# ***PyHawk*** **User Manual**

**(Version1.0)**

**Yi Wu[1], Fan Yang[1,2,*], ShuHao Liu[1], Ehsan Forootan[2]**

[1]School of Physics, Huazhong University of Science and Technology, Wuhan 430074, China

[2]Geodesy Group, Department of Sustainability and Planning, Aalborg University, Aalborg 9000, Denmark

**Update: November 20, 2024**

Corresponding author: Fan Yang, `fany@plan.aau.dk`

## 1 Overview

PyHawk is an open-source toolbox tailored to the low-low satellite-to-satellite tracking (ll-sst) missions. It is built on an advanced object-oriented Python framework, aiming to provide user-friendly services to especially the non-expert users in geodesy community. PyHawk comprises a comprehensive data processing chain that handles various stages of ll-sst tasks, including payload data pre-processing, background force modeling, orbit integration, inter-satellite range calibration, and Time-Varying Gravity (TVG) recovery for the Gravity Recovery and Climate Experiment (GRACE) mission (2002-2017) and its successor mission (GRACE-FO, 2018-present).

The main features of the software are:

- It adopts advanced data exchange mechanisms, enabling users to interact with the software through configuration files, which eliminates the need to modify source code and reduces the risk or complexity associated with code changes.
- The software has been optimized to achieve computational efficiency comparable to that of Fortran and C++, languages commonly used in existing GRACE (-FO) toolboxes
- PyHawk is compatible with multiple platforms and allows for easy and fast installation on systems such as Windows, Linux or clusters.
- PyHawk has a modular structure, where code is highly decoupled internally for an easy use, comprehension and extension.

## 2 Dependence

For the sake of high numerical efficiency, a hybrid programming strategy is adopted for this project, please see section 8.1 of the main text for more information. Python is known to be not good at looping, and the most time-consuming part that heavily relies on looping is found to be the gravity acceleration computation, which we thereby realize by C++ first and compile into a library to be called by python. Similarly, we have another important dependency, that is, the Standards Of Fundamental Astronomy (SOFA, `http://www.iausofa.org/`), please see Section 8.1 for more information. In PyHawk, we compile the C version of SOFA into dynamic libraries (.so for Linux, .dll for Windows) and use Cython to call them from Python.

In addition, other python dependencies like numpy, tqdm, scipy, h5py, numba, Quaternion, jplephem, and matplotlib are all open-source and available online. To avoid potential conflict, Python $>$ 3.8 is strongly recommended.

## 3 Installation

The project can be firstly downloaded from GitHub (`https://github.com/NCSGgroup/PyHawk.git`). Then, a Python environment with required dependencies must be created, where two options are offered: (1) without Conda (`https://www.anaconda.com/download`), (2) with Conda.

For installation without Conda:

**Step 1:** First, make sure you have at least Python version 3.8 installed on your system:

```
python --version
```

**Step 2:** Then, download pip, the Python package installer:

```
python -m pip install pip
```

**Step 3:** Install other dependencies in sequence:

```
pip install numpy.
pip install tqdm.
pip install scipy.
pip install h5py.
pip install numba.
pip install Quaternion.
pip install jplephem.
pip install matplotlib.
```

For installation with Conda (suggested):

**Step 1:** `conda create -n py-hawk python=3.8.10`

**Step 2:** `source activate py-hawk`

**Step 3:** `pip install -r requirments.txt`

**Troubleshooting:** PyHawk has a dependence on two essential dynamic libraries as addressed in previous section. For this, we have provided the pre-compiled dynamic libraries (created individually for Linux and Windows) to simplify the installation. However, these pre-built libraries may occasionally fail because of incompatibility between the libraries and running platform. If this is the case, we suggest you to use the provided C (C++) source code together with the compiling script (PyHawk/lib) to automatically compile a library compatible with your own platform:

```
Open a terminal under PyHawk/lib, and execute the 'make' command.
```

This can automatically generate a library of 'GravHar.so' if Linux is the intended platform; however, for Windows, you may need an extra software (Visual C++) to generate the 'GravHar.dll' library. For another important dependency 'sofa', please see the official website for its source code and cookbook (`http://www.iausofa.org/`). Please also feel free to reach out to us about any further issues.

## 4 Sample data

Essential sample data for running the demos are provided that include (but not limited to):

- **GRACE-(FO) Level-1b data:** A few months data for demo. Suggested default path: /PyHawk/data.
- **De-aliasing product:** A few months data. Suggested default path: /PyHawk/data/AOD/RL06.
- **EOP:** Suggested default path: /PyHawk/data/eop.
- **Load Love Number:** Suggested default path: /PyHawk/data/LoveNumber.
- **Static Gravity:** Suggested default path: /PyHawk/data/StaticGravityField.
- **Ephemerids:** Suggested default path: /PyHawk/data/Ephemerides.
- **Auxiliary:** Suggested default path: /PyHawk/data/Auxiliary.
- **Benchmark:** Suggested default path: /PyHawk/data/Benchmark.

Be aware that these sample data are of large size, so that we put them at an open data repository (`https://doi.org/10.5281/zenodo.14205243`) for use when running the demos.

# 5 Quick Start

Here we present three demo (demo1, demo2, demo3), which are available under /PyHawk/demo, to showcase the usage of PyHawk. For the sake of generality, we design three specific demos to cover the major interests of the users: (1) demo-1 for orbit determination (2) demo-2 for background force modeling and (3) for the gravity field modeling. Be aware that, since PyHawk is a comprehensive toolbox for various research objectives, it is unlikely to be exhaustive to list all operations here. Nevertheless, one can see the detailed description at the comments of each demo in its scripts.

To be able to run the demo, please be aware that sample data as addressed in previous section are required. And for a quick start, we suggest to put all these sample data at default path: /PyHawk/data/. Nevertheless, as an advanced user, one can place data at any desired place as long as the setting files are well configured. Here is a brief tutorial of three demos.

## 5.1 Demo-1

This demo is the basis of orbit determination. For the sake of computational efficiency, we will demonstrate the calculation of only one arc segment (6-hourly dataset). In this case, no parallelization is required so that the computation should be affordable for personal computer. The time for completing this demo is around 5 minutes, which may slightly varies with the platform.

**Step 1:** Before running, you need to confirm the input data path and date span by opening the "interfaceConfig.json" configuration file.

**Step 2:** Navigate to the /PyHawk/demo/, open a terminal and execute the command:

```
python demo_1.py
```

**Step 3:** The output consists of calibrated satellite orbits for each arc segment, stored in HDF5 format, typically located in directory of /PyHawk/temp/StateVector. The screenshot of running refers to Fig. 1.

```
GRACE level 1b reading for SatID.A: 100%|          | 1/1 [00:00<00:00,  2.87it/s]
GRACE level 1b reading for SatID.B: 100%|          | 1/1 [00:00<00:00,  2.94it/s]
Kinematic orbit reading for SatID.A: 100%|          | 1/1 [00:00<00:00, 110.30it/s]
Kinematic orbit reading for SatID.B: 100%|          | 1/1 [00:00<00:00, 113.53it/s]
Orbit conversion from ITRS into GCRS: 100%|          | 17280/17280 [00:11<00:00, 1528.03it/s]
Arc unpacking for ACC_A: 100%|          | 4/4 [00:00<00:00, 300.82it/s]
Arc unpacking for ACC_B: 100%|          | 4/4 [00:00<00:00, 386.79it/s]
Arc unpacking for SCA_A: 100%|          | 4/4 [00:00<00:00, 372.45it/s]
Arc unpacking for SCA_B: 100%|          | 4/4 [00:00<00:00, 374.56it/s]
Arc unpacking for GNV_A: 100%|          | 4/4 [00:00<00:00, 1059.37it/s]
Arc unpacking for GNV_B: 100%|          | 4/4 [00:00<00:00, 1053.98it/s]
Arc unpacking for KBR_A: 100%|          | 4/4 [00:00<00:00, 1187.10it/s]
Orbit integration: 100%|          | 4311/4311 [03:20<00:00, 21.53it/s]
Orbit integration: 100%|          | 4311/4311 [03:21<00:00, 21.41it/s]
```

**Figure 1.** The screenshot of orbit determination process.

### 5.2 Demo-2

This shows how to calculate each background force model and quickly generates benchmark test result as an illustration of the accuracy of the software, please see section 2.2 of the main text for more information. A complete process of this demo takes about 4 minutes.

**Step 1:** Navigate to the /PyHawk/demo/, open a terminal and execute the command:

```
python demo_2.py
```

**Step 2:** The output includes the benchmark test of acceleration caused by each individual force model, which can be found at directory of /PyHawk/result/benchmarktest/ and one example can refer to Fig. 2.

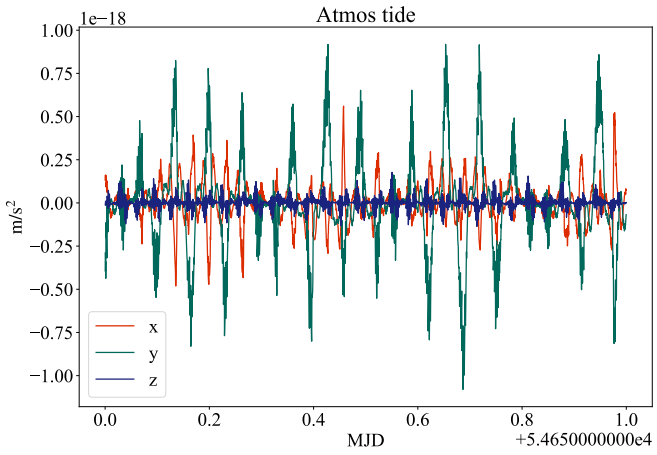


**Figure 2.** One example of demo-2 outputs: benchmark test for atmospheric tides, where three-axis acceleration differences are demonstrated.

### 5.3 Demo-3

This demonstrates the process of temporal gravity recovery, which is also the primary objective of the ll-sst missions such as GRACE, GRACE-FO and future satellite gravity missions. Figure 3 illustrates how this demo is proceeding. Similarly, as an demonstration, we choose to recover the gravity from one arc (6-hourly data) rather than the one-month arc for gaining an efficiency. In this manner, parallelization is no longer required and computation should be affordable for personal computer. The completion time is estimated to be around 9 minutes.

**Step 1:** Navigate to the /PyHawk/demo/, open a terminal and execute the command:

```
python demo_3.py
```

**Step 2:** The output consists of standardized spherical harmonic coefficients of the gravity field, which are stored in the directory of /PyHawk/result/SH/.

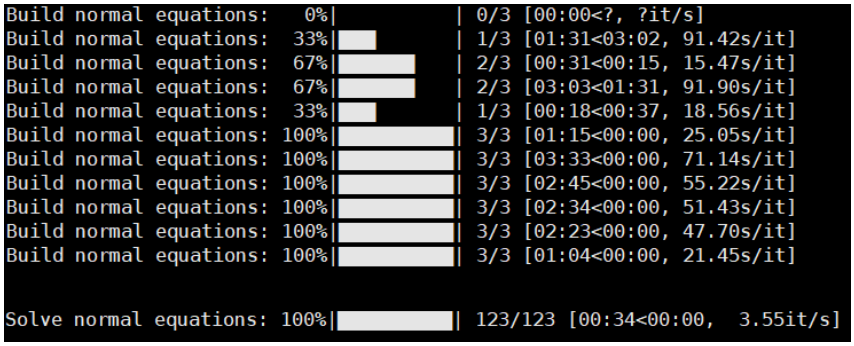


**Figure 3.** Screenshot of temporal gravity recovery process.

## 6 Configuration parameters

PyHawk is designed as a modular structure, which primarily comprise four modules: (1) Data Pre-processing, (2) Background Force Modeling, (3) Orbit Integration and Range Calibration, and (4) Parameter Estimation, please see Section 2 of the main text for more details. Each of these modules involves numerous optional parameters that make PyHawk a flexible toolbox. Due to the complexity, we classify these parameters into various groups, with each being handled by an individual configuration file (managed by JSON), see below:

- **AccelerometerConfig.json:** for the accelerometer calibration, see Table 1.

- **AdjustOrbitConfig.json:** for the orbit calibration and KBRR calibration, see Table 2.
- **ProcessControlConfig.json:** for the workflow controlling, like selecting functional modules and allocating parallel resources, see Table 3.
- **ForceModelConfig.json:** for the selection of desired force model and configuring the corresponding parameters, see Table 4.
- **ConstantsConfig.json:** for configuring the values of all constants that are involved in the code, such as the graviational constant (GM) of the Earth, see Table 5.
- **DesignParameterConfig.json:** for the definition of the global and local parameter in the design matrix, see Table 6.
- **InterfaceConfig.json:** for selecting the desired raw data and configuring the access (decoding) to the raw data, see Table 7.
- **NEQConfig.json:** for the configuration of the normal equation, see Table 8.
- **ODEConfig.json:** for selecting and configuring the ordinary differential equation parameters, see Table 9.
- **SolverConfig.json:** for configuring the gravity field solution parameters, see Table 10.
- **FrameConfig.json:** for configuring the reference time and coordinate frame, see Table 11.

**Table 1.** Description of Accelerometer configuration

| Parameter | Descriptions |
|---|---|
| Initial Bias | Initial values of bias. |
| Initial Scale | Initial values of scale. |
| Scale | whether to calibrate scale factor. |
| Bias | whether to calibrate bias factor. |
| Bias-linear | linear fitting. |
| Bias-quadratic | quadratic fitting. |
| Time interval | Frequency of parameter estimation. |

**Table 2.** Description of adjust orbit and KBRR configuration

| Parameter | Descriptions |
|---|---|
| Outlier Limit | the outlier threshold. |
| Outlier Times | the outlier range (e.g., 3 $\sigma$). |
| Parameter Fitting | the choice of calibration parameters (e.g., bias plus constant/trend). |
| Iterations | number of calibration iterations. |
| Arc Length | calibration length division (e.g., 1.5h/3h). |

**Table 3.** Description of process control configuration

| Parameter | Descriptions |
|---|---|
| Module Control | module selection (e.g., preprocessing and orbit integration). |
| Parallel Control | parallel node allocation (e.g., pre-processing: 30 nodes). |
| Satellite Mission | Satellite mission type (e.g., GRACE or GRACE-FO). |

**Table 4.** Description of force model configuration

| Parameter | Descriptions |
|---|---|
| Include Force Type | force model selection (e.g., consider static model without atmospheric tide). |
| LoveNumber Config | lovenumber selection and degree control (e.g., Wang and max degree: 180). |
| Pole Tide Config | pole tide parameter control (e.g., consider only ocean pole and max degree: 180). |
| Solid Tide Config | solid tide parameter settings (e.g., consider zero tide correction). |
| Atmos Tide Config | atmospheric tide parameter selection (e.g., consider P1, S1, K1 without S2). |
| Ocean Tide Config | ocean tide parameter selection (e.g., select EOT11a or FES2014). |
| Three Body Config | three-body parameters selection (e.g., consider J2, without Mars, etc.). |
| De-aliasing Config | dealiasing parameter selection (e.g., consider OCN and time resolution (3/6), etc.). |
| Static Model Config | static model parameter selection (e.g., select GIF 48 or GGM05C). |
| Relativity Config | relativity parameter selection (e.g., consider only Schwarzschild term). |

**Table 5.** Description of constants configuration

| Parameter | Descriptions |
|---|---|
| J2000-MJD | simplified Julian date starting point (e.g., 51544.5). |
| Difference-Tai-GPS | The difference between TAI (International Atomic Time) and GPS Time (e.g., 19 seconds). |
| Difference-Tai-TT | the difference between TAI and TT (Terrestrial Time) (e.g., 32.184 seconds.). |

**Table 6.** Description of design parameter configuration

| Parameter | Descriptions |
|---|---|
| Transition Matrix | state transition matrix parameter settings (e.g., set as local parameters, with parameter number being six or nine);. |
| Accelerometer | accelerometer calibration parameter settings (e.g., set as local parameters, with parameter number being six or nine);. |
| Stokes Coefficients | spherical harmonic coefficient parameter settings (e.g., set as global parameters, with order and degree set to 60).. |

**Table 7.** Description of interface configuration

| Parameter | Descriptions |
|---|---|
| Date span | defines the start and end dates for the time period of interest in the format [start date, end date]. |
| Date delete | defines the start and end dates for the time period of useless in the format [start date, end date]. |
| Include payloads | A dictionary specifying which payloads (instruments) to include for processing (e.g., include ACC and SCA). |
| Payloads sampling rate | this defines the sampling rate for each payload (e.g., ACC: 1s, KBR: 5s). |
| Arc length | define the arc segment length (e.g., 3h/6h). |
| Path of files | specifies the file paths for various data inputs and outputs. These are used for locating payload data, raw data, reports, and logs. |

**Table 8.** Description of normal equation configuration

| Parameter | Descriptions |
|---|---|
| Orbit Config | Determine whether to include orbital information and whether to use a simplified dynamic or kinematic orbit when constructing the normal equation. |
| SST Config | when constructing the normal equation, determine whether KBRR information should be considered. |

**Table 9.** Description of ordinary differential equation configuration

| Parameter | Descriptions |
|---|---|
| Complex Config | define single step method and multi-step integrator(e.g., single: RKN; multi: GaussJackson). |
| Single Step Config | define detailed parameters of the single step method integrator (e.g., stepsize: 5s). |
| Multi Step Config | define detailed parameters of the multi-step method integrator (e.g., order: 8, limit of integral calibration times: 10). |
| Accuracy | threshold for stopping iteration. |
| Kinematic orbit | configure the external kinematic orbits. |

**Table 10.** Description of solver configuration

| Parameter | Descriptions |
|---|---|
| Orbit-Kin Factor | setting the weight factors for the orbit and kbr (e.g., Factor: $1e^{10}$). |
| SST Option | select the type of SST data to be used in the computation (e.g., range rate). |
| Output path | define the directory to save outputs. |
| Product format | select the convention to formulate the gravity field file. |

**Table 11.** Description of reference frame configuration

| Parameter | Descriptions |
|---|---|
| EOP Path Config | define the path of EOP and its related files. |
| Coordinate Transformation method | define which method to implement the coordinate transformation. |
| EOP parameters Config | define the choice of EOP |
| Ephemerides Configuration | choices of Ephemerides: DE421, DE430, DE440 etc. |

## 7 Software Architecture

PyHawk has a comprehensive structure due to the complexity of the processing chain of GRACE(-FO) missions. However, a well-designed software architecture can substantially reduce maintenance costs, enhance user-friendliness, and significantly increase software readability. To this end, PyHawk follows the popular and modern layered architecture, where five layers are defined and used to interact with each other. All these layers are decoupled, with each running at its own logic, but five layers are supportive with each other. Please see Fig. 4 for an overview of this architecture.

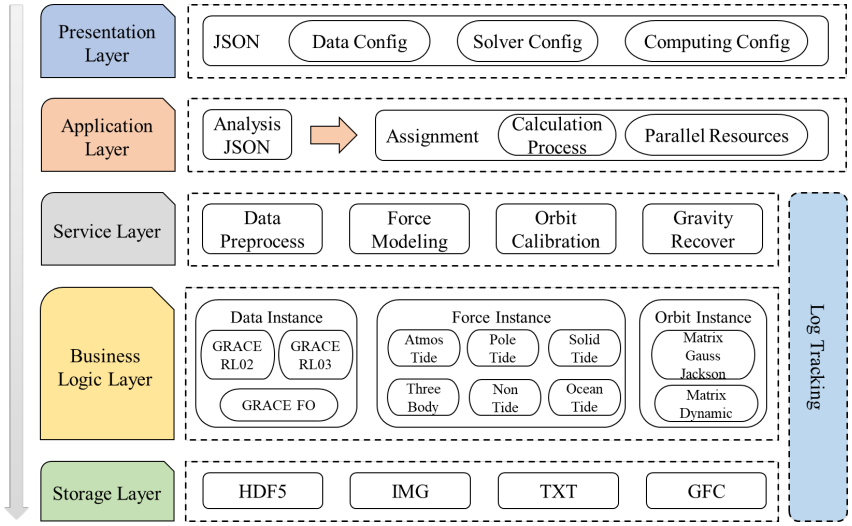


**Figure 4.** A five-layered hierarchy architecture for PyHawk: (1) presentation layer–responsible for the user interactions, e.g., the setting files via JSON; (2) application layer–to coordinate and process user requests, e.g., decoding JSON input and assigning a job; (3) service layer–to define/generate a workflow according to the assigned job; (4) business logic layer–the basic and unit function that implement the job; (5) data storage layer–responsible for the persistent data storage.

Thanks to this modern architecture, users only need to interact with PyHawk at the presentation layer, without any risk of modifying the core code in the underlying layers. In addition, in this manner, visualization of the presentation layer can be also straightforward, e.g.,, handling the JSON setting files on the client end or the browser end (this has not been realized yet but will be integrated soon in coming release). More importantly, the layered architecture ensures a flexible extension of PyHawk in future, for example, towards the next-generation satellite gravity missions.

# 8 High performance computation

Python is a high-level programming language known for its readability, simplicity, and flexibility, featuring a rich set of libraries and frameworks with cross-platform capabilities. However, as an interpreted language, Python tends to be slower than C or Fortran. To achieve high-performance computing (acceleration) with Python, we have adopted multiple strategies to optimize the code, see the overview at Fig. 5.

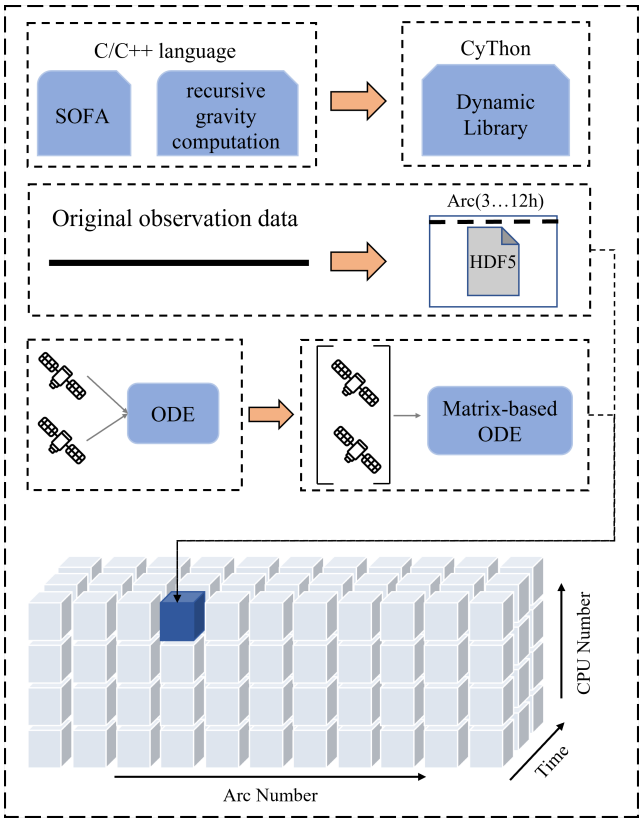


**Figure 5.** Implemented strategies for computation acceleration: (1) hybrid programming and effient data management; (2) multi-channel ODE; (3) parallel computation.

## 8.1 hybrid programming and efficient data management

The SOFA package, developed and maintained by the International Astronomical Union (IAU), provides standardized algorithms for precise astronomical computations and data processing. It is a super fast toolbox that supports multiple platforms and programming languages, including C and Fortran. In PyHawk, tasks like coordinate and time reference conversions rely on this reliable SOFA toolbox as well. However, since SOFA lacks a Python

version, PyHawk addresses this by compiling the C version of SOFA into dynamic libraries (.so files for Linux and .dll files for Windows) and invoking them in Python via Cython.

In addition, it is known that Python's loop efficiency is much lower than that of C++, making it much more time-consuming when the job involves numerous looping operations. We have optimized the code by avoiding loop in majority of PyHawk, but unfortunately this can not be bypassed during the computation of non-spherical gravitational geopotential/force where the recursive is dependent on the previous output. Therefore, like SOFA, we program this part of computation in C++ first, and then compile it into a library to be called by Cython.

PyHawk automates the compilation of these libraries, streamlining updates to SOFA and the aforementioned algorithm. Using Cython, the software combines C's high performance with Python's usability, enabling efficient, maintainable, and scalable implementations for complex calculations.

In addition, due to the complexity of data processing of ll-sst missions, there could be giant data, either as the interim data or final output, that significantly affect the efficiency of the software. To realize an efficent data management, PyHawk employs the HDF5 format due to its support for large-scale, multi-dimensional data, cross-platform compatibility (Linux, Windows, HPC), and integration with multiple programming languages (Python, C, Fortran). HDF5 enables data compression, chunk storage, and simultaneous multi-process access, facilitating efficient parallel computations. This approach minimizes file clutter, optimizes storage, and reduces read times, making it ideal for scientific computing and analysis.

### 8.2 Multi-channel ODE

Orbital integration, essential for predicting spacecraft trajectories, faces increasing challenges with the rise of multi-satellite missions like NGGM. Traditional single-satellite methods struggle to efficiently handle multiple satellites, leading to significant computational overhead, especially for (near) real-time monitoring. To address these limitations, this study introduces a novel multi-channel ODE solution.

In this approach, the states (position and velocity) of all satellites are consolidated into a state matrix, enabling simultaneous updates. Similarly, the forces acting on each satellite

are represented as a force matrix, accounting for both conservative and non-conservative forces. At each time step, matrix operations update all satellite states in parallel, leveraging computational resources more effectively and avoiding delays from sequential processing. This parallelism, coupled with continuous state matrix iteration, forms the foundation of the numerical integration process.

To evaluate the efficiency of this method, an experiment compared the multi-channel integrator with traditional approaches. Data lengths ranged from 500 to 1000, divided into 30 groups. The traditional integrator performed three computations per group, while the multi-channel integrator expanded data into a 3D matrix and computed once using the Gauss-Jackson 8th order method. Results demonstrated a 30% improvement in computational efficiency (Fig. 6).

This method significantly reduces computation time while maintaining accuracy and adapts seamlessly to varying satellite numbers and configurations, making it ideal for large-scale satellite missions. Beyond orbital integration, it has potential applications in fields like physical simulations and engineering computations. PyHawk employs this matrix-based method to enhance computational efficiency, ensuring compatibility with next-generation gravity satellite processing and expanding its application potential across scientific and engineering domains.

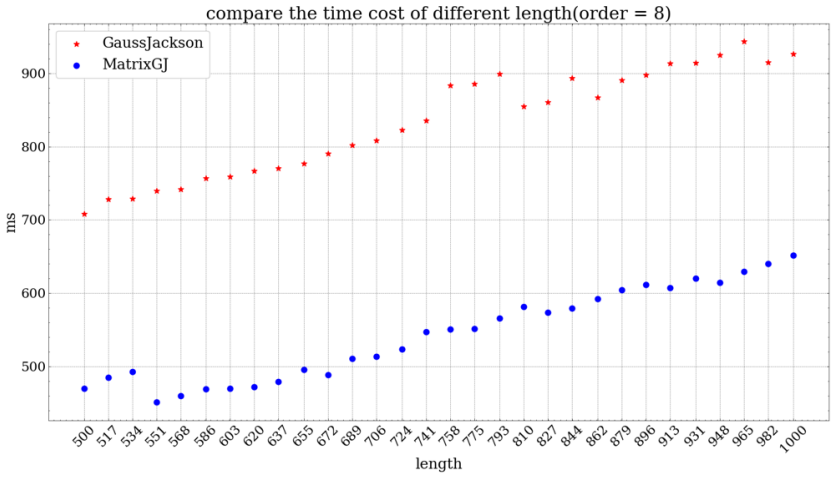


**Figure 6.** Comparison in terms of computational efficiency: the matrix based GJ (i.e., the multi-channel method) versus the traditional GJ.

### 8.3 Parallel computation

PyHawk is challenged with growing data and computational needs. For acceleration, PyHawk requires to implement multiprocessing technique, as which is common for other similar software in geodesy community. Other reasons for choosing multiprocessing (instead of other techniques) for parallel computation include:

- High Resource Utilization: Maximize CPU usage for improved performance.
- Stability: Independent processes ensure minimal interference and enhanced system reliability.
- Enables dynamic task allocation, boosting efficiency.

This can be done by Python's own multiprocessing module, or alternatively by extra python toolbox such as the Message Passing Interface (MPI, and will be incorporated in future). Leveraging process pools and the map function from the multiprocessing module, PyHawk dynamically allocates tasks to each process, and users can configure the amount of processes for specified job/task through the configuration file. On a supercomputer with 240GB RAM and 56 cores, PyHawk achieves monthly time-varying gravity fields in approximately 2 hours, where the employed parallelization includes 30 processes for orbit/range calibration, and 15 processes for gravity recovery. This robust design ensures efficient handling of large-scale data, meeting the demands of modern satellite missions.

## 9 Output description

PyHawk is designed to provide a complete ll-SST data processing chain, where a number of temporary/output related to each processing step can be optional to save for inspection or later use. These files are stored generally in HDF5 format, which is a self-description data container that can be easily accessed. The major outputs are summarized in Table 12.

Here, as the most important output, we present a simple description of the temporal gravity field obtained from PyHawk, see Fig. 7. The gravity field data is organized following the standard convention, i.e., GFC, as recommended by the ICGEM (International Centre for Global Earth Models). As one can see from Fig 7 that the data comprises two parts: a header where basic informaton are recorded, and another main body where the spherical harmonic coefficients as well as their uncertainties (unavailable at the moment) are sorted

**Table 12.** A summary of output files

| Parameter | Descriptions |
|---|---|
| State Vector | calibrated orbits |
| Preprocessed Data | Level-1b data after preprocessing |
| Along-track Range Rate | prefit KBRR-residuals before and after calibration |
| Orbit Adjustment | calibrated local parameters such as satellite's initial state vectors and accelerometer parameters |
| Non Conservative Force | non-conservative force after calibration |
| Normal Equation | Normal equation matrix. |
| Design Matrix | Design matrix and the observations in terms of orbit and KBRR |
| Temporal Gravity Field | Dimensionless spherical harmonic coefficients up to certain degree and order. |

by the degree and order. In this file, L denotes the degree, M represents the order, and C and S are the corresponding spherical harmonic coefficients.

```
*******************************************************************************
model converted into ICGEM-format at: Wed Oct 23 18:01:39 2024
*******************************************************************************

begin_of_head ===================================================================
product_type                gravity_field
modelname                   GSM-2_2009-01-01-2009-01-31_GRAC_PyHawk_BA01_0600
radius                      6.3781363000E+06
earth_gravity_constant      3.9860044150E+14
max_degree                  60
norm                        fully_normalized
tide_system                 zero_tide
errors                      formal

key       L      M         C                  S              sigma C        sigma S
end_of_head ===================================================================
gfc       0      0  +1.0000000000E+00  +0.0000000000E+00    +0.0000E+00    +0.0000E+00
gfc       1      0  +0.0000000000E+00  +0.0000000000E+00    +0.0000E+00    +0.0000E+00
gfc       2      0  -4.8416957422E-04  +0.0000000000E+00    +0.0000E+00    +0.0000E+00
gfc       3      0  +9.5723689797E-07  +0.0000000000E+00    +0.0000E+00    +0.0000E+00
gfc       4      0  +5.4000888726E-07  +0.0000000000E+00    +0.0000E+00    +0.0000E+00
gfc       5      0  +6.8614209754E-08  +0.0000000000E+00    +0.0000E+00    +0.0000E+00
gfc       6      0  -1.5000131515E-07  +0.0000000000E+00    +0.0000E+00    +0.0000E+00
gfc       7      0  +9.0488915998E-08  +0.0000000000E+00    +0.0000E+00    +0.0000E+00
gfc       8      0  +4.9467856886E-08  +0.0000000000E+00    +0.0000E+00    +0.0000E+00
gfc       9      0  +2.8005894590E-08  +0.0000000000E+00    +0.0000E+00    +0.0000E+00
```

**Figure 7.** Temporal gravity field in terms of spherical harmonic coefficient up to degree and order of 60, following the convention of ICGEM.