Supplementary Information

A Deep Learning Pipeline for Large Earthquake Analysis using High-Rate Global Navigation Satellite System Data

Claudia Quinteros-Cartaya[1], Javier Quintero-Arenas[1,2], Andrea Padilla-Lafarga[3], Carlos Moraila[3], Johannes Faber[1,4], Wei Li[1], Jonas Köhler[1,5], Nishtha Srivastava[1,5*]

[1]Frankfurt Institute for Advanced Studies, Ruth-Moufang-Straße 1, Frankfurt am Main, 60438, Hessen, Germany.
[2]Institute of Computer Sciences, Goethe University, Robert-Mayer-Str. 11-15, Frankfurt am Main, 60325, Hessen, Germany.
[3]Faculty of the Earth and Space Sciences, Autonomous University of Sinaloa, Universitarios Ote. S/N, Cd Universitaria, Culiacan Rosales, 80040, State, Mexico.
[4]Institute for Theoretical Physics, Goethe University, Max-von-Laue-Str. 1, Frankfurt am Main, 60438, Hessen, Germany.
[5]Institute of Geosciences, Goethe University, Altenhöferallee 1, Frankfurt am Main, 60438, Hessen, Germany.

*Corresponding author(s). E-mail(s): N.Srivastava@em.uni-frankfurt.de;
Contributing authors: quinteros@fias.uni-frankfurt.de; jaqarenas@fias.uni-frankfurt.de; andreapadilla@uas.edu.mx; cmoraila@uas.edu.mx; faber@fias.uni-frankfurt.de; wli@fias.uni-frankfurt.de; jkoehler@fias.uni-frankfurt.de;

Contents of this file: Tutorial

1. Integrated Workflow for Earthquake Detection and Magnitude Estimation with DetEQ and MagEs.
2. Thresholds for Earthquake Detection.
3. DetEQ and MagEs Models for independent analysis.

Additional Supporting Information

1. This pipeline has been integrated into SAIPy package as a gnss module, available at https://github.com/srivastavaresearchgroup/SAIPy
2. The GNSS data used in this supplementary example was retrieved from the TUSAGA-Aktif (https://www.tusaga-aktif.gov.tr/, accessed on February 2023).

Turorial

1 Integrated Workflow for Earthquake Detection and Magnitude Estimation with DetEQ and MagEs

This section provides a step-by-step guide on how to use DetEQ for earthquake detection and MagEs for magnitude estimation in a Jupyter Notebook. The workflow covers data loading, preprocessing, detection, and magnitude estimation.

1.1 Load Required Libraries

Import the Python libraries and modules/packages needed to be used in the Notebook (Figure 1). Ensure that the actual path where your GNSS module is stored is correctly set.

```
[ ]: # Libraries
     import os
     from datetime import datetime

     import gnss.gnss_packagetools as gnss_pck
     import gnss.gnss_load_data as ld
     import gnss.det_eq.ae_utils as d_utils
     from gnss.gnss_utils import *
```

```
[ ]: os.environ['TF_CPP_MIN_LOG_LEVEL'] = '1'

     %reload_ext autoreload
     %autoreload 2
     %matplotlib widget
```

Fig. 1 Libraries, modules, and configurations

Optionally, the behavior of the environment and the Jupyter Notebook can be configured just after the libraries are loaded. In this example, automatic module reloading

is activated, and automatically reloads all imported modules, so changes are reflected without needing to restart the kernel. Also, interactive plots are enabled using the widget backend.

1.2 Set Path and file names

Before starting data loading and processing, verify that you are using the correct file paths and datasets (Figure 2). In this step, you check the current working location and define the paths to the models, input data files, and the output directory where the results will be saved. If the current location is not in your work directory, make sure that you change it to a new right path.

```
[ ]: # Get the current working directory
current_path = os.getcwd()

print("Current working directory:", current_path)
```

```
[ ]: # Optional

# In case that it is necessary to set a new working path
new_path = '/change/to/your/actual/working/path/'

# Change the working directory
os.chdir(new_path)

# Verify the change
print("New working directory:", os.getcwd())
```

```
[ ]: # Model's Path
deteq = './gnss/det_eq/saved_model/DetEq.pt' #('/change/to/Deteq_model/file/
  ↪path/')
dir_mages = './gnss/mages/saved_model/' #('/change/to/MagEs_model/folder/path/')

# Data
dir_data = './data_test/' #('/change/to/your/data/path/')

# Output directory
subnm = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
# folder name with the current time to be unique
parent_outdir = './output_results/Turkey2023_'+subnm+'/' #('/change/to/your/
  ↪ouput/folder/path/')

# Create output folder
os.mkdir(parent_outdir)
```

```
[ ]: #Custumize your input data info

#Date of the data - for output file names
date = '2023-02-06'

# Station names
stat_names = ["adn2", "akle", "elaz", "feek", "mrsi", "siv1"]
file_paths = [f"{dir_data}Turkey2023/enu_2023037_{station}" for station in␣
  ↪stat_names]
```

Fig. 2 Working directories and files.

Double-check the paths before running any commands that modify or remove directories that you may need to preserve. Hence, in this example, the output directory

name is set using the current time to avoid overwriting any previous output from earlier experiments.

1.3 Load and Preprocess Data

In this step, the GNSS data are loaded and preprocessed using functions from the `gnss.gnss_load_data` and `gnss.det_eq.ae_utils` modules. The `load_data_sf` function handles the mseed and numpy formats, while `load_data_txt` is used for file formats such as track, pos, and other standard text files.

You could also customize the data loading process to fit your specific format, ensuring that the structure consists of four columns: the first column represents time in seconds (within a day), followed by displacement values in meters for the East, North, and Vertical components.

Here in the example (Figure 3), 'text' specifically refers to a standard format containing four columns: time (in seconds) and displacements in three components that follow the order East, North, and Vertical. For track files, the `load_data_txt` function automatically reorders the original columns: North, East, Vertical, to the order: East, North, Vertical.

In the `load_data_txt` function, the `max_time=86400` parameter corresponds to the total of seconds within 1 day that will be used to fix the length of the input array. In cases where data are missing, either due to recording gaps or because the start time occurs after the beginning of the day and/or the end time is earlier than the full 24-hour period, the array is filled with NaN values during the loading process to maintain the time series structure and facilitate data management. Alternatively, `max_time` can be set to a value smaller than 86400 if a shorter duration is preferred.

Since the model operates with meter units, unit conversion may be required. For this example `load_data_txt`, use `conv_metric=100` to convert from centimeters to meters. If the original data are in counts and need to be converted to meters, apply `conv_metric=100 x gain`. If no conversion is necessary, set `conv_metric=False`.

Visualizing the plots and/or saving the figures in this step is optional.

The data is stored in dictionaries: `raw_data` contains the original time series for each station, while `detrended_data` holds the processed signals after detrending. The detrending is performed using a sliding window to remove long-period trends.

For practical purposes, you can extract a specific section of the data by setting the start and end times in seconds within a 24-hour period (Figure 3). If you prefer to process the entire loaded data, simply set the start time to 0 and the end time to the length of the loaded data.

If you want to visualize the final prepared data before processing, you can use the `plot_data` function from the `gnss_load_data` module. The time axes formatting can be set to 'HH:MM:SS' or 'seconds' (Figure 4).

1.4 Run the GNSS Monitoring Pipeline (DetEQ - MagEs)

Just before calling the integrated workflow monitoring function, it is important to configure the input settings that define the key processing parameters (Figure 5).

The data of each station are stored in the `network_data` variable.

```
[ ]: # If you want to define the maximum time or an sub-interval of interest (for testing)
print('to_eqk1 =', seconds_passed_in_day('01:17:37')) # eqk1 origin time 4657
print('to_eqk2 =', seconds_passed_in_day('10:24:50')) # eqk2 origin time 37490
```

```
[ ]: # Dictionaries to store data
raw_data = {}
detrended_data = {}

# Load and pre-process data
for station, file_path in zip(stat_names, file_paths):
    print(f"Loading {station} station")
    raw_data[station] = ld.load_data_txt(file_path, formatf='text',
                                max_time=43200, # from 0 to 24 * 3600 seconds in 1 day
                                plot=False, save_plot=False,
                                outpath=parent_outdir,
                                stat_name=station,
                                conv_metric=100 # cm to m
                            )

    # Apply detrending
    print(f"Detrending ...")
    detrended_data[station] = d_utils.get_complete_detrend(
        raw_data[station], window_size=6, step_size=1, max_context_size=60
    )
# detrended_data["adn2"].shape
# (86400, 3)
```

```
[ ]: # start, end in seconds in 1 day (0 is 00:00:00 and 86400 seconds is 23:59:00 )
#start, end = 3000, 6000 # for example, specific section near the earthquake.
start, end = 0, len(detrended_data[stat_names[0]]) # 43200 Original data until 12:00:00

# Create a subset of the detrended data
detrended_subset = {station: data[start:end] for station, data in detrended_data.items()}

# Now you can access individual processed data with detrended_subset["adn2"], etc.
```

Fig. 3 Loading data files and extracting a section of the total data time window.

```
[ ]: # Plot time series
for j in range(len(stat_names)):
    # start and end in seconds (0 is 00:00:00 and 86400 seconds is 23:59:00)
    fig = ld.plot_data(detrended_data[stat_names[j]], start, end,
                       stat_names[j], date,
                       save=True, save_path=parent_outdir+'Data_plots/',
                       format_axes='HH:MM:SS', label = ['E','N','U'],
                       ylim=[-0.45,0.45])
```

Fig. 4 Visualization of data.

DetEQ is performed using predefined threshold values for each station, `thresh` variable, which is a parameter used to identify seismic events in the time series. These thresholds can be customized for specific stations, but their estimation requires additional processing time. Therefore, we suggest determining them in advance, as explained in the next section (Section 2). For testing purposes, in this example, trial values are used for all stations.

```
[ ]: # Data and Parameters
network_data = [detrended_subset[stat] for stat in stat_names]
thresh = [[0.00, 0.034]] * len(network_data) # trial thresh values for detection
network_thresh = 5 # minimum number of stations for detection
latency_win = 60 # latency between stations for detection in seconds
gap = 360 # Minimum gap between detections (in seconds).
n_det = 5 # N consecutive triggers to confirm the detection in a station
eval_time  = 60 # total time evaluation for magnitude

# Create output files for deteq
out_file = parent_outdir+'deteq_output_times'
open(out_file+'.txt', 'w').close()
open(out_file+'_every_stat.txt', 'w').close()

# Create a Readme file with key process parameters.
readme_content = f"""# Data Processing Pipeline  DetEQ - MagEs

## Overview
This README provides key process parameters for the data processing pipeline.

## Process Parameters
- **Date of the data**: {date}
- **Stations names**: {stat_names}
- **Processing time window (seconds)**: {start, end}
- **Minimum number of stations for detection**: {network_thresh}
- **Thresh values for detection**: {thresh}
- **Latency between stations for detection (seconds)**: {latency_win}
- **Minimum gap between detections (in seconds)**: {gap}
- **N consecutive triggers to confirm the detection in a station**: {n_det}
- **Total time evaluation for magnitude estimation (seconds)**: {eval_time}

"""

# Save to README.md
with open(parent_outdir+'README.md', 'w') as file:
    file.write(readme_content)
```

Fig. 5 Input Settings and Readme file creation.

DetEQ also needs a specific threshold number of stations, `network_thresh`, which is the minimum number of triggered stations to confirm detection. If you have a dense network, you can define the minimum number of stations required to detect the event based on as many possible stations near the seismogenic area. Otherwise, consider setting the `network_thresh` to 3 or 4 stations.

The `latency_win` parameter defines the maximum allowable time difference (in seconds) between detection times at different stations to be associated with the same earthquake. And the `n_det` parameter defines how many consecutive triggers you consider for detection (maximum 6 seconds).

Additionally, for the magnitude estimation by time update, the maximum evaluation window, `eval_time`, is 65 seconds by default, but you can set it up to 390 seconds.

A Readme file is generated for future reference and for reproducibility of the experiment.

Since the input parameters are defined, the module for the integrated process DetEQ-MagEs is performed using the `gnss.gnss_packagetools` module, through the `GNSS_monitor` function (Figure 6).

```
[ ]: # Call GNSS monitoring Pipeline
results = gnss_pck.GNSS_monitor(parent_outdir, out_file, deteq, dir_mages,
                                network_data, stat_names, date, start, thresh,
                                network_thresh=network_thresh,
                                latency_win=latency_win,
                                gap=gap, n_det=n_det,
                                eval_time=eval_time,
                                save_output_plots=True)

# Results stored in a dictionary
#results["output"] --> label_dict, trg_intervals, complete_anom_score,␣
 ↪first_stat_data, detect_df
#results["t_stations"] --> time detections
#results["mag_t_stations"] --> magnitude and times
```

Fig. 6 Function for integrated monitoring DetEQ-MagEs.

The `GNSS_monitor` function first loads the DetEQ autoencoder model, which runs in a separate thread using the `eval_network` function, and concurrently, the `deteq_mages_monitor` function allows the MagEs model to proceed for magnitude estimation when an earthquake has been detected. MagEs is activated through the `gnss.mages.gnss_make_predictions` module, and thus, `mag_predict_secbysec` function processes the data over the defined evaluation maximum window (`eval_time`), refining the magnitude estimate second by second. The results are stored in an output file for further analysis.

After detection, the processing thread is synchronized to ensure that all detections are completed before proceeding. Optionally, detection and magnitude plots can be generated (`save_output_plots=True`).

1.5 Outputs

The results are stored in an output file for visualization and further analysis. The output consists of two text files generated by DetEQ: one containing the commonly evaluated interval of positive detections and another listing detection times per station. From MagEs, results are stored in subfolders for each station, containing two text files: one detailing all estimated magnitudes per second after detection time, and another summarizing representative magnitude results. In the end, a Summary text (Figure 7) is saved with the final information.

Additionally, various plots can be saved by station, including data visualization, detection intervals (Figure 8), and magnitude curves over time (Figure 9.

```
                    ** SUMMARY **

Event 1: 2023-02-06
  Station ADN2 (Time: 01:18:39): 8.4, 8.3
  Station AKLE (Time: 01:18:41): 7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5
  Station ELAZ (Time: 01:18:37): 6.0, 7.9, 8.1, 8.0
  Station FEEK (Time: 01:18:12): 7.3, 7.8, 7.8
  Station MRSI (Time: 01:19:11): 7.9, 8.0, 8.0
  Station SIV1 (Time: 01:18:33): 8.2, 8.1, 8.1, 8.1, 8.1

** Median Magnitude +/- absolute deviation: 7.8 +/- 0.3 **
--------------------------------------------------------------------------------

Event 2: 2023-02-06
  Station ADN2 (Time: 10:25:34): 7.3, 7.9, 7.8, 7.8
  Station AKLE (Time: 10:25:59): 7.6, 7.6, 7.6, 7.7, 7.7
  Station ELAZ (Time: 10:25:36): 7.9, 8.0, 7.8, 7.9, 7.9
  Station FEEK (Time: 10:25:10): 8.5, 8.4, 8.5
  Station SIV1 (Time: 10:25:39): 7.6, 7.7, 7.8

** Median Magnitude +/- absolute deviation: 7.7 +/- 0.2 **
--------------------------------------------------------------------------------
```

Fig. 7 Summary file content

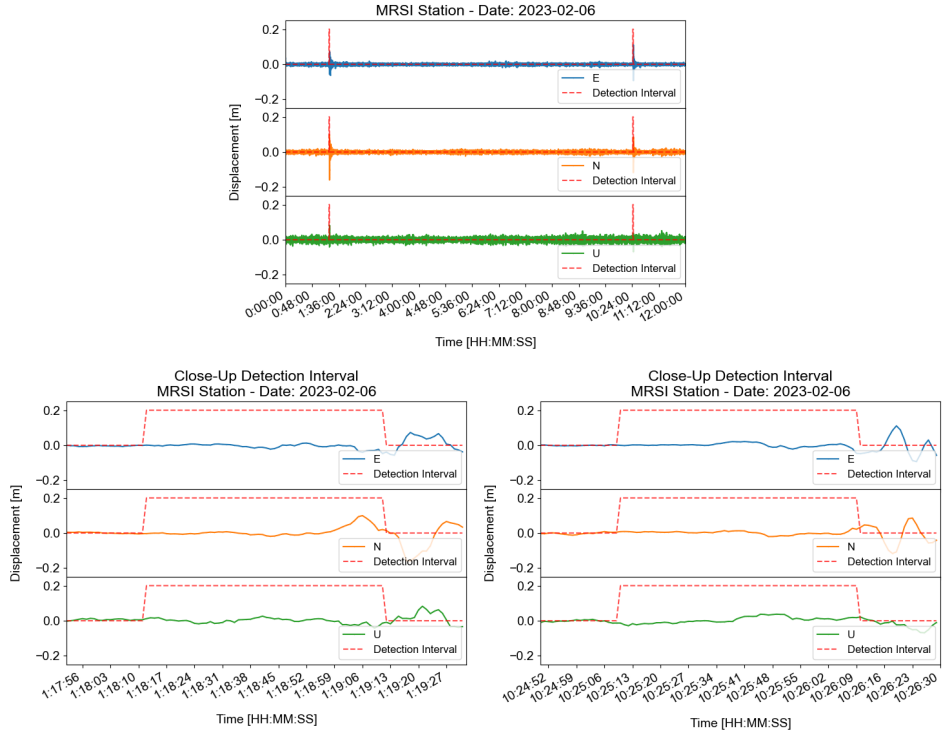

Fig. 8 Detection plots Station MRSI. Example of the detection of the Turkey Earthquakes, Mw 7.8 and 7.6, on 6th February, 2023. In the top of the figure is the complete stream with the detection intervals. In the bottom, is the close-up of every detection interval.

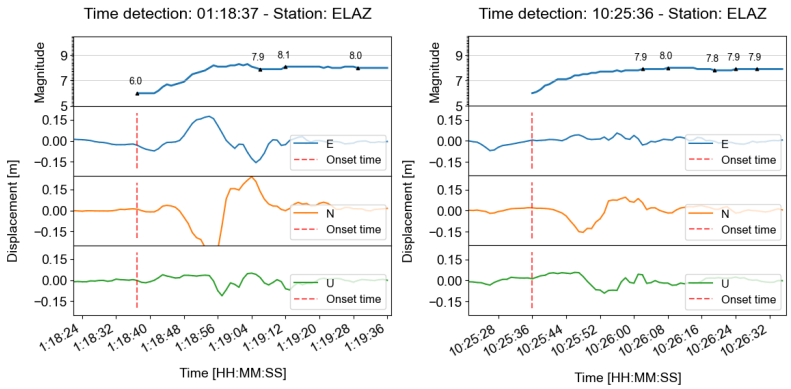

Fig. 9 Magnitude estimation plots of the Station ELAZ. Example of the Turkey Earthquakes, Mw 7.8 and Mw 7.6, on 6th February, 2023.

2 Thresholds for Earthquake Detection

The maximum `thresh` value define the sensitivity of the DetEQ model and should be calibrated using a noise-only time window. Since this process takes additional computation time, we recommend performing it in advance, as described in this section.

Ensure that the necessary libraries and modules are loaded before running the threshold estimation (Figure 10).

```
[ ]: # Libraries
import os
import numpy as np

import torch
import torch.optim as optim
from torch.optim import RAdam

import gnss.gnss_load_data as ld
from gnss.det_eq.ae_models import Autoencoder
import gnss.det_eq.ae_utils as d_utils
```

```
[ ]: %reload_ext autoreload
%autoreload 2
```

```
[ ]: device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

Fig. 10 Libraries and modules for the Jupyter Notebook to estimate the thresh values.

For this process, the data are loaded and preprocessed following the same procedure as described above, in the Load and Preprocess Data (Section 1.3). From the loaded data, a 60 minute noise-only interval is selected, ensuring that the chosen time window contains no earthquake signals or significant disturbances. Note that in this example (Figure 11), we use different intervals for two noisy stations (CYL2 and ERGN) compared to the rest, as it is necessary to select an interval with minimal noise.

High noise levels lead to higher threshold values, requiring a trade-off: a threshold that is too high may cause true seismic events to be missed, as weaker signals could fall below the detection threshold. On the other hand, setting the threshold too low may cause false detections, even when using minimal noise intervals. However, false detections are further minimized by requiring detections from multiple stations within the network.

For a more robust threshold determination, we recommend using multiple samples per station to perform a statistical estimation, such as calculating the mean, to obtain a representative threshold value, even for all stations.

The DetEQ model is first initialized before executing `get_thresh` function from `gnss.det_eq.ae_utils` module. Only for the threshold estimation process, it is necessary to replace NaN values with zeros.

```
[ ]: input_size= 18 # 6 seconds and 3 channels (6x3)

model = Autoencoder(input_size)

criterion = d_utils.CombinedLoss()
optimizer = optim.RAdam(model.parameters(), lr=0.001)
model.load_state_dict(torch.load(deteq))


thresh_search_interval_1=[400,4000] # 3600 len (60 minutes), only noise. to_eqk␣
 ↪= 4657 & 37490
thresh_search_interval_2=[8100,11700]
thresh_search_interval_3=[21000,24600]
stats_data = [detrended_data[stat] for stat in stat_names]

thresh = []

for i in range(len(stat_names)):
    if stat_names[i]== "cyl2":
        lw_lim, hg_lim = thresh_search_interval_2[0],␣
 ↪thresh_search_interval_2[1]
    elif stat_names[i]== "ergn":
        lw_lim, hg_lim = thresh_search_interval_3[0],␣
 ↪thresh_search_interval_3[1]
    else:
        lw_lim, hg_lim = thresh_search_interval_1[0],␣
 ↪thresh_search_interval_1[1]

    data = stats_data[i][lw_lim:hg_lim]
    data = np.nan_to_num(data)
    thresh_st = d_utils.get_thresh(model, criterion, data)
    thresh.append(thresh_st)

print('Thresh list: ', thresh)
```

Fig. 11 Function for the thresh values estimation process.

The output is a list of the minimum and maximum threshold values for each station.(Figure 12).

```
min max thresh:  [0.00018904644246707557, 0.02541411970020098]
min max thresh:  [0.000578285198959921, 0.033713450901059455]
min max thresh:  [-0.0005639280846373004, 0.01847214869990911]
min max thresh:  [0.00014653823917113958, 0.022130500814350095]
min max thresh:  [0.0006982521737313888, 0.033416414102863376]
min max thresh:  [0.00041600275107972636, 0.02836394144352328]
min max thresh:  [0.00020700202186880404, 0.025221705825134774]
min max thresh:  [1.904171346229827e-05, 0.02344051226280826]
```

Fig. 12 Example of output list of thresh values.

3 DetEQ and MagEs Models for independent analysis

DetEQ can be used independently via the `eval_network` function from the `gnss.det_eq.ae_utils` module.

MagEs, on the other hand, can be used independently through two functions: `mag_prediction` and `mag_predict_secbysec`, both from the `gnss.mages.gnss_make_predictions` module. The `mag_prediction` function is designed for magnitude estimation using data from 1 to 3 stations, assuming the earthquake location is already known. In this case, the processing time window is fixed on the basis of the epicentral distance. If the detection time is not available, the input variable is set to None and the time is calculated theoretically. The `mag_predict_secbysec` function is used for magnitude estimation on a second-by-second basis once the detection time is already known.

For more details, examples in the Jupyter Notebook are available in the SAIPy repository.