **Исключения**: Проверки на существование файла, диапазон времени для интерполяции и наличие данных отрабатывают.
* **Сериализация**: to\_dict и from\_dict корректно обрабатывают orbit\_file, orbit\_data (с преобразованием NumPy в списки) и kepler\_elements (с датой в ISO формате).
* **Полнота**: Все методы реализованы, включая сложные вычисления орбит.
* **Замечания**: Зависимость от numpy и scipy требует их наличия в среде. Код полностью рабочий, но тестирование интерполяции требует реальных OEM-файлов.

**9. Telescopes (telescopes.py)**

* **Работоспособность**: Управление списком телескопов (включая Telescope и SpaceTelescope) работает.
* **Исключения**: Проверка уникальности кода в set\_telescope и валидность индексов отрабатывают.
* **Сериализация**: to\_dict и from\_dict различают типы телескопов через "type".
* **Полнота**: Реализованы все методы, включая \_\_len\_\_ и \_\_repr\_\_.
* **Замечания**: Нет проблем, класс готов.

**10. Observation (observation.py)**

* **Работоспособность**: Создание наблюдения и управление его компонентами работают.
* **Исключения**: Проверки типов для всех компонентов и положительности step\_time присутствуют.
* **Сериализация**: to\_dict и from\_dict корректно обрабатывают все вложенные объекты.
* **Полнота**: Методы для получения времени начала/конца и активации/деактивации всех компонентов реализованы.
* **Замечания**: Нет ошибок, класс полностью готов.