Aftershocks

Benjamin Mason

Earthquakes are caused by the sudden release of energy initiated at a rupture below the surface. After an initial earthquake, the *mainshock*, the region surrounding the initial rupture might be unstable, causing secondary earthquakes, the *aftershocks*. We will study a dataset of earthquakes, and model the probability of aftershocks based on quantities such as the distance to the mainshock rupture.

Question 1

We have several tables with information about earthquakes. `all_events.csv` contains the `date`, location (latitude `lat` and longitude `lon`), identifier `id`, intensity `mw` and seismic moment `moment` of many earthquakes. The tables in the folder `aftershocks/` contain the mechanical stresses `s1,...,s6` at different locations surrounding a mainshock, and a column indicating if an aftershock was identified at that location (0 if aftershock was not recorded, 1 otherwise). The table `selectedEvents.csv` contains a list of identifiers `id` and a list of the files with the corresponding aftershock tables.

(a) Create a new dataframe with four columns: `date`, `file`, `lat`, `lon`, `mw`, `aftershocks` with a row for each of the selected events, containing the date (from `all_events.csv`), the file containing the aftershock information (from `selectedEvents.csv`), the location of the mainshock, the intensity and the total number of aftershocks. Make sure the new dataframe is sorted by date, and display the first few rows using `head`.

```python
import pandas as pd
import numpy as np

# DataFrame transcripts of resp. files
all_events = pd.read_csv("all_events.csv", index_col="id")
selected_events = pd.read_csv("selectedEvents.csv", index_col="id")

# Each extract_info is respective to the criteria of the question see Q1c and
 ↪  Q2b
```

```
def extract_info(row_id):
    specific_event = all_events.loc[row_id] # Locates a selected event in
↪   all_events
    aftershock_data = pd.read_csv("aftershocks/" + specific_event.name +
↪   "_grid.csv")
    aftershocks_count = aftershock_data["aftershock"].sum() # Sums the
↪   aftershock column
    return {
        "date": specific_event["date"],
        "file": selected_events.loc[row_id, "file"],
        "lat": specific_event["lat"],
        "lon": specific_event["lon"],
        "mw": specific_event["mw"],
        "aftershocks": aftershocks_count
    }

# Applies extract_info to each row
data = list(map(extract_info, selected_events.index))

# Convert the list of dictionaries to a DataFrame
df = pd.DataFrame(data)
df["date"] = pd.to_datetime(df["date"], format="%m/%d/%Y") # Converts the
↪   date into a datetime type
# Sort the DataFrame by date
sorted_df = df.sort_values("date")
sorted_df.head()
```

	date	file	lat	lon	mw	aftershocks
2	1989-10-18	1989LOMAPR01WALD_grid.csv	37.0410	-121.8830	6.94	79.0
5	1994-01-17	1994NORTHR01WALD_grid.csv	34.2130	-118.5370	6.80	76.0
9	1997-05-10	1997ZIRKUH01SUDH_grid.csv	33.8200	59.8000	7.20	34.0
7	1998-08-16	1998HIDASW09IDEx_grid.csv	36.3222	137.6327	5.13	6.0
1	2000-10-06	2000TOTTOR01IWAT_grid.csv	35.2690	133.3570	6.86	6.0

(b) Implement a function `process_stress(fi, fu)` that receives the name of an aftershock file `fi` and a function `fu`. `fu` receives six arguments (the stress components `s1,...,s6`), and returns a single value. `process_stress` returns a data frame with columns `x`, `y`, `fu` and `aftershock`, with values from the corresponding aftershock file, and the outputs of the function `fu` for each row. Apply it to the event `2001BHUJIN01YAGI` with $f(s_1, \dots, s_6) = \sum_i |s_i|$, and display the first few rows of the result with `head`.

```python
import pandas as pd
import numpy as np

def process_stress(fi, fu):
    df_file = pd.read_csv(fi)
    # Adds a new column with the result from fu with given inputs s1,s2,..
    df_file["fu"] = df_file.apply(lambda row: fu(row["s1"], row["s2"],
 ↪  row["s3"], row["s4"], row["s5"], row["s6"]), axis=1)
    # Returns the DataFrame with only the selected columns
    return df_file[["x", "y", "fu", "aftershock"]]

def fu(s1, s2, s3, s4, s5, s6):
    return np.sum(np.abs([s1, s2, s3, s4, s5, s6]))

df = process_stress("aftershocks/2001BHUJIN01YAGI_grid.csv", fu)
df.head()
```

	x	y	fu	aftershock
0	547594.439578	2.503102e+06	86315.950044	0.0
1	552594.439578	2.503102e+06	93082.647564	0.0
2	557594.439578	2.503102e+06	99307.713303	0.0
3	562594.439578	2.503102e+06	104760.596201	0.0
4	567594.439578	2.503102e+06	109217.784340	0.0

(c) Create new dataframe with four columns, `file` (from `selectedEvents.csv`), `lat`, `lon`, and `moment` (from `all_events.csv`). Sort it by the column `file` and display the first few rows with `head`.

```python
import pandas as pd

selected_events = pd.read_csv("selectedEvents.csv", index_col="id") # Sets id
 ↪  as the index column
all_events = pd.read_csv("all_events.csv", index_col="id")

def extract_info(row_id):
    specific_event = all_events.loc[row_id] # Locates a selected event in
 ↪  all_events
    return {
        "file": selected_events.loc[row_id, "file"],
        "lat": specific_event["lat"],
        "lon": specific_event["lon"],
        "moment": specific_event["moment"],
    }

# Applies extract_info to each row
data = list(map(extract_info, selected_events.index))

df = pd.DataFrame(data)
sorted_df = df.sort_values("file") # Sorts by file
sorted_df.head()
```

	file	lat	lon	moment
2	1989LOMAPR01WALD_grid.csv	37.0410	-121.8830	2.890000e+19
5	1994NORTHR01WALD_grid.csv	34.2130	-118.5370	1.750000e+19
9	1997ZIRKUH01SUDH_grid.csv	33.8200	59.8000	7.640000e+19
7	1998HIDASW09IDEx_grid.csv	36.3222	137.6327	5.660000e+16
1	2000TOTTOR01IWAT_grid.csv	35.2690	133.3570	2.160000e+19

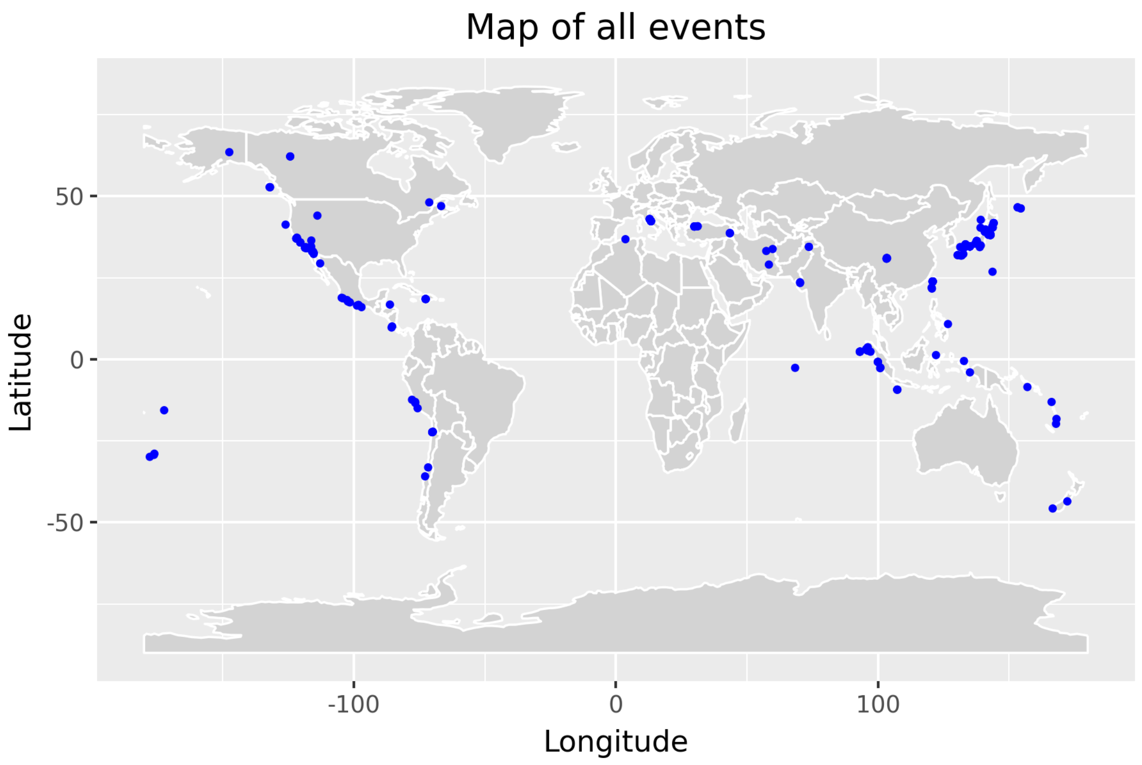
Question 2

Note: if you are not familiar with any of the *geoms* required for this question, check the documentation of `ggplot` or `plotnine`, either with the RStudio help or searching the online documentation.

(a) Use `geom_map` (Python) or `geom_sf` (R) and the file `worldMap.shp` to plot a map of all the events in `all_events.csv`, a point for each event. Note: in R, you will need to read `worldMap.shp` first using the function `st_read` from the library `sf`; in Python, read `worldMap.shp` using `geopandas.read_file`.

```
import geopandas as gpd
import pandas as pd
from plotnine import *

all_events = pd.read_csv("all_events.csv") # Turns all events into a
 ↪  DataFrame
countries = gpd.read_file("worldMap.shp") # Turns world map into a geo
 ↪  DataFrame

p = (
ggplot(countries)
```

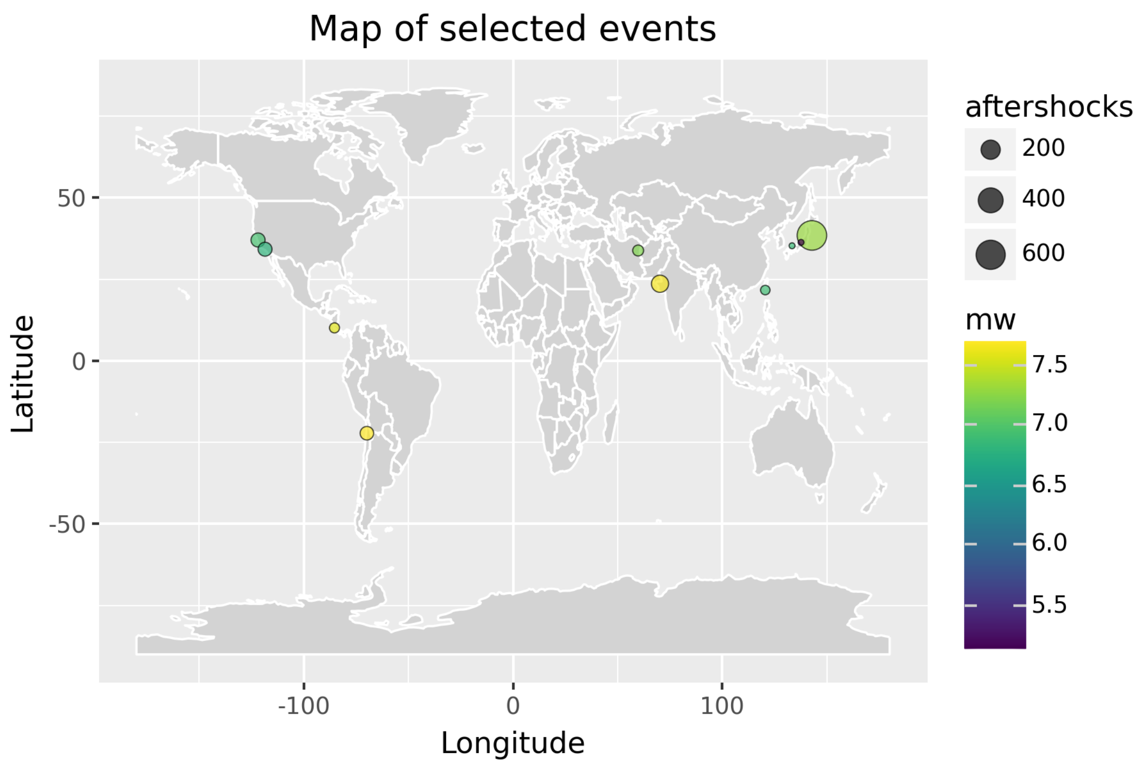
```
    + geom_map(fill="lightgray", color="white")
    + geom_point(all_events, aes(x="lon", y="lat"), color="blue", size=0.75)
    + labs(x="Longitude", y="Latitude", title="Map of all events") # Labels
    + theme(figure_size=(6, 4))
)

p.show()
```



(b) Use `geom_map` (Python) or `geom_sf` (R) with `worldMap.shp` to plot a map with a point for each event in `selectedEvents.csv`. Use colour to represent the intensity, and size to represent the number of associated aftershocks. Note: in R, you will need to read `worldMap.shp` first using the function `st_read` from the library `sf`; in Python, read `worldMap.shp` using `geopandas.read_file`.

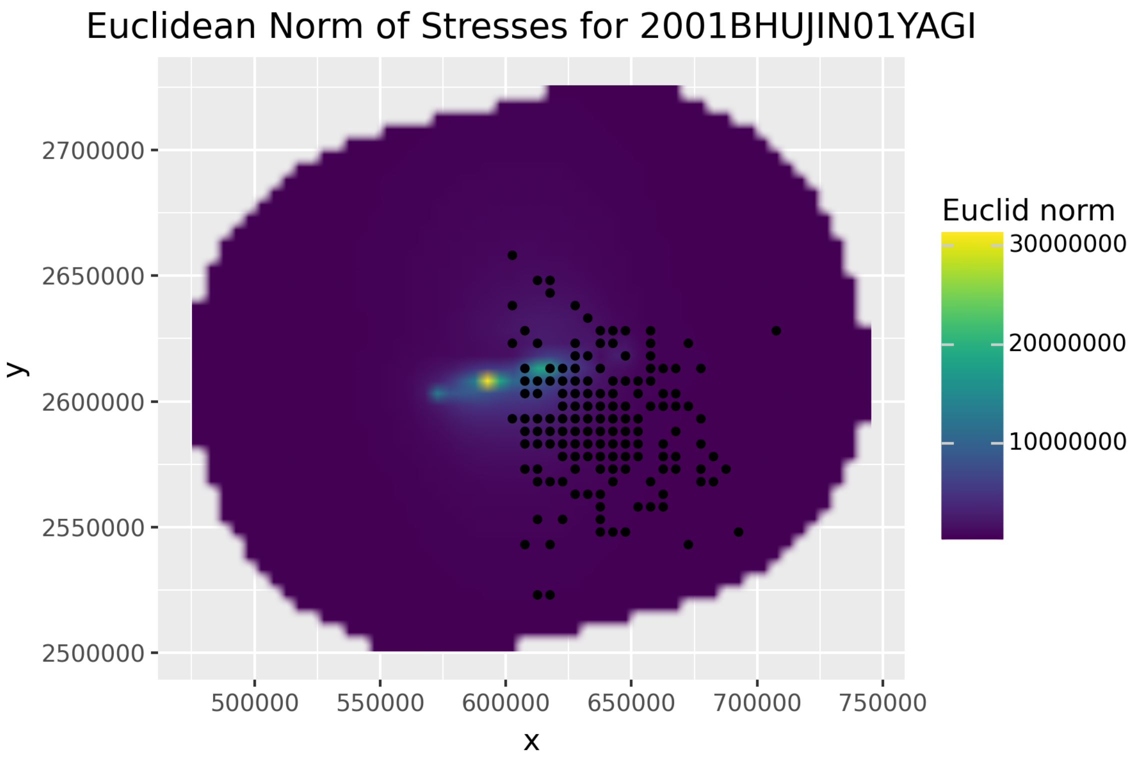
```
import geopandas as gpd
import pandas as pd
import numpy as np
from plotnine import *
```

```
selected_events = pd.read_csv("selectedEvents.csv", index_col="id") # Sets id
 ↪  as the index
all_events = pd.read_csv("all_events.csv", index_col="id")
countries = gpd.read_file("worldMap.shp")

def extract_info(row_id):
    specific_event = all_events.loc[row_id] # Locates a selected event in
 ↪  all_events
    aftershock_data = pd.read_csv("aftershocks/" + specific_event.name +
 ↪  "_grid.csv")
    aftershocks_count = aftershock_data["aftershock"].sum() # Sums the
 ↪  aftershock column
    return {
        "lat": specific_event["lat"],
        "lon": specific_event["lon"],
        "mw": specific_event["mw"],
        "aftershocks": aftershocks_count
    }

# Applies extract_info to each row
data = list(map(extract_info, selected_events.index))

# Convert the list of dictionaries to a DataFrame
df = pd.DataFrame(data)
# Sorted by aftershock quantity - descending so we have no big points plotted
 ↪  over small points
sorted_df = df.sort_values(by="aftershocks", ascending=False)

p = (
ggplot()
    + geom_map(data=countries, fill="lightgray", color="white")
    # Plots the points from sorted_df where size and fill colour correlate to
     ↪  number of aftershocks and the intensity - Given a black border to
     ↪  make more distinct
    + geom_point(data=sorted_df, mapping=aes(x="lon", y="lat",
     ↪  size="aftershocks",fill="mw"), color="black", stroke=0.25, alpha=0.7)
    + labs(x="Longitude", y="Latitude", title="Map of selected events") #
     ↪  Labels
    + theme(legend_position="right", figure_size=(6, 4))
)

p.show()
```



(c) Plot the Euclidean norm of the stresses for `2001BHUJIN01YAGI` at the (x, y) coordinates in the corresponding file, using colour for the value of the norm, and include black points at the location of the aftershocks.

```
import pandas as pd
from plotnine import *
import numpy as np

# Calcs the Euclidean norm with given data from s1, s2,..., s6
def euclid_norm(df, cols = ["s1", "s2", "s3", "s4", "s5", "s6"]):
    return np.linalg.norm(df[cols].to_numpy(), axis=1)

specific_events = pd.read_csv("aftershocks/2001BHUJIN01YAGI_grid.csv")
# Adds a new column called norm with the Euclidean norm of the stresses
specific_events["Euclid norm"] = euclid_norm(specific_events)

p = (
    ggplot(specific_events)
    + geom_raster(aes(x="x", y="y", fill="Euclid norm"), interpolate=True)
    # Overlay points for aftershocks
    + geom_point(aes(x="x", y="y"),
     ↪  data=specific_events[specific_events["aftershock"] == 1.0],
     ↪  color="black", size=1)
    + labs(title="Euclidean Norm of Stresses for 2001BHUJIN01YAGI") # Labels
    + theme(figure_size=(6, 4))
)

p.show()
```



Question 3

We are going to model the probability of an aftershock with

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}, \tag{1}$$

where x will be a variable that we use to make the prediction. We are going to find the *best* parameter values $\beta_0, beta_1$ to model the data of a given main event, by finding the values of

β_0, β_1 that minimise

$$f(\beta_0, \beta_1) = \sum_k -y_k \log(p(x_k; \beta_0, \beta_1)) - (1 - y_k) \log(1 - p(x_k, \beta_0, \beta_1)). \tag{2}$$

This expresion corresponds to the negative log-likelihood of a model. Here $y_k \in \{0, 1\}$ is the observed outcome (no aftershock or aftershock present), and x_k is our *predictor* variable, that we will define based on information about the earthquake.

(a) Implement a function `fit(X,Y,gamma)` that receives the vectors with values x_k and y_k, and a step `gamma` for the gradient descent method, and returns β_0, β_1 obtained the gradient descent method with starting point $(0, 0)$. Test it by computing the values for `TODO` using the Euclidean norm of the stresses as X and the value of the column `aftershock` as Y.

```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings("ignore") # Removes the exp overflow warning

# Prob. of an aftershock func.
def p_aftershock(x, beta0, beta1):
    z = beta0 + beta1 * x
    exp_neg_z = np.exp(-z)
    return 1 / (1 + exp_neg_z)

# Negative log-likelihood func.
def n_log_lik(X, Y, beta0, beta1):
    p = p_aftershock(X, beta0, beta1)
    # Usage of dot applies this calc over vectors rather than indiv. values
    return -(Y.dot(np.log(p)) - (1-Y).dot(np.log(1-p)))

# Returns the gradient of b0,b1
def gradient(X, Y, beta0, beta1):
    p = p_aftershock(X, beta0, beta1)
    grad_beta0, grad_beta1 = np.sum(p - Y), np.sum((p - Y) * X)
    return grad_beta0, grad_beta1

#Returns values for b0,b1 after we reach stopping cond.
def fit(X, Y, gamma):
    beta0, beta1 = 0, 0
    for _ in range(int(1/gamma)): # Stopping cond. 1/gamma
        grad_beta0, grad_beta1 = gradient(X, Y, beta0, beta1)
        beta0 -= gamma * grad_beta0
        beta1 -= gamma * grad_beta1
    return beta0, beta1

specific_data = pd.read_csv("aftershocks/2001BHUJIN01YAGI_grid.csv")
data_aftershocks = specific_data["aftershock"].to_numpy()
X = euclid_norm(specific_data) #Function defined in Q2C
Y = data_aftershocks

gamma = 0.001  # Step size for gradient descent
beta0, beta1 = fit(X, Y, gamma)
# I do not believe that this converges as intended - Issue with data
↪ potentially?
print("Optimal beta values (b0, b1 resp.):", beta0, beta1)
```

```
Optimal beta values (b0, b1 resp.): -328.3600000000003 -340753.5288408412
```

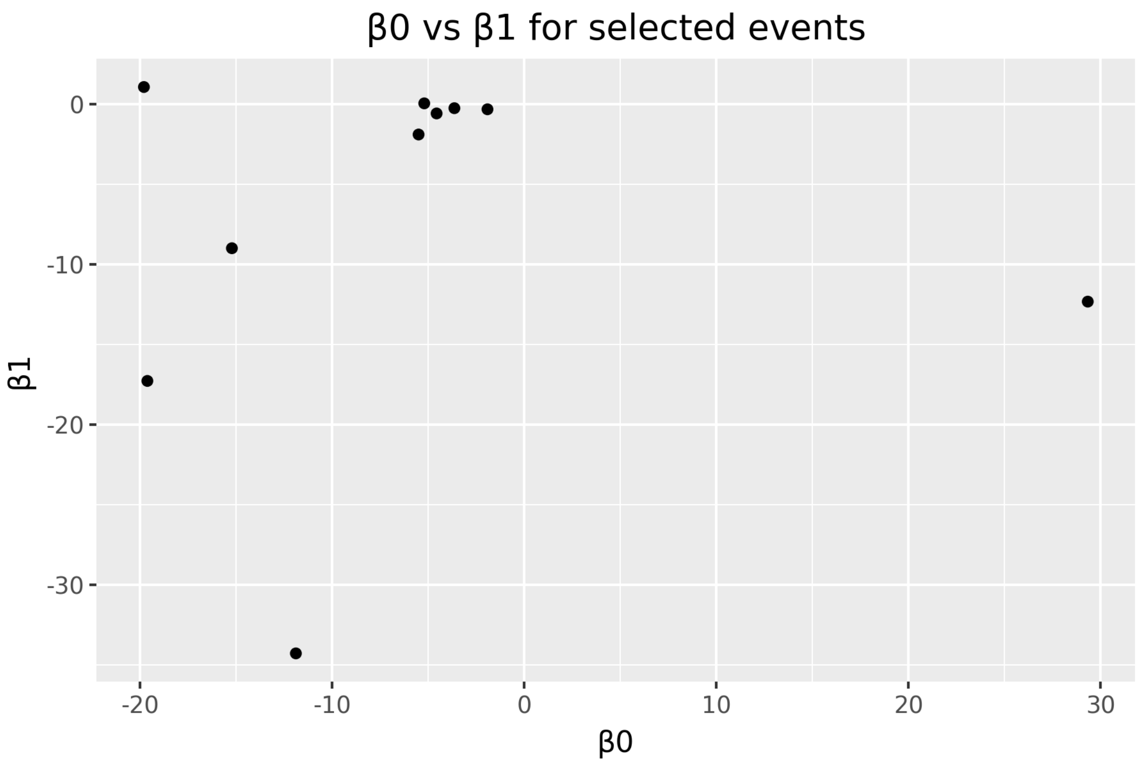
(b) Implement a function `fit_file(fi,fu,gamma)` that finds the optimal values of β_0, β_1 using gradient descent as before, using the data in the aftershock file `fi`, and the function `fu` on the stresses (defined as in Question 1b). Test it by computing the values for `TODO` using the Euclidean norm of the stresses as X and the value of the column `aftershock` as Y.

```
import numpy as np
import pandas as pd

def fit_file(fi, fu, gamma):
    specific_data = pd.read_csv(fi)
    data_aftershocks = specific_data["aftershock"].to_numpy() # To NumPy
↪ array
    X = euclid_norm(specific_data) # Function defined in Q2C
    Y = data_aftershocks
    return fit(X, Y, gamma) # Function defined in Q3A

gamma = 0.001  # Step size for gradient descent
beta0, beta1 = fit_file("aftershocks/2001BHUJIN01YAGI_grid.csv", fu, gamma)
print("Optimal beta values (b0, b1 resp.):", beta0, beta1)
```

```
Optimal beta values (b0, b1 resp.): -328.3600000000003 -340753.5288408412
```

(c) Implement a function factory `fit_file_factory(fu,gamma)` to fix the values of `fu` and `gamma` in `fit_file`. Compute the values of β_0, β_1 for all events in *selectedEvents*, using $f(s_1, \ldots, s_6) = \log(\sum_i |s_i|)$ and $gamma = 10^{-3}$. Plot the results with β_0 in the x-axis and β_1 in the y-axis, one point for each event.

```python
import pandas as pd
import numpy as np
from plotnine import *

# logfu is a function that takes 6 params and returns the log of the sum of
 ↪  the abs. stresses
logfu = lambda s1,s2,s3,s4,s5,s6: np.log(fu(s1,s2,s3,s4,s5,s6))

def fit_file(fi, fu, gamma):
    specific_data = pd.read_csv("aftershocks/" + fi)
    data_aftershocks = specific_data["aftershock"].to_numpy()
    # Applies logfu to each row and returns a NumPy array
    X = specific_data.apply(lambda row: logfu(row["s1"], row["s2"],
 ↪  row["s3"], row["s4"], row["s5"], row["s6"]), axis=1).to_numpy()
    Y = data_aftershocks
    return fit(X, Y, gamma) # Function defined in Q3A

def fit_file_factory(fu, gamma):
    # Returns fit_file which only takes 1 param (fi)
    return lambda fi: fit_file(fi, fu, gamma)

def plot_results(beta_df):
    p = (
    ggplot(beta_df, aes(x="beta0", y="beta1"))
        + geom_point()
        + labs(x=" 0", y=" 1", title=" 0 vs  1 for selected events") # Labels
        + theme(figure_size=(6, 4))
    )
    p.show()

selected_events = pd.read_csv("selectedEvents.csv")
gamma = 0.001
# fit_file_fixed is the version of fit_file that only takes 1 param (fi)
fit_file_fixed = fit_file_factory(fu, gamma) # Function fu defined in Q1B
beta_values = []
# Applies fit_file_fixed to each file path in selected_events
for file_path in selected_events["file"]:
    beta0, beta1 = fit_file_fixed(file_path)
    beta_values.append((beta0, beta1))

beta_df = pd.DataFrame(beta_values, columns=["beta0", "beta1"])
plot_results(beta_df)
```



Question 4

The logistic regression model from Question 3 can be extended to more variables, by defining the probability

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2)}}. \tag{3}$$

(a) Write a function `moment_distance(fi)` that receives the name of an aftershock file, and returns a dataframe with three columns: the mainshock seismic log-moment (log of `moment` in `all_events.csv`), the distance between the mainshock and the possible aftershock location computed (assume that the mainshock is at the centre of the grid of points in the aftershock file), and column with the presence/abscence of an aftershock. Use the column names `moment`, `distance`, `aftershock`, and note that the `moment` is the same for all the rows, since we are looking only at one mainshock event. Display the first few rows of the dataframe obtained by applying this function to TODO.

```python
import numpy as np
import pandas as pd

def moment_distance(fi):
    specific_event = pd.read_csv(fi)
    all_events = pd.read_csv("all_events.csv")
    file_name = fi.split("/")[-1] # Removes the aftershocks/ part of fi
    file_id = file_name.split("_")[0] # Removes the _grid.csv part of fi
    related_row = all_events[all_events["id"] == file_id]
    log_moments = np.log(related_row.iloc[0]["moment"])
    dists_from_mainshock = np.sqrt(specific_event["x"]**2 +
 ↪  specific_event["y"]**2).to_numpy()

    df = pd.DataFrame({"moment": log_moments,
                    "distance": dists_from_mainshock,
                    "aftershock": specific_event["aftershock"].to_numpy()})
    return df

moment_distance("aftershocks/2001BHUJIN01YAGI_grid.csv").head()
```

	moment	distance	aftershock
0	47.290076	2.562300e+06	0.0
1	47.290076	2.563373e+06	0.0
2	47.290076	2.564455e+06	0.0
3	47.290076	2.565547e+06	0.0
4	47.290076	2.566648e+06	0.0

(b) Implement a function `fit2(X1,X2,Y)` that minimises the negative log-likelihood function f in Question 3 and returns the values of $\beta_0, \beta_1, \beta_2$. Use `optim` (in R) or `scipy.optimize.minimize` in Python, and **do not** use the derivative of f. Obtain the values of $\beta_0, \beta_1, \beta_2$ for `2001BHUJIN01YAGI` using `moment` for x_1, `distance` for x_2 and `aftershock` for y.

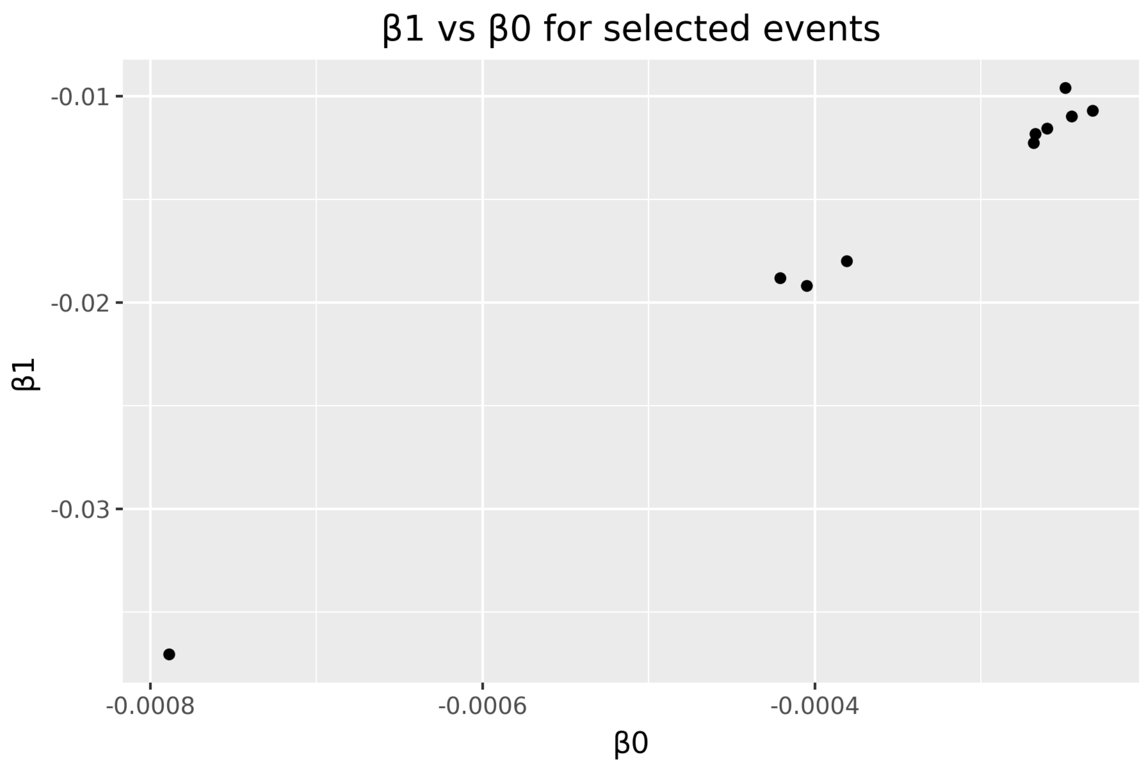
```python
import numpy as np
import pandas as pd
from scipy.optimize import minimize
```

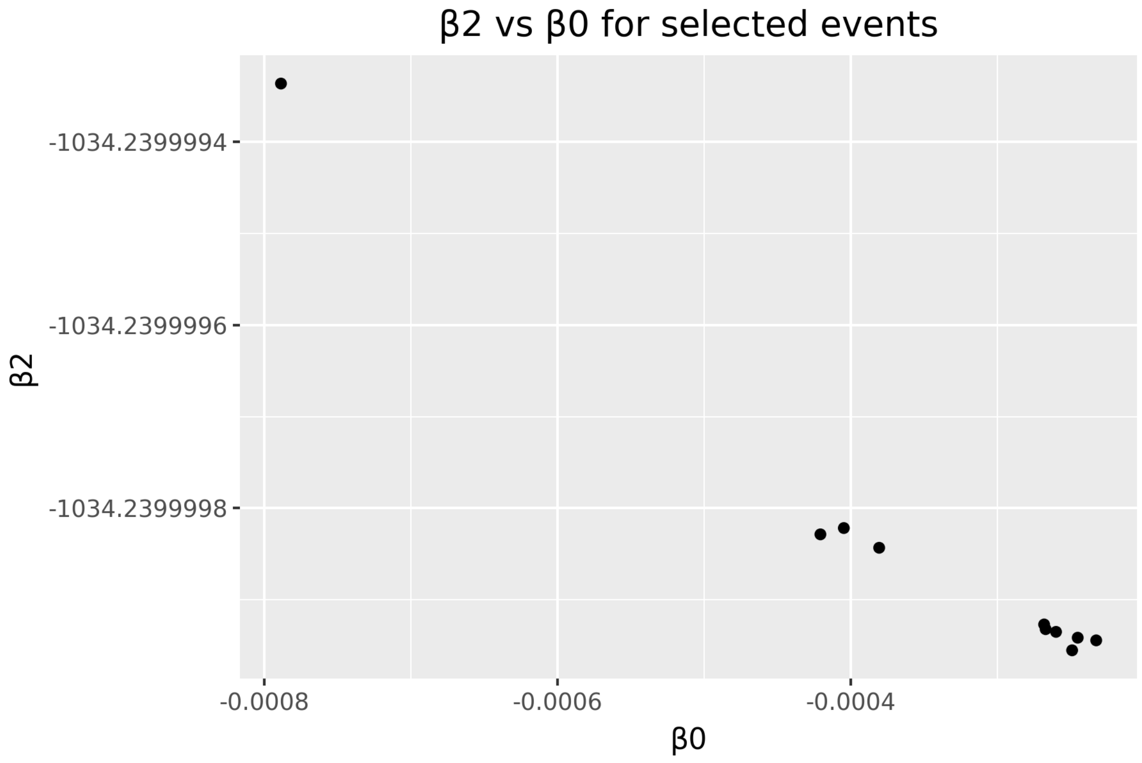
```python
# 2nd version of the negative log likelihood function
def n_log_lik2(beta, X1, X2, Y):
    beta0, beta1, beta2 = beta
    z = beta0 + beta1 * X1 + beta2 * X2
    p = 1 / (1 + np.exp(-z))
    # Usage of dot applies this calc over vectors rather than indiv. values
    return -(Y.dot(np.log(p)) + (1 - Y).dot(np.log(1 - p)))

def fit2(X1, X2, Y):
    beta_initial = np.zeros(3) # np.array([0, 0, 0])
    result = minimize(n_log_lik2, beta_initial, args=(X1, X2, Y))
    beta_optimal = result.x
    return beta_optimal

df = moment_distance("aftershocks/2001BHUJIN01YAGI_grid.csv") # Function
↪ defined in Q4A
X1 = df["moment"].to_numpy()
X2 = df["distance"].to_numpy()
Y = df["aftershock"].to_numpy()

beta_values = fit2(X1, X2, Y)
print("Optimal beta values (b0, b1, b2 resp.):\n", beta_values)
```

```
Optimal beta values (b0, b1, b2 resp.):
 [-3.80596928e-04 -1.79984614e-02 -1.03424000e+03]
```

(c) Implement a function `fit2_file(fi)` that returns the values of $\beta_0, \beta_1, \beta_2$ for the aftershock file `fi` using `moment` for x_1, `distance` for x_2 and `aftershock` for y. Plot the values of β_1 vs β_0 and β_2 vs β_0 in two separate plots, one point for each event in `selectedEvents.csv`.

```python
import pandas as pd
import numpy as np
from scipy.optimize import minimize
from plotnine import *

def fit2_file(fi):
    specific_data = moment_distance("aftershocks/" + fi) # Function defined
↪ in Q4A
    X1 = specific_data["moment"].to_numpy() # X1, X2, Y are all numpy arrays
↪ of the resp. columns
    X2 = specific_data["distance"].to_numpy()
    Y = specific_data["aftershock"].to_numpy()
    beta_initial = np.zeros(3) # np.array([0, 0, 0])
    result = minimize(n_log_lik2, beta_initial, args=(X1, X2, Y))
    return result.x

selected_events = pd.read_csv("selectedEvents.csv")
beta_values = []
# Applies fit2_file to each file path located in selected_events
for file_path in selected_events["file"]:
    beta_values.append(fit2_file(file_path))

beta_df = pd.DataFrame(beta_values, columns=["beta0", "beta1", "beta2"])
p1 = (
ggplot(beta_df, aes(x="beta0", y="beta1"))
    + geom_point()
    + labs(x=" 0", y=" 1", title=" 1 vs  0 for selected events") # Labels
    + theme(figure_size=(6, 4))
)
p1.show()
p2 = (
ggplot(beta_df, aes(x="beta0", y="beta2"))
    + geom_point()
    + labs(x=" 0", y=" 2", title=" 2 vs  0 for selected events") # Labels
    + theme(figure_size=(6, 4))
)
p2.show()
```


β1 vs β0 for selected events
-0.01
-0.02
-0.03
β1
-0.0008
-0.0006
-0.0004
β0


β2 vs β0 for selected events
-1034.2399994
-1034.2399996
-1034.2399998
β2
-0.0008
-0.0006
-0.0004
β0