Early Earthquake and Tsunami Warning Viewer

AQA A-Level Computing NEA Report

Yicheng Shao (Eason)

Candidate Number: [1234]

October 16, 2024

Abstract

Give a brief summary outline of your project.

Contents

List of Tables

List of Figures

List of Listings

Chapter 1

Analysis

Overview

This section introduces the background for the EEW system in place in Japan, consisting of EEW Forecasts and EEW Warnings, and together with the tsunami warnings. Concepts such as intensities, magnitude and epicentres are defined. Two key users and targets are identified, specifically passionate geologists and earthquake-sensitive industries, each having different requirements. A client in the former was interviewed, together with a thorough analysis and comparison of some existing solutions such as SREV, JQuake, KEVI and Quarog. A detailed requirement specification and the critical path is also outlined, consisting of DM-D.S.S. parsing, real-time monitoring, past-earthquake viewing and joint functionalities.

1.1 Background Information

1.1.1 The Early Earthquake Warning System

Earthquake is one of the most common natural disasters in the whole world, and direct consequences of earthquakes include tsunamis which could be catastrophic.

Japan, sitting on the intersection of the Eurasian, the Philippine and the North-American plates, is the countries with most earthquakes. Historically, the Great Kantō Earthquake (関東大震災) in 1923, the Great East Japan Earthquake (東日本大震災) in 2011 (a.k.a. the Tōhoku Earthquake) and the recent 2024 Noto Peninsula Earthquake (能登半島地震) all caused hundreds of deaths, both due to the result of the earthquake(s) and the resulting tsunami.

To provide protection to its residents, the Japan Meteorological Agency (JMA, 気象庁), together with the National Research Institute for Earth Science and Disaster Resilience (NIED, 防災科研) placed thousands of **earthquake sensors** across Japan (the Hi-net, the K-NET, the KiK-net and the F-NET), with several of them lying deep in the sea bed, measuring displacement, velocity and acceleration, which are connected to multiple servers, including two located in Ōsaka and Tōkyo.

Using data obtained from the sensors, computers do some complicated algorithms (mentioned below) to send out **early earthquake warnings (EEWs, 緊急地震速報)** automatically within milliseconds. There are two types of EEWs:

1. **EEW (Forecast, 予報).** Sent out to **highly-dependent industries** (e.g. rail industry, power plants) and **subscribed users**, when maximum intensity level of more than 3, or a magnitude of more than 3.5 is expected.
2. **EEW (Warning, 警報).** Sent out to **everyone** via TV, Radio, Mobile Phone, SMS, etc., when a maximum intensity level of more than 4 is expected.

After the earthquake, JMA staff will determine the location and severity of tsunami warnings to be issued, if necessary.

1.1.2 Earthquake Terminology

- **Intensity (震度).** The intensity describes the intensity vibration of a point due to an earthquake. It is not unique to an earthquake - **different places can have different intensities** due to the

distance to the epicentre, and intensity will also change over time. JMA measures intensity using **9 levels: 1, 2, 3, 4, 5–, 5+, 6–, 6+ and 7** in increasing order.

- **Magnitude/Scale.** The magnitude of an earthquake describes the energy released in the earthquake in a logarithmic scale. **It is unique to an earthquake.**
- **Epicentre/Hypocentre.** The epicentre is the surface point directly above the true centre of the earthquake.
- **Focal Depth.** The focal depth is the depth of the true centre of the earthquake.
- **P-Wave and S-Wave.** These are seismic waves, sourced from the true centre of the earthquake, travelling at different speeds, with Primary (P)-Wave travelling faster and Secondary (S)-Wave travelling slower.

1.2 Problem Area

The main goal of this application is to provide a visualisation of the earthquake/tsunami related data feed(s) provided by JMA's affiliated institution, Disaster Mitigation Data Send Service (DM-D.S.S). There are numerous apps providing a list of recent earthquakes, the real-time data measured by the sensors, and the real time earthquake warning displayed on a map, but rarely are there good apps that combine all those features together satisfyingly, with just the necessary features the author needs.

Some applications are no longer being updated due to change in the user's policy of the related data feed. Furthermore, most of the apps available are only in Japanese, not in English or my home language Chinese, which can create trouble for the author to understand.

1.3 Client and End User

The primary target of this application will be passionate geographers and geologists who are interested in the study of earthquake observations and predictions. The age group of this vary all the way from primary-school students to adults, including the author who has been amazed by the technology since the age of 12. They could take any employment, ranging from students to full-time jobs. Their proficiency usually varies, since there are people new to this field who probably does not have much knowledge, so the interface of the application should be relatively user-friendly and understandable, hiding unnecessary technical complexities.

Another target client could be industries which highly rely on earthquake predictions due to the risk imposed by earthquakes. High-speed railway and nuclear power plants are good examples of this. Therefore, the staff in charge monitoring will usually have higher proficiency and would like more detailed data of the earthquake. However, they will only need the necessary data from earthquakes happening close to them and only require intensity data of the point in interest (e.g. the power plant). To put this into content, an earthquake happening 1000 km away from them does not need to be fed into their system, while they would like to see the intensity of the shock and the arrival time of the seismic waves to decide the actions. In fact, the author really likes investigating on the rail industry, whose infrastructure could be greatly affected by earthquakes.

Table 1.1 compares relative features of these two target users/clients.

Feature	Primary Target	Secondary Target
Description	Passionate Geologists	Earthquake-Sensitive Industries
Age	Varies (Middle School – Adults)	Work Age
Reason	Monitor Live & Latest Earthquakes	Monitor Risks to Infrastructure
Proficiency	Varies (Beginner – Amateur)	Trained Professional
General Requirements	Monitor Overall Movement	Alert about intensity and arrival time at specific points

Table 1.1: Comparison of target users and clients

1.4 Research Methodology

1.4.1 Client Interview

The author interviewed my friend Wesley Ma, who is a passionate geologist on earthquake studies and also monitor earthquakes regularly.

1. *Which earthquake monitoring apps do you use?*

   Response: JQuake, SREV, Quarog, KEVI, Kyoshin-Monitor (Support discontinued), Kiwi Monitor (Support discontinued)

2. *Do you subscribe/pay to services such as DM-D.S.S. to use earthquake monitoring apps, and do you think it is worth the price?*

   Response: Yes. However, DM-D.S.S. is a little expensive. However, the price becomes more affordable considering the information provided by the subscription.

3. *Do you watch YouTube livestreams on earthquake monitoring?*

   Response: I do not usually watch the live streams, as almost no one who has already has a monitoring app will use the stream. They provide mostly the same information as the applications, just real-time streaming the windows.

4. *Why do you use earthquake monitoring apps?*

   Response: To monitor the earthquake. This is derived from my interest in broadcasting culture in Japan. This eventually led me to be intrigued with the development of earthquake monitoring technologies and theories in Japan.

5. *How often do you use earthquake monitoring apps (e.g. all the time/after school/only after big earthquakes)?*

   Response: After big earthquakes. But I usually open one or two apps for all-day monitoring to catch potential major (or medium) earthquakes.

6. *Describe the advantages and disadvantages of each of them, mentioning the specific features.*

   Response:

   - SREV is a good one, but it is only available in a browser with no app.
   - Kyoshin-Monitor and Kiwi Monitor are relatively more stable, but the source is not the same as that from the former two apps, and its support is also discontinued.
   - Quarog has weak response time and interacting interface.
   - JQuake is the most developed app, which includes nearly all functions that can be thought of. However sometimes the connection of WebSocket is unstable.
   - KEVI does not have the sound files configured by default. Some information are not displayed clearly enough.

7. *What features do you use the most/least?*

   Response: Basic earthquake notifications, and should be including sufficient and prompt information.

8. *What features are redundant in the earthquake monitoring apps you use?*

   Response: KEVI has a weather monitoring function, which could be redundant. But that could be caused by the different purposes of the app. Hence, no further comments. It is not a bad thing.

9. *What are the critical features of an earthquake monitoring app?*

   Response: To provide accurate, prompt and detailed information according to the source released by the JMA, the information display interface should be easy to read and understand. This is not only helping the people who like to monitor, but more importantly, provides the easiest way to the people who really need to seek information to minimise the harm brought by earthquakes and successive disaster.

10. *What additional features would you like to have in those existing apps?*

    Response: Summarise all the useful features from different apps into one app. Stable connection. Nothing else.

1.4.2 Existing Applications and Solutions

Based on the applications the author uses and the feedback from interviewee, there are the following commonly-used applications:

- JQuake
- Scratch Real-time Earthquake Viewer (SREV), available in compiled form at kotoho7/scratch-realtime-earthquake-viewer-page
- Kyoshin EEW Viewer for Ingen (KEVI), available at ingen084/KyoshinEewViewerIngen
- Quarog

1.4.2.1 Supported Platforms

Supported platforms of those apps are listed in Table 1.2. In particular, note that SREV is a web-based and GitHub Pages-hosted application therefore supporting all platforms. KEVI is written in .NET Framework and supports the second most platforms, with JQuake not supporting Linux and Quarog only supporting Windows.

Platform	JQuake	SREV	KEVI	Quarog
Windows	✓	✓	✓	✓
macOS	✓	✓	✓	
Linux	✓	✓	✓	
Android/iOS		✓		

Table 1.2: Supported platforms of existing solutions

1.4.2.2 Earthquake Monitoring

An overview feature table of monitoring is included in Table 1.3. In particular, note that due to the nature of SREV being Scratch-programmed and web-based, it reached a special agreement with DM-D.S.S. to use the API without the need of all users paying for this, since it is hard to integrate such function into a web application.

Quarog is a relatively new application and only supports basic functionalities of EEW Viewing and past earthquake listing, as shown in Figure 1.2. JQuake is solely dedicated to earthquake and tsunami monitoring as shown in Figure 1.1, while KEVI has features like rain clouds map and natural disaster warning which is beyond the scope of this analysis (which is a burden to users only requiring earthquake monitoring and a waste of storage space).

It is worth noting that for SREV, DM-D.S.S. is integrated with special permission with no login required. For JQuake, although it has tsunami warnings/special warnings, the tsunami forecast is not available.

Feature	JQuake	SREV	KEVI	Quarog
NIED Real-time Shake Support	✓	✓	✓	
DM-D.S.S. WebSocket Support	✓	✓	✓	✓
Real-time Sensor Data	✓	✓	✓	
Vibration Alert	✓	✓	✓	
Past Earthquake List	✓	✓	✓	✓
Past Earthquake Details		✓	✓	✓
Tsunami Warning	✓	✓	✓	
Real-time EEW	✓	✓	✓	✓
Calculated Seismic Wavefronts	✓	✓	✓	✓
User-Defined Key Monitor Point	✓		✓	
Sub-Map for the Okinawa Area	✓		✓	
Replay	✓		✓	

Table 1.3: Feature comparison in monitoring of existing solutions

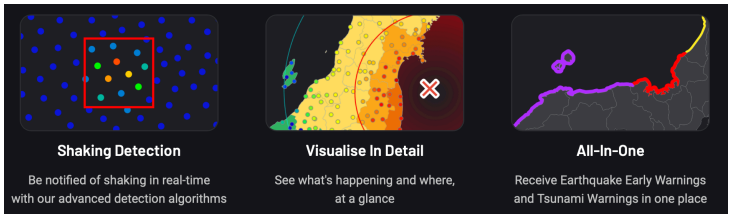

Figure 1.1: Feature introduction of JQuake, screenshot from website.

Figure 1.2: Feature introduction of Quarog, screenshot from website. Top-left: Past earthquake information; Top-right: Real-time EEW; Bottom-Left: Past earthquake list; Bottom-Right: Details of EEW.

1.4.2.3 Configuration Options

There are also a variety of configuration options available for all apps, as listed in Table 1.4. Both KEVI and Quarog supports the adjustment of the colour theme, and Quarog even supports changing the style of how blocks are displayed and coloured as shown in Figure 1.3 and 1.4. Playing a sound on the speaker is also common among the apps to remind the user of earthquakes.

Feature	JQuake	SREV	KEVI	Quarog
DM-D.S.S. Login	✓		✓	✓
Sound Alert	✓	✓	✓	✓
System Notification			✓	
Colour Theme			✓	✓
Map Colouring Style		✓		✓

Table 1.4: Feature comparison in configuration and customisability of existing solutions

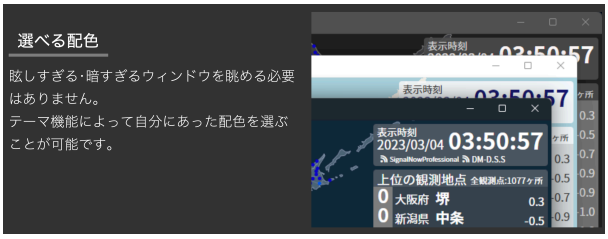

Figure 1.3: Customisable colour scheme of KEVI, screenshot from website.

Figure 1.4: Customisability of Quarog, screenshot from website.

1.5 Features of proposed solution

Based on the potential user/client interview results, and the research into the existing solutions, the following key features should appear in the solution, since they are essential to earthquake monitoring applications:

1. DM-D.S.S. Login functionality (for the data source)
2. Real-Time Sensor Shake Intensity Data (w/ vibration alert)
3. EEW Visualisation (w/ Calculated Seismic Wavefronts)
4. Past Earthquake List (w/ Option to review details)
5. Tsunami Warning Visualisation (w/ Related Sounds)

However, due to the limitation of time and the difficulty of implementation, the following functionalities are not the key to implementation, which include mostly the customisation parts, ranked in decreasing importance:

1. User-Defined Key Monitor Point
2. Customisable Sound Alert
3. Customisable Colour Theme
4. Customisable Map Colouring Style

The following features might not be available due to the time constraints and the complexity behind such system:

1. Replay (due to a server needing to store past data)
2. Sub-Map for the Okinawa Area (due to the difficulty in implementation)
3. Map Zooming Feature (due to the complexity in the map colouring functionality)

With those key features implemented, the application should mirror all essential functionalities of an earthquake monitoring application, as compared to the four existing solutions above.

1.6 Critical Path

The functionality of this application can essentially be divided into $1 + 2 + 1 + n$ steps, where the 1 is to receive information from the API/WebSocket provided by DM-D.S.S. and NIED, and 2 is to, briefly saying:

1. Real-time Monitoring: Produce a real-time map to monitor real time vibration and plot real-time EEW/tsunami warnings
2. Past-earthquake Viewing: Produce a menu to select an earthquake to display information on the map.

The next 1 is to merge these two functions, specifically the functionality to switch back to the real-time monitoring option immediately when a new EEW is released (to make sure the user does not miss any information on real-time EEWs while looking at past earthquake information).

The final n is to implement a setting page for customisation, and some extra additional features.

1.6.1 Part 1. NIED and DM-D.S.S. API/WebSocket Connection

1. Investigate into the list of APIs/WebSockets that should be used by the application.
2. Implement classes/DTOs to convert the information into C# objects.
3. Implement classes to transfer real-time WebSocket XML/JSON to the classes and DTOs.
4. Use simple API Key to test the functionality of this sub-system.
5. Implement OAuth 2 login to reduce complexity and increase security.

1.6.2 Part 2a. Real-time Monitoring Map

1. Implement a Japan map in the application.
2. Achieve and store the locations of the monitoring points in Japan, using necessary information from JMA, NIED and DM-D.S.S.
3. Colour the monitoring points using a colour scheme on the map.
4. Investigate into and implement an algorithm to detect shake in a certain area of the map.
5. Achieve and store the names of areas of earthquake epicentres from JMA.
6. Investigate into and implement an algorithm to calculate seismic wave fronts.
7. Plot and display real time EEWs, considering special cases such as:
   - Cancellation of EEW;
   - Upgrade from EEW Forecast to EEW Warning;
   - Multiple Earthquakes (hence displaying epicentres with labels).
8. Implement functionality of colouring areas on the map with the maximum expected intensity.
9. Achieve and store the names of shorelines from JMA.
10. Plot and display real time tsunami warnings.

1.6.3 Part 2b. Past-earthquake Viewing

1. Implement a side-list of a list of path earthquakes.
2. Provide a button to view the detailed information on the earthquake (e.g. magnitude, epicentre, time).
3. Implement functionality to plot the detected maximum intensities on the map, from where they are detected.
4. Provide external link to view earthquake in the JMA website.

1.6.4 Part 3. Joint functionality

1. Provide sidebar to switch between real-time monitoring and past-earthquake viewing.
2. Implement automatic functionality to switch back to real-time earthquake monitoring when a shake over a certain magnitude is detected, or an EEW/Tsunami Warning is being published.

1.6.5 Part 4. Setting page for customisation and some more

1. Implement customisable voice and sound playing (e.g. when EEW (over certain magnitude) is published, when an earthquake information detail is received, etc.)
2. Implement customisable colour scheme for the colouring of different intensity scales, with several built in default.
3. Implement key monitoring point with special warnings when an earthquake with more than a certain intensity is expected to hit that point.
4. Implement a map-zooming feature.
5. Implement a sub-map for the Okinawa Area.

1.7 Requirements Specification

A detailed specification is defined here that is to be aimed to be fulfilled at the end of the project, split into four sections:

1. **NIED and DM-D.S.S. Functionality** – Corresponding to **Part 1**, in Table 1.6
2. **Real-Time Earthquake Monitoring** – Corresponding to **Part 2a**, in Table 1.7
3. **Past-earthquake Viewing** – Corresponding to **Part 2b**, in Table 1.8
4. **GUI Design** – Corresponding to **Part 3, 4**, in Table 1.9

The objectives here are SMART, meaning they are specific, measurable, achievable, realistic and timely.

Note that those marked with an * means they are optional.

Table 1.5 for types of testing. [M] afterwards will stand for a necessary manual testing (i.e. specific user inputs), while (M) will stand for supplementary manual testing (i.e. will have stand-alone tests as well as manual tests).

Test Method	Abbr.
Unit Testing	UT
Integrated Testing	IT
Performance Testing	PT

Table 1.5: Measurement methods

Req. №	Description	Success Criteria	Testing
1(i)	Login to DM-D.S.S. using an API Key	Login successful	UT [M]
1(ii)	Call HTTP-Based APIs	Successful calls	UT (M)
1(iii)	Connect to WebSocket Data Feed	Successful calls	UT [M]
1(iv)	Obtain stable connection on WS	Connected for 30min	PT (M)
1(v)	Successfully parse JSON and XML into C# objects	Information successfully parsed	UT
1(vi)	Exception handling for incorrect JSON and XML	Exceptions thrown	UT
1(vii)	Exception handling for failed connections	Exceptions thrown	UT (M)
1(viii)	Integrated functionality from login to providing objects	Correct objects created	IT [M]
1(ix)*	Login to DM-D.S.S. using OAuth2	Login successful	UT (M)

Table 1.6: Requirements for Part 1

Req. №	Description	Success Criteria	Testing
2a(i)	Display real time shake-intensities on the map	Coloured points at correct locations	IT
2a(ii)	Implement algorithm to display shake detection on the map	Use squares to indicate shake detected in area	UT, IT, PT
2a(iii)	Display the time at the corner of the UI	Time displayed with less than 100ms error	UT
2a(iv)	Display real-time EEW when issued	Epicentre at correct position	IT
2a(v)	Update/Cancel EEW when appropriate	Correctly updated and plotted	IT
2a(vi)	Calculate and display seismic wavefronts by algorithm	Calculated and plotted without significant delay	UT, IT, PT
2a(vii)	Colour map with expected maximum intensity of EEW	Correctly coloured	UT, IT
2a(viii)	Display the exp. magnitude, location, depth and intensity when EEW issued	Correctly formatted and displayed	UT, IT
2a(ix)	Provide additional EEW information (e.g. detailed time, algorithm used)	Correctly formatted and displayed	IT [M]
2a(x)	Display tsunami warnings when issued	Coloured at correct location	UT
2a(xi)*	Display real-time shake data at a user-defined point	Display the name of the point with acceleration and intensity	IT

Table 1.7: Requirements for Part 2(a)

Req. №	Description	Success Criteria	Testing
2b(i)	Display past-earthquake side list using data from DM-D.S.S.	Displayed and updated	UT, IT
2b(ii)	Display the time, location, depth, intensity and magnitude of earthquake on the side list	Correctly formatted and displayed	UT, IT
2b(iii)	Colouring of the map by the maximum intensity of an earthquake when prompted	Correctly coloured	UT, IT
2b(iv)	Display details of shake and other additional information provided by JMA when prompted on a sub-window	Correctly formatted and displayed	UT, IT
2b(v)	Provide link to external weather services for details	Linked to correct website and earthquake	UT [M]
2b(vi)	Provide link to JMA Earthquake details	Linked to correct earthquake	UT [M]

Table 1.8: Requirements for Part 2(b)

Req. №	Description	Success Criteria	Testing
3(i)	Provide sidebar to switch between real-time, past earthquake and settings	Sidebar functioning	IT [M]
3(ii)	Switch to real-time monitoring map when EEWs are issued	Switch with delay less than 1s	IT, PT
3(iii)	Provide an easy-to-use GUI	Potential user gives rating of more than 7 out of 10	[M]
3(iv)*	Provide page to design colour scheme	Functional design page and applied to the whole UI	UT
3(v)*	Play sound when events happen and provide option to customise sound files	Correct sound played when required	UT, IT
3(vi)*	Provide option to define a point on the map to monitor	Functioning as specified in Part 2a	UT, IT
3(vii)*	Provide zooming feature	Map functioning as specified in Part 2	UT, IT
3(viii)*	Provide sub-map to display Okinawa Area	Map functioning as specified in Part 2	UT, IT

Table 1.9: Requirements for Part 3

Chapter 2

Design

Overview

This section includes a breakdown of the application into sections of data processing, GUI functionalities and joint functionalities. The use of external sources including DM-D.S.S. and NIED data sources is discussed, together with the relevant formats (XML, JSON) and the objects related. A UML class diagram is included to discuss OOP relations of classes, records (record classes) and enums including use of inheritance, composition, association and aggregation. They also implement different interfaces. An outline of the design of the user interface is included. The expected hardware requirements of the systems are also listed, but any laptop with an up-to-date operating system (running Windows or macOS) should be able to run the program.

2.1 Data Structures/Data modelling

2.1.1 External Data Sources

There are two data sources this program will use: the NIED and the DM-D.S.S. Specifically, the former one is used to achieve the real-time shake data of the sensor points which were set up by the government (whose data is free to use), and the latter one is used to achieve past earthquake information and EEW information sent out by the JMA (which is pay-to-use). Note that DM-D.S.S. does also provide the real-time intensity data of the observation points, however it is pay-to-use only for companies and institutions on request. Therefore, it will not be feasible to use this data source in the program since one of and the principle target users is people passionate in monitoring earthquakes.

2.1.1.1 NIED Data Source

As mentioned before, NIED has numerous 'earthquake observation nets' across Japan. Specifically, there is the K-NET and the KiK-net, which is dedicated to the observation of strong seismic motion. The K-NET consists of approximately 1000 sensors located across Japan, while the KiK-net also includes some sensors which are located within the earth, which will often have different readings compared to those located on the surface. They are extremely capable of detecting strong motion of ground. Furthermore, the K-NET and the KiK-net provides real-time intensity data webpage of two types, the 'Kyoshin' (強震) monitor and the long-period ground motion (LPGM, 長周期地震動) monitor (not working at the time of investigation). The Hi-net stands for high-sensitivity seismograph network, and it is dedicated for observation of minor motions of the ground. They release the waveforms to those who are researching seismic movements. As for the F-net which stands for the Full Range Seismograph Network of Japan, which is used to analyse the mechanism of a certain earthquake by analysing movements. None of the three nets provide a real-time API data feed.

Having compared the functionalities described above of the K/KiK-net, Hi-net and F-net and how they feed the data sources, the most suitable data source to reflect real-time motion of ground movements will be the **K/KiK-net**'s data feed, since it detects strong ground movements and is available real-time for the purpose of the application. (This is also the data source that JQuake and KEVI use in fact.)

In fact, in addition to these three networks, there are also the S-net, the DONET and the N-net, which detects the ground seismic movements in the sea. These data were adapted by SREV, but this is

beyond the scope of this NEA analysis.

A comparison from the official website of MOWLAS (Monitoring of Waves on Land and Seafloor) of the three nets are included in Figure 2.1 and a map of the distribution of the sensors are included in Figure 2.2.

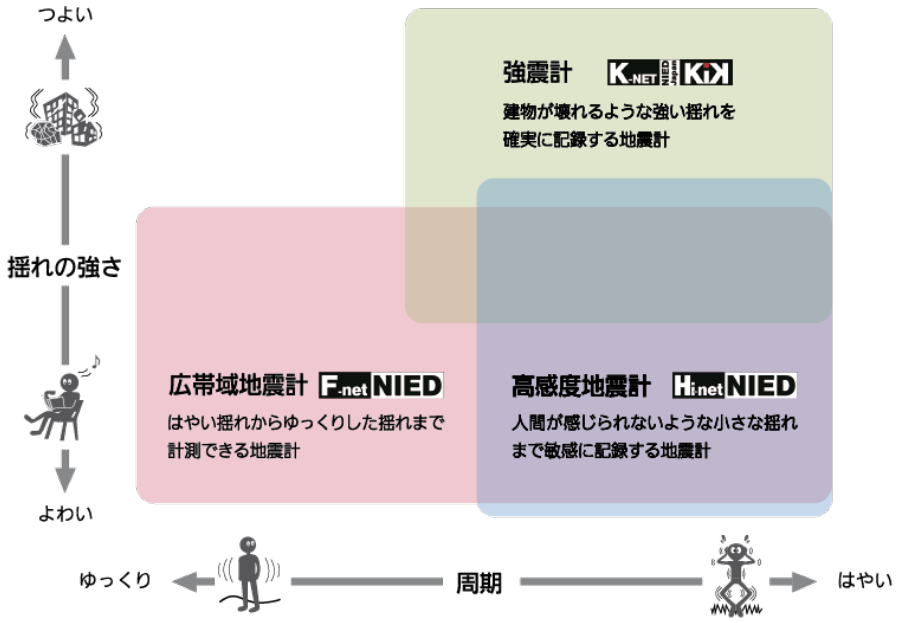

Figure 2.1: A comparison of the K-NET, F-net and Hi-net.

Figure 2.2: Distribution of the sensors of different nets.

Achieving image format data source The data source fed by the 'Kyoshin' Monitor is split into 8 types (detailed below in Table 2.1), and each type split into 2 types of data sources, surface sensors and borehole (earth) sensors, with codes in Table 2.2. The link to the GIF image is in the following format:

`http://www.kmoni.bosai.go.jp/data/map_img/RealTimeImg/[#1]_[#2]/[yyyyMMdd]/[yyyyMMdd][hhmmss].[#1]_[#2].gif`

In the link, the `[yyyyMMdd]` and the `[hhmmss]` part should be replaced with the date and time respectively (in JST, UTC+8), and the `#1` replaced with the codes detailed below for the data types, and `#2` replaced with the codes detailed below for data sources. An example of the imaged achieved is in Figure 2.3.

Data Type	Description/Meaning	Code in `#1`
Real-time Shindo	Real-time Measured Intensity	`jma`
PGA	Peak (Maximal) Ground Acceleration	`acmap`
PGV	Peak (Maximal) Ground Velocity	`vcmap`
PGD	Peak (Maximal) Ground Displacement	`dcmap`
Response 0.125Hz	Response spectrum for 0.125Hz PGV	`rsp0125`
Response 0.250Hz	Response spectrum for 0.250Hz PGV	`rsp0250`
Response 0.500Hz	Response spectrum for 0.500Hz PGV	`rsp0500`
Response 1.000Hz	Response spectrum for 1.000Hz PGV	`rsp1000`
Response 2.000Hz	Response spectrum for 2.000Hz PGV	`rsp2000`
Response 4.000Hz	Response spectrum for 4.000Hz PGV	`rsp4000`

Table 2.1: Data available in 'Kyoshin' monitor

Sensor Type	Description/Meaning	Code in `#2`
Surface	K-NET and KiK-net sensors	`s`
Borehole	KiK-net sensors within earth	`b`

Table 2.2: Sensors available in 'Kyoshin' monitor

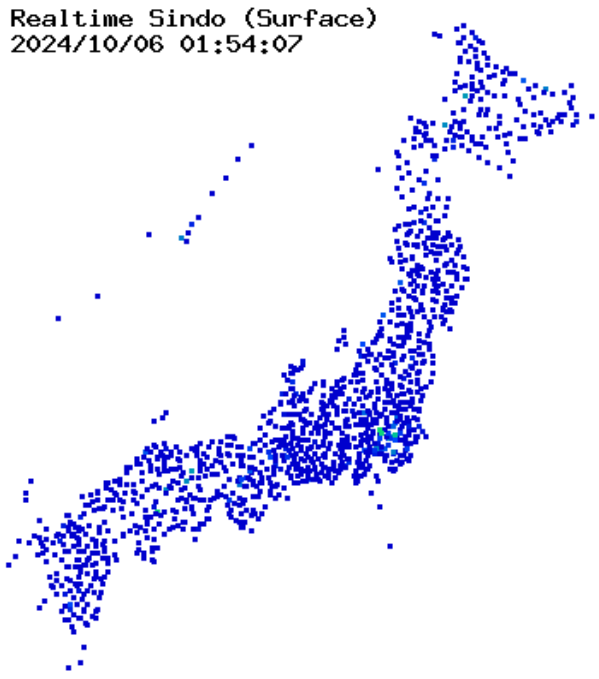

Figure 2.3: Sample GIF image achieved from 'Kyoshin' Monitor

Extracting colour for each observation point Unfortunately, it seems to the author (and is widely accepted in the EEW monitoring app development society) that the position of the points (squares) on the image does not follow any significant pattern of position, i.e. there is no obvious conversion of

coordinates to us from the official longitude/latitude locations to the positions on the image. Therefore, a manual conversion one-to-one mapping has to be developed.

NIED does have an official released list of observation points, which include their names and positions. This list has around 1700 of those observation points. However, in the actual image (like those in Figure 2.3), there are only 1000 of those in use in real time, consistent with K-NET's official introduction, and the rest 700 of those are invalid observation points. Therefore, it will be worth removing them from the list of earthquake monitoring points, before attempting to make the dictionary.

Unfortunately, 1000 is still quite a lot for us to deal with. Luckily, Ingen who used a similar approach to develop the KEVI application has already made such a mapping inside his open-source application in the file ShindoObsPoints.mpk.lz4 within ingen084/KyoshinEewViewerIngen , and even developed an editor for this at ingen084/KyoshinShindoPlaceEditor .

Due to the limited time for this NEA, the author will primarily use the pre-determined observation points for the K-net, and will use the existing application to map the points for KiK-net, which is still a considerable amount of work, but significantly less.

This paragraph referred to this blog article written by Ingen.

Converting colour to number format for further processing The true numerical data does not seem fully necessary at the first glance (since we might just as well just achieve the colour from the image and just plot them on the map, without the need to convert to a colour and back). However, for us to detect the shake in certain regions, it is necessary for us to achieve the numerical value to run the algorithm on it. Nevertheless, it is just good to have the number for us to have the numerical value for potential future developments.

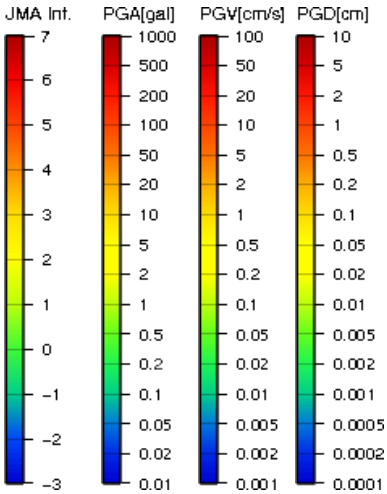

Figure 2.4: Scale colours of different measurements

As shown in Figure 2.4, the NIED 'Kyoshin' Monitor does indeed provide a scale of colours and reference to numerical values. However, there is a chance that a certain colour is not 'exactly' mapped on the scale, and further concerning that it is very slow and difficult to 'loop over' a colour legend, it is necessary to have an algorithmic-approach (numerical mapping-based approach) to map the colours in the colour space to numerical values (and back) is necessary.

Abstraction of colour scale Notice that the scale for PGA/PGV/PGD follow a logarithmic scale, while measured intensity follows a linear scale (though noting that the way intensity and magnitude is calculated is logarithmic as well). Therefore, if we normalise the vertical distance from the bottom of the axis h to $0 \leq h \leq 1$ (i.e. $h = 0$ at the bottom of the scale, $h = 1$ at the top of the scale), and if we denote intensity using I in JMA scale, PGA as a in gal, PGV as v in cm per second, and PGD as s in

cm, from the scale, the following transforming formulae obviously hold:

$$\begin{aligned} I = 10h - 3 &\iff h = \frac{I+3}{10}, \\ a = 10^{5h-2} &\iff h = \frac{\lg a + 2}{5}, \\ v = 10^{5h-3} &\iff h = \frac{\lg v + 3}{5}, \\ x = 10^{5h-4} &\iff h = \frac{\lg x + 4}{5}. \end{aligned}$$

However, it is worth noting that NIED did use $1, 2, 5, 10$ on the logarithmic scale at equal intervals, so it is not a perfect logarithmic scale. The author is unsure why they designed the scale like this, nor if it's an intended approximation. Nevertheless, the logarithmic scale is a good enough approximation.

The next step is to develop a mapping from this colour space $\mathcal{C}$ to h, which of course should be invertible. Denote this as $f : [0, 1] \to \mathcal{C}$.

Describing colour numerically We consider using a suitable base to decompose $\mathcal{C}$. The colour of the given scale is an immediate suggestion to use a base containing **hue**, which in fact is designed to describe how human perceive colour, and unlike RGB and CMYK which uses principle colours to describe colour. A hue scale is shown in Figure 2.5.

Figure 2.5: The hue scale in HSL/HSV encoding

Therefore, a colour in the colour space $\mathcal{C}$ can be represented as a 3-D vector $\mathcal{C} \ni C = (H, S, V)$, where $H \in [0, 360)$ in degrees is the hue value, $S \in [0, 1]$ stands for the saturation, and $V \in [0, 1]$ stands for the value (a brightness). And hence we will be able to decompose f into three components $f = (f_H, f_S, f_V)$.

Figure 2.6 plots the values of H, S and V against h (this is the graph of f and its components) of discrete values of h, and depending on the result we will attempt some fit/regression to a suitable function.

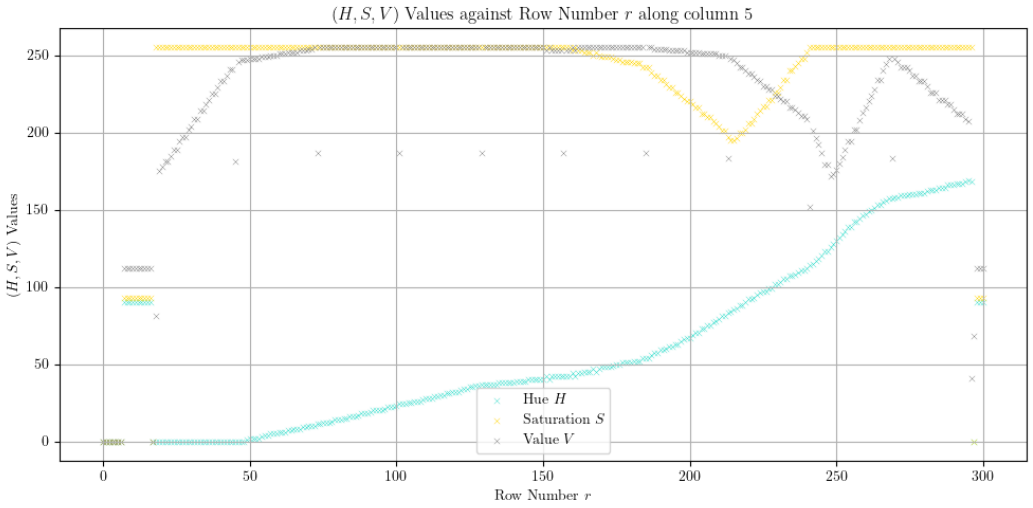

Figure 2.6: The values of (H, S, V) against pixel row r

Notice that in this plot, all values of (H, S, V) in fact range from 0 to 255.

It is worth noting that the scale has some space on the top (to show the type), and some space at the bottom. Notice that when the row $r = 17$ and $r = 297$ have values significantly different, so we extract the rows $r = 18$ and $r = 296$ to correspond (linearly) to $h = 1$ and $h = 0$, i.e.,

$$h = 1 - \frac{r - 18}{278}.$$

Figure 2.7 shows the result of this transformation being applied.

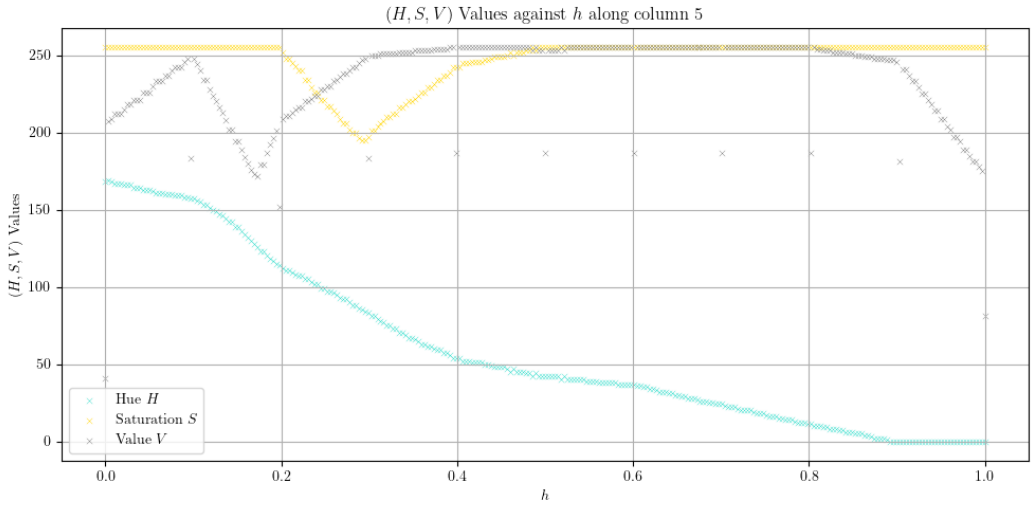

Figure 2.7: The values of (H, S, V) against normalised height h

From here onwards, all values of (H, S, V) will be adjusted to be within the range which they should be in, i.e. $H \in [0, 360), S \in [0, 1], V \in [0, 1]$.

Finding f_H in terms of h We consider finding f_H first, which is the cyan line. Notice that its trend can be split into 4 parts:

- $h \in [0, 0.1]$: linear;
- $h \in [0.1, 0.6]$: curving, ideally a cubic;
- $h \in [0.6, 0.9]$: linear;
- $h \in [0.9, 1]$: constant (0).

h	$H = f_H(h)$
0	237
0.1	222
0.6	51
0.9	0
1	0

Table 2.3: Initial values for f_H

Furthermore, boundary conditions in Table 2.3 are applied to ensure that the function is continuous and nicely-behaving while matching the existing data. We use the following function to apply the fit:

$$f_H(h) = \begin{cases} -150h + 237, & h \in [0, 0.1], \\ \odot, & h \in [0.1, 0.6], \\ -170h + 153, & h \in [0.6, 0.9], \\ 0, & h \in [0.9, 1]. \end{cases}$$

Here,

$$\begin{aligned}\odot = &\frac{222\cdot(h-0.3)\cdot(h-0.4)\cdot(h-0.6)}{(0.1-0.3)\cdot(0.1-0.4)\cdot(0.1-0.6)}\\ &+\frac{y_1\cdot(h-0.1)\cdot(h-0.4)\cdot(h-0.6)}{(0.3-0.1)\cdot(0.3-0.4)\cdot(0.3-0.6)}\\ &+\frac{y_2\cdot(h-0.1)\cdot(h-0.3)\cdot(h-0.6)}{(0.4-0.1)\cdot(0.4-0.3)\cdot(0.4-0.6)}\\ &+\frac{51\cdot(h-0.1)\cdot(h-0.3)\cdot(h-0.4)}{(0.6-0.1)\cdot(0.6-0.3)\cdot(0.6-0.4)}.\end{aligned}$$

Here, m_1 is the gradient of the line for $h \in [0, 0.1]$, $y_1 = f_H(0.3), y_2 = f_H(0.4)$ for $h \in [0.1, 0.6]$ (using Lagrange Polynomial), and the equation between $h \in [0.6, 0.9]$ is in fact fixed due to the initial conditions.

By applying a curve fit to the original data, the following results are obtained:

$$(y_1, y_2) = (115, 79.5).$$

Plotting H and $f_H(h)$ against h gives us Figure 2.8, which is decent.

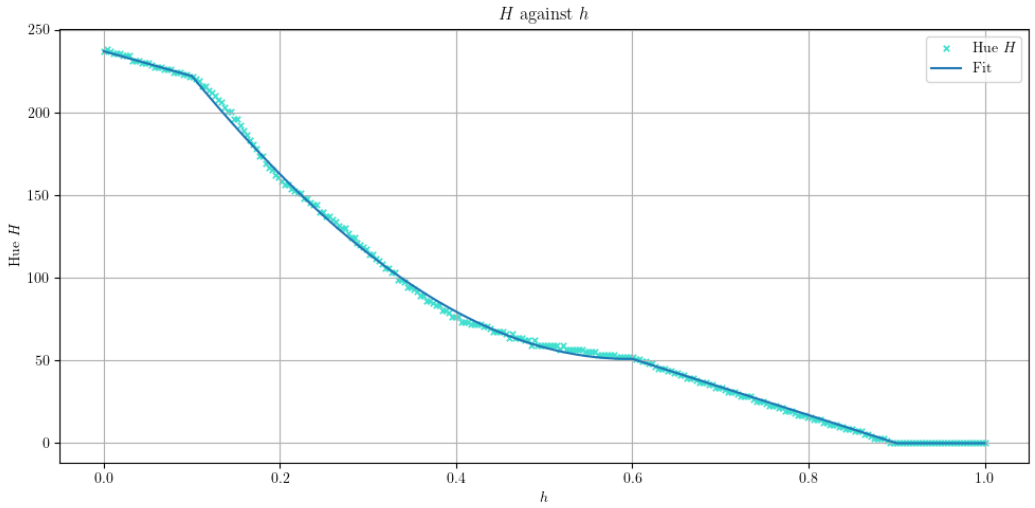

Figure 2.8: The fit result for $f_H : h \mapsto H$

Finding f_S in terms of h As for $f_S(h)$, the obvious thing to do is to split it into 5 (4) piecewise functions, specifically $f_S = 1$ for $h \in [0, 0.2] \cup [0.5, 1]$, and three linear functions for $h \in [0.2, 0.29], h \in [0.29, 0.4]$ and $h \in [0.4, 0.5]$. Initial values are included in Table 2.4.

h	$S = f_S(h)$
0	1
0.2	1
0.29	0.765
0.4	0.95
0.5	1
1	1

Table 2.4: Initial values for f_S

This gives us that

$$f_S(h) = \begin{cases} 1, & h \in [0, 0.2], \\ -2.611h + 1.522 & h \in [0.2, 0.29], \\ 1.682h + 0.277, & h \in [0.29, 0.4], \\ 0.5h + 0.75 & h \in [0.4, 0.5], \\ 1, & h \in [0.5, 1]. \end{cases}$$

Plotting this out gives Figure 2.9.

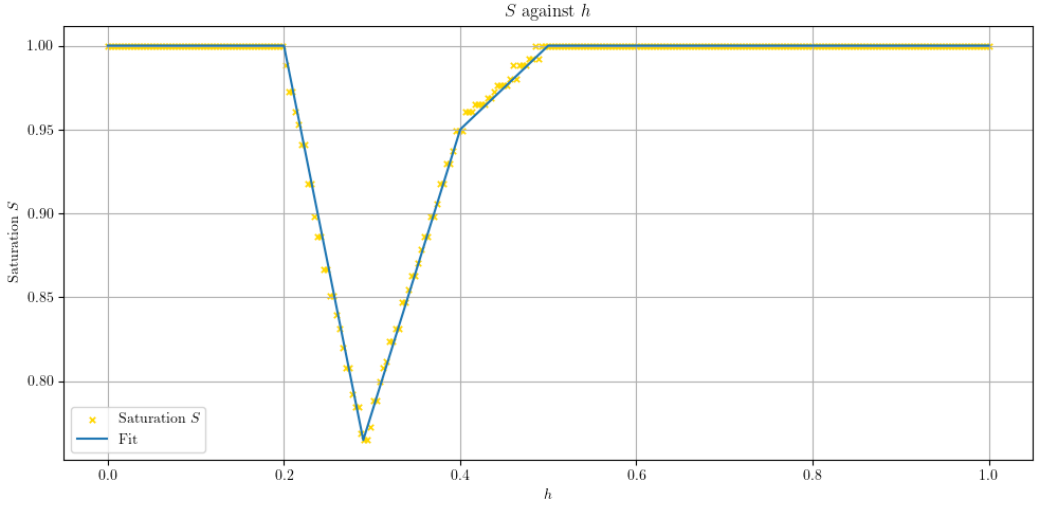

Figure 2.9: The fit result for $f_S : h \mapsto S$

Finding f_V in terms of h As for $f_V(h)$, we shall divide it into even more piecewise linear functions. Specifically, I chose to divide the interval $[0, 1]$ at $0.1, 0.172, 0.2, 0.3, 0.4, 0.8$ and 0.9. Initial values are included in Table 2.5.

h	$V = f_V(h)$
0	0.8
0.1	0.98
0.172	0.66
0.2	0.82
0.3	0.98
0.4	1
0.8	1
0.9	0.97
1	0.68

Table 2.5: Initial values for f_V

This gives us the piecewise function

$$f_V(h) = \begin{cases} 1.8h + 0.8, & h \in [0, 0.1], \\ -4.444h + 1.424, & h \in [0.1, 0.172], \\ 5.714h - 0.323, & h \in [0.172, 0.2], \\ 1.6h + 0.5, & h \in [0.2, 0.3], \\ 0.2h + 0.92, & h \in [0.3, 0.4], \\ 1, & h \in [0.4, 0.8], \\ -0.3h + 1.24, & h \in [0.8, 0.9], \\ -2.9h + 3.58, & h \in [0.9, 1]. \end{cases}$$

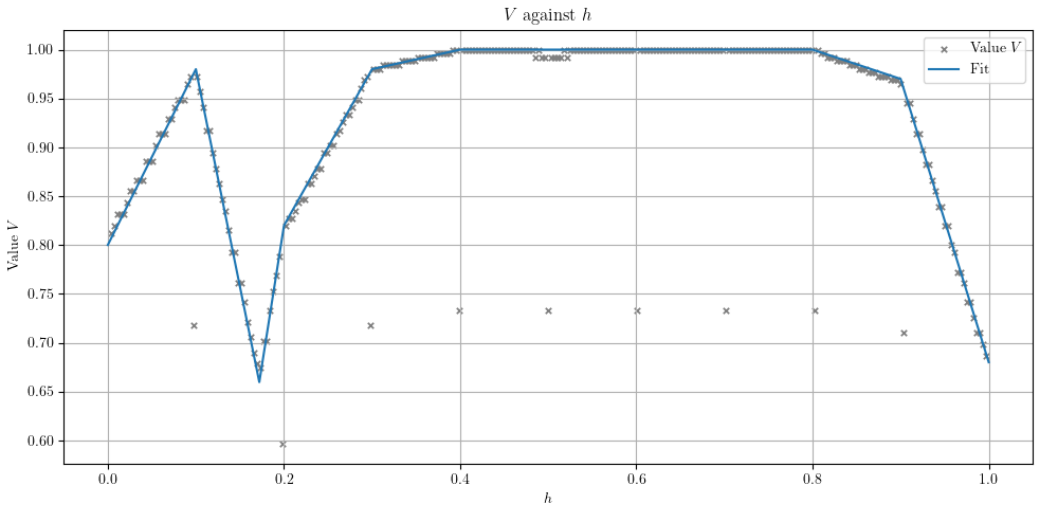

Figure 2.10: The fit result for $f_V : h \mapsto V$

Plotting this out gives us Figure 2.10.

Note that in this plot, V when $h = 0$ or $h = 1$ is excluded, since just like every $h = 0.1k$ for some $k \in \mathbb{N}$, they are anomalies created by the horizontal black line in the scale.

Finding f^{-1} To find $f^{-1} : \mathcal{C} \to [0, 1]$, we do not need necessarily to find an expression of h in terms of (H, V, S). If we notice that f_H is one-to-one on $h \in [0, 0.9]$, and f_V is one-to-one on $h \in [0.9, 1]$, we can use f_H^{-1} to determine h from H only if H is non-zero, and use f_V^{-1} otherwise.

$$f^{-1}(H, S, V) = \begin{cases} f_H^{-1}(H), & H \neq 0, \\ f_V^{-1}(V), & H = 0. \end{cases}$$

Notice that for $h \in [0.1, 0.6]$, f_H is a cubic and is not easily invertible. However, it would be plausible to use a binary-search algorithm to find h based on H since it is monotonic, and it is within a reasonable amount of time, to relatively good accuracy. Otherwise, on the linear parts, it is fine to simply mathematically invert it.

Flowchart of data and sidenotes To summarise, we discussed the mapping from the colour space$\mathcal{C}$ to the normalised height h, and back, and we also discussed how h is related with the measured intensity I, the PGA a, the PGV v, and the PGD x. They can be transformed forwards and backwards using simple mathematical explicit relations, and specifically for $f_H^{-1}(H)$ will use a binary search algorithm.

Figure 2.11 shows the data flow, Figure 2.12 shows the relation between abstract variables, and 2.13 shows the result of colour generated compared with the original.

It is worth noting that an existing NuGet Library, ingen084/KyoshinMonitorLib which is designated to manage intensities, as well as extracting intensities from the 'Kyoshin' monitor. This NEA did refer to this for some guidance but is not dependent on this library, and its necessary functionalities within the scope of this NEA is realised again using the author's own code. The developer of this library did also mention that it is quite purpose-built so might not be suitable for general use.

It is also worth noting that, technically, scraping the data from the 'Kyoshin' monitor page of NIED is not explicitly allowed, but not explicitly banned either. However, extracting and displaying numerical data in the application is strictly banned by the NIED, and therefore the numerical values will only serve as internal values of the application and will not be displayed in any way.

This paragraph referred to the blog article written by NoneType1, author of JQuake. The code used for this section is in Listing A.1.

2.1.1.2 DM-D.S.S. Data Source

DM-D.S.S. is a well-structured official data source with low latency and reliable information and services. This is going to be the primary data source for most part of the application.

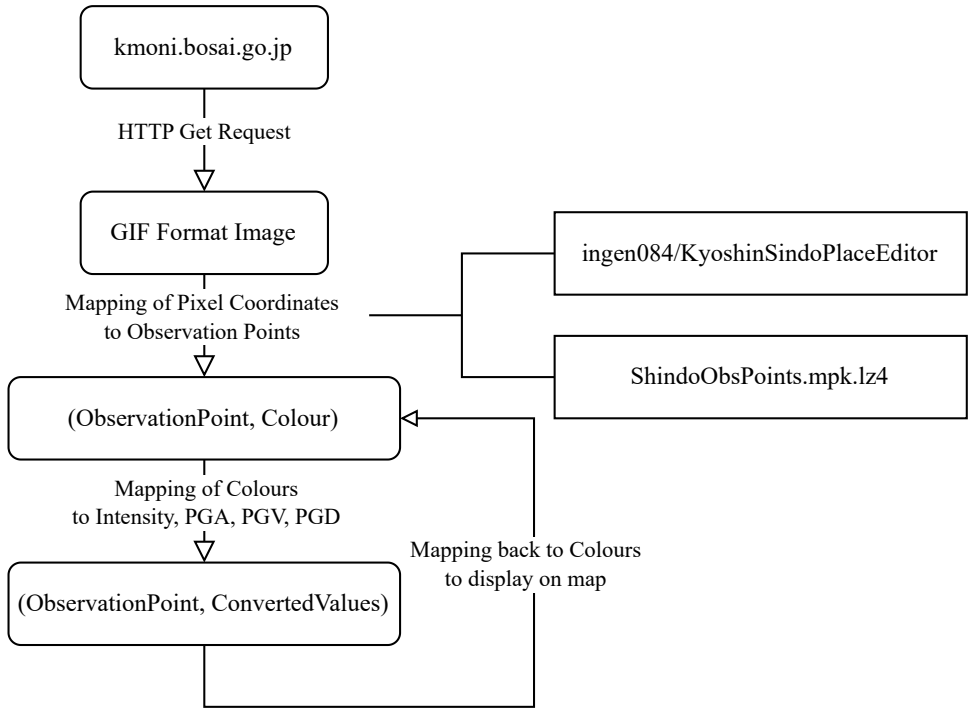

Figure 2.11: Flow of data in NIED data sources

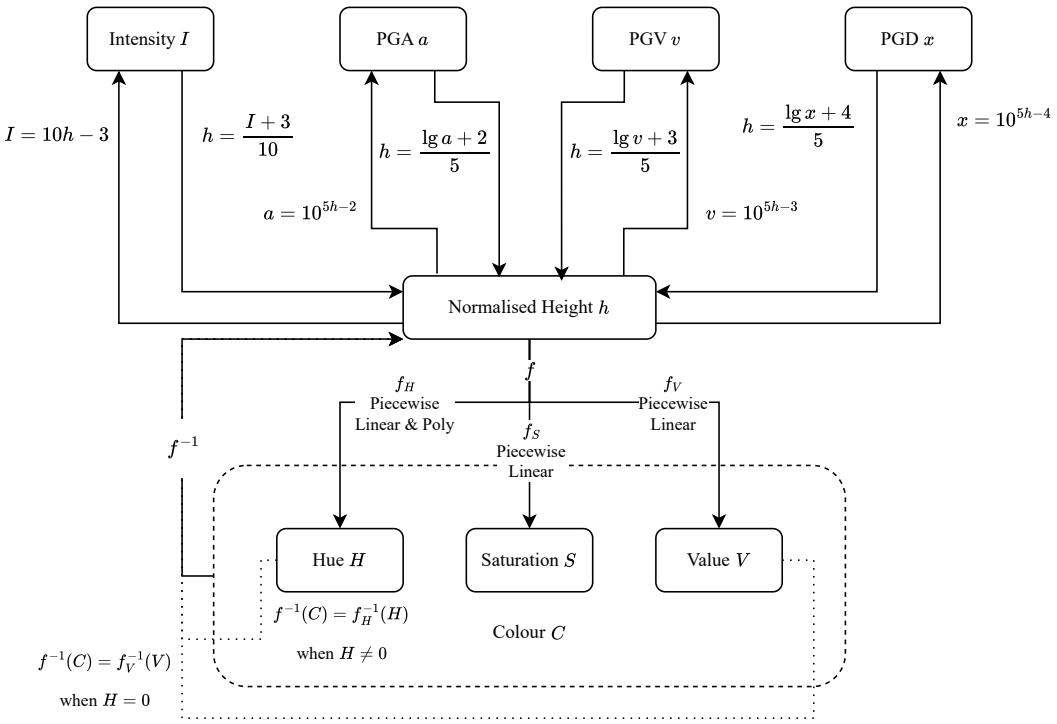

Figure 2.12: Relation between abstract variables

Their APIs are split into two types: HTTP based requests and WebSocket based connections. HTTP based requests are typically for more static information, while WebSocket connections are for live time-essential data feeds, such as the EEW warnings and latest earthquake information.

Authorisation There are two types of authorisation that DM-D.S.S. supports, API Keys and OAuth2 Access Tokens.

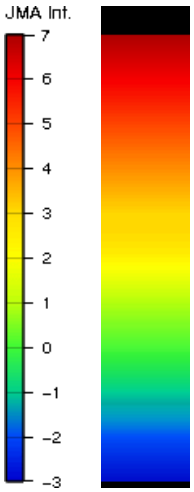

Figure 2.13: Colour generated using fitted functions

API Keys access tokens are extremely easy to program, since it simply uses Basic BasicBath64 Authorisation in the header, and uses the key as the username (without a password). It uses the Basic BasicBase64 Authorisation encoding. However, this introduces an extra layer of complexity for the users, since they would have to go to the settings of the DM-D.S.S. webpage and achieve an API Key to paste into the application, as shown in Figure 2.14.

Figure 2.14: Control panel for API Keys

As for OAuth2, it will be much simpler for the users, since it will provide the user with a login interface on the website, and ask them to give the program certain permissions, which is just a few simple clicks. Rather than the user sharing the credentials with the application, they are shared between the authorisation server (DM-D.S.S.) and the application directly, without the need for the user to deal with such human-unreadable codes.

How the OAuth2 works for DM-D.S.S. is outlined below:

For the purposes of this NEA, we will primarily use the API Key way of accessing DM-D.S.S. since it will be easier to code and debug, and OAuth2 will introduce quite a lot of complexity to the program. However, the program should be designed to be able to modify to OAuth2 authentication without much modification, and if time permits OAuth2 will be implemented in the application.

HTTP Based Requests The base URL of all requests is `https://api.dmdata.jp/v2/` which will be indicated as `base://` from now on.

Table 2.6 shows the necessary permissions would be necessary for the application to function. How each of them functions will be discussed below.

Standard Return Information and Errors There are two status that an API call could return: a successful `ok` status, or an unsuccessful `error` status.

Listing 2.1 shows the JSON for a successful `ok` response, and Listing 2.2 shows the JSON for a `error` response.

In both responses, there is an attribute `responseId` which gives the unique ID for each response as a `string`, and an attribute `responseTime` which gives the date and time of response in `ISO8601Time`

Permission Code	Permission Details
`contract.list`	Get the list of subscriptions the user has.
`gd.earthquake`	Get list of past earthquakes.
`parameter.earthquake`	Get details of observation points for earthquake intensities.
`socket.start`	Start a new WebSocket connection.
`socket.list`	Get list of existing WebSocket connections.
`socket.close`	Close an existing WebSocket connection.
`telegram.list`	Get list of telegrams released by the JMA.
`telegram.data`	Get specific telegram released by the JMA.
`telegram.get.earthquake`	Allows program to access telegrams on earthquake information.
`eew.get.forecast`	Allows program to access telegrams on EEW forecasts (including warnings).

Table 2.6: Necessary access permissions for DM-D.S.S.

```
{
    "responseId": "66d23c0cede77d82",
    "responseTime": "2021-04-01T00:00:00.000Z",
    "status": "ok"
}
```

Listing 2.1: `ok` status for API

```
{
    "responseId": "66d23c0cede77d82",
    "responseTime": "2021-04-01T00:00:00.000Z",
    "status": "error",
    "error": {
        "message": "Authentication required.",
        "code": 401
    }
}
```

Listing 2.2: `error` status for API

format.

The `status` attribute in `string` indicates whether it is a successful response (in which case it will be `"ok"`) and if it is an error response (in which case it will be `"error"` and gives the relevant HTTP error as well). The error will be indicated in the object `error` and will give the relevant message and code.

The list of standard errors together with their meanings is discussed in Table 2.7.

Error Code	Error Message
400	The query parameters are required.
400	The post parameters are required.
400	Unexpected data of search query `cursorToken`.
401	Authorisation required.
403	Insufficient scope for ....
403	Requests are not allowed.

Table 2.7: Standard errors for API.

From here onwards, the `responseId`, `responseTime` and `status` will be removed from the sample response by default.

Cursor Token For some API data calls, and specifically for `telegram.list` that we are going to use, there is too much data to be returned in one API call. Therefore, in a response, there will be a specified attribute named `nextToken` indicated with type `string`, sometimes as well as `nextPooloing` and `nextPoolingInterval`, as shown in Listing 2.3 as an example call of

`base://telegram?type=VXSE53`.

```
{
    "responseId": "8359288359fc5bb9",
    "responseTime": "2021-04-01T00:00:00.000Z",
    "status": "ok",
    "items": [
        {},
        {},
        {}
    ],
    "nextToken": "bmV4dCAgICAgICAgNTc0MzI",
    "nextPooling": "cG9sbGluZyAgICAgNzM0MzE",
    "nextPoolingInterval": 1600
}
```

Listing 2.3: Cursor token sample JSON.

Specifically, the calls for `socket.list` and `gd.eew` will only return a `nextToken` for the next call, while for `telegram.list` and `gd.earthquake` it will return a `nextPooling` as well.

If one would like to do the next call to find the next few items of the list, the `cursorToken` parameter should be specified as the same as the `nextToken` or `nextPooling`, with all the rest of the parameters identical to the previous request, i.e.,

`base://telegram?type=VXSE53&cursorToken=bmV4dCAgICAgICAgNTc0MzI`

if the `nextToken` is used, and

`base://telegram?type=VXSE53&cursorToken=cG9sbGluZyAgICAgNzM0MzE`

if `nextPooling` is used (which should be after `nextPoolingInterval` in milliseconds).

The document suggested that, due to performance issues, the `nextPooling` should be used wherever possible. Note that the `nextToken` and `nextPooling` will change for every single call of the API.

From here onwards, in the API sample results, `nextToken`, `nextPooling` and `nextPoolingInterval` will not be included.

Contract API There is only one contract-related API, which is `contract.list`. It is an HTTP GET request, on `base://contract` with no parameters.

It will return a list of contracts (subscriptions) of DM-D.S.S., whether or not the user has subscribed to it.

Listing 2.4 shows a sample return of this API call.

Here, `items` is the list of contracts (subscription plans), with each of its element being an object representing a plan. For each of them, there are the properties:

- `id` (nullable) which stands for the subscription ID (could be `null` if the user is not subscribed);
- `planId` which stands for the subscription plan ID, which is unique for each plan;
- `planName` which stands for the name of the subscription plan;
- `classifications` which stands for the classification code of the subscription plan;
- `price.day` and `price.month` which stands for the price of the plan per day/per month;
- `start` (nullable) which stands for the start of the subscription;

```
{
    "items": [
        {
            "id": 92,
            "planId": 1,
            "planName": " 地震・津波関連",
            "classification": "telegram.earthquake",
            "price": {
                "day": 15,
                "month": 350
            },
            "start": "2021-01-01T01:01:00.000Z",
            "isValid": true,
            "connectionCounts": 1
        }
    ]
}
```

Listing 2.4: Contract list sample JSON.

- `isValid` which is a `boolean` to represent whether the plan is valid; and
- `connectionCounts` which stands for the number of extra WebSocket connections this subscription plan provides.

Note that how `id` and `start` can be `null` since it lists all the subscriptions, whether or not the user has been subscribed to it.

This API will be used to determine if the user has an appropriate subscription plan to use the application.

WebSocket APIs There are three WebSocket-Related APIs, specifically, `socket.start`, to start a socket, `socket.list` to list all sockets, and `socket.close`, to close a socket. They are all called on `base://socket`.

`socket.start` is a POST method which has a request body in JSON, as shown in Listing 2.5.

```
{
    "classifications": [
        "eew.forecast",
        "telegram.earthquake"
    ],
    "types": [
        "VXSE45",
        "VXSE51",
        "VXSE52",
        "VXSE53"
    ],
    "test": "including",
    "appName": "Application Test",
    "formatMode": "json"
}
```

Listing 2.5: Socket start sample request JSON.

The attributes are:

- `classifications`, which is a list of classifications of the subscription plans (which include all the types below);

- `types` (optional), which is a list of telegram wished to be transmitted through the WebSocket;
- `test` (optional, default to `"no"`), which indicates whether test telegrams should be received (`"including"`) or not (`"no"`);
- `appName` (optional), which is the name of the application to be recorded with the WebSocket; and
- `formatMode` (optional, default to `"raw"`), which is either `"json"` or `"raw"` depending on the desired return type.

The detailed classifications and types will be discussed in the next section on WebSocket connections. When the `types` is excluded, the WebSocket will return all the telegrams included in the classification.

It will have a response as in Listing 2.6.

```
{
    "ticket": "Tik....",
    "websocket": {
        "id": 0,
        "url": "wss://ws003.api.dmdata.jp/v2/websocket?ticket=Tik....",
        "protocol": [
            "dmdata.v2"
        ],
        "expiration": 300
    },
    "classifications": [
        "eew.forecast",
        "telegram.earthquake"
    ],
    "test": "including",
    "types": [
        "VXSE45",
        "VXSE51",
        "VXSE52",
        "VXSE53"
    ],
    "formats": [
        "xml",
        "a/n",
        "binary"
    ],
    "appName": "Application Test"
}
```

Listing 2.6: Socket start sample response JSON.

Some attributes are exactly the same as the JSON sent has header, `classifications`, `types` (nullable), `test` and `appName`. The additional attributes are:

- `ticket`, which is a code in `string` for the connection. This is also included in the `websocket` object.
- `websocket`, which is the object containing details for the connections. Its attribute `id` is unique for each connection and `url` is the WebSocket (`wss://`) URL to use to connect to. `protocal` being a constant array containing `"dmdata.v2"` indicating that this is the 2nd version of the protocol. `expiration` is in seconds the time the connection will expire if no transmission handshakes are made (see below), and is always `300` as constant.
- `formats`, which is a list of formats used to encode the data transmitted in the WebSocket. This is a constant list including `"xml"`, `"a/n"` and `"binary"` if `formatMode` is set to `"raw"`, and a list with only `"json"` if set to `"json"` in the request JSON.

Error Code	Error Message
400	The body of the request is not JSON.
400	At least one element of `classifications` is required.
400	The `types` is not a string or has more than 30 elements.
400	The `appName` is up to 24 bytes.
400	You have entered a string that is not defined in `formatMode`.
402	No contract.
409	The maximum number of simultaneous connections is full.

Table 2.8: Errors for `socket.start`.

Apart from the standard errors, it will also output errors shown in Table 2.8

`socket.close` is a DELETE method, which specifies the ID of the WebSocket port to be closed in the URL parameter `:id`, e.g., `base://socket/30`. If the result is successful, the request will not return anything. If the status is error, it will return a standard error (detailed above), or a `404` error, which indicates that the specified WebSocket `id` is not found.

`socket.list` is a GET method which supports `cursorToken` (optional) as discussed above. There are other parameters in the query:

- `id` (optional), which stands for the ID of the WebSocket connection that details is wished to be retrieved;
- `status` (optional), a `string` indicating the status of the WebSocket, including `open` which stands for operating live, `waiting` which means idle, and `closed` standing for closed connections;
- `limit` (optional, default 20), of type `integer` with a maximum of 100, indicating the number of WebSockets listed in a single response.

Listing 2.7 shows a response from the call. The `items` is a list of objects, each representing a WebSocket, with the following properties:

- `id`, `ticket`, `classifications`, `test`, `types`, `formats`, `appName`, identical to described above;
- `start`, a `ISO0601Time` representing the time of the start of the connection;
- `end`, a nullable `ISO0601Time` representing the time of the end of the connection;
- `ping`, a nullable `ISO0601Time` representing the previous time of ping-pong;
- `ipAddress`, a nullable `string` representing the IP address of the source of connection;
- `server`, a nullable `string` representing the connected WebSocket server; and
- `status`, a `string` representing the connection status.

These will be used to manage (open new, list existing and close) WebSockets in the application.

Parameter API The program would need to have a list of observation points of earthquakes to plot on the graph. Therefore, we would need to call `parameter.earthquake`, which is available at `base://parameter/earthquake/station`.

This is a GET request with no parameters to specify, and will return all stations in one go. Listing 2.8 shows a sample response.

The `changeTime` and `version` of this list of parameters is often the same for calls since the list is rarely updated, and they give the version of the list and the last time of the update.

For each object in `items`, they represent an earthquake observation point. The attributes are:

- `region` and `city`, both containing `code`, `name` and `kana` (Kana (カナ, 仮名) represents the pronunciation), which is unique for each region and city;
- `noCode`, the unique code, and `code`, the code used in XML (WebSocket data feeds);
- the `name` and `kana`, which are not necessarily unique for each observation point;

```
{
    "items": [
        {
            "id": 0,
            "ticket": "Tik....",
            "types": [
                "VXSE45",
                "VXSE51",
                "VXSE52",
                "VXSE53"
            ],
            "test": "including",
            "classifications": [
                "eew.forecast",
                "telegram.earthquake"
            ],
            "ipAddress": "192.168.0.0",
            "status": "open",
            "server": "websocket-03",
            "start": "2021-04-01T00:00:00.000Z",
            "end": null,
            "ping": "2021-04-01T00:00:00.000Z",
            "appName": "ApplicationTest"
        }
    ]
}
```


Listing 2.7: Socket list sample response JSON.

- `status` standing for whether the station is in use. `" 現"` (now) means the data hasn't changed in the update, `" 変更"` (change) means the data has been altered, `" 新規"` (new) stands for a new observation point, and `" 廃止"` (abolished) means the observation point has been discontinued;
- the `owner`, representing the owner of the observation point; and
- the `latitude` and `longitude`, which as the name suggests, represent the position of the observation point.

Due to the nature of JSON being very long and barely updated (an email will be sent to users of DM-D.S.S. every time its updated in fact), it will be stored locally in the application's local storage, and updated automatically every day only. Upon program launch, the application will read the local data into the main memory.

Telegram API JMA release telegrams (mostly in XML format) through DM-D.S.S., and to achieve them, we can use the `telegram.list` GET request, which is available at `base://telegram`. It supports `cursorToken` with pooling.

Furthermore, the following query parameters are available:

- `type` (optional), which indicates the type of telegram wishing to retrieve. Note that there can be a maximum of 5 types of telegrams specified due to the need for a perfect match;
- `xmlReport` (optional, default `false`), which specifies whether some details of the XML report should be included in the JSON;
- `test` (optional, default `no`), which specifies whether test telegrams should be included. Options are `no`, `including` and `only`;
- `formatMode` (optional, default `raw`) indicates whether the XML reports should be converted to JSON;

```
{
    "changeTime": "2024-07-18T12:00:00+09:00",
    "version": "20240718",
    "items": [
        {
            "region": {
                "code": "100",
                "name": " 石狩地方北部",
                "kana": " イシカリチホウホクブ"
            },
            "city": {
                "code": "0123500",
                "name": " 石狩市",
                "kana": " イシカリシ"
            },
            "noCode": "1000000",
            "code": "0123500",
            "name": " 石狩市花川",
            "kana": " イシカリシハナカワ",
            "status": " 現",
            "owner": " 気象庁",
            "latitude": "43.1714",
            "longitude": "141.3156"
        }
    ]
}
```


Listing 2.8: Earthquake parameter sample response JSON.

- `limit` (optional, default 20) indicates the number of telegrams returned in one response.

Listing 2.9 gives a sample response of this API call.
In the `items` list, each object represents a telegram, with certain properties:

- `serial` stands for the serial ID of this transmission, while `id` is a unique ID for the telegram (non-serial);
- `classification` gives the classification of this telegram. For the purpose of this application, all classification will be of type `telegram.earthquake`;
- `head` gives the head of this telegram, including the `type`, the `author`, the `time`, and whether it is `test`;
- `receivedTime` gives the time of this telegram being released,
- `xmlReport` details parts of information encoded in the XML, if the `xmlReport` option in the query is set to true;
- `url` gives the URL for the detailed information of this telegram (that can be retrieved using another GET request), and `format` gives its format.

Apart from standard errors, it might also return a `400` error with message 'You have entered a string that is not defined in `formatMode`.'

Past Earthquake API The API `gd.earthquake` allows us to use a GET request to obtain a series of earthquake on `base://gd/earthquake`, with the option to obtain a list, or to obtain only a certain event.

These APIs will be used to display a list of past earthquakes. For each object representing an earthquake, this includes:

```
{
    "items": [
        {
            "serial": 0,
            "classification": "telegram.weather",
            "id": "123456789abcdef...",
            "head": {
                "type": "VPWW54",
                "author": "JPTD",
                "time": "2020-01-01T00:00:00.000Z",
                "test": false
            },
            "receivedTime": "2020-01-01T00:00:02.000Z",
            "xmlReport": {
                "control": {
                    "title": " 気象警報・注意報（Ｈ２７）",
                    "dateTime": "2020-02-27T00:00:00Z",
                    "status": " 通常",
                    "editorialOffice": " 気象庁本庁",
                    "publishingOffice": " 気象庁予報部"
                },
                "head": {
                    "title": " 東京都気象警報・注意報",
                    "reportDateTime": "2020-02-27T09:00:00+09:00",
                    "targetDateTime": "2020-02-27T09:00:00+09:00",
                    "eventId": null,
                    "serial": null,
                    "infoType": " 発表",
                    "infoKind": " 気象警報・注意報",
                    "infoKindVersion": "1.2_1",
                    "headline": " 注意報を解除します。"
                }
            },
            "format": "xml",
            "url": "https://data.api.dmdata.jp/v1/123456789abcdef..."
        }
    ]
}
```


Listing 2.9: Telegram list sample response JSON.

- Its unique `id`, `eventId`, `originTime` (if it is not a report on the intensity), and `arrivalTime`;
- its `type` (which is `normal` for earthquakes in Japan, and `distant` for earthquake reported by other countries);
- its `hypocenter`, including the `code`, `name`, the `coordinate` and the `depth`;
- its `magnitude` (an object representing units and values); and
- its `maxInt`, the maximum intensity observed.

It might also include LPGM information, in `maxLgint` and `lgCategory`.

If the call `base://gd/earthquake/:eventId` is used, it will return the object representing the earthquake, as well as a list of telegram XML, which was detailed above in the `telegram.list` section. However, it is worth noting that all data here will be parsed into JSON using DM-D.S.S.'s JSON Schema.

This API will be called to obtain a list of past earthquakes when the program launches, and at regular intervals. This is because the WebSocket connection will feed in information on earthquakes as well, so there is no need to call this API every second.

```
{
    "items": [
        {
            "id": 1584,
            "type": "normal",
            "eventId": "20210808085414",
            "originTime": "2021-08-08T08:54:00+09:00",
            "arrivalTime": "2021-08-08T08:54:00+09:00",
            "hypocenter": {
                "code": "787",
                "name": " 鹿児島湾",
                "coordinate": {
                    "latitude": {
                        "text": "31.3°N",
                        "value": "31.3000"
                    },
                    "longitude": {
                        "text": "130.6°E",
                        "value": "130.6000"
                    },
                    "height": {
                        "type": " 高さ",
                        "unit": "m",
                        "value": "0"
                    },
                    "geodeticSystem": " 日本測地系"
                },
                "depth": {
                    "type": " 深さ",
                    "unit": "km",
                    "value": "0",
                    "condition": " ごく浅い"
                }
            },
            "magnitude": {
                "type": " マグニチュード",
                "unit": "Mj",
                "value": "2.6"
            },
            "maxInt": "2"
        }
    ]
}
```


Listing 2.10: Past earthquake list sample response JSON.

WebSocket Connections WebSocket connections are essential to real-time applications, since it provides a real-time feed from the server to the client on latest data, without the need of the client to query (which consumes a lot of resources). DM-D.S.S. provides WebSockets to initiate real-time connection to the server to achieve the latest telegrams and earthquake warnings.

The URL of the WebSocket will be `wss://{#1}.api.dmdata.jp/v2/websocket`, where the `#1` part could be `ws001` to `ws004` (which will be assigned in the `socket.start` API call).

WebSocket Responses There are four types of responses that a WebSocket could return, indicated in the `type` attribute:

- a `start`. This only happens when the WebSocket is started, and this includes the exact same

information in the `socket.start` call in Listing 2.6, with an additional `time` to indicate the time of the response;

- a (server-side initiated) `ping`. This will happen regularly, and the server will send a JSON in the format in Listing 2.11.

```
{
    "type": "ping",
    "pingId": "012345"
}
```

Listing 2.11: WebSocket Ping JSON.

Upon receiving, the client should return a `pong` JSON in the same format, with the same `pingId`, as in Listing 2.12.

```
{
    "type": "pong",
    "pingId": "012345"
}
```

Listing 2.12: WebSocket Pong JSON.

This is in place to ensure that there are no 'dead' WebSocket connections, and the server will close the WebSocket connection after 120s of no response in such ping-pong calls, to not take up the limited number of WebSocket connection slots;

- a client-side initiated `ping` as in Listing 2.11, which the server will return a `pong` as in Listing 2.12. This works almost identically as before, apart from that such `pingId` is optional. This is in place to ensure that the application would be able to know if the connection is broken, for example due to unstable internet connections, and would be able to restart another WebSocket connection;
- a `error`, which will be the format in Listing 2.13. There will also be a boolean, `close`, to indicate whether the WebSocket connection is closed or not.

```
{
    "type": "error",
    "error": "Server error.",
    "code": 4503,
    "close": true
}
```

Listing 2.13: WebSocket Error JSON.

A table of possible WebSocket errors is included in 2.9, with the relevant WebSocket error codes.

The program should be designed to be able to handle these errors, and restart the connection if necessary;

- a `data`, an example shown in Listing 2.14. This will include a `classification`, a version `version`, and a unique `id`. An array named `passing` gives the `name` and `time` of each passing location of this data piece inside the server. Furthermore, the `head` gives some key information, and `xmlReport` gives certain details in the XML body. `format`, `compression` (`gzip`, `zip` or `null`) and `encoding` (`base64` or `utf-8`) gives information on how the `body` should be processed and interpreted.

  This gives the key piece of information that will be passed between the data component of the software and the UI component of the software;

Error Code	Error Details
4400	Compulsory parameters are not specified when starting WebSocket.
4404	Ticket is not specified when starting WebSocket.
4409	Number of connection limit is reached.
4503	Internal server error.
4640	`pingId` does not match for a server-initiated `ping`.
4641	Incorrect data (non-JSON) received from client.
4808	Client requested closure of WebSocket.
4807	Contracts are discontinued in connection.

Table 2.9: Errors for WebSocket connection.

JSON v.s. XML Lots of data are serialised in JSON, but the JMA provides its data in XML format.DM-D.S.S. does also provide JSON formatted schemas for most of the XML data files which are also processed real time. However, there is a high chance of delay (up to 1 second) if the JSON format is used, not to say the potential of an error in processing which lead to undefined behaviour in the JSON format. Therefore, to control the delay and ensure that data is real-time, the WebSockets in this application will use the original format in XML and parse internally in the program.

XML Data Sources

Sidenotes It is worth noting that an existing NuGet Library, ingen084/DmdataSharp supports dealing with the DM-D.S.S. data flow and converting them to C# objects (and exceptions). However, for the purposes of this NEA, we will implement our own way to interact with the APIs and the correlated C# DTOs. The developer of this library did also mention that it is quite purpose-built so might not be suitable for general use.

This section referred to the official document for DM-D.S.S.

2.1.2 OOP Model

The data achieved by the previous two data sources will be parsed into objects (classes, records) to transfer between different components of the program. Those objects will then be transferred to the GUI component and be displayed.

The Class Diagram following the UML convention for the HTTP-based API calls is shown in Figure 2.15

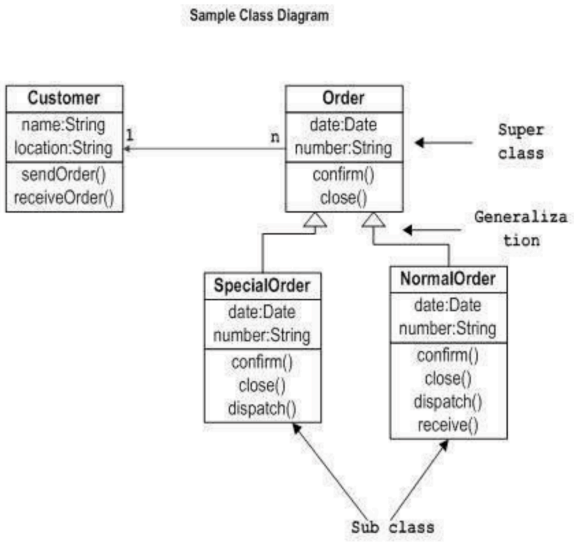

Figure 2.15: Class Diagram.

```
{
    "type": "data",
    "version": "2.0",
    "classification": "telegram.weather",
    "id": "123456789abcdef...",
    "passing": [
        {
            "name": "websocket-01",
            "time": "2020-01-01T00:00:00.120Z"
        }
    ],
    "head": {
        "type": "VPWW54",
        "author": "JPTD",
        "time": "2020-01-01T00:00:00.000Z",
        "test": false,
        "xml": true
    },
    "xmlReport": {
        "control": {
            "title": " 気象警報・注意報（Ｈ２７）",
            "dateTime": "2020-02-27T00:00:00Z",
            "status": " 通常",
            "editorialOffice": " 気象庁本庁",
            "publishingOffice": " 気象庁予報部"
        },
        "head": {
            "title": " 東京都気象警報・注意報",
            "reportDateTime": "2020-02-27T09:00:00+09:00",
            "targetDateTime": "2020-02-27T09:00:00+09:00",
            "eventId": null,
            "serial": null,
            "infoType": " 発表",
            "infoKind": " 気象警報・注意報",
            "infoKindVersion": "1.2_1",
            "headline": " 注意報を解除します。"
        }
    },
    "format": "xml",
    "compression": "gzip",
    "encoding": "base64",
    "body": "H4sIAAAAAAAAA..."
}
```


Listing 2.14: WebSocket Data JSON.

2.1.3 Algorithms

There are two key 'algorithms' that this application will use: the first one used to calculate the position of the P/S wavefronts, and the other one used to detect shake on the real-time monitoring screen.

2.1.3.1 Wavefront Calculation

Earthquake waves do not travel at uniform rate as they spread out due to the decrease in energy, and so there is no uniform velocity that the seismic waves travel in. We cannot do a simple equation like $s = vt$ to calculate the distance of the wavefront of a seismic wave from the epicentre.

However, JMA provides tables for the time that seismic waves take to travel a certain distance in

their data bank, together with the format definition. It will be possible for us to look up the table and plot the seismic waves at correct distance at a certain time.

Comparing the different wave travel tables that JMA provides, the most suitable one for this application will be the **JMA 2001**, since it is designed for general purpose uses for both land-based and ocean-based earthquakes, and does not take into account the effect of the altitude of the observation points. Listing 2.15 provides a preview of the first 10 lines of this file.

```
P␣␣␣␣0.000␣S␣␣␣␣0.000␣␣␣0␣␣␣␣␣␣0
P␣␣␣␣0.416␣S␣␣␣␣0.703␣␣␣0␣␣␣␣␣␣2
P␣␣␣␣0.831␣S␣␣␣␣1.403␣␣␣0␣␣␣␣␣␣4
P␣␣␣␣1.243␣S␣␣␣␣2.099␣␣␣0␣␣␣␣␣␣6
P␣␣␣␣1.650␣S␣␣␣␣2.786␣␣␣0␣␣␣␣␣␣8
P␣␣␣␣2.052␣S␣␣␣␣3.465␣␣␣0␣␣␣␣␣10
P␣␣␣␣2.448␣S␣␣␣␣4.135␣␣␣0␣␣␣␣␣12
P␣␣␣␣2.838␣S␣␣␣␣4.794␣␣␣0␣␣␣␣␣14
P␣␣␣␣3.221␣S␣␣␣␣5.444␣␣␣0␣␣␣␣␣16
P␣␣␣␣3.600␣S␣␣␣␣6.084␣␣␣0␣␣␣␣␣18
```

Listing 2.15: JMA 2001 Wave Travel Tables.

The table is a lookup table from horizontal distance to time in terms of the depth of the epicentre and the type of wave (P or S) to the time needed to travel such distance. Each row represents the P-Wave, time taken for the P-wave to travel (in seconds), S-wave, time taken for the S-wave to travel (in seconds), depth of epicentre (in kilometres), distance from epicentres (in kilometres).

The depth and the distance from epicentre is provided in the intervals specified in 2.10. They are provided for distances up to 2000 kilometres and depths up to 700 kilometres.

Distance	Interval for Depth	Interval for distance
0 ~ 50	2	2
50 ~ 200	5	5
200 ~ 700	10	10
700 ~ 2000	–	10

Table 2.10: Distance intervals in JMA 2001.

However, this does not provide us with the distance we would like, which is drawing a circle based on how long it has been since the earthquake has happened – the other way around. Luckily, for a fixed depth, the list is sorted since the distance is always increasing, so is the time. We can apply a binary search on the list given a depth (by first pinpointing the start and the end rows for that depth, and this is trivial since there is a fixed number of rows for each depth), and find the neighbouring two times such that the current time elapsed from the earthquake happening is between them. After doing this, a linear interpolation will be done on the neighbouring time-distance (t, d) pairs to find the accurate distance (radius) corresponding with the time and the depth.

This file will be read by the program upon launch every time and stored in the active memory when the program is running.

2.1.3.2 Shake Detection

As discussed in the analysis section, one of the common features of applications with real-time intensity display map is the shake detecting feature. Specifically, if the measured intensities within a certain region shows an increasing tendency, the application will display a box around that area to indicate there is a chance that the earthquake will happen.

However, there is a chance that certain observation points will malfunction and suddenly show a big increase in intensity, which does not indicate an earthquake, and we would like to prevent this from happening. Furthermore, we wish to be able to distinguish between a big earthquake and a small earthquake on the display when such shake event has occurred.

It is worth thinking how human would see an earthquake from the real-time intensity map. First, we would notice a certain observation point has an increase in the intensity for a certain observation

point. Then, we will naturally look at the observation points near it, and see whether they also have an increase, or if that was just an anomaly. If the points nearby also has increasing intensity, then we would think it is probably an event, and we will look at the observation points near that, etc. If after a certain time period, that the intensity is no longer increasing, then we may say that the shake event is over.

The algorithm used in KEVI used a similar idea, and this is the algorithm that we are going to implement in our solution. Specifically, it will create a list of observation points that are near each of the observation points (we will refer to them as neighbouring points) upon launch of the program. When there is a significant increase of the measured value of an observation point, the program will look at the neighbouring points, and if they show an increase (not necessarily significant), this will create a new event including the observation point and the neighbouring points. If, further, that the neighbouring points also observe a significant increase in intensity, then they will be added to the event, and their neighbouring points will be added to the list of neighbouring points as well.

An event could be represented by three things: the time it has happened, the observation point in the event, and the neighbouring points of those points. The neighbouring points will become observation points themselves if they have significant increase in intensity, and then their neighbouring points will become some new neighbouring points. If no such increase has been found, then after a certain time period of waiting, this event would end, and it could be seen that the primary motion of the earthquake is over.

Classification of the significance of an event is straightforward – it can be determined by the maximum intensity observed within the event. A suitable scale would be split at intensity 1 (which is which human can feel a shake), and at intensity 3 (which is when earthquake might actually cause damage), and it has to have minimum measured intensity 0.0. Most real-time measurements have negative measured intensity, so this should not lead to detection of anomalies.

Figure 2.16 shows the concept of an observation point and neighbouring points together with a flowchart of how the algorithm operates. The time constants of 10 seconds and 30 seconds are chosen to reflect the behaviour of common applications including JQuake and KEVI, but should be left to be adjustable in the code to ensure the maintainability.

This section referred to this blog article written by Ingen.

2.2 User Interface

The nature of this application decided that the UI will be a graphical interface. Comparing MAUI in .NET with Avalonia, due to the fact that MAUI is more suitable for a mobile touch-based platform (that also works on desktop), while Avalonia is designed to support the desktop paradigm, the latter will be used to implement the graphical interface.

As discussed in the previous sections, there are three main pages that the program will include: the page for real-time observations, the page for viewing past earthquakes, and the page for controls and settings.

A sidebar is used to switch between the three pages.

Note that the following mock-up images are for demonstration purposes only – it is not an actual earthquake.

2.2.1 Real-Time Monitoring Screen

The real-time monitoring screen holds the following three functionalities:

- real-time intensity display (including shake detection boxes);
- tsunami warning display; and
- EEW display (including predicted intensity colouring and real-time wavefront positions).

Figure 2.17 shows a design of this page that will be implemented. The real-time intensity colour points, the current time and the colour scale will be displayed at all times.

The shake detection boxes will only be displayed when there is shake detected as described in the algorithm section, and would be coloured into red, yellow or green in accordance to the intensity of the event.

The tsunami warning will be displayed only if it is released (or updated). It will consistently flash at 0.5 Hz to warn the users and to signify its presence.

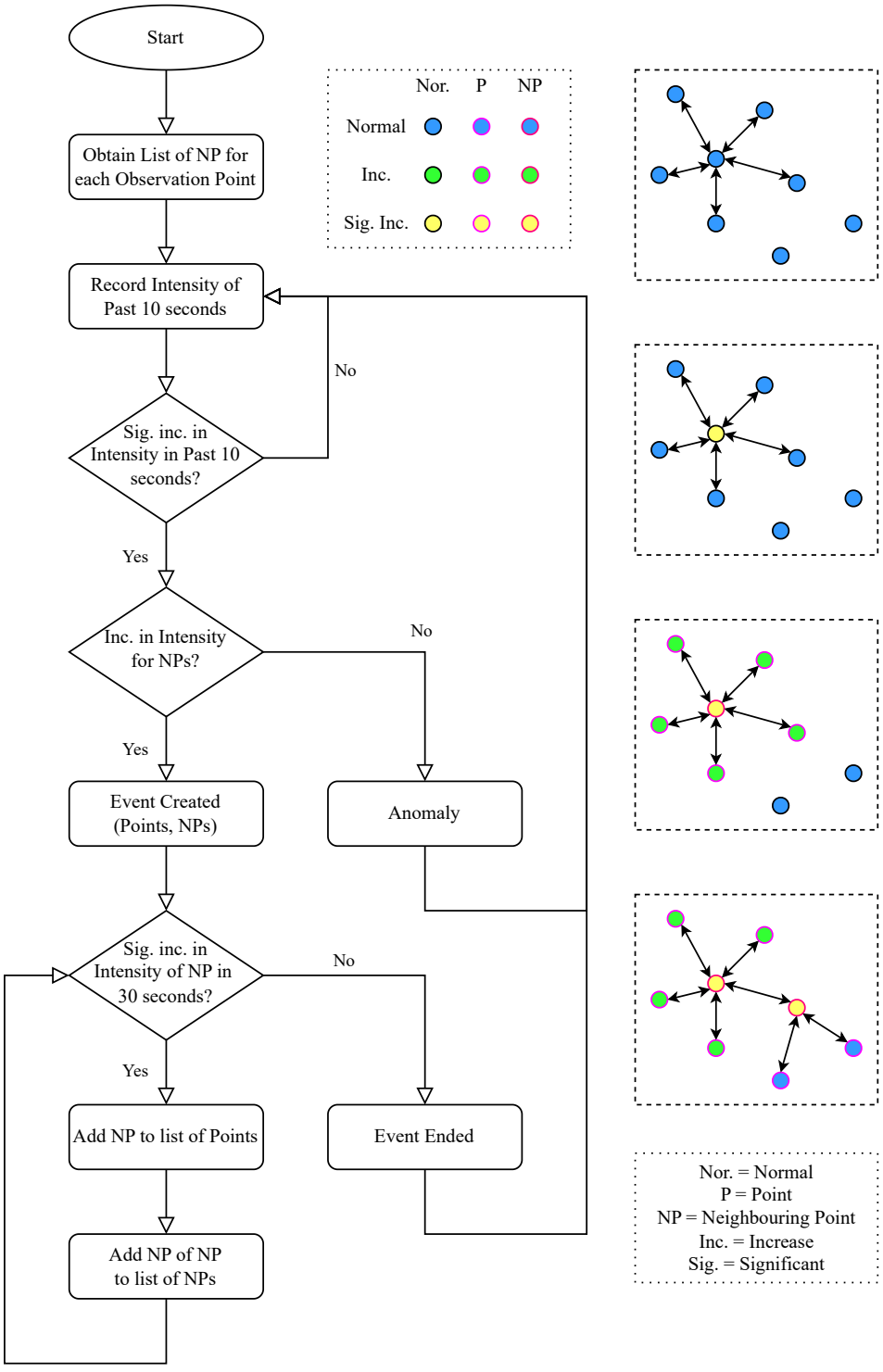

Figure 2.16: Shake detection flowchart and diagram.

The EEW warning display will only be present when an EEW is ongoing, until 1 minute after the final EEW warning. The top-left box will display key properties of the earthquake (including the predicted maximum intensity in the corresponding colour), with relatively smaller text displaying the details, such as the shake detection algorithm used (e.g. IPF/PLUM etc.). The map will be coloured in accordance to the predicted intensities released in the EEW.

There are no buttons or interactions for the user to interact with. However, this page will have a relatively high refresh rate to ensure that the data displayed is up-to-date.

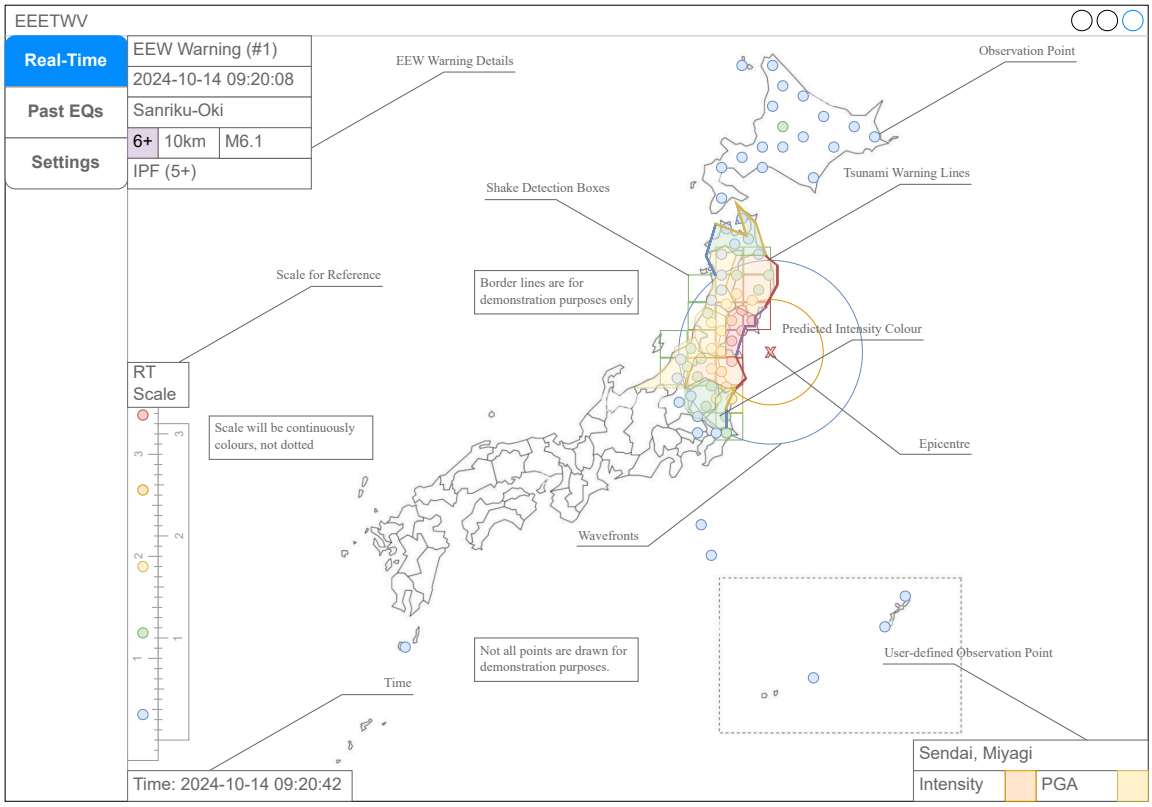

Figure 2.17: Design of GUI for Real-Time Page.

2.2.2 Past Earthquakes

The past-earthquake screen regularly displays a sidebar to select the earthquake from, with by default the latest earthquake selected, with a map in the middle. Figure 2.18 shows a design of this page.

For each earthquake, the sidebar should display the time of the earthquake, the epicentre of the earthquake (position and depth), the maximum intensity of the earthquake (in the correct corresponding colour), and its magnitude.

In the centre, there should be a map colouring the observed intensity of the earthquake, together with a cross to indicate the position of the epicentre.

On the left, there should be the details displayed in a bigger font, and an expandable sidebar to see the intensity of each observation point, in a hierarchy according to the province and the county. There should be two buttons, one corresponding to the JMA website to view the details, and another to jump to a weather service provider to view the details of the earthquake.

2.2.3 Customisation Page

In the customisation page, there should be a place to input an API Key from DM-D.S.S., and in the future this should be upgraded to a button to provide a login using OAuth 2 and the default browser.

There should also be a page to customise sound files for different events, a page to customise the intensity colour scheme, and to define a key point on the map to observe the intensity.

Figure 2.19 shows a mock-up of the design of the customisation page.

2.2.4 Joint Functionalities and Technicalities

There is a joint functionality between the real-time page and the past earthquakes page, that is, when a new EEW/Tsunami Warning is released, the application should automatically jump to the real-time page to display the latest earthquake information to the user.

Since this a GUI-based application, there will be a significant use of asynchronous programming and events. Specifically, EEWs should be treated as events for the GUI application to handle and display,

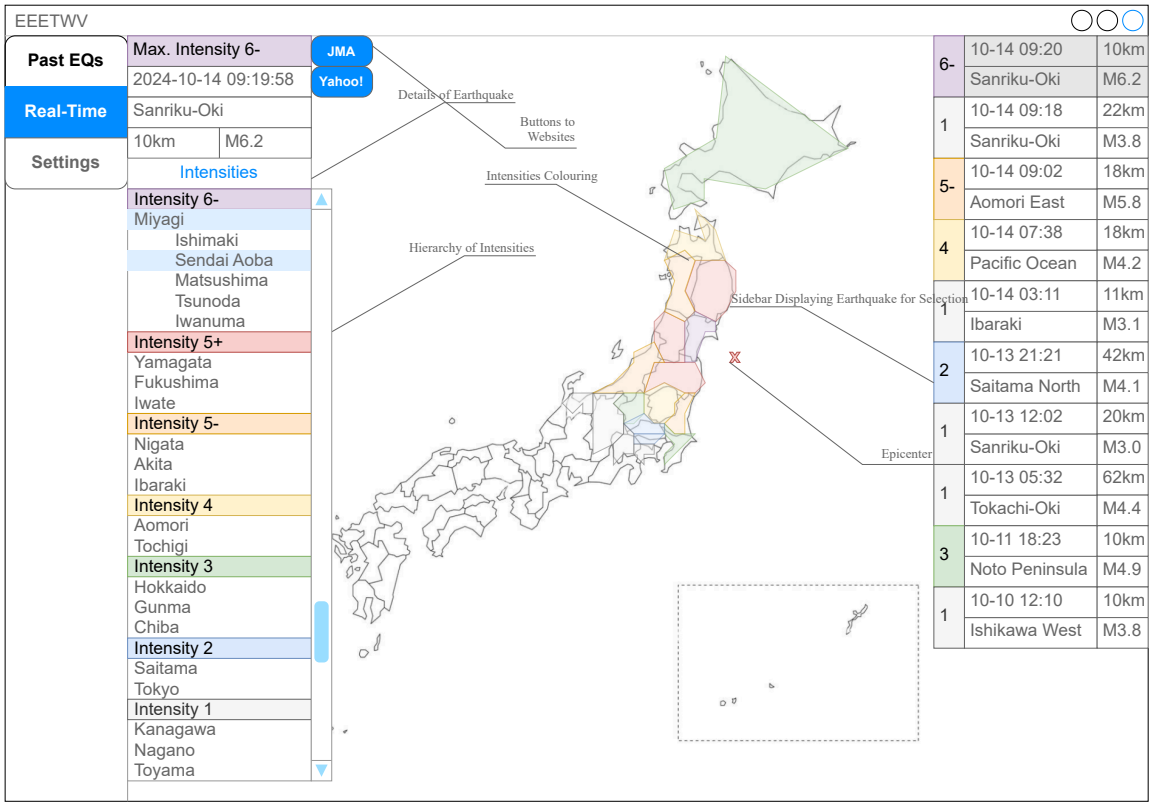

Figure 2.18: Design of GUI for Past Earthquakes Page.

and user inputs are also events that the program should respond to.

2.3 Hierarchy Chart

As discussed in the analysis section, the program consists of three parts: data-parsing from external data sources, GUI functionalities and joint functionalities, where the GUI part will be divided into two parts focusing on real-time monitoring and past-earthquake information, respectively. A detailed discussion into how different modules can be further split up while at the same time being interacted was discussed, and Figure 2.20 is a hierarchy diagram for the whole application. This shows how **decomposition** technique is applied to reduce a sophisticated problem into more attackable problems.

The whole application (apart from the polynomial fitting part which was done in Python) will be written in C# using the .NET Core Version 8.0, since the .NET Core is designed oriented around OOP techniques.

The built-in .NET parsers for XML and JSON will be used to convert into DTOs.

The GUI will be designed using Avalonia, which is designed to develop cross-platform applications in .NET.

2.4 Hardware & Software Requirements

Table 2.11 outlines basic hardware requirements for the application.

The amount of RAM is due to the amount of data that is processed (and considering other applications running as well). Storage space is not a substantial requirement of this application, while the CPU has to be of high standards to process all the data. To display the application properly (with appropriate size), a display of 1080p 12” is recommended.

It will be able to run on up-to-date Windows, macOS and Linux distributions (both x64 and ARM) due to the cross-platform nature of .NET, but x86 platforms will not be supported.

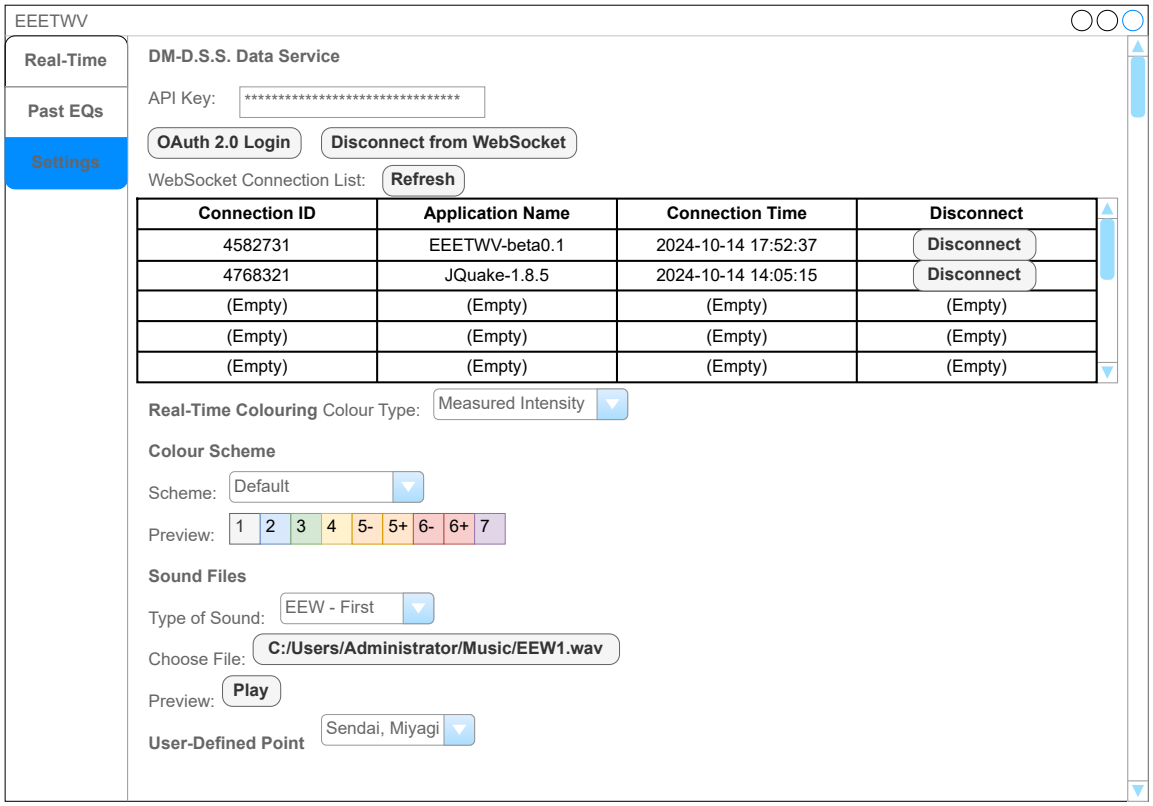

Figure 2.19: Design of GUI for Customisation Page.

Type	Minimum	Recommended
RAM	4 GB	8 GB
Storage Space	1 GB	2 GB
CPU	Intel Core 8th Gen (or equiv.)	Intel Core 10th Gen (or equiv.)
Display Resolution	720p	1080p
Display Size	9”	12”

Table 2.11: Hardware Specification for App

The author uses a macOS 15 (beta) machine with 2.5K 13” display, Intel Core i5-1038NG7 (with Intel Iris Plus Graphics) and 16 GB of RAM (MacBook Pro 2020, 4 Thunderbolt Ports) and a Windows 11 (beta) machine with 2.5K 15” display, AMD Ryzen 7 5800H and 32 GB of RAM (with RTX 3070 for Laptop) (Legion R9000K 2021) to test the application, and with 2.5K 24” external display as well. An Ubuntu 22.04 virtual machine will be used to test the compatibility for Linux systems.

The application will be self-contained (i.e. comes with .NET runtime) to prevent the user from unnecessary technical complications.

The device needs to have stable connection to the internet using Wi-Fi/Ethernet/other means, to establish connection with the relevant APIs.

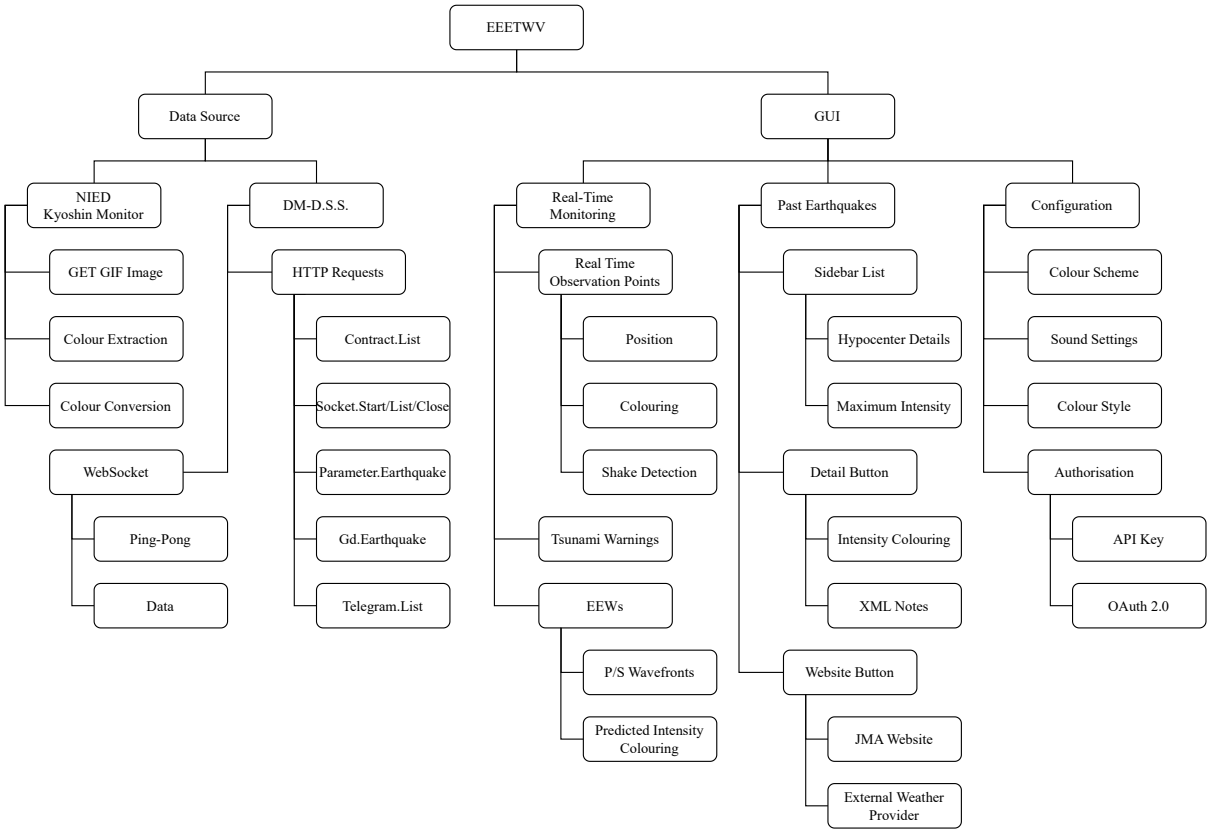

Figure 2.20: Hierarchy chart of the whole application

Chapter 3

Technical Implementation

3.1 Key Code Segments

3.1.1 Data structures

Implementation of ADTs and OOP Classes to be demonstrated.

3.1.2 Modularity

Code should be created and tested in separate modules that are integrated later. Use subheadings for each module, define the purpose of the module, and show unit testing of the module.

3.1.3 Defensive Programming/Robustness

Exception handling

Chapter 4

Testing

Consider how you will test your project. You should devise a test strategy that encompasses a range of methods.

4.1 Test Strategy

- Unit testing (of individual functions)
- Integration testing (e.g. different modules/class files)
- Robustness (demonstrating defensive programming skills/exception handling)
- Requirements testing (against your initial requirements - a table with test number, description, test data, expected result, evidence (screenshot/video time link) would be suitable)
- Independent end user beta testing (this will assist with your evaluation)

4.2 Testing Video

- You can include a video to assist (but you will need to reference the time point at which relevant evidence appears)
- If you include a video you will need to have it publicly available.
- It is suggested that you include a QR code in your testing to give a link to it the video (for the moderator) rather than just giving a long URL on its own.

4.3 System Tests (against original requirements' specification)

You need to give evidence in support of requirements that have been met e.g. reference to a relevant test/screenshot/relevant code.

Requirement №	Description	Success Criteria	Tests + Evidence

Table 4.1: Table of Tests.

Chapter 5

Evaluation

5.1 Requirements Specification Evaluation

Personal evaluation

- Copy and paste your original requirements from your project analysis
- You need to review each requirement and comment objectively on whether it was *fully met/partially met/not met*.

Requirement №	Description	Success Criteria	Fully/Partial/Not met (Reflective Comment)

Table 5.1: Table of Evaluation.

5.2 Independent End-User Feedback

End user/client evaluation

- there **must** be meaningful end user feedback
- You should hold a review meeting with your end user
- Write down any key feedback that they give you. E.g. Agreement that a particular requirement has been meet/comments as to aspects that they find suboptimal/comments as to additions they would like to see

Requirement №	Description	Acceptance Y/N	Additional Comments

Table 5.2: Table of Feedback.

5.3 Improvements

You need to give consideration to a number of potential future improvements that could be made. They may arise from either your experience or from feedback given to you by your end user. Ideally at least one should be in response to end user feedback.

- Write a paragraph for each potential improvement/change
- The improvements/changes could result from additional functionality that has been identified as being beneficial or could be as a result of required efficiencies if some processes are clunky or require faster run-times
- You should then comment on how the proposed change could be implemented moving forward. i.e. what would need to be changed/developed and how? You are not expected to actually make any changes; just comment on the possibilities.

Appendix A

Code Listing

```
# %% [markdown]
# # Polynomial fitting for $f : [0, 1] \to \mathcal{C}$

# %%
from PIL import Image, PyAccess # for image reading
import matplotlib.pyplot as plt # to plot graphs
from scipy import optimize # to do fitting

plt.rcParams['text.usetex'] = True # LaTeX in plots
plt.rcParams['mathtext.fontset'] = 'stix'
plt.rcParams['font.family'] = 'STIXGeneral'

# %% [markdown]
# ## Plot $C = (H, S, V)$

# %%
orig_img_path: str = 'jma-scale.png'
orig_img: Image.Image = Image.open(orig_img_path) # read original image

orig_img_hsv: Image.Image = orig_img.convert('HSV') # convert to HSV colour format

img_width: int
img_height: int
img_width, img_height = orig_img.size # get size of image

img_width, img_height

# %%
# read the (H, S, V) in a fixed column

column: int = 5
hsv_list: list[tuple[int, int, int]] = [(orig_img_hsv.getpixel((column, y))) for y
↪  in range(img_height)] # type: ignore

hsv_list

# %% [markdown]
# ### Plot $f_H, f_S, f_V$ against $r$

# %%
r_list: list[int] = list(range(img_height))

# split up list into separate lists
```

```
hsv_h_list: list[int] = [c[0] for c in hsv_list]
hsv_s_list: list[int] = [c[1] for c in hsv_list]
hsv_v_list: list[int] = [c[2] for c in hsv_list]

# %%
# setup options for plotting graphs

marker_size: int = 15
marker_char: str = 'x'
line_width: float = 0.3

fig_size: tuple[int, int] = (10, 5)

h_colour: str = 'turquoise'
s_colour: str = 'gold'
v_colour: str = 'gray'

h_label: str = 'Hue $H$'
s_label: str = 'Saturation $S$'
v_label: str = 'Value $V$'

# %%
def init_plt(xlabel: str, ylabel: str, title: str) -> None:
    plt.figure(figsize = fig_size)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.grid(True)
    plt.tight_layout()

# %%
# plot graph of (H, S, V) against r

init_plt('Row Number $r$', '$(H, S, V)$ Values', f'$(H, S, V)$ Values against Row
↪  Number $r$ along column {column}')

plt.scatter(r_list, hsv_h_list, label=h_label, color=h_colour, s=marker_size,
↪  marker=marker_char, linewidths=line_width)
plt.scatter(r_list, hsv_s_list, label=s_label, color=s_colour, s=marker_size,
↪  marker=marker_char, linewidths=line_width)
plt.scatter(r_list, hsv_v_list, label=v_label, color=v_colour, s=marker_size,
↪  marker=marker_char, linewidths=line_width)

plt.legend()

plt.savefig('hsv-against-row.png')

# %%
plt.close()

# %% [markdown]
# ### Plot $f_H, f_S, f_V$ against $h$

# %%
# setup the initial values for h and r

h_0 = 18
```

```
h_1 = 296 + 1
r_len = h_1 - h_0

def r_to_h(r: int) -> float:
    return 1 - ((r - h_0) / (r_len - 1))

h_list: list[float] = [r_to_h(r) for r in r_list] # convert list of r to list of h

h_r_list: list[float] = h_list[h_0:h_1] # extract the values of h within range

hsv_h_r_list: list[int] = hsv_h_list[h_0:h_1] # extract the values of (H, S, V)
↪  within range
hsv_s_r_list: list[int] = hsv_s_list[h_0:h_1]
hsv_v_r_list: list[int] = hsv_v_list[h_0:h_1]

h_r_list

# %%
# plot graph of (H, S, V) against h

init_plt('$h$', '$(H, S, V)$ Values', f'$(H, S, V)$ Values against $h$ along column
↪  {column}')

plt.scatter(h_r_list, hsv_h_r_list, label=h_label, color=h_colour, s=marker_size,
↪  marker=marker_char, linewidth=line_width)
plt.scatter(h_r_list, hsv_s_r_list, label=s_label, color=s_colour, s=marker_size,
↪  marker=marker_char, linewidth=line_width)
plt.scatter(h_r_list, hsv_v_r_list, label=v_label, color=v_colour, s=marker_size,
↪  marker=marker_char, linewidth=line_width)

plt.legend()

plt.savefig('hsv-against-h.png')

# %%
plt.close()

# %% [markdown]
# ## Regression

# %% [markdown]
# ### Prepare Lists to do Regression

# %%
# normalise values of h, s, v to our desired values

hsv_h_n_list: list[float] = [hsv_h_r_list[i] / 255 * 360 for i in range(279)]
hsv_s_n_list: list[float] = [hsv_s_r_list[i] / 255 * 1 for i in range(279)]
hsv_v_n_list: list[float] = [hsv_v_r_list[i] / 255 * 1 for i in range(279)]

# %% [markdown]
# ### Regression on $f_H$

# %%
# define f_H with unkwnown parameters y_1, y_2 which operates on a list

def f_h_params(xl: list[float], y_1: float, y_2: float) -> list[float]:
    yl: list[float] = [0 for _ in xl]
```

```
    for i, x in enumerate(xl):
        if 0 <= x <= 0.1:
            yl[i] =  -150 * x + 237
        elif 0.1 <= x <= 0.6:
            yl[i] = (
                ((222  * (x - 0.3) * (x - 0.4) * (x - 0.6)) / ((0.1 - 0.3) * (0.1 -
                ↪  0.4) * (0.1 - 0.6))) +
                ((y_1 * (x - 0.1) * (x - 0.4) * (x - 0.6)) / ((0.3 - 0.1) * (0.3 -
                ↪  0.4) * (0.3 - 0.6))) +
                ((y_2 * (x - 0.1) * (x - 0.3) * (x - 0.6)) / ((0.4 - 0.1) * (0.4 -
                ↪  0.3) * (0.4 - 0.6))) +
                ((51   * (x - 0.1) * (x - 0.3) * (x - 0.4)) / ((0.6 - 0.1) * (0.6 -
                ↪  0.3) * (0.6 - 0.4)))
            )
        elif 0.6 <= x <= 0.9:
            yl[i] = -170 * x + 153
        elif 0.9 <= x <= 1:
            yl[i] = 0
        else:
            yl[i] = 0
    return yl

# %%
h_init: list[int] = [0, 0] # initial guess to indicate number of paameters

h_params: list[int]

h_params, _ = optimize.curve_fit(f_h_params, h_r_list, hsv_h_n_list, p0=h_init) #
↪  optimised parameters

h_params

# %%
def f_h(x) -> list[float]: # this is a partial function which gives in the fitted
↪  parameters
    return f_h_params(x, *h_params)

h_p_list: list[float] = [i / 1000 for i in range(0, 1001, 1)] # list to make sure
↪  the plotted graph looks quite smooth
hsv_h_fitted_list: list[float] = f_h(h_p_list) # fitted line

# %%
line_width = 1.25 # make the markers look a bit thicker

# %%
# plot result of fit

init_plt('$h$', h_label, '$H$ against $h$')

plt.scatter(h_r_list, hsv_h_n_list, label=h_label, color=h_colour, s=marker_size,
↪  marker=marker_char, linewidth=line_width)
plt.plot(h_p_list, hsv_h_fitted_list, label='Fit')

plt.legend()
plt.savefig('h-against-h.png')

# %%
plt.close()
```

```
# %% [markdown]
# ### Regression on $f_S$

# %%
# define f_S

def f_s(xl: list[float]) -> list[float]:
    yl: list[float] = [0 for _ in xl]
    for i, x in enumerate(xl):
        if 0 <= x <= 0.2:
            yl[i] = 1
        elif 0.2 <= x <= 0.29:
            yl[i] = -2.611 * x + 1.522
        elif 0.29 <= x <= 0.4:
            yl[i] = 1.682 * x + 0.277
        elif 0.4 <= x <= 0.5:
            yl[i] = 0.5 * x + 0.75
        elif 0.5 <= x <= 1:
            yl[i] = 1
        else:
            yl[i] = 0
    return yl

hsv_s_fitted_list: list[float] = f_s(h_p_list) # fitted line

# %%
# plot result of fit

init_plt('$h$', s_label, '$S$ against $h$')

plt.scatter(h_r_list, hsv_s_n_list, label=s_label, color=s_colour, s=marker_size,
↪  marker=marker_char, linewidth=line_width)
plt.plot(h_p_list, hsv_s_fitted_list, label='Fit')

plt.legend()
plt.savefig('s-against-h.png')

# %% [markdown]
# ### Regression on $f_V$

# %%
# define f_V

def f_v(xl: list[float]) -> list[float]:
    yl: list[float] = [0 for _ in xl]
    for i, x in enumerate(xl):
        if 0 <= x <= 0.1:
            yl[i] = 1.8 * x + 0.8
        elif 0.1 <= x <= 0.172:
            yl[i] = -4.444 * x + 1.424
        elif 0.172 <= x <= 0.2:
            yl[i] = 5.714 * x - 0.323
        elif 0.2 <= x <= 0.3:
            yl[i] = 1.6 * x + 0.5
        elif 0.3 <= x <= 0.4:
            yl[i] = 0.2 * x + 0.92
        elif 0.4 <= x <= 0.8:
```

```
            yl[i] = 1
        elif 0.8 <= x <= 0.9:
            yl[i] = -0.3 * x + 1.24
        elif 0.9 <= x <= 1:
            yl[i] = -2.9 * x + 3.58
        else:
            yl[i] = 0
    return yl

hsv_v_fitted_list: list[float] = f_v(h_p_list) # fitted line

# %%
# plot result of fit

init_plt('$h$', v_label, '$V$ against $h$')

plt.scatter(h_r_list[1:r_len - 1], hsv_v_n_list[1:r_len - 1], label=v_label,
↪  color=v_colour, s=marker_size, marker=marker_char, linewidth=line_width) # the
↪  first and last points are excluded from the plot
plt.plot(h_p_list, hsv_v_fitted_list, label='Fit')

plt.legend()
plt.savefig('v-against-h.png')

# %% [markdown]
# ## Back-Generate the image

# %%
# setup an HSV formatted image and get its pixels

image_generated: Image.Image = Image.new('HSV', (img_width, img_height))
generated_pixels: PyAccess.PyAccess = image_generated.load() # type: ignore

# returns a list of tuples of (H, S, V) for a column

def f(yl: list[int]) -> list[tuple[int, int, int]]:
    hl: list[float] = [r_to_h(y) for y in yl]

    hsv_h: list[float] = f_h(hl)
    hsv_s: list[float] = f_s(hl)
    hsv_v: list[float] = f_v(hl)

    hsv_h_i: list[int] = [int(h / 360 * 255) for h in hsv_h]
    hsv_s_i: list[int] = [int(s * 255) for s in hsv_s]
    hsv_v_i: list[int] = [int(v * 255) for v in hsv_v]

    return [(hsv_h_i[i], hsv_s_i[i], hsv_v_i[i]) for i in yl]

my_pixels: list[list[tuple[int, int, int]]] = [f(list(range(img_height))) for _ in
↪  range(img_width)]

# copy this into the generated_pixels reference

for x in range(img_width):
    for y in range(img_height):
        generated_pixels[x, y] = my_pixels[x][y]

# convert to RGB and output
```

```
image_generated_rgb: Image.Image = image_generated.convert('RGB')
image_generated_rgb.save('generated-colour.png')
```

Listing A.1: Code for Polynomial Fit of Colour