

Синтез радиоизображений

Инструкции по синтезу радиоизображений, полученных Сибирским Радиодогелиографом с помощью программного обеспечения, разработанного в радиоастрофизическом отделе ИСЗФ СО РАН.

- [Синтез радиоизображений с помощью пакета `srh_synth`](#)
- [Синтез радиоизображений с помощью пакета `srhdata`](#)
- [Универсальный пакет для синтеза изображений `srhimages`](#)

Синтез радиоизображений с помощью пакета srh_synth

Установка пакета `srh_synth` на свой компьютер

ПО для синтеза радиоизображений SRG написано на языке программирования `Python 3`, поэтому прежде чем его использовать, вам необходимо установить интерпретатор этого языка.

Если вы синтезируете изображения на одном из серверов отдела радиоастрофизики ИСЗФ СО РАН, то Продолжите чтение инструкции с раздела Синтез радиоизображений, так как у вас уже установлены все необходимые пакеты.

Установка Python 3 в Linux

В большинстве популярных дистрибутивов Linux Интерпретатор `Python 3` обычно установлен по умолчанию. Проверить установлен ли Python на вашей машине можно, запустив следующую команду в терминале:

```
python --version
```

или

```
python3 --version
```

Если `Python` установлен, то вы команда выведет версию установленного Интерпретатора. Мы рекомендуем использовать `Python 3.10` или более свежую версию. Если же в вашей системе Python 3 не установлен, то установите его с помощью менеджера пакетов вашего

дистрибутива. Например, в Ubuntu для этого нужно в терминале выполнить следующую команду.

```
sudo apt install python3
```

Установка Python 3 в Windows

Интерпретатор языка `Python` не входит в состав операционной системы Windows, поэтому его придётся устанавливать отдельно. Это можно сделать двумя способами.

1. Установить "чистый" интерпретатор Python запустив инсталлятор, скачанный с [официального сайта Python](#).
2. Установить Python в составе менеджера пакетов `Anaconda` и `Miniconda`. В первом случае вместе с интерпретатором языка будут также установлен большой набор пакетов предназначенных для научных вычислений и визуализации данных. В случае же `Miniconda` будет установлен только интерпретатор языка Python и менеджер пакетов `conda`.

Создание виртуального окружения

Мы рекомендуем устанавливать ПО для синтеза радиоизображений в отдельном виртуальном окружении (virtual environment), чтобы избежать конфликта зависимостей. В случае, если вы планируете использовать Python только для синтеза радиоизображений, этот шаг можно пропустить.

TO BE DONE

Установка пакета `srh_synth`

Пакет `srh_synth` в текущей версии использует часть функций универсального пакета `srhimages` и устанавливается вместе с ним командой:

```
pip install -U "srhimages[anfinogentov] @ git+https://git.iszf.irk.ru/fedenev/srhimages"
```

Альтернативно можно установить пакет `srhimages` с полным набором опций.

```
pip install -U "srhimages[all] @ git+https://git.iszf.irk.ru/fedenev/srhimages"
```

Тогда вместе с пакетом `srh_synth` будет доступен также и код для синтеза изображений `srhdata`, разрабатываемый Марией Глоба. Если вы используете виртуальное окружение, то перед запуском команд установки его нужно активировать.

Синтез радиоизображений

Синтез радиоизображений состоит из двух этапов:

1. Калибровка коэффициентов усиления антенн - функция `srh_synth.srh_calib`
2. Синтез изображений - функция `srh_synth.srh_synth` В начале работы необходимо импортировать импортировать эти функции.

```
from srh_synth import srh_synth, srh_calib
```

Калибровка коэффициентов усиления антенн

Калибровка выполняется для заданного интервала времени. При этом калибруются не все изображения, а только некоторые. По умолчанию калибруются изображения с интервалом 5 минут. Затем полученные калибровочные коэффициенты подвергаются постобработке в ходе которой устраняются пространственные сдвиги между отдельными изображениями и выравниваются амплитуды коэффициентов усиления. Полученные калибровочные данные можно затем использовать для построения изображений внутри выбранного интервала на заданных частотах с произвольной скважностью и произвольным временем накопления.

Примеры команд для калибровки:

Калибровка для частот 6 и 6.4 ГГц для одного момента времени 2024-06-01 03:00:

```
calibrations = srh_calib("2024-06-01 03:00", frequency = [6000, 6400])
```

Калибровка наблюдений за весь день для частоты 6 ГГц:

```
calibrations = srh_calib("2024-06-01 00:00", "2024-06-01 12:00", frequency = [6000])
```

Калибровка наблюдений за два часа для частоты 16 ГГц:

```
calibrations = srh_calib("2024-06-01 02:00", "2024-06-01 04:00", frequency = [16000])
```

Калибровка наблюдений за весь день для частоты 3 ГГц с сохранением калибровок в файл 'calib.json':

```
calibrations = srh_calib("2024-06-01 00:00", "2024-06-01 12:00", frequency = [3000], write_to='calib.json')
```

Калибровка данных решётки 6-12 ГГц для одного момента времени 2024-06-01 03:00:

```
calibrations = srh_calib("2024-06-01 03:00", frequency = 'SRH0612')
```

Полный список параметров функции `srh_calib`:

- `start_time` - начало интервала времени для калибровки, например '2024-05-14 00:30'
- `stop_time` - конец интервала времени для калибровки, например '2024-05-14 03:00'
- `frequency` - список частот, которые надо откалибровать, например [3200, 4000]. Значение по умолчанию 'all' -- калибровать все частоты, 'SRH0306', 'SRH0612', 'SRH1224' -- калибровать все частоты соответствующей решётки.
- `n_proc` - Количество параллельно обрабатываемых калибровок. По умолчанию 16. Если это число больше числа имеющихся в системе процессорных ядер, то этот параметр уменьшается до числа ядер в системе. При назначении числа параллельных процессов, нужно иметь ввиду, что каждый такой процесс потребляет около 1 Гб оперативной памяти.
- `cadence` - скважность калибровок в секундах. По умолчанию калибруются данные с интервалом в 300 с. Менять не рекомендуется.
- `int_time` - Время накопления в секундах для каждого калибровочного изображения. По умолчанию 30 с. Менять не рекомендуется
- `parallel` - запускать калибровки параллельно в дочерних процессах. По умолчанию True. В случае установки в False, калибровки будут выполняться последовательно в основном процессе.
- `post_process` - постобработка калибровок с целью устранения 'дрожания' и 'уезжания' изображений. Кроме того выравнивается амплитуда коэффициентов усиления антенн
- `raw_dir` - директория в которой код программа будет искать сырые данные СРГ и куда их будет скачивать при необходимости. Рекомендуется для всех своих задач использовать одну и ту же директорию на достаточно емком диске.
- `db_file` - имя файла с базой данных SQLite для сохранения калибровок. По умолчанию калибровки сохраняются в файл 'calibrations.db' в текущем каталоге. Обратите внимание, что в БД создается запись для каждой комбинации даты наблюдения и частоты наблюдения.
- `write_to` - имя файла для сохранения калибровок в формате JSON. По умолчанию калибровки сохраняются только в локальную базу данных SQLite.
- `recalibrate` - по умолчанию `False`: программа берёт готовые калибровки из локальной БД SQLite, если они уже делались ранее. Если вы хотите пересчитать

калибровки, то установите этот параметр в `True`

Возвращаемое значение: Функция `srh_calib` возвращает словарь, в котором находятся результаты калибровки разобранные по частотным каналам и отсортированные по времени. Данную переменную нужно использовать для синтеза изображений.

Синтез радиозображений

После выполнения калибровки можно приступить непосредственно к построению изображений.

Примеры команд синтеза радиозображений

Синтез изображений на двух частотах для одного момента времени с использованием калибровок, полученных функцией `srh_calib`:

```
srh_synth("2024-06-01 03:00", frequency = [6000, 6400], calibrations=calibrations)
```

Синтез серии изображений на одной частоте для заданного интервала времени с временным разрешением 60 секунд, используя калибровки из файла 'calib.json':

```
srh_synth("2024-06-01 02:30", "2024-06-01 03:30", frequency=[3000], cadence=60, calibrations='calib.json')
```

Синтез изображений для заданного интервала времени с накоплением 30 секунд и временным разрешением 300 секунд. Изображения сохраняются в папку "srh20240601", имена сохранённых файлов будут напечатаны в консоли:

```
files = srh_synth("2024-06-01 03:00", int_time=30, cadence=300, calibrations=calibrations, out_dir='srh20240601')
print(files)
```

Полный список параметров функции `srh_synth`:

- `start_time` - начало интервала времени для калибровки, например '2024-05-14 00:30'
- `stop_time` - конец интервала времени для калибровки, например '2024-05-14 03:00'
- `int_time` - время интегрирования для каждого изображения в секундах. По умолчанию изображения строятся из одной наблюдательной записи (скана).
- `cadence` - скважность (временное разрешение) синтезируемых изображений. По умолчанию 0 - изображения строятся с максимально возможной скважностью.
- `frequency` - список частот, на которых надо построить изображения, например [3200, 4000]. Значение по умолчанию 'all' -- строить изображения на всех частотах. Данные на выбранных частотах должны быть предварительно откалиброваны функцией `srh_calib`

- `out_dir` - путь к каталогу, куда будут записаны построенные изображения. Если в данном каталоге уже имеются файлы изображений, они будут перезаписаны при совпадении имён. По умолчанию изображения записываются в текущий каталог
- `raw_dir` - директория в которой код программа будет искать сырые данные SRG и куда их будет скачивать при необходимости. Рекомендуется для всех своих задач использовать одну и ту же директорию на достаточно емком диске.
- `calibrations` - калибровочные данные, возвращаемые функцией `srh_calib`, или путь к JSON файлу с сохранёнными калибровками.
- `naxis` - размер изображения в пикселях (по умолчанию 512)
- `cdel` - размер одного пикселя в секундах дуги (по умолчанию 5.)
- `save_RL` - сохранять изображения в правой (R) и левой (L) круговой поляризации. По умолчанию False - сохраняется интенсивность (I) и поляризация (V).
- `no_channel_dirs` - не создавать подкаталоги для каждого частотного канала. По умолчанию False - подкаталоги создаются
- `save_psf` - сохранять диаграмму направленности для каждого изображения в отдельный FITS файл
- `n_proc` - Количество параллельно синтезируемых изображений. По умолчанию 16. Если это число больше числа имеющихся в системе процессорных ядер, то этот параметр уменьшается до числа ядер в системе.
- `round_beam` - использовать симметричную круглую "чистую" диаграмму направленности при "чистке" изображений. Данная опция экспериментальная. По умолчанию False. Применение этой опции приводит к уменьшению пространственного разрешения изображения.
- `high_resolution` - Искусственно увеличить пространственное разрешения изображения за счёт уменьшения размера "чистой" диаграммы направленности. Данная опция экспериментальная. Её использование имеет смысл только для заведомо компактных источников. А интерпретация полученных изображений возможна только с оговорками.
- `compressed` - сохранять изображения в сжатом формате FITS. По умолчанию True.
- `parallel` - запускать синтез изображений параллельно в дочерних процессах. По умолчанию True. В случае установки в False, изображения будут строиться последовательно в основном процессе.
- `clean_weights` - Путь к FITS файлу с радиомизображением Солнца на низкой частоте, на которое нет перекрытия порядков или двухмерный numpy массив с весами, которые будут использоваться алгоритмом CLEAN при поиске точки с максимальной интенсивностью. Массив должен иметь те же размеры, что и изображение, то есть `[naxis, naxis]`. В случае, если в этом параметре передается путь к FITS файлу, изображение из файла будет автоматически преобразовано в веса для CLEAN и отмасштабировано для требуемых значений NAXIS и CDEL.

Возвращаемое значение: Функция `srh_synth` возвращает список имен файлов синтезированных изображений.

Синтез радиоизображений с помощью пакета srhdata

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 3.13, код Марии Глобы - пока только на 3.10! Поэтому желательно устанавливать именно ту версию, которую нужно. Для расчётов на удалённых машинах версия питона не особо важна, но рекомендуется использовать что-то новое.

```
conda create -c conda-forge -n srhsynth python=3.13 uv git
conda activate srhsynth

uv pip uninstall srhimages srh_synth srhdata
uv pip install -U "srhimages[anfinogentov,global] @
git+https://git.iszf.irk.ru/fedenev/srhimages"
# или [global], [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
```

Установка из каталога с git-репозиторием (для разработки)

```
# git clone https://git.iszf.irk.ru/fedenev/srhimages
# cd ./srhimages
git pull -a
uv pip uninstall srhimages srhsynth
uv cache clean
```

```
uv pip install -U -e "srhimages[anfinogentov] @ ."
```

Использование уже установленного окружения на сервере ИСЗФ

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

Интерфейс командной строки

```
srhimages create_synth_tasks --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__":
    # for result in 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:

- `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.

2. `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", 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.
- **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="single-day.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__":
    for i, results1 in enumerate(srhimages.run_computation("./single-day.json", "globa")):
        results1.to_json_file(f"./single-day-results-{i}.json")

# изображения 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__":
    for i, calibrations in enumerate(srhimages.run_computation("calibs.json", "globa", \
        skip_images=True)):
        if len(calibrations) == 0:
            continue
        freq = calibrations.frequencies[0]
        calibrations.to_json_file(f"calibs_ready_{i}_{freq}.json")

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

# если всё хорошо, то в файлах 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__":
    for i, results in enumerate(srhimages.run_computation("synth.json", "anfinogentov")):
        if len(results) > 0:
            dt = results.dates[0]
            freq = results.frequencies[0]
            results.to_json_file(f"result_{freq}_{dt}_{i}.json")

```

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

Загрузка калибровок на сервер

```
from srhimages import TaskCollection
calibrated_tasks = TaskCollection("synth_progress.json")
calibrated_tasks.extract_gains().upload()
```