

Универсальный пакет для синтеза изображений

srhimages

Программа `srhimages` предоставляет функции для синтеза радиоизображений из данных Сибирского Радиогелиографа (SRH).

Установка

Код Сергея Анфиногентова работает на python 3.10 до 3.12, код Марии Глобы - только на 3.10! Поэтому желательно устанавливать именно 3.10. Для расчётов на удалённых машинах может быть допустимо установить любую версию питона. При расчётах через Dask желательно так же использовать 3.10. На Windows на данный момент пока что работает только код Сергея Анфиногентова.

```
conda env create -n srhsynth python=3.10
conda activate srhsynth

pip install -U "srhimages[all] @ git+https://git.iszf.irk.ru/fedenev/srhimages"
# или [globa], [anfinogentov] вместо [all], чтобы выбрать только конкретный расчётный код
```

Если производится свежая установка с использованием расчётного кода Марии Глобы, то требуется обновить данные CASA:

```
python3 -m casaconfig --update-all
```

Использование уже установленного окружения на сервере ИСЗФ

```
conda activate /opt/miniconda3/envs/srhsynth/
```

Интерфейс командной строки

```
srhimages create_synth_tasks --help
srhimages run_computation --help
```

Использование в своих скриптах

```
import srhimages

help(srhimages.create_synth_tasks)
help(srhimages.run_computation)
```

Обратите внимание, что если запуск кода происходит через Python скрипт `.py`, то требуется запускать расчёты следующим образом:

```
import srhimages

if __name__ == "__main__":
    # srhimages.run_computation(.....)
    pass
```

Этот костыль связан с особым механизмом работы систем параллельных расчётов в Python. При запуске кода в Jupyter, скорее всего, это не понадобится.

Описание команд и принципы работы

Расчётные задачи

Программа работает в парадигме так называемых расчётных задач ("tasks"). Каждая из задач представляет собой Python-словарь или его JSON-представление и описывает, из каких данных (сырых файлов и сканов внутри них) должно получиться итоговое изображение, куда оно будет сохранено и с какими параметрами синтезировано.

Актуальная спецификация формата расчётной задачи есть [в исходном коде](#) в формате `jsonschema`. Спецификация API для работы с амплитудно-фазовыми калибровками антенн [находится](#) в репозитории [badary-services](#).

Интерфейс программы:

1. `server_calibrate(task_list, algorithm, save_to=None, search_window="15min")`
 - **Назначение:** Обрабатывает список некалиброванных задач, чтобы назначить им калибровки, предоставленные командой SRH и находящиеся на API-сервере.
 - **Аргументы:**

- `task_list`: Список задач для вычисления или путь к файлу JSON с списком задач.
- `algorithm`: "globa" или "anfinogentov".
- `save_to` (необязательно): Путь к файлу JSON для сохранения списка задач.
- `search_window` (необязательно): Временное окно $(-search_window/2, +search_window/2)$, в котором серверная калибровка считается допустимой для использования. По умолчанию: "15min".

• **Возвращает:**

- `calibrated_tasks`: Список задач с доступным объектом ["gains"] в каждой из них, если соответствующие серверные калибровки были найдены, в противном случае – задачи, которые были переданы изначально.

2. `create_synth_tasks(time1, time2=None, cadence="15min", frequencies="all", resample_from=None, save_to=None, average_width=20, average_unit="scans", average_position="after", average_mode = "visibilities", output_polarizations="IV", naxis=512, cdelt=4.9, clean_disk=True, compressed=True, smooth_gains=False)`

- **Назначение:** Создает список (некалиброванных) задач по синтезу радиоизображений с телескопа.

• **Аргументы:**

- **time1** (str): Время начала наблюдения в формате 'ГГГГ-ММ-ДД ЧЧ:ММ:СС'.
- **time2** (str или None, опционально): Время окончания в том же формате.
- **cadence** (str, опционально): Временной интервал между каждым наблюдением в формате "NNmin" или "NNs". По умолчанию "15min".
- **frequencies** (list(int) или str, опционально): Список частот для наблюдения в МГц или "all" в виде строки, или диапазон вида "3000-5000", или список вида [3000, "6000-8000", "SRH1224"].
- **resample_from** (list или str): Список задач или путь к файлу, содержащему список задач в формате JSON, откуда брать калибровки. Доступные частоты в списке resample должны совпадать с запрошенными пользователем частотами.
- **save_to** (str, опционально): Путь к json файлу для сохранения списка задач.
- **average_width** (int или float, опционально): Количество сканов или секунд для усреднения. По умолчанию 20.
- **average_unit** (str, опционально): Может быть 'scans' или 'seconds'. По умолчанию 'scans'.
- **average_position** (str, опционально): Временное окно для усреднения. Может быть 'after', 'before' или 'center'. По умолчанию 'after'.
- **average_mode** (str, опционально): Режим усреднения. Может быть 'visibilities', 'gridding' или 'images'. По умолчанию 'visibilities'.
- **output_polarizations** (str, опционально): Может быть 'IV' или 'RL'.
- **naxis** (int, опционально): Количество пикселей вдоль каждой оси выходного изображения. По умолчанию 512.
- **cdelt** (float, опционально): Размер каждого пикселя в угловых секундах. По умолчанию 4.9 для СРГ.
- **clean_disk** (bool, опционально): Флаг для включения/выключения очистки "грязного" изображения. По умолчанию True.

- **compressed** (bool, опционально): Флаг для включения/выключения сжатия FITS выходного изображения. По умолчанию True.
- **smooth_gains** (bool, опционально): Предпочтение сплайн-интерполяции калибровок вместо ближайших соседей при передискретизации. Используйте True для калибровок за весь день.

- **Возвращает:**

- Список задач синтеза (каждая задача – Python dict), например, для отправки на кластер или для локального вычисления.

3. `run_computation(task_list, algorithm, cache_dir="./images/raw/", out_dir="./images/out/", ftp_server="https://ftp.rao.istp.ac.ru", n_threads=5, calibrate="prefer_server", progress_save_to=None, calibrations_search_window="15min", skip_postprocessing=False, skip_images=False, input_dir=None, cluster_object="local", run_id=None)`

- **task_list** (list или str): Список задач для вычисления или путь к файлу, содержащему список задач в формате JSON.
- **algorithm** (str): "globa" или "anfinogentov".
- **cache_dir** (str): Директория для хранения загруженных сырых файлов СРГ.
- **out_dir** (str): Директория для сохранения синтезированных изображений.
- **ftp_server** (str): Адрес сервера для загрузки файлов. По умолчанию загрузка осуществляется с использованием HTTPS (адрес начинается со схемы https://).
- **n_threads** (int): Количество потоков для вычислений (по умолчанию 5).
- **calibrate** (str): "prefer_server" или "from_scratch". По умолчанию "prefer_server". "from_scratch" удаляет все уже имеющиеся калибровки в списке задач и заставляет калиброваться всё заново.
- **progress_save_to** (str, опционально): Путь к json файлу для сохранения результирующего списка задач (на основе исходного списка задач). По умолчанию None (прогресс сохраняется в файл со случайным именем в текущем каталоге).
- **calibrations_search_window** (str, опционально): Временное окно (-search_window/2, +search_window/2), в котором калибровка с сервера считается действительной. По умолчанию: "15min".
- **skip_postprocessing** (bool, опционально): Пропустить выравнивание и сглаживание амплитуды/фазы усилений для антенн СРГ. По умолчанию False, True не рекомендуется.
- **skip_images** (bool, опционально): Только откалибровать (и выполнить постобработку) задачи и не выполнять этап CLEAN и синтез изображений. По умолчанию: False.
- **input_dir** (str, опционально): Директория, содержащая входные файлы, в случае, если SRH NAS смонтирован там. Для локальных расчётов всегда должно быть None.
- **cluster_object** (str или object): Тип кластерного объекта для использования для вычислений, 'local' для локальных вычислений (по умолчанию 'local') и 'badary' для кластера в Бадарах. Другие варианты включают передачу пользовательских объектов Dask.distributed (например, SSHCluster).
- **run_id** (str, опционально): Префикс строки для Redis для хранения прогресса задачи (и списка задач) во время вычислений. По умолчанию None, что приведёт к случайной строке.

Возвращает:

- **results:** Список задач с результатами вычислений, либо с ошибками.

Примечание:

- Если `task_list` является строкой или объектом `pathlib.Path`, он будет загружен как JSON файл.
- Загружает необработанные файлы с FTP сервера в директорию кэша.
- Калибрует задачи, используя серверные калибровки или локально.
- Постобработка калибровок выполняется методом, разработанным Сергеем Анфиногентовым.
- Все ошибки в вычислениях будут сохранены в возвращаемом объекте или сохранены в результирующем файле в формате JSON.

Пример использования

Простой случай нескольких изображений в течение дня

```
import srhimages

time1, time2 = "2024-06-01 02:00:00", "2024-06-01 02:00:05"
frequencies = [2800, 3000]

task_list = srhimages.create_synth_tasks(time1, time2, cadence="3s", frequencies=frequencies,\
    save_to="synth_1.json",
    average_width=5, average_unit="seconds", average_mode = "visibilities", average_position="after",\
    output_polarizations = "IV", naxis=512, cdelt=4.9, clean_disk=True, compressed=True)

if __name__ == "__main__":
    results = srhimages.run_computation("./synth_1.json", "globa", progress_save_to="./results.json")

# результаты появятся в каталоге ./images/out
```

Обработка солнечной вспышки

Здесь требуется уже откалибровать данные на достаточно большом интервале времени, поскольку во время вспышки резко снижается вклад коротких баз радиотелескопа. Поэтому расчёт будет проходить в 2 этапа:

Первый этап - калибровка на длинном интервале времени раз в 5 минут, чтобы отследить тренд "уплывания" коэффициентов усиления антенн в течение дня. Обязательно используется постобработка калибровок и их сглаживание. Рекомендуются интервалы от нескольких часов, самое идеальное - весь день.

```

import srhimages

freq_list = [3000, "5500-6200"]
time1, time2 = "2024-02-06 02:10:00", "2024-02-06 04:00:00"

calib_tasks = srhimages.create_synth_tasks(time1, time2, cadence="5min",\
    frequencies=freq_list, save_to="calibs.json")

# в файле calibs.json появятся задания для вычисления первоначальных калибровок

if __name__ == "__main__":
    computed_calibrations = srhimages.run_computation("calibs.json", "globa",\
    progress_save_to="calibs_ready.json", skip_images=True)

    # параметр skip_images нужен, потому что
    # нам нужны только калибровки, а не сами изображения

# если всё хорошо, то в файле calibs_ready.json будут готовые калибровки.

```

Второй этап. Когда у нас уже имеются постобработанные калибровки, то можно на их основе посчитать картинки с максимальным временным разрешением. `cadence="0s"` означает, что используется максимально возможное временное разрешение. Список частот должен быть тем же самым, на котором шла калибровка. Параметр `resample_from` означает, что для новых заданий берутся уже посчитанные калибровки и переносятся на новую сетку по времени. `smooth_gains` означает, что интерполяция коэффициентов усиления антенн на нужный кадр будет производиться с помощью сплайна, а не ближайшего соседа. Эта настройка рекомендуется, когда временное разрешение финальных изображений чаще, чем разрешение калибровок.

```

synth_tasks = srhimages.create_synth_tasks(time1, time2, cadence="0s", frequencies=freq_list,\
    resample_from="calibs_ready.json", smooth_gains=True, save_to="synth.json")

if __name__ == "__main__":
    results = srhimages.run_computation("synth.json", "globa",\
    progress_save_to="synth_progress.json")

```

В списке выполненных (или невыполненных) задач, который появится в переменной `results` или в файле `synth_progress.json`, будут уже прописаны полные пути к полученным изображениям

Загрузка калибровок на сервер

```
from srhimages.calibrations import CalibrationsApi
import json

with open("synth_2.json", "r") as fp:
    calibrated_tasks = json.load(fp)

cal_api = CalibrationsApi(password="...")

for task in calibrated_tasks:
    cal_api.create(task["gains"])
```

Revision #22

Created 16 August 2024 08:47:46 by Виктор Феденёв

Updated 10 May 2025 11:04:00 by Виктор Феденёв