PyGAMIT-Bridge
User Manual

A Python Toolkit for Bridging Modern GNSS Data Formats
with GAMIT/GLOBK Processing

Version 0.1.0

February 2025

Jinzhen Han

Sungkyunkwan University, South Korea

https://github.com/geumjin99/pygamit-bridge

Contents

Introduction

PyGAMIT-Bridge is a lightweight Python toolkit designed to bridge the compatibility gap between modern International GNSS Service (IGS) data infrastructure and the GAMIT/GLOBK software suite. Since 2020, three major changes in the GNSS data ecosystem have introduced friction into routine GAMIT processing workflows:

1. **CDDIS Earthdata Authentication** — NASA's CDDIS discontinued anonymous FTP access in October 2020, requiring Earthdata Login. Failed authentication results in an HTML login page disguised as a data file, causing silent downstream failures.
2. **IGS Long Filenames** — Starting GPS Week 2238 (November 2022), IGS adopted new long product filenames, while GAMIT's internal scripts expect legacy short filenames.
3. **RINEX 3/4 Incompatibility** — GAMIT 10.71's `makexp` module cannot correctly parse RINEX 3's `SYS / # / OBS TYPES` header, causing empty batch files and breaking the processing chain.

PyGAMIT-Bridge addresses these issues through three focused modules, as summarized in Table 1.

Table 1: PyGAMIT-Bridge module overview

Module	Problem Addressed	Key Source File
1. Smart Downloader	CDDIS Earthdata authentication	`downloader.py`
2. Format Bridge	RINEX 3/4 incompatibility	`converter.py`
	Empty `makexp` batch file	`batch_fallback.py`
	IGS long filename mapping	`preprocessor.py`
3. Output Parser	Non-standardized GAMIT output	`parser.py`

Design Philosophy

- **Pure Python, Standard Library Only** — No third-party dependencies. Requires only Python $\geq$ 3.7.
- **Thin Wrapper** — Serves as a preprocessing and postprocessing layer around existing GAMIT workflows, without modifying GAMIT's Fortran source code.
- **Modular** — Each module can be used independently.

File Structure

```
pygamit-bridge/
|-- setup.py # Installation script
|-- bin/ # (reserved for future scripts)
|-- examples/
| `-- process_day.py # End-to-end usage example
`-- pygamit_bridge/
    |-- __init__.py # Package metadata (v0.1.0)
    |-- cli.py # Unified CLI entry point
    |-- downloader.py # Module 1: CDDIS smart downloader
    |-- converter.py # Module 2a: RINEX 3 -> 2 converter
    |-- batch_fallback.py # Module 2b: makexp batch fallback
    |-- preprocessor.py # Module 2c: Product preprocessing
```

```
    |-- parser.py # Module 3: Output parser
    `-- utils.py # GPS time utilities
```

Listing 1: Project directory layout

Installation

System Requirements

- Python $\geq$ 3.7
- GAMIT/GLOBK 10.71 (installed and functional)
- `CRX2RNX` utility for Compact RINEX decompression
- `wget` command-line tool (for data downloading)
- NASA Earthdata account (`https://urs.earthdata.nasa.gov`)

Installation Steps

```
# Clone the repository
git clone https://github.com/geumjin99/pygamit-bridge.git
cd pygamit-bridge

# Install in development mode
pip install -e .

# Verify installation
pygamit-bridge --version
# Output: 0.1.0
```

Listing 2: Installation from source

Earthdata Configuration

Before using the downloader module, you must configure NASA Earthdata credentials. The toolkit uses `wget` with cookie-based authentication.

```
# Create a .netrc file for Earthdata authentication
echo "machine urs.earthdata.nasa.gov login <USERNAME> password <PASSWORD>" \
    >> ~/.netrc
chmod 0600 ~/.netrc

# Create .urs_cookies file (wget will populate it automatically)
touch ~/.urs_cookies
```

Listing 3: Earthdata cookie setup

Warning

The `.netrc` file contains your credentials in plain text. Ensure proper file permissions (`chmod 0600`) and never commit it to version control.

Module 1: Smart Downloader

The `downloader` module (`downloader.py`) automates retrieval of GNSS observation data and IGS precise products from CDDIS with Earthdata cookie-based authentication.

Key Features

- **Earthdata Authentication** — Wraps `wget` with cookie parameters (`--load-cookies`, `--save-cookies`, `--auth-no-challenge`).
- **Silent Failure Detection** — Verifies downloaded files using two checks:
  1. **Gzip magic number** (`0x1f8b`) confirms the file is a valid compressed archive.
  2. **HTML content sniffing** scans the first 512 bytes for HTML tags (`<html>`, `<head>`, `<!doctype>`).
- **Dual-Format Retrieval** — First attempts the IGS long filename convention (post-2022), iterating over 14 common country codes (ATA, AUS, JPN, USA, etc.), then falls back to legacy short filenames.
- **Skip Existing Files** — Checks for already-downloaded valid files to avoid redundant transfers.

Country Code Iteration

When constructing RINEX 3 long filenames, the station's country code is required but not always known. The downloader iterates over a built-in list of common IGS station country codes:

```
ATA, AUS, JPN, USA, ZAF, CHL, FRA,
NZL, NOR, DEU, GBR, ARG, RUS, CHN
```

Listing 4: Built-in country codes

Note

For stations in countries not in this list, the downloader will fall back to the short filename format. This fallback ensures data retrieval succeeds for most IGS stations without requiring users to maintain station–country lookup tables.

API Reference

`download_rinex()`

```
def download_rinex(station, year, doy, output_dir):
    """Download single-station single-day RINEX observation data.

    Args:
        station (str): 4-character station name (e.g., 'mcm4')
        year (int): 4-digit year
        doy (int): Day of year (1--366)
        output_dir (str): Output root directory

    Returns:
        str: Status string
             'OK: <filename> (<size>KB)'
```

```
            'EXISTS: <filename>'
            'FAIL: <station> <year> <doy>'
    """
```

Listing 5: download_rinex function signature

The output directory structure is: `<output_dir>/<year>/<doy>/`

`download_products()`

```
def download_products(year, doy, output_dir):
    """Download IGS precise products (SP3, CLK, ERP) and
    broadcast ephemeris.

    Args:
        year (int): 4-digit year
        doy (int): Day of year
        output_dir (str): Product output root directory

    Returns:
        list: Status strings for each product
    """
```

Listing 6: download_products function signature

Downloaded products include:

- `SP3` — Precise orbit (15-min interval)
- `CLK` — Precise clock (30-sec interval)
- `ERP` — Earth rotation parameters
- `BRDC` — Broadcast navigation ephemeris

CLI Usage

```
# Download RINEX and products for 4 stations, 7 days
pygamit-bridge download \
    --stations mcm4,cas1,dav1,maw1 \
    --year 2025 --start-doy 1 --end-doy 7 \
    --output ./data/rinex \
    --products-output ./data/products

# Download only station data (no products)
pygamit-bridge download \
    --stations mcm4 \
    --year 2025 --start-doy 1 --end-doy 1 \
    --stations-only

# Download only products (no station data)
pygamit-bridge download \
    --stations mcm4 \
    --year 2025 --start-doy 1 --end-doy 1 \
    --products-only
```

Listing 7: Download command examples

Table 2: `download` subcommand arguments

Argument	Required	Default	Description
`--stations`	Yes	—	Comma-separated station list
`--year`	Yes	—	4-digit year
`--start-doy`	No	1	Start day of year
`--end-doy`	No	1	End day of year
`--output`	No	`./data/rinex`	RINEX output directory
`--products-output`	No	`./data/products`	Products output directory
`--stations-only`	No	False	Download only station data
`--products-only`	No	False	Download only products

Module 2: Format Bridge

The Format Bridge consists of three sub-modules that handle the cascading format transformations required between modern IGS data and GAMIT's internal expectations.

RINEX 3-to-2 Converter (`converter.py`)

Problem

GAMIT 10.71's `makexp` module cannot correctly parse the `SYS / # / OBS TYPES` header records introduced in RINEX 3, which declare observation types grouped by satellite system. This causes `makexp` to produce an empty batch file, breaking the entire processing chain.

Solution

The converter performs a complete format transformation:

1. **Header conversion**: Maps per-system observation type declarations (`SYS / # / OBS TYPES`) to the unified `# / TYPES OF OBSERV` format of RINEX 2.11.
2. **Observation type mapping**: Translates 52 three-character RINEX 3 observation codes to their RINEX 2 two-character equivalents, with de-duplication and order preservation.
3. **Data section conversion**: Restructures per-satellite single-line records (RINEX 3) to the traditional multi-line format with epoch header containing the satellite list (RINEX 2).

Observation Type Mapping Table

Table 3 shows the complete RINEX 3 to RINEX 2 observation type mapping implemented in the converter.

Table 3: RINEX 3 → 2 observation type mapping (GPS only)

RINEX 3	RINEX 2	RINEX 3	RINEX 2	RINEX 3	RINEX 2
C1C	C1	C1S	C1	C1L	C1
C1X	C1	C1P	P1	C1W	P1
L1C	L1	L1S	L1	L1L	L1
L1X	L1	L1P	L1	L1W	L1
D1C	D1	D1S	D1	D1L	D1

Continued on next page

RINEX 3	RINEX 2	RINEX 3	RINEX 2	RINEX 3	RINEX 2
D1X	D1				
S1C	S1	S1S	S1	S1L	S1
S1X	S1				
C2C	C2	C2S	C2	C2L	C2
C2X	C2	C2P	P2	C2W	P2
L2C	L2	L2S	L2	L2L	L2
L2X	L2	L2P	L2	L2W	L2
D2C	D2	D2S	D2	D2L	D2
D2X	D2				
S2C	S2	S2S	S2	S2L	S2
S2X	S2				
C5I	C5	C5Q	C5	C5X	C5
L5I	L5	L5Q	L5	L5X	L5
D5I	D5	D5Q	D5	D5X	D5
S5I	S5	S5Q	S5	S5X	S5

Note

The converter only retains GPS (G-system) observations. Multi-GNSS observations (GLONASS, Galileo, BeiDou) are discarded during conversion, as GAMIT's double-difference processing is typically configured for GPS-only or requires system-specific handling.

API Reference

```
def convert_rinex3_to_rinex2(input_file, output_file):
    """Complete RINEX 3.x to RINEX 2.11 format conversion.

    If the input file is already RINEX 2, it is copied directly.

    Args:
        input_file (str): Path to RINEX 3.x input file
        output_file (str): Path for RINEX 2.11 output file

    Returns:
        bool: True if conversion succeeded
    """
```

Listing 8: convert_rinex3_to_rinex2 function

CLI Usage

```
pygamit-bridge convert \
    --input MCM400ATA_R_20250010000_01D_30S_MO.rnx \
    --output mcm40010.25o
```

Listing 9: Convert command example

Table 4: `convert` subcommand arguments

Argument	Required	Alias	Description
`--input`	Yes	`-i`	RINEX 3 input file path
`--output`	Yes	`-o`	RINEX 2 output file path

Batch File Fallback (`batch_fallback.py`)

Problem

Even after RINEX 3-to-2 conversion, certain receiver type and firmware version combinations cause GAMIT's `makexp` to produce an empty `.makex.batch` file, preventing X-file generation.

Solution

The `batch_fallback` module directly parses RINEX headers to extract receiver metadata and generates a syntactically valid batch file. It includes a built-in receiver type mapping table:

Table 5: Receiver type to GAMIT abbreviation mapping

Receiver Manufacturer	GAMIT Abbreviation
SEPT (Septentrio)	SEP
JAVAD	JAV
TRIMBLE	TRM
LEICA	LEI
ASHTECH	ASH
TOPCON	TOP
NOVATEL	NOV
TPS	TPS
ROGUE	ROG
(Other)	*First 3 characters*

API Reference

```
def generate_makex_batch(expt, year, doy, orbt='igsg',
                         nav_file=None, xver='5',
                         rinex_dir='./'):
    """Generate GAMIT makex batch file content.

    Args:
        expt (str): Experiment name (4-char, e.g., 'anta')
        year (int): 4-digit year
        doy (str): Day of year (zero-padded, e.g., '001')
        orbt (str): Orbit type identifier (default 'igsg')
        nav_file (str): Navigation filename; auto-generated
                        if None
        xver (str): X-file version (default '5')
        rinex_dir (str): Directory containing RINEX files

    Returns:
        str: Batch file content, or None on failure
    """
```

```
def write_batch_file(expt, year, doy, output_dir='./',
                     **kwargs):
    """Generate and write makex batch file to disk.

    Args:
        expt (str): Experiment name
        year (int): 4-digit year
        doy (str): Day of year
        output_dir (str): Output directory
        **kwargs: Forwarded to generate_makex_batch()

    Returns:
        str: Path to batch file, or None on failure
    """
```

Listing 10: Batch fallback functions

Preprocessor (`preprocessor.py`)

The preprocessor module replaces traditional shell scripts with a Python-native pipeline for preparing data for GAMIT processing.

Processing Pipeline

For each station, the preprocessor performs:

1. **Decompress** Compact RINEX (`.crx.gz`) → `.crx` (gunzip) → `.rnx` (`CRX2RNX`)
2. **Convert** RINEX 3 → RINEX 2.11 (using the converter module)
3. **Rename** to GAMIT short filename format (`ssss{doy}0.{yr2}o`)

For products:

1. **Decompress** `.gz` files
2. **Create symlinks** from IGS long filenames to GAMIT short format:
   - `IGS0OPSFIN_*ORB.SP3` → `igs{week}{dow}.sp3`
   - `IGS0OPSFIN_*CLK.CLK` → `igs{week}{dow}.clk`

API Reference

```
def prepare_rinex(year, doy, data_dir, expt_dir,
                  stations=None):
    """Preprocess RINEX data: decompress, convert, rename.

    Args:
        year (int): 4-digit year
        doy (int): Day of year
        data_dir (str): Raw data root directory (with
                        year/doy subdirectories)
        expt_dir (str): GAMIT experiment directory
        stations (list): Station filter (None = auto-detect)

    Returns:
        int: Number of stations successfully processed
```

```
    """

def prepare_products(year, doy, data_dir, expt_dir):
    """Prepare IGS products with long-to-short filename
    mapping.

    Args:
        year (int): 4-digit year
        doy (int): Day of year
        data_dir (str): Product data root directory
        expt_dir (str): GAMIT experiment directory

    Returns:
        int: Number of product files processed
    """

def prepare_broadcast(year, doy, data_dir, expt_dir):
    """Prepare broadcast ephemeris file.

    Returns:
        bool: True if successful
    """

def link_tables(gg_dir, expt_dir, template_dir=None):
    """Symlink GAMIT global tables and local configuration
    to the experiment directory.

    Args:
        gg_dir (str): GAMIT installation root (e.g., ~/gg)
        expt_dir (str): GAMIT experiment directory
        template_dir (str): Optional local config template dir
    """
```

Listing 11: Preprocessor functions

CLI Usage

```
pygamit-bridge preprocess \
    --year 2025 --doy 1 \
    --data-dir ./data/rinex \
    --products-dir ./data/products \
    --expt-dir ./gamit/expt/2025001 \
    --gg-dir ~/gg \
    --stations mcm4,cas1,dav1,maw1
```

Listing 12: Preprocess command example

Module 3: Output Parser

The `parser` module (`parser.py`) provides standardized extraction of GAMIT processing results from o-files, q-files, and summary files into structured CSV and JSON formats.

Extracted Data Categories

ZTD (Zenith Total Delay)

Extracts `ATMZEN` records from GAMIT o-files. Handles both:

Table 6: `preprocess` subcommand arguments

Argument	Required	Default	Description
`--year`	Yes	—	4-digit year
`--doy`	Yes	—	Day of year
`--data-dir`	Yes	—	Raw data directory
`--products-dir`	No	None	Products directory
`--expt-dir`	Yes	—	GAMIT experiment directory
`--stations`	No	None	Station filter (comma-separated)
`--gg-dir`	No	None	GAMIT installation directory

- **Daily mean values** — Lines without epoch index. The final column contains the absolute ZTD in meters.
- **Piecewise-linear segments** — Lines with epoch index (1–13, for 2-hour intervals). The adjustment values are added to the daily mean to obtain absolute ZTD.

The parser correctly handles Fortran `D`-format exponents (e.g., `-0.1066D-01`).

```
# Daily mean ZTD (no epoch index)
13*CAS1 ATMZEN m 2.2655438832-0.1066D-01 ...
# Piecewise segment (epoch 1)
17*CAS1 ATMZEN m 1 0.0000000000-0.2437D-01 ...
```

Listing 13: Example GAMIT ATMZEN output lines

Station Coordinates

Extracts geocentric spherical coordinates (GEOC latitude, longitude, radius) from o-file lines matching patterns like:

```
1*CAS1 GEOC LAT dms S66:08:28.75542 0.1971D-02 ...
2*CAS1 GEOC LONG dms E110:31:10.94427-0.2388D-02 ...
3*CAS1 RADIUS km 6360.2587609893-0.2210D-02 ...
```

Listing 14: Example GAMIT coordinate output

Each coordinate component includes: final value, adjustment (meters), and formal error (meters).

Baselines

Extracts inter-station baseline lengths and formal errors from o-file baseline sections.

Quality Metrics

Extracted from summary files and o-files:

- Prefit and postfit **nrms** (normalized root-mean-square)
- Phase ambiguity counts: total, WL-fixed, NL-fixed
- Ambiguity **fixing rates**: WL% and NL%
- Number of double-difference **observations** and **parameters**

Note

If quality metrics are not found in summary files, the parser falls back to q-files for nrms extraction.

API Reference

```
def parse_ztd(session_dir, expt='anta'):
    """Extract ZTD estimates from GAMIT o-files.

    Returns:
        list: [{'station': str, 'epoch_idx': int,
                'ztd_m': float, 'ztd_mm': float,
                'adjustment_m': float,
                'sigma_m': float, 'sigma_mm': float}, ...]
    """

def parse_positions(session_dir, expt='anta'):
    """Extract station coordinate estimates.

    Returns:
        dict: {station: {'lat': str, 'lon': str,
                         'radius_km': float,
                         'lat_adj_m': float,
                         'lon_adj_m': float,
                         'radius_adj_m': float,
                         'lat_sigma_m': float, ...}}
    """

def parse_baselines(session_dir, expt='anta'):
    """Extract baseline lengths and formal errors.

    Returns:
        list: [{'from': str, 'to': str,
                'length_m': float, 'sigma_m': float}, ...]
    """

def parse_summary(session_dir, expt='anta'):
    """Extract quality metrics from summary/q-files.

    Returns:
        dict: {'nrms': float, 'postfit_nrms': float,
               'num_ambiguities': int,
               'wl_fixed': int, 'nl_fixed': int,
               'wl_rate': float, 'nl_rate': float,
               'num_observations': int,
               'num_parameters': int}
    """

def parse_session(session_dir, expt='anta'):
    """One-stop parsing of all GAMIT session outputs.

    Returns:
        dict: {'ztd': [...], 'positions': {...},
               'baselines': [...], 'summary': {...}}
    """

def export_csv(results, output_path):
```

```
    """Export ZTD data to CSV format."""

def export_json(results, output_path):
    """Export all parsed results to JSON format."""
```

Listing 15: Parser functions

CLI Usage

```
# Parse and export to JSON
pygamit-bridge parse \
    --session-dir ./gamit/expt/2025001 \
    --expt anta \
    --output results.json

# Parse and export to CSV (ZTD data only)
pygamit-bridge parse \
    --session-dir ./gamit/expt/2025001 \
    -o ztd_results.csv
```

Listing 16: Parse command examples

Table 7: `parse` subcommand arguments

Argument	Required	Default	Description
`--session-dir`	Yes	—	Session output directory
`--expt`	No	`anta`	Experiment name prefix
`--output`	No	None	Export path (.csv or .json)

Output Format Examples

JSON Output

```
{
  "ztd": [
    {
      "station": "CAS1",
      "epoch_idx": 0,
      "ztd_m": 2.25488495,
      "ztd_mm": 2254.9,
      "adjustment_m": -0.01066,
      "sigma_m": 0.005965,
      "sigma_mm": 6.0
    },
    ...
  ],
  "positions": {
    "CAS1": {
      "lat": "S66:08:28.75536",
      "lon": "E110:31:10.94408",
      "radius_km": 6360.25875878,
      "lat_adj_m": 0.00197,
      "lon_adj_m": -0.00239,
      "radius_adj_m": -0.00221,
```

```
      ...
    }
  },
  "baselines": [...],
  "summary": {
    "nrms": 0.41331,
    "postfit_nrms": 0.23542,
    "num_ambiguities": 89,
    "wl_fixed": 87,
    "nl_fixed": 76,
    "wl_rate": 97.8,
    "nl_rate": 85.4,
    "num_observations": 6110,
    "num_parameters": 268
  }
}
```

Listing 17: Example JSON output structure

CSV Output

The CSV export contains ZTD time series data with the following columns:

Table 8: CSV output columns

Column	Description
`station`	4-character station name
`epoch_idx`	Epoch index (0 = daily mean)
`ztd_mm`	Zenith total delay (mm)
`sigma_mm`	Formal error (mm)
`ztd_m`	Zenith total delay (m)
`sigma_m`	Formal error (m)
`adjustment_m`	Adjustment from a priori (m)

Utility Functions

The `utils.py` module provides GPS time system conversions and file validation functions used throughout the toolkit.

```
def doy_to_date(year, doy):
    """Convert day-of-year to datetime object."""

def date_to_doy(dt):
    """Convert datetime to (year, doy) tuple."""

def date_to_gps_week(year, month, day):
    """Calculate GPS week and day-of-week.
    Returns: (gps_week, dow) where dow 0=Sunday."""

def doy_to_gps_week(year, doy):
    """Calculate GPS week from year and DOY.
    Returns: (gps_week, dow)."""

def is_gzip(filepath):
    """Check if file is valid gzip (magic 0x1f8b)."""
```

```
def is_html(filepath):
    """Check if file content is HTML (CDDIS redirect)."""

def find_gamit_home():
    """Auto-detect GAMIT installation path.
    Checks: $GAMIT_HOME, ~/gg, /opt/gg, /usr/local/gg.
    Raises FileNotFoundError if not found."""

def station_name_short(long_name):
    """Extract 4-char short name from RINEX 3 long name.
    e.g., 'MCM400ATA_R_...' -> 'mcm4'."""
```

Listing 18: Utility function signatures

Complete Workflow

This section demonstrates a complete end-to-end processing workflow using PyGAMIT-Bridge with four Antarctic IGS stations.

Step-by-Step CLI Workflow

```
# === Step 1: Download data ===
pygamit-bridge download \
    --stations mcm4,cas1,dav1,maw1 \
    --year 2025 --start-doy 1 --end-doy 1 \
    --output ./data/rinex \
    --products-output ./data/products

# === Step 2: Preprocess for GAMIT ===
pygamit-bridge preprocess \
    --year 2025 --doy 1 \
    --data-dir ./data/rinex \
    --products-dir ./data/products \
    --expt-dir ./gamit/expt/2025001 \
    --gg-dir ~/gg

# === Step 3: Run GAMIT (standard command) ===
cd ./gamit/expt/2025001
sh_gamit -d 2025 1 -expt anta -noftp

# === Step 4: Parse results ===
pygamit-bridge parse \
    --session-dir ./gamit/expt/2025001 \
    --expt anta \
    -o results.json
```

Listing 19: Complete workflow using CLI

Step-by-Step Python API Workflow

```
from pygamit_bridge.downloader import (
    download_rinex, download_products
)
from pygamit_bridge.preprocessor import (
```

```
    prepare_rinex, prepare_products,
    prepare_broadcast, link_tables
)
from pygamit_bridge.batch_fallback import write_batch_file
from pygamit_bridge.parser import (
    parse_session, export_json
)

# --- Configuration ---
YEAR = 2025
DOY = 1
STATIONS = ['mcm4', 'cas1', 'dav1', 'maw1']
DATA_DIR = './data/rinex'
PROD_DIR = './data/products'
EXPT_DIR = './gamit/expt/2025001'

# --- Step 1: Download ---
for stn in STATIONS:
    result = download_rinex(stn, YEAR, DOY, DATA_DIR)
    print(f" {stn}: {result}")

prod_results = download_products(YEAR, DOY, PROD_DIR)
for r in prod_results:
    print(f" {r}")

# --- Step 2: Preprocess ---
n = prepare_rinex(YEAR, DOY, DATA_DIR, EXPT_DIR, STATIONS)
print(f"RINEX: {n} stations processed")

m = prepare_products(YEAR, DOY, PROD_DIR, EXPT_DIR)
print(f"Products: {m} files ready")

prepare_broadcast(YEAR, DOY, DATA_DIR, EXPT_DIR)
link_tables('~/gg', EXPT_DIR)

# Generate fallback batch file (insurance)
write_batch_file('anta', YEAR, f"{DOY:03d}",
                 output_dir=EXPT_DIR,
                 rinex_dir=EXPT_DIR)

# --- Step 3: Run GAMIT (externally) ---
# sh_gamit -d 2025 1 -expt anta -noftp

# --- Step 4: Parse results ---
results = parse_session(EXPT_DIR, expt='anta')
print(f"ZTD records: {len(results['ztd'])}")
print(f"Stations: {len(results['positions'])}")
print(f"nrms: {results['summary']['nrms']}")
export_json(results, 'results.json')
```

Listing 20: Complete workflow using Python API

Example Script

A ready-to-use example script is provided at `examples/process_day.py`. Usage:

```
python3 examples/process_day.py \
    --year 2025 --doy 1 \
```

```
    --stations mcm4,cas1,dav1,maw1 \
    --data-dir ./data/rinex \
    --products-dir ./data/products \
    --expt-dir ./gamit/expt/2025001 \
    --expt anta
```

Listing 21: Running the example script

Troubleshooting

Download Issues

`FAIL: mcm4 2025 001`
The downloader could not retrieve data for this station. Common causes:

- Earthdata credentials not configured (check `~/.netrc`)
- `~/.urs_cookies` file does not exist
- Station is temporarily unavailable on CDDIS
- Network connectivity issues

Downloaded file is an HTML page
The toolkit's built-in HTML detection should reject these files automatically. If this persists, verify your Earthdata credentials at `https://urs.earthdata.nasa.gov`.

Tip

Test your Earthdata setup manually:

```
wget --no-check-certificate \
    --load-cookies ~/.urs_cookies \
    --save-cookies ~/.urs_cookies \
    --auth-no-challenge \
    --keep-session-cookies \
    -O test.gz \
    https://cddis.nasa.gov/archive/gnss/data/daily/2025/001/25d/
        MCM400ATA_R_20250010000_01D_30S_MO.crx.gz
file test.gz # Should report "gzip compressed data"
```

RINEX Conversion Issues

`convert_rinex3_to_rinex2()` returns `False`
The file may not contain GPS observations. The converter only maps GPS observation types; if the file contains only GLONASS or Galileo data, the conversion will fail.

`CRX2RNX` not found
The preprocessor searches for `CRX2RNX` in the following locations:

1. System PATH (`CRX2RNX` or `crx2rnx`)
2. `~/gg/bin/crx2rnx`
3. `/usr/local/bin/crx2rnx`

Ensure the utility is installed in one of these locations.

makexp Batch File Issues

Empty batch file after conversion

Use the batch fallback generator as a safety net. After preprocessing, check whether the `.makex.batch` file was generated:

```
cat ./gamit/expt/2025001/anta.makex.batch
```

If the file is empty or missing, the `preprocess` command will automatically generate a fallback batch file.

Parsing Issues

No ZTD records extracted

Check that:

- The `--expt` prefix matches your GAMIT experiment name (default: `anta`)
- O-files exist in the session directory (files named `oanta*.*`)
- The GAMIT run completed successfully

nrms is `None`

The parser first searches summary files, then falls back to q-files. If the GAMIT run terminated abnormally, these files may not have been generated.

CLI Reference Summary

```
pygamit-bridge --help
pygamit-bridge --version
```

Listing 22: Global CLI help

Table 9: Available subcommands

Subcommand	Description
`download`	Download GNSS data and products from CDDIS
`convert`	Convert RINEX 3 to RINEX 2.11
`preprocess`	Preprocess data for GAMIT processing
`parse`	Parse GAMIT output files

For detailed help on any subcommand:

```
pygamit-bridge <subcommand> --help
```

License

PyGAMIT-Bridge is released under the MIT License. See the `LICENSE` file in the repository for details.

Repository: `https://github.com/geumjin99/pygamit-bridge`

Bug Reports: `https://github.com/geumjin99/pygamit-bridge/issues`