team-astraios-documentation

October 5, 2024

1 Documentation for Seismic Event Detector Code

1.1 Overview

This program is a graphical user interface (GUI) application that analyzes seismic data from `.mseed` files using the ObsPy library. It identifies significant seismic events based on velocity peaks and the STA/LTA (Short-Term Average/Long-Term Average) triggering method. The results are plotted and can be exported to a CSV file.

1.2 Code Explanation

1.2.1 Importing Required Libraries

```
import tkinter as tk
from tkinter import filedialog
from tkinter import messagebox
import numpy as np
from obspy import read
from obspy.signal.trigger import classic_sta_lta
import matplotlib.pyplot as plt
from scipy.signal import find_peaks
from datetime import timedelta
import pandas as pd
```

- **tkinter**: For creating the GUI.
- **numpy**: For numerical operations.
- **obspy**: For reading and processing seismic data.
- **matplotlib**: For plotting data.
- **scipy**: For finding peaks in data.
- **datetime**: For handling time-related operations.
- **pandas**: For managing and exporting data in tabular form.

1.2.2 File Selection Function

```
def select_file():
    filepath = filedialog.askopenfilename(title="Select Test File", filetypes=[("mseed files",
    entry.delete(0, tk.END)
    entry.insert(0, filepath)
    popup_root.lift()
    popup_root.focus_force()
```

- **Purpose**: This function opens a file dialog for the user to select a `.mseed` file. It then displays the selected file path in an entry field.

1.2.3 Running the Analysis

```
def run_analysis():
    filepath = entry.get()
    if not filepath:
        messagebox.showerror("Error", "Please select a file")
        return
    try:
        analysis(filepath)
    except Exception as e:
        messagebox.showerror("Error", str(e))
    finally:
        popup_root.destroy()
```

- **Purpose**: This function retrieves the selected file path from the entry field. If no file is selected, it shows an error message. It calls the `analysis()` function to process the file and handles any exceptions that may occur.

1.2.4 Main Analysis Function

```
def analysis(filepath):
    st = read(filepath)
```

- **Purpose**: Reads the seismic data from the selected file into an ObsPy stream object.

Filtering the Data

```
    # Apply the bandpass filter
    minfreq = 0.5
    maxfreq = 1.0
    st_filt = st.copy()
    st_filt.filter('bandpass', freqmin=minfreq, freqmax=maxfreq)
    tr_filt = st_filt.traces[0].copy()
    tr_times_filt = tr_filt.times()
    tr_data_filt = tr_filt.data
```

- **Purpose**: Applies a bandpass filter to the data, allowing frequencies between 0.5 Hz and 1.0 Hz. It extracts the filtered trace data and its corresponding time values.

Identifying the Highest Velocity Peak

```
    max_vel_index = np.argmax(csv_data)
    max_vel_time = csv_times[max_vel_index]
    print(f"Highest peak in velocity found at time {max_vel_time} s with value {csv_data[max_ve
```

- **Purpose**: Finds the index of the maximum velocity peak and its corresponding time in the filtered data. It prints the time and value of the peak.

STA/LTA Detection Algorithm

```
df = tr_filt.stats.sampling_rate
sta_len = 120
lta_len = 600
cft = classic_sta_lta(tr_data_filt, int(sta_len * df), int(lta_len * df))
```

- **Purpose**: Calculates the STA/LTA characteristic function using specified short-term and long-term window lengths (120s and 600s).

Finding Peaks in the STA/LTA Function

```
cft_peaks, _ = find_peaks(cft)
cft_peak_times = tr_times_filt[cft_peaks]
cft_peak_values = cft[cft_peaks]
```

- **Purpose**: Identifies the peaks in the STA/LTA function, along with their times and values.

Validating Peaks Around the Velocity Peak

```
window_size = 1000
valid_peaks_indices = np.where(np.abs(cft_peak_times - max_vel_time) <= window_size)[0]
```

- **Purpose**: Filters the STA/LTA peaks to find those that occur within a specified time window around the identified velocity peak.

Selecting the Best Trigger

```
if len(valid_peaks_indices) == 0:
    print("No valid STA/LTA peak found within the specified range.")
    messagebox.showerror("Error", "No valid STA/LTA peak found within the specified range.
    return

highest_peak_index_in_window = valid_peaks_indices[np.argmax(cft_peak_values[valid_peaks_i
closest_trigger_time = cft_peak_times[highest_peak_index_in_window]
print(f"Closest highest STA/LTA trigger found at time {closest_trigger_time} s")
```

- **Purpose**: Checks for valid peaks in the specified window. If found, it selects the highest STA/LTA peak and prints its time.

Plotting Results

```
plt.figure(figsize=(12, 6))
plt.plot(csv_times, csv_data)
plt.axvline(x=closest_trigger_time, color='green', label='Best Trigger On')
plt.xlim([min(csv_times), max(csv_times)])
plt.ylabel('Velocity (m/s)')
plt.xlabel('Time (s)')
plt.title(f'{filepath}', fontweight='bold')
plt.legend()
plt.tight_layout()
plt.show()
```

- **Purpose**: Creates a plot of the seismic velocity data, marking the identified best trigger time with a vertical line.

Exporting Results to CSV

```
fname = filepath
starttime = tr_filt.stats.starttime.datetime
best_trigger_time_abs = starttime + timedelta(seconds=closest_trigger_time)
best_trigger_time_str = best_trigger_time_abs.strftime('%Y-%m-%dT%H:%M:%S.%f')

detect_df = pd.DataFrame(data={
    'filename': [fname],
    'time_abs(%Y-%m-%dT%H:%M:%S.%f)': [best_trigger_time_str],
    'time_rel(sec)': [closest_trigger_time]
})

output_path = filedialog.asksaveasfilename(defaultextension=".csv", filetypes=[("CSV files'
if output_path:
    detect_df.to_csv(output_path, index=False)
    print(f"Catalog exported to {output_path}")
```

- **Purpose**: Compiles the results (file name, absolute time of the best trigger, and relative time) into a DataFrame and prompts the user to save it as a CSV file.

1.2.5 GUI Setup

```
root = tk.Tk()
root.title("Seismic Event Detector by Schimmel")
root.state("zoomed")
```

- **Purpose**: Initializes the main GUI window, sets its title, and sizes it.

Adding Labels and Buttons

```
heading = tk.Label(root, text="Welcome to Seismic Event Detector", font=("Arial", 40, "bold"))
heading.place(relx=0.5, rely=0.2, anchor=tk.CENTER)

subheading = tk.Label(root, text="by Team Astraios", font=("Arial", 20, "bold"))
subheading.place(relx=0.5, rely=0.3, anchor=tk.CENTER)

button = tk.Button(root, text="Analyze Data", font=("Arial", 20), command=lambda: popup())
button.place(relx=0.5, rely=0.5, anchor=tk.CENTER)
```

- **Purpose**: Creates and positions labels and buttons in the GUI. The button initiates the file selection popup when clicked.

Popup for File Selection

```
def popup():
    global popup_root
    popup_root = tk.Toplevel(root)
```

```
popup_root.title("Select Test File")
popup_root.resizable(False, False)

label = tk.Label(popup_root, text="Please select a test file (.mseed file)")
label.pack()

global entry
entry = tk.Entry(popup_root, width=50)
entry.pack()

browse_button = tk.Button(popup_root, text="Browse", command=select_file)
browse_button.pack()

run_button = tk.Button(popup_root, text="Run", command=run_analysis)
run_button.pack()
```

- **Purpose**: Creates a popup window for selecting the `.mseed` file. It includes a label, an entry field for displaying the selected file path, and buttons for browsing and running the analysis.

1.2.6 Running the GUI

```
root.mainloop()
```

- **Purpose**: Starts the GUI event loop, allowing the application to respond to user interactions.

1.3 Conclusion

This code implements a comprehensive tool for detecting seismic events based on velocity and STA/LTA analysis. It provides a user-friendly interface for selecting data files, performing analysis, visualizing results, and exporting findings to a CSV format.

1.4 Authors and Sources

This tool was developed by Schimmel Hafeez of Team Astraios for NASA's Space Apps Challenge 2024, Pakistan, based on provided demos and resources from NASA.